《计算机体系结构》

实验报告

院　　系：计算机与信息学院

专　　业：物联网工程

年　　级：16 级

课程名称：计算机体系结构

学　　号：2016217725

姓　　名：吴逸韬

指导教师：李建华

2019 年 5 月 16 日

实验一、 流水线相关和指令调度

- 题目内容

流水线相关

1. 用 WinMIPS64 模拟器执行下列三个程序（任选一个）：

求阶乘程序 factorial.s

插入排序程序 isort.s

乘法计算程序 mult .s

分别以步进、连续、设置断点的方式运行程序，观察程序在流水线中的执行情况，观察 CPU 中寄存器和存储器的内容。熟练掌握 WinDLX 的操作和使用。

2. 用 MIPS64 汇编语言编写代码文件*.s，程序中应包括结构相关。用 WinMIPS64 运行该程序，通过模拟：

找出存在结构相关的指令对以及导致结构相关的部件；

记录由结构相关引起的暂停时钟周期数，计算暂停时钟周期数占总执行周期数的百分比；

论述结构相关对 CPU 性能的影响，讨论解决结构相关的方法。

3. 用 MIPS64 汇编语言编写代码文件*.s，程序中应包括数据相关。在不采用定向技术的情况下，用 WinMIPS64 运行存在数据相关的程序。记录数据相关引起的暂停时钟周期数以及程序执行的总时钟周期数，计算暂停时钟周期数占总执行周期数的百分比。

4. 在采用定向技术的情况下，用 WinMIPS64 再次运行程序。重复上述 3 中的工作，并计 算采用定向技术后性能提高的倍数。

指令调度

1．用指令调度技术解决流水线中的结构相关与数据相关

（1） 用 MIPS64 汇编语言编写代码文件*.s，程序中应包括数据相关与结构相关（你可以 自己设置各个功能单元的延迟时间）

（2） 用 WinMIPS64 运行你所写的程序。记录程序执行过程中各种相关发生的次数、发 生相关的指令组合，以及程序执行的总时钟周期数；

（3） 采用指令调度技术对程序进行指令调度，消除相关；

（4） 用 WinMIPS64 运行调度后的程序，观察程序在流水线中的执行情况，记录程序执 行的总时钟周期数；

（5） 根据记录结果，比较调度前和调度后的性能。论述指令调度对于提高 CPU 性能 的意义。

2. 用循环展开、寄存器换名以及指令调度提高性能

（1） 用 MIPS64 汇编语言编写代码文件*.s，程序中包含一个循环次数为 4 的整数倍 的简单循环；

（2） 用 WinMIPS64 运行该程序。记录执行过程中各种相关发生的次数以及程序执行的 时钟周期数；

（3） 将循环展开 3 次，将 4 个循环体组成的代码代替原来的循环体，并对程序做相应 的修改。然后对新的循环体进行寄存器换名和指令调度；

（4） 用 WinMIPS64 运行修改后的程序，记录执行过程中各种相关发生的次数以及程序 执行的总时钟周期数；

（5） 根据记录结果，比较循环展开、指令调度前后的性能。

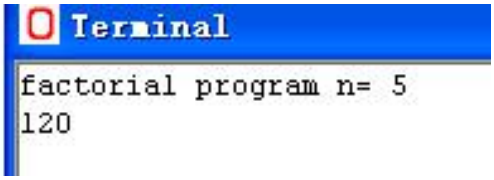
- 实验结果与分析

1、流水线相关

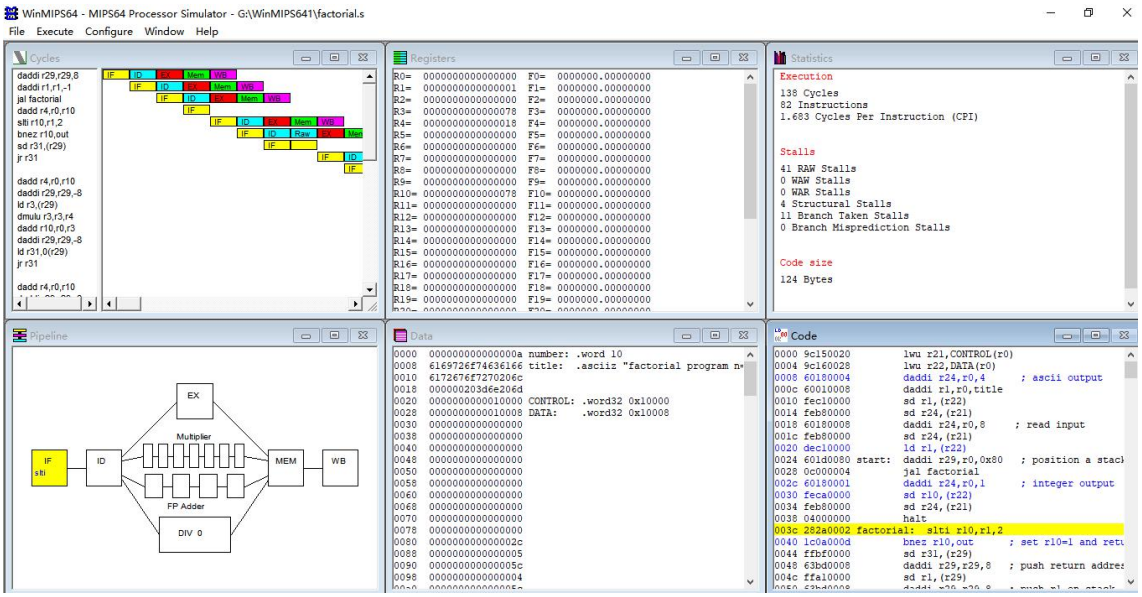
用 WinDLX 模拟器执行求阶乘程序 factorial.s 。该程序从标准输入读入一个整数，求其阶乘，然后将

