SimpleScalar 实验报告 2

1.

分析报告中提供的 HelloWorld 测试程序的测试结果，标注各指标的含义，同时说明各个工具起到作用，分析运行 HelloWorld 时 bin/sslittle-na-sstrix-gcc hello.c simplesim-3.0/sim-safe a.out 两条命令的含义及交叉编译概念。

1.1 测试结果中各指标的含义如下：

Sim_num_insn 执行的指令总数 7855
Sim_num_refs 执行的加载和存储的总数 4306
Sim_elapsed_time 模拟的总时间，以秒计 1
Sim_inst_rate 模拟速度（指令/秒）7855.0000
Ld_text_base 程序正文（代码）的段基址 0x00400000
Ld_text_size 程序正文（代码）的大小 以字节为单位 70144
Ld_data_base 程序初始化的数据段基址 0x10000000
Ld_data_size 程序初始化的.data 和未初始化的.bss 的大小 以字节为单位 8192
Ld_stack_base 程序堆栈段基址 堆栈的最高地址 0x7fffc000
Ld_stack_size 程序初始化堆栈大小 16384
Ld_prog_entry 程序入口地址 0x00400140
Ld_environ_base 程序环境基址地址 0x7fff8000
Ld_target_big_endian 目标可执行的字节次序 如果为大尾，则非零 0
Mem.page_count 分配的总页数 26
Mem.page_mem 分配的内存页面大小 104K
Mem.ptab_misses 一级页表缺页总数 26
Mem,ptab_accesses 访问页表总数 475078
Mem.ptab_miss_rate 一级页面失效率 0.0001

1.2 分析各个工具起到的作用 ：（主要是依据各个工具自带的说明及部分网上资料）

查看 Simplescalar\simplesim-3.0\目录，发现有如下几个模拟工具，现对各工具的功能作简要分析：

Sim-fast: 最快速、基本的模拟工具，仅做functional simulation，假设指令依次执行，没有cache也没有debug功能，多用于初期开发测试使用。默认情况下，这个模拟器不执行指令错误检查，结果，任何错误都会作为模拟器运行错误显示，可能导致sim-fast执行不正确或当掉。

Sim-safe: 在工具集中，是最简单的最友好的模拟器，检查所有的指令错误，不讲究速度。

Sim-bpred: 实现一个分支预测分析器。

Sim-cache: 这个工具实现cache模拟功能，为用户选择的cache和快表设置生成cache统计，其中可能包含两级指令和数据cache ，还有一级指令和数据快表，不会生成时间信息。

Sim-eio: 这个模拟器支持生成外部事件跟踪（EIO traces）和断点文件。外部事件跟踪俘获程序的执行，并且允许被打包到一个单独的文件，以备以后的再次执

行。这个模拟器也提供在外部事件跟踪执行中在任意一点做断点。断点文件可被用于在程序运行中启动 simplescalar 模拟器。

Sim-outorder: 最完整的工具。支持依序和乱序执行，branch predictor，memory hierarchy，function unit 个数等参数设定。这个模拟器追踪潜在的所有流水（pipeline）操作。

Sim-profile: 也叫functional simulation，但提供较完整的模拟参数，可依照使用者之设定，决定所要模拟之项目，如instruction classes and addresses、text symbols、memory accesses、branches and data segment symbols 以方便使用者整理收集数据材料。

1.3 分析两条命令的含义

bin/sslittle-na-sstrix-gcc hello.c指令是使用simplescalar附带的编译器编译hello.c文件，生成可执行文件a.out，这种可执行文件并不是通常意义下的可执行文件，它的可执行性是相对于模拟器程序而已的。**（在这里我们猜测生成的可执行文件默认为a.out，因为在下条指令中，我们试着把a.out改为其他的名字，都无法找到文件）回答：是的a.out是默认生成的，bin/sslittle-na-sstrix-gcc hello_world.c -I/usr/include -o hello_world.out 类似的命令是可以生成相应名字的out文件的。在一般的模拟器中，尤其是开源的，生成的文件名都是a.out，如NS-2。**

simplesim-3.0/sim-safe a.out指令是利用sim-safe模拟器来模拟运行刚才生成的可执行文件a.out，并列出运行的测试结果和各项指标。