结果输出。

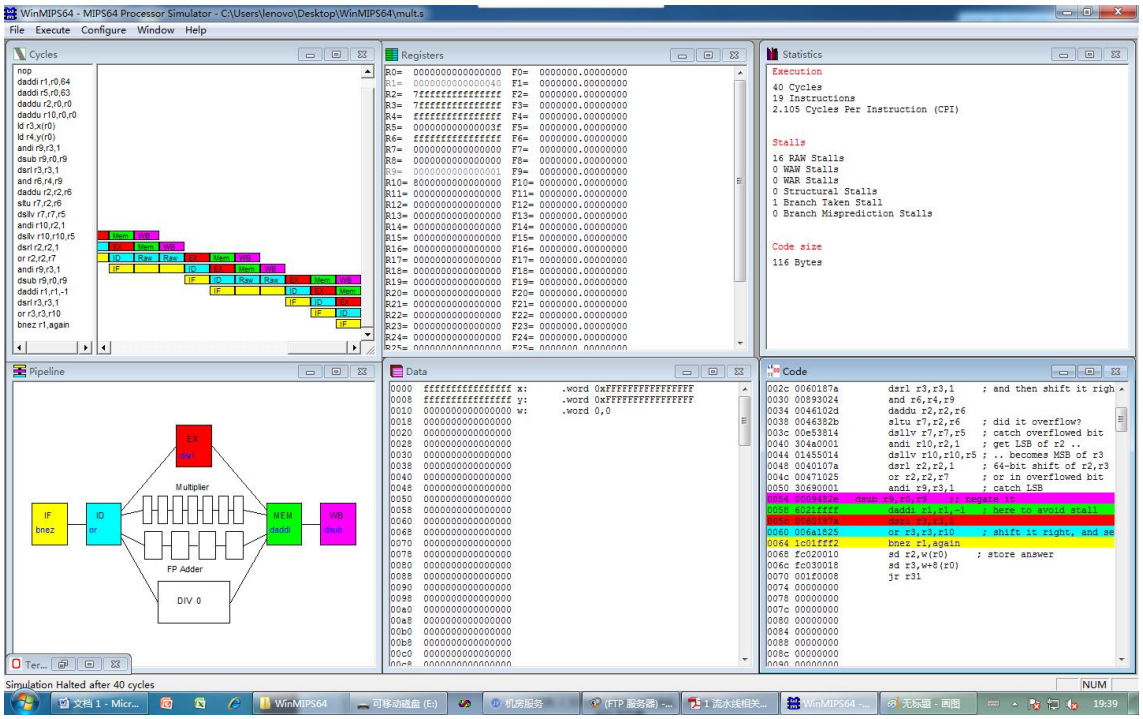
执行结果如下：



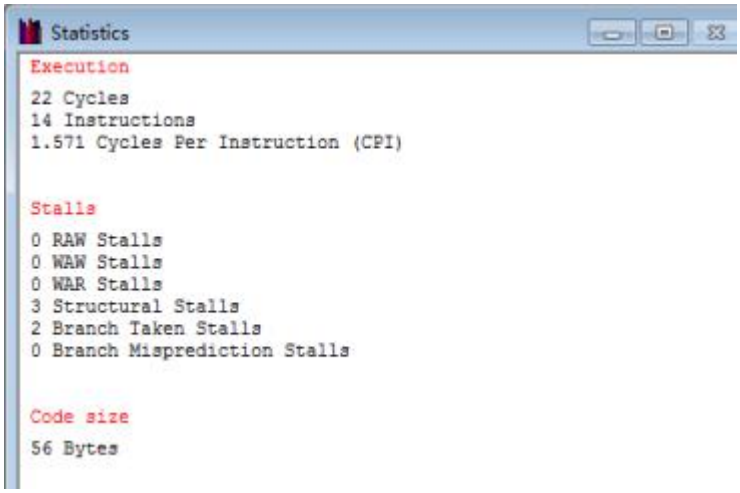
连续执行：



断点执行：



结构相关：



程序执行了 22 个周期，结构相关引起的时钟周期 structural stall 为 3 个。 暂停时钟周期数占总执行周期数的百分比为 13.7%。

为了避免结构相关，可以考虑采用资源重复的方法，比如，在流水线机器中设置相互独立的指令存储器和数据存储器，也可以将 cache 分割成指令 cache 和数据 cache。