1.4 交叉编译的概念

交叉编译：简单地说，就是在一个平台上生成另一个平台上的可执行代码。这里需要注意的是所谓平台，实际上包含两个概念：体系结构（Architecture）、操作系统（Operating System）。同一个体系结构可以运行不同的操作系统；同样，同一个操作系统也可以在不同的体系结构上运行。在进行嵌入式系统的开发时，运行程序的目标平台通常具有有限的存储空间和运算能力，比如常见的 ARM 平台，其一般的静态存储空间大概是 16 到 32 MB，而 CPU 的主频大概在 100MHz 到 500MHz 之间。这种情况下，在 ARM 平台上进行本机编译就不太可能了，这是因为一般的编译工具链（compilation tool chain）需要很大的存储空间，并需要很强的 CPU 运算能力。为了解决这个问题，交叉编译就应运而生了。通过交叉编译工具，我们就可以在 CPU 能力很强、存储控件足够的主机平台上（比如 PC 上）编译出针对其他平台的可执行程序。

2. 撰写分析SimpleScalar Simulator模拟程序的整个流程的分析报告。

（参考《处理器体系结构模拟器SimpleScalar分析与优化》）

SimpleScalar Simulator模拟程序的工作过程如图所示。

现以sim-fast工具为例，来介绍一下模拟程序的流程：

Sim-fast是执行速度最快，最不关心模拟过程细节信息的子模拟器程序。它采用顺序执行指令的方式，没有指令并行；不支持cache的使用，也不进行指令正确性检查，由程序员保证每条指令的正确性；不支持模拟器本身内嵌的Dlite！调试器（类似于gdb调试器）。为了模拟器的速度优化，在缺省情况下，sim-fast模拟器不进行时间统计，不对指令的有关信息（ 如指令总数及访存指令数目）进行统计。当然，可以修改模拟器源程序，通过改变其设

置，使模拟器更加符合设计人员的需求。

（1）模拟器初始化

模拟器执行的入口点是hello.c文件中定义的main（）主函数。当模拟器开始执行时，首先执行模拟器初始化工作，它主要包括：

1、显示版本信息。

2、处理用户对模拟器的配置选项。simplescalar模拟器提供了许多选项供用户设置，如配置cache的大小及结构、前瞻执行策略的选择等，从而方便用户得到关心的信息。

3、存储器的初始化。simplescalar模拟器提供了2GB的虚存空间，用来存放被模拟程序的正文段和数据段等。执行初始化后，被模拟程序的目标代码全部载入模拟器的内存中。

4、寄存器初始化。simplescalar提供了32个整数、32个单精度浮点和16个双精度浮点寄存器，另有六个控制寄存器。

5、相应子模拟器的初始化。五个子模拟器的初始化复杂程度不一，实现的功能也不尽相同。

（2）执行模拟。

运行时模拟器的sim-main（）函数：首先将被模拟程序的全部正文段译码，然后从程序的入口点处开始执行模拟。Sim_main（）程序是每个子模拟器运行的核心部分。Sim-fast中sim_main（）函数的主要工作是：

1、从模拟器的内存空间中取一条在前面主程序中存入的指令；

2、分析指令代码；

3、执行指令，并在执行过程中，依据用户的配置，将用户关心的信息进行统计；

4、循环执行步骤1到步骤3，直到被模拟的程序执行结束；

5、跳转返回到模拟器的主程序。

（3）结束处理

执行完毕后进行结束处理，并将模拟过程中依据用户的选项配置记录的执行信息，如执行时间、各种类型指令的分类统计、前瞻执行的命中/不命中情况等结果输出。

Sim-safe是sim-fast的孪生兄弟，实现基本与sim-fast一致，但它们又是相互独立的。

Sim-main（ ）函数流程与sim-fast中的sim-main（）函数大体相同，主要的区别在于在执行指令时，sim-safe 进行指令的齐整性（Alignment）检查，对所有的访存指令首先检查是否合法，而且sim-safe增加了访存指令的分析，并支持Dilte调试器。

3. 分析 sim-outorder 的乱序流水线实现，并注意观察与五级单流水的不同点，撰写分析报告。（参考网上的资料）

有五种很重要的功能单元支持sim_outorder对指令序列的乱序执行：**保留站**与**重定序缓冲（RUU）**、**Load/Store队列（LSQ）**、**取指队列**、**输入输出相关链**和**寄存器忙闲表**。它们在simplescalar中是通过五种数据结构来实现的。

RUU单元实现寄存器的同步和通讯功能，它将再定序缓冲和保留站统一起来，作为一个循环队列来管理。RUU队列记录了指令的操作类型、源操作数、数据有效性标识。其中的数据项在指令发射时分配，在提交时回收；当寄存器数据和存储器数据相关性满足时，实现乱序流出；

Load/Store队列处理存储器的相关性问题。如果store操作是猜测执行的，其值就被放入队列中。当所有之前的写入地址都已知之后，Load操作就可以访存。如果地址匹配，load操作可以在存储系统或者Load/Store队列中以前的store值的允许下进行。

取指队列是由取指段建立，在调度段译码并调度的指令队列；没有被调度的指令仍留在其中。它是用一个结构数组来实现的。

输入输出相关链，是用来记录前一条指令的输出数据（结果操作数）与后几条指令的输入数据（源操作数）的相关性的链表。

寄存器忙闲表，是用来记录当前各个寄存器被哪一条指令占用的结构数组。

具体的乱序过程如下：

A.取指段：根据配置的各种参量的要求，从指令一级Cache里预取指令，加入到取指队列里。如果在一级cache里找不到指令，同时配置了二级cache，就试图从二级cache里再找，否则就从存储器里寻找。

1. 根据分支预测的要求、cache 容量的支持、事先配置的取指队列的大小，确定预取多少条指令。
2. 在地址有效的条件下，取出指令，并根据指令一级CACHE的延时和指令一级TLB的延时计算出其取指延时的大小。
3. 若是分支指令，则要根据事先配置的分支预测策略预测下一条指令地址；若不是，指令地址自加一。
4. 把这一条指令加入取指队列里，更新取指队列。

B.调度段：从取指队列调度指令。指令首先被译码，然后为其分配RUU资源，判断是否存在数据相关性。如果不存在就可以发射出去，存在的话仍旧留在RUU队列里等待发射。若是访存指令则分配LSQ资源，最后更新输入输出的相关链。

C.发射执行段：检查从调度段发射出来的指令所需的功能部件是否可用（结构相关性），如果可用则将其发射执行。

1. 查看指令所需数据、功能部件是否准备好；
2. 如果是 store 指令，执行之。由于数据可先存在LSQ队列中，执行时间为零，实际的访存操作在 ruu_commit() 中执行。其他指令则需先查看有无功能部件。
3. 如果是 load 指令，要确定 cache 访问的延时，先扫描LSQ队列看其前有无访存地址相同的 store 指令。如果有，那么store 指令存的数据就是 load 指令要取的数据，因而访存延时为一周期；如果没有就并行访问数据 cache 和数据TLB，访存延时为二者中较大者。
4. 如果是非访存指令，操作时间为其功能部件的执行时间；不需功能部件的指令，操作时间为一个周期。如果是空指令操作时间为零。

D.LSQ队列更新：此过程是找出下一条数据相关性被满足了的指令，并将其发射。而这是通过检查LSQ队列，查找存储器阻塞的情况来实现的。

E.写回段：完成把功能部件的输出数据（结果操作数）写入RUU（ register update unit ）的任务。就这点来说，模拟器根据正在完成的指令的输出数据，确定取指队列中的后续指令是否有输入数据与其相关，如果是这样，将把这条指令从取指队列中调度出来进行发射。

F.提交段：这个阶段把已经完成的结果从RUU和LSQ提交到寄存器文件中，并且LSQ中的 store 指令将把其存储数据提交到数据 cache 中。

1. RUU和LSQ中有结果可提交，就执行提交。
2. 让LSQ中的 store 指令把其存储数据提交到数据 cache 中并计算其操作时间，其中要考虑TLB的延时。
3. 按序把已经完成的结果从RUU和LSQ提交到寄存器文件中，并更新RUU和LSQ。