数据相关：

不采用定向技术：

```
Execution
187 Cycles
63 Instructions
2.968 Cycles Per Instruction (CPI)

Stalls
109 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
11 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
32 Bytes
```

程序执行了 187 个周期，数据相关引起的时钟周期 RAW stall 为 109 个。 暂停时钟周期数占总执行周期数的百分比为 58%。

采用定向技术：

```
Execution
65 Cycles
43 Instructions
1.512 Cycles Per Instruction (CPI)

Stalls
13 RAW Stalls
0 WAW Stalls
0 WAR Stalls
3 Structural Stalls
2 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
68 Bytes
```

程序执行了 65 个周期，数据相关引起的时钟周期 RAW stall 为 13 个。 暂停时钟周期数占总执行周期数的百分比为 20%。

可见通过定向技术，减少了数据相关，缩短了程序的执行周期，整个性能为原来的 2.87 倍。

指令调度：

调度前：

```
Execution
36 Cycles
17 Instructions
2.118 Cycles Per Instruction (CPI)

Stalls
4 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
2 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
68 Bytes
```

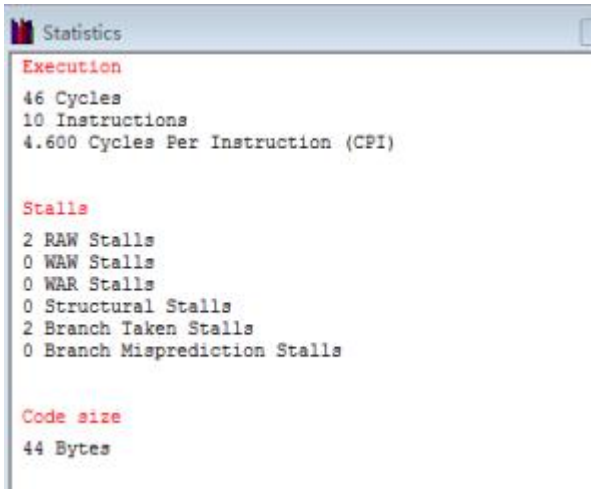
调度后：

```
Execution
30 Cycles
17 Instructions
1.765 Cycles Per Instruction (CPI)

Stalls
2 RAW Stalls
0 WAW Stalls
0 WAR Stalls
1 Structural Stall
2 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
68 Bytes
```

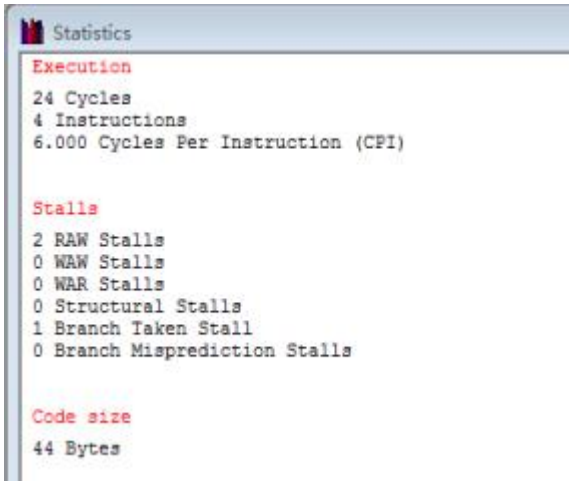
可以看出经过调度之后 ，运行周期从 36 减少到 30，而且减少了相关。

循环展开：

展开前：



展开后：



可以看出经过调度之后 ，运行周期从 46 少到 24，而且减少了相关。

- 关键程序代码

（1）求阶乘程序 factorial.s

```
.data
number: .word 10
title:  .asciiz "factorial program n= "
CONTROL: .word32 0x10000
DATA:    .word32 0x10008
        .text
        lwu r21,CONTROL(r0)
        lwu r22,DATA(r0)
        daddi r24,r0,4      ; ascii output
        daddi r1,r0,title
        sd r1,(r22)
        sd r24,(r21)
        daddi r24,r0,8      ; read input
        sd r24,(r21)
        ld r1,(r22)
start:  daddi r29,r0,0x80   ; position a stack in data memory, use r29 as stack pointer
        jal factorial
```

```
        daddi r24,r0,1      ; integer output
        sd r10,(r22)
        sd r24,(r21)
        halt
; parameter passed in r1, return value in r10
factorial:  slti r10,r1,2
        bnez r10,out      ; set r10=1 and return if r1=1
        sd r31,(r29)
        daddi r29,r29,8   ; push return address onto stack
        sd r1,(r29)
        daddi r29,r29,8   ; push r1 on stack
        daddi r1,r1,-1    ; r1 = r1-1
        jal factorial     ; recurse...
        dadd r4,r0,r10
        daddi r29,r29,-8
        ld r3,(r29)       ; pop n off the stack

        dmulu r3,r3,r4    ; multiply r1 x factorial(r1-1)
        dadd r10,r0,r3    ; move product r3 to r10
        daddi r29,r29,-8  ; pop return address
        ld r31,0(r29)
out:    jr r31
```

（2）数据相关：

```
LHI R2, (A>>16) & 0xFFFF
ADDUI R2, R2, A & 0xFFFF
LHI R3, (B>>16)&0xFFFF
ADDUI R3, R3, B&0xFFFF
loop:
LW R1, 0 (R2)
ADD R1, R1, R3
SW 0(R2), R1
LW R5, 0 (R1)
ADDI R5, R5, #10
ADDI R2, R2, #4
SUB R4, R3, R2
BNEZ R4, loop
TRAP #0
A: .word 0, 4, 8, 12, 16, 20, 24, 28, 32, 36
B: .word 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```

（3）结构相关：