五级流水线分别为取指、译码、执行、存储和回写五个阶段，而乱序过程比五级流水线多了一个提交段。。。。。。

4. 以 Alpha 为例，用 sim-cache 模拟运行 tests-alpha 目录下的程序。设计实验，分析 cache 不同配置参数，如 cache 容量，相联度和块大小对失效率的影响。要求测试配置至少有四组。测试程序至少四个。

4.1.使用sim-cache模拟运行test-alpha目录下的几个程序，结果如下：

Test-dirent.c 测试结果如下图所示：

```
sim: ** starting functional simulation w/ caches **
Usage: test-dirent <path>
```

Test-fmath.c 测试结果如下图所示：

```
sim: ** starting functional simulation w/ caches **
q=4 (int)x=12 (int)y=29
z=144
z=841
z=13
z=13
l=6
l=36
*lp=216
z=144.000000
q=4 x=12.000000 (int)x=12 y=29.000000 (int)y=29
q = 16  x = 11.700001  y = 23.400000
```

Test-math.c 测试结果如下图所示：

```
sim: ** starting functional simulation w/ caches **
pow(12.0, 2.0) == 144.000000
pow(10.0, 3.0) == 1000.000000
pow(10.0, -3.0) == 0.001000
str: 123.456
x: 123.000000
str: 123.456
x: 123.456000
str: 123.456
x: 123.456000
123.456 123.456000 123 1000
sinh(2.0) = 3.62686
sinh(3.0) = 10.01787
h=3.60555
atan2(3,2) = 0.98279
pow(3.60555,4.0) = 169
169 / exp(0.98279 * 5) = 1.24102
3.93117 + 5*log(3.60555) = 10.34355
cos(10.34355) = -0.6068,  sin(10.34355) = -0.79486
x     0.5x
x0.5     x
x   0.5x
-1e-17 != -1e-17 Worked!
```

Test-printf.c 测试结果太长，就不再附图了。

4.2设计实验

为分析cache中4个配置参数（sets数量，块大小，相联度，块替换策略）对失效率的影响，我们设计并进行了以下实验：

我们选择了4个测试程序，在每个程序运行前修改相应的cache配置参数（其余参数保持不变），并观察运行后的失效率的变化。通过这些变化来判断该配置对对cache失效率的影响。

4.2.1 Sets number 参数测试

利用语句：–cache:dl1 dl1:4096:32:1:l 修改cache配置如下，作为参考配置:

Sets number=4096
Block size=32
Associativity=1
Replacement strategy =LRU

选择测试程序一：test-printf.c

运行–cache:dl1 dl1:4096:32:1:l 时模拟统计结果如下：
Dl1.accessses 534043
Dl1.hits 533485
Dl1.misses 558
Dl1.miss_rate 0.0010
修改cache Sets number为2048,-cache:dl1 dl1:2048:32:1:l 数据如下：
Dl1.accessses 53403
Dl1.hits 533455
Dl1.misses 588
Dl1.miss_rate 0.0011

修改Sets number为128,-cache:dl1 dl1:128:32:1:l 数据如下：
Dl1.accessses 53403
Dl1.hits 531336
Dl1.misses 2707
Dl1.miss_rate 0.0051
失效率明显变大
修改Sets number为8192,-cache:dl1 dl1:8192:32:1:l 数据如下：
Dl1.accessses 53403
Dl1.hits 533485
Dl1.misses 558
Dl1.miss_rate 0.0010

从3可以看出，当Sets number大于4096时，失效率不再发生变化.

选择测试程序二：test-math.c
修改Sets number为4096 –cache:dl1 dl1:4096:32:1:l
Dl1.accessses 57571
Dl1.hits 57026
Dl1.misses 545
Dl1.miss_rate 0.0095
修改Sets number为8192 –cache:dl1 dl1:8192:32:1:l
Dl1.accessses 57571
Dl1.hits 57026
Dl1.misses 545
Dl1.miss_rate 0.0095
修改cache Sets number为512 –cache:dl1 dl1:512:32:1:l
Dl1.accessses 57571
Dl1.hits 57007
Dl1.misses 564
Dl1.miss_rate 0.0098
修改cache Sets number为128 –cache:dl1 dl1:128:32:1:l
Dl1.accessses 57571
Dl1.hits 56419
Dl1.misses 1152
Dl1.miss_rate 0.0200