```
ADDI R5, R5, 1
SUBI R4, R4, 1
AND R3, R3, R3
XOR R7, R7, R7
```

```
ADDI R8, R8, 1
ADDI R9, R9, 1
MULT R1,R5,R4
MULT R2,R3,R7
```

（4）循环展开：

```
LHI     R2, (A>>16)&0xFFFF
ADDUI   R2, R2, A&0xFFFF
LHI     R3, (B>>16)&0xFFFF
ADDUI   R3, R3, B&0xFFFF
ADDU    R4, R0, R3
SUBI    R4, R4, #8
SUBI    R4, R4, #8
SUBI    R4, R4, #8
SUBI    R4, R4, #8
TRAP    #0
A:      .double 1, 2, 3, 4
B:      .double 1, 2, 3, 4
```

- 实验心得

本次实验使用 WinMIPS64 进行。

通过本次实验，我掌握了 WinMIPS64 模拟器的使用，加深对计算机流水线基本概念的理解，进一步了解 MIPS 基本流水线各段的功能以及基本操作，同时加深对数据相关、结构相关的理解，了解相关对 CPU 性能的影响，学会了解决数据相关的方法，掌握如何使用定向技术来减少数据相关带来的暂停。加深对循环级并行性、指令调度技术、循环展开技术的理解，可以用循环展开、指令调度等技术来解决流水线中的相关问题以及了解了其对 CPU 性能的改进。

在使用指令调度消除相关的过程中，我也加深了对指令相关的影响的理解，同时大致掌握了各种消除相关的算法思想。

实验二、分支预测

- 题目内容

SimpleScalar 分支预测的实现方法：先进行分支方向探测，即是否采取分支（当然跳转指令和调用返回指令不用作这一步），接着是生成分支地址，对于调返指令，直接在 RAS 上作相关操作，普通分支指令则要利用 BTB 来进行地址探测，命中则生成地址。然后对两步综合，地址命中且分支预测为采取，返回分支目标地址；地址不命中且分支预测为采取，返回 1;只要分支预测为不采取，就返回 0。

重点分析针对条件分支指令的方向探测方法，主要有 6 种，三种静态：

taken,nottanken,perfect;三种动态：bimod,2-level,combined。静态的方法顾名思义，只是 perfect 这种，按它的原意是不预测，直接把真正采取的下一条指令填入 npc，而且它确实不需要调用。

对于三种动态方法，分别说明如下：

bimod 是最普通的，即采用一个 2bit 宽的分支方向预测表，按分支地址查找，2bit 分支预测器的判断和更新与课本上的一致。这种方式只有一个参数，就是分支预测表的长度。

2-level 要复杂一些，它采用两级表格式，第一级是分支历史表，存放各组分支历史寄存器的值，第二级是全局/局部分支模式表，（全局或局部应是由表长相对于分支历史寄存器的长决定），它存放各分支历史模式的 2bit 预测器。在判断时用当前分支指令对应的历史寄存器值去索引二级表得到相应预测器值。更新时，把当前分支的方向左移入历史寄存器，并对使用过的 2bit 预测器作更新。它有四个参数，前三个是一级表长度，二级表长度，历史寄存器宽度，最后一个是异或标志。如果为 1，则将历史寄存器的值与当前分支指令地址异或，用其结果再去索引二级模式表。

实验步骤

(1)进入 SimpleScalar 目录(simplesim-3.0)。

(2)用 sim-bpred 仿真器运行

spec95_little 中的测试程序：ccl, compress95, go, perl，或者从 SPEC2000 INT 中任选 4 个程序，分别采用 4 种不同的分支预测方法，即 bimod 方式，two-level adaptive 方式，always taken 方式，always not taken 方式，并对前两种分别使用下表中两种参数配置：分析仿真器输出的关于分支预测的统计参数集，填写表格，并对各仿真器的能力给出相应说明。

命令格式为：./sim-bpred {-option} executable_benchmark -argument

- 实验结果与分析

ccl.ss 程序

	Always taken	always not taken	bimod(512)	Bimod(1024)	Two level (1,1024,8,0)	Two level (1,64,6,1)
sim_total_insn	45903	45903	45617	45903	45903	45903
sim_total_refs	11678	11678	11578	11678	11678	11678
sim_num_branches	10032	10032	9983	10032	10032	10032
sim_elapsed_time	1	1	1	1	1	1
sim_inst_rate	45903	45903	45617	45903	45903	45903
sim_IPB	4.5757	4.5757	4.5695	4.5757	4.5757	4.5757
bpred_bimod.lookups	10032	10032	9983	10032	10032	10032
bpred_bimod.updates	10032	10032	9983	10032	10032	10032
bpred_bimod.addr_hits	6925	5053	8360	8403	8414	8414

bpred_bimod.dir_hirts	6925	5053	8613	8655	8666	8666
bpred_bimod.misses	3107	4979	1370	1377	1366	1366
bpred_bimod.jr_hits	679	679	672	676	676	676
bpred_bimod.jr_seen	679	679	674	679	679	679
bpred_bimod.jr_non_ras_hits.PP	679	679	3	3	3	3
bpred_bimod.jr_non_ras_seen.PP	679	679	4	4	4	4
bpred_bimod.bpred_addr_rate	0.6903	0.5037	0.8374	0.8376	0.8387	0.8387
bpred_bimod.bpred_dir_rate	0.6903	0.5037	0.8628	0.8627	0.8638	0.8638
bpred_bimod.bpred_jr_rate	1.0000	1.0000	0.9970	0.9956	0.9956	0.9956
bpred_bimod.bpred_jr_non_ras_rate.PP	1.0000	1.0000	0.7500	0.7500	0.7500	0.7500
bpred_bimod.retstack_pushes	0	0	674	679	679	679
bpred_bimod.retstack_pops	0	0	670	675	675	675
bpred_bimod.used_ras.PP	0	0	670	675	675	675
bpred_bimod.ras_hits.PP	0	0	669	673	673	673
bpred_bimod.ras_rate.PP	<error:divide by zero>	<error:divide by zero>	0.9985	0.9970	0.9970	0.9970

compress95.ss 程序

	Always taken	always not taken	bimod(512)	Bimod(1024)	Two level (1, 1024, 8, 0)	Two level (1, 64, 6, 1)
sim_total_insn	80432368	80432368	80432368	80432368	80432368	80432368
sim_total_refs	28767283	28767283	28767283	28767283	28767283	28767283
sim_num_branches	14361029	14361029	14361029	14361029	14361029	14361029
sim_elapsed_time	2	2	2	3	2	3
sim_inst_rate	40216148	40216148	40216148	26810789.3333	40216148	26810789.3333
sim_IPB	5.6007	5.6007	5.6007	5.6007	5.6007	5.6007
bpred_bimod.lookups	14361029	14361029	14361029	14361029	14361029	14361029
bpred_bimod.updates	14361029	14361029	14361029	14361029	14361029	14361029
bpred_bimod.addr_hits	10280057	9462801	12911908	12912448	12912662	5.6007
bpred_bimod.dir_hits	10280057	9462801	12912217	12912757	12912968	14361029
bpred_bimod.misses	4082972	4898228	1448812	1448272	1448061	14361029
bpred_bimod.jr_hits	2208777	2208777	2208652	2208652	2208652	5.6007

bpred_bimod.jr_seen	2208777	2208777	2208777	2208777	2208777	14361029
bpred_bimod.jr_non_ras_hits.PP	2208777	2208777	61	61	61	14361029
bpred_bimod.jr_non_ras_seen.PP	2208777	2208777	185	185	185	5.600
bpred_bimod.bpred_addr_rate	0.7158	0.6589	0.8991	0.8991	0.8991	0.8991
bpred_bimod.bpred_dir_rate	0.7158	0.6589	0.8991	0.8991	0.8991	0.8991
bpred_bimod.bpred_jr_rate	1.0000	1.0000	0.9999	0.9999	0.9999	0.9999
bpred_bimod.bpred_jr_non_ras_rate.PP	1.0000	1.0000	0.3297	0.3297	0.3297	0.3297
bpred_bimod.retstack_pushes	0	0	2208594	2208594	2208594	2208594
bpred_bimod.retstack_pops	0	0	2208593	2208593	2208593	2208593
bpred_bimod.used_ras.PP	0	0	2208592	2208592	2208592	2208592
bpred_bimod.ras_hits.PP	0	0	2208591	2208591	2208591	2208591
bpred_bimod.ras_rate.PP	<error:divide by zero>	<error:divide by zero>	1.0000	1.0000	1.0000	1.0000

go.ss 程序

	Always taken	always not taken	bimod(512)	Bimod(1024)	Two level (1,1024,8,0)	Two level (1,64,6,1)
sim_total_insn	920716	920716	920716	920716	920716	920716
sim_total_refs	217545	217545	217545	217545	217545	217545
sim_num_branches	150035	150035	150035	150035	150035	150035
sim_elapsed_time	1	1	1	1	1	1
sim_inst_rate	920716	920716	920716	920716	920716	920716
sim_IPB	6.1367	6.1367	6.1367	6.1367	6.1367	6.1367
bpred_bimod.lookups	150035	150035	150035	150035	150035	150035
bpred_bimod.updates	150035	150035	150035	150035	150035	150035
bpred_bimod.addr_hits	116995	55264	136551	136560	136560	136560
bpred_bimod.dir_hirts	116995	55264	137228	137229	137227	137227
bpred_bimod.misses	33040	94771	12807	12806	12808	12808
bpred_bimod.jr_hits	8929	8929	7729	7729	7729	7729
bpred_bimod.jr_seen	8929	8929	8929	8929	8929	8929
bpred_bimod.jr_non_ras_hits.PP	8929	8929	5074	5074	5074	5074

bpred_bimod.jr_non_ras_seen.PP	8929	8929	6274	6274	6274	6274
bpred_bimod.bpred_addr_rate	0.7798	0.3683	0.9101	0.9101	0.9102	0.9102
bpred_bimod.bpred_dir_rate	0.7798	0.3683	0.9146	0.9146	0.9146	0.9146
bpred_bimod.bpred_jr_rate	1.0000	1.0000	0.8656	0.8656	0.8656	0.8656
bpred_bimod.bpred_jr_non_ras_rate.PP	1.0000	1.0000	0.8087	0.8087	0.8087	0.8087
bpred_bimod.retstack_pushes	0	0	2660	2660	2660	2660
bpred_bimod.retstack_pops	0	0	2665	2665	2665	2665
bpred_bimod.used_ras.PP	0	0	2665	2665	2665	2665
bpred_bimod.ras_hits.PP	0	0	2665	2665	2665	2665
bpred_bimod.ras_rate.PP	<error:divide by zero>	<error:divide by zero>	1.0000	1.0000	1.0000	1.0000

perl.ss 程序

	Always taken	always not taken	bimod(512)	Bimod(1024)	Two level (1,1024,8,0)	Two level (1,64,6,1)
sim_total_insn	26153	26153	26153	26153	26153	26153
sim_total_refs	11326	11326	11326	11326	11326	11326
sim_num_branches	4145	4145	4145	4145	4145	4145
sim_elapsed_time	1	1	1	1	1	1
sim_inst_rate	26153	26153	26153	26153	26153	26153
sim_IPB	6.3095	6.3095	6.3095	6.3095	6.3095	6.3095
bpred_bimod.lookups	4145	4145	4145	4145	4145	4145
bpred_bimod.updates	4145	4145	4145	4145	4145	4145
bpred_bimod.addr_hits	2644	2116	3437	3449	3469	3469
bpred_bimod.dir_hirts	2644	2166	3700	3708	3728	3728
bpred_bimod.misses	1501	2029	445	437	417	417
bpred_bimod.jr_hits	247	247	239	239	239	239
bpred_bimod.jr_seen	247	247	247	247	247	247
bpred_bimod.jr_non_ras_hits.PP	247	247	1	1	1	1
bpred_bimod.jr_non_ras_seen.PP	247	247	6	6	6	6

bpred_bimod.bpred_addr_rate	0.6379	0.5105	0.8292	0.8321	0.8369	0.8369
bpred_bimod.bpred_dir_rate	0.6379	0.5105	0.8926	0.8946	0.8994	0.8994
bpred_bimod.bpred_jr_rate	1.0000	1.0000	0.9676	0.9676	0.9676	0.9676
bpred_bimod.bpred_jr_non_ras_rate.PP	1.0000	1.0000	0.1667	0.1667	0.1667	0.1667
bpred_bimod.retstack_pushes	0	0	245	245	245	245
bpred_bimod.retstack_pops	0	0	241	241	241	241
bpred_bimod.used_ras.PP	0	0	241	241	241	241
bpred_bimod.ras_hits.PP	0	0	238	238	238	238
bpred_bimod.ras_rate.PP	<error:divide by zero>	<error:divide by zero>	0.9876	0.9876	0.9876	0.9876

- 实验心得

在本次实验中，使用分支预测模拟器 sim-bpred，在 4 种预测器类型及不同的参数配置下运行测试程序，并比较、分析结果，使我加深对动态分支预测机制的理解，并了解各种分支预测实现方式的优劣。

实验三、缓存性能分析

- 题目内容

1. 安装和测试 SimpleScalar 模拟器(利用模拟器自带的测试程序进行测试)。

2. 在基本配置情况下运行 SPEC 2000 基准测试集下面的 4 个程序(请指明自己选的是哪些测试程序)统计 Cache 失效次数，并统计 L2 缓存的失效次数（注：配置二级缓存结构）。

3. 改变 Cache 容量（*2，*4，*8，*64），运行相同的测试程序，并统计 L2 缓存的失效次数计算失效率，并进行分析。

4. 改变 Cache 的相联度（2 路，4 路，8 路，16 路，64 路），运行 1 中所选择的测试程序，并统计 L2 缓存的失效次数计算失效率，并进行分析。

5. 改变 Cache 块大小（*2，*4，*8，*64），运行 1 中所选择的测试程序，并统计 L2 缓存的失效次数计算失效率，并进行分析。

- 实验结果与分析

改变 cache 容量：

测试的程序是 simplescalar 自带的测试程序 test-math。

.运行的界面如下图所示：

cache 容量*2 时，

```
[root@PC2012100520KZC root]# cd /root/simplescalar/simplesim-3.0/tests-pisa/bin.little
[root@PC2012100520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:32:32:2:l test-mat
h
```

```
dl1.accesses                 57466 # total number of accesses
dl1.hits                     56071 # total number of hits
dl1.misses                    1395 # total number of misses
dl1.replacements              1331 # total number of replacements
dl1.writebacks                 827 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.0243 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.0232 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0144 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

cache 容量*4 时，

```
[root@PC2012100520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:64:32:2:l test-mat
h
```

```
dl1.accesses                 57466 # total number of accesses
dl1.hits                     56626 # total number of hits
dl1.misses                     840 # total number of misses
dl1.replacements               712 # total number of replacements
dl1.writebacks                 548 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.0146 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.0124 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0095 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

cache 容量*8 时，

```
[root@PC201210520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:128:32:2:l test-ma
th

dl1.accesses                 57466 # total number of accesses
dl1.hits                     56797 # total number of hits
dl1.misses                     669 # total number of misses
dl1.replacements               413 # total number of replacements
dl1.writebacks                 359 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.0116 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.0072 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0062 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

cache 容量为*64 时，

```
[root@PC201210520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:1024:32:2:l test-m
ath

dl1.accesses                 57466 # total number of accesses
dl1.hits                     56924 # total number of hits
dl1.misses                     542 # total number of misses
dl1.replacements                 0 # total number of replacements
dl1.writebacks                   0 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.0094 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

结果分析：

由以上截图可得，cache 容量为

*2：失效次数 1395 次，失效率 2.43%

*4：失效次数 840 次，失效率 1.46%

*8：失效次数 669 次，失效率 1.16%

*64：失效次数 542 次，失效率 0.94%

即随着 cache 容量的不断增大，其失效次数和失效率在一定程度上有所减小，原因是增大了 cache 容量后，会减少了容量失效。但当容量增大到一定值后，失效率不再减小。

改变 Cache 的相联度：

测试的程序是 simplescalar 自带的测试程序 test-fmath。

运行的界面如下图所示：

相联度为 2 路时：

```
[root@PC201210520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:512:32:1:l test-fm
ath

dl1.accesses                 16639 # total number of accesses
dl1.hits                     16147 # total number of hits
dl1.misses                     492 # total number of misses
dl1.replacements                83 # total number of replacements
dl1.writebacks                  73 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.0296 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.0050 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0044 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

相联度为 4 路时：

```
[root@PC201210052OKZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dll dll:256:32:2:l test-fm
ath

dll.accesses                 16639 # total number of accesses
dll.hits                     16147 # total number of hits
dll.misses                     492 # total number of misses
dll.replacements                76 # total number of replacements
dll.writebacks                  68 # total number of writebacks
dll.invalidations                0 # total number of invalidations
dll.miss_rate               0.0296 # miss rate (i.e., misses/ref)
dll.repl_rate               0.0046 # replacement rate (i.e., repls/ref)
dll.wb_rate                 0.0041 # writeback rate (i.e., wrbks/ref)
dll.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

相联度为 8 路时：

```
[root@PC201210052OKZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dll dll:128:32:4:l test-fm
ath

dll.accesses                 16639 # total number of accesses
dll.hits                     16163 # total number of hits
dll.misses                     476 # total number of misses
dll.replacements                28 # total number of replacements
dll.writebacks                  22 # total number of writebacks
dll.invalidations                0 # total number of invalidations
dll.miss_rate               0.0286 # miss rate (i.e., misses/ref)
dll.repl_rate               0.0017 # replacement rate (i.e., repls/ref)
dll.wb_rate                 0.0013 # writeback rate (i.e., wrbks/ref)
dll.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

相联度为 16 路时：

```
[root@PC201210052OKZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dll dll:64:32:8:l test-fma
th
dll.accesses                 16639 # total number of accesses
dll.hits                     16167 # total number of hits
dll.misses                     472 # total number of misses
dll.replacements                10 # total number of replacements
dll.writebacks                   7 # total number of writebacks
dll.invalidations                0 # total number of invalidations
dll.miss_rate               0.0284 # miss rate (i.e., misses/ref)
dll.repl_rate               0.0006 # replacement rate (i.e., repls/ref)
dll.wb_rate                 0.0004 # writeback rate (i.e., wrbks/ref)
dll.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

相联度为 64 路时：

```
[root@PC201210052OKZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dll dll:8:32:64:l test-fma
th
dll.accesses                 16639 # total number of accesses
dll.hits                     16169 # total number of hits
dll.misses                     470 # total number of misses
dll.replacements                 0 # total number of replacements
dll.writebacks                   0 # total number of writebacks
dll.invalidations                0 # total number of invalidations
dll.miss_rate               0.0282 # miss rate (i.e., misses/ref)
dll.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
dll.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
dll.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

结果分析：

由以上截图可得，cache 相连度为

2 路：失效次数 492 次，失效率 2.96%

4 路：失效次数 492 次，失效率 2.96%
8 路：失效次数 476 次，失效率 2.86%
16 路：失效此时 472 次，失效率 2.84%
64 路：失效次数 470 次，失效率 2.84%

可以分析得出，随着 cache 相联度的增大，各程序中 cache 失效率均大体呈下降趋势。因为随着相联度的提升，冲突失效会减小，却也会增大容量失效。

改变 Cache 块大小：
块大小*2 时：

```
[root@PC2012100520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:2048:8:2:l test-p
intf
dl1.hits                     529374 # total number of hits
dl1.misses                     2050 # total number of misses
dl1.replacements                  2 # total number of replacements
dl1.writebacks                    0 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0039 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0000 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
```

块大小*4 时：

```
[root@PC2012100520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:1024:16:2:l test-p
rintf
dl1.accesses                 531424 # total number of accesses
dl1.hits                     530363 # total number of hits
dl1.misses                     1061 # total number of misses
dl1.replacements                  2 # total number of replacements
dl1.writebacks                    0 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0020 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0000 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
```

块大小*8 时：

```
[root@PC2012100520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:512:32:2:l test-pr
intf
dl1.accesses                 531424 # total number of accesses
dl1.hits                     530864 # total number of hits
dl1.misses                      560 # total number of misses
dl1.replacements                  3 # total number of replacements
dl1.writebacks                    0 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0011 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0000 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
```

块大小*64 时：

```
[root@PC2012100520KZC bin.little]# /root/simplescalar/simplesim-3.0/sim-cache -cache:dl1 dl1:256:64:2:l test-pr
intf
```

```
dl1.accesses                 531424 # total number of accesses
dl1.hits                     531129 # total number of hits
dl1.misses                      295 # total number of misses
dl1.replacements                  2 # total number of replacements
dl1.writebacks                    0 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0006 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0000 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
```

结果分析：

当块大小为：

8byte：失效次数为 2050 次，失效率 0.39%

16byte：失效次数为 1061 次，失效率 0.20%

32byte：失效次数为 560 次，失效率 0.11%

64byte：失效次数为 295 次，失效率 0.06%

分析得出，在一定范围内，增大cache 块大小的确能够有效降低失效率，因为增加块大小会减少强制性失效，但当块大小增大到一定值时，失效率将增大。出现这种现象的原因是在增大块大小的同时，块的数量在随之减少，所以会增加冲突失效。

- 实验心得

本次实验使用 SimpleScalar 模拟器通过本次实验，我加深了对 Cache 的基本概念、基本组织结构以及基本工作原理的理解，了解了 Cache 的容量、相联度、块大小对 Cache 性能的影响，掌握了降低 Cache 失效率的各种方法，以及这些方法对 Cache 性能提高的好处，同时也理解了 Cache 失效的产生原因以及 Cache 的三种失效，对 cache 这一结构的理解更加深入。

- 成绩评定

指导老师：李建华