选择测试程序三：test-fmath.c
修改cache Sets number为4096 –cache:dl1 dl1:4096:32:1:l
Dl1.accessses 16663
Dl1.hits 16196
Dl1.misses 467
Dl1.miss_rate 0.0280

修改cache Sets number为2048 –cache:dl1 dl1:2048:32:1:l
Dl1.accessses 16663
Dl1.hits 16196
Dl1.misses 467
Dl1.miss_rate 0.0280

修改cache Sets number为512 –cache:dl1 dl1:512:32:1:l
Dl1.accessses 16663
Dl1.hits 16175
Dl1.misses 488
Dl1.miss_rate 0.0293

修改cache Sets number为128 –cache:dl1 dl1:128:32:1:l
Dl1.accessses 16663
Dl1.hits 16045
Dl1.misses 618
Dl1.miss_rate 0.0371

选择测试程序四：test- dirent.c
修改cache Sets number为4096 –cache:dl1 dl1:4096:32:1:l
Dl1.accessses 4173
Dl1.hits 3739
Dl1.misses 434
Dl1.miss_rate 0.1040

修改cache Sets number为8192 –cache:dl1 dl1:8192:32:1:l
Dl1.accessses 4173
Dl1.hits 3739
Dl1.misses 434
Dl1.miss_rate 0.1040

修改cache Sets number为512 –cache:dl1 dl1:512:32:1:l
Dl1.accessses 4173
Dl1.hits 3721
Dl1.misses 452
Dl1.miss_rate 0.1083

修改cache Sets number为128 –cache:dl1 dl1:128:32:1:l
Dl1.accessses 4173
Dl1.hits 3718
Dl1.misses 455
Dl1.miss_rate 0.1090
实验结论：通过对4个测试程序的测试，发现有一下特点：当cache中sets的数量减少时，会引起失效率的提高；而当实验中发现，当sets的数量高于1024时，失业率将不再降低.

4.2.2 块大小参数测试
与上一参数的测试过程相同，就不再赘述。
选择测试程序一：test-printf.c
修改块大小为16，-cache:dl1 dl1:4096:16:1:l 数据如下
Dl1.accessses 53403
Dl1.hits 532946
Dl1.misses 1097
Dl1.miss_rate 0.0021

修改块大小为8，-cache:dl1 dl1:4096:8:1:l 数据如下：
Dl1.accessses 53403
Dl1.hits 531974
Dl1.misses 2069
Dl1.miss_rate 0.0039

修改块大小为64，-cache:dl1 dl1:4096:64:1:l 数据如下：
Dl1.accessses 53403
Dl1.hits 533748
Dl1.misses 295
Dl1.miss_rate 0.0006

修改块大小为128，-cache:dl1 dl1:4096:128:1:l 数据如下：

出现严重错误。

选择测试程序二：test-math.c
修改块大小为16 –cache:dl1 dl1:4096:16:1:l
Dl1.accessses 57571
Dl1.hits 56548
Dl1.misses 1023
Dl1.miss_rate 0.0178

修改块大小为8 –cache:dl1 dl1:4096:8:1:l
Dl1.accessses 57571
Dl1.hits 55621
Dl1.misses 1950
Dl1.miss_rate 0.0339

修改块大小为64 –cache:dl1 dl1:4096:64:1:l
Dl1.accessses 57571
Dl1.hits 57272
Dl1.misses 299
Dl1.miss_rate 0.0052

选择测试程序三：test-fmath.c
修改块大小为16 –cache:dl1 dl1:4096:16:1:l
Dl1.accessses 16663
Dl1.hits 15766
Dl1.misses 897
Dl1.miss_rate 0.0538

修改块大小为8 –cache:dl1 dl1:4096:8:1:l
Dl1.accessses 16663
Dl1.hits 14914
Dl1.misses 1749
Dl1.miss_rate 0.1050

修改块大小为64 –cache:dl1 dl1:4096:64:1:l
Dl1.accessses 16663
Dl1.hits 16411

Dl1.misses 252
Dl1.miss_rate 0.0151

选择测试程序四：test- dirent.c
修改块大小为16 –cache:dl1 dl1:4096:16:1:l
Dl1.accessses 4173
Dl1.hits 3325
Dl1.misses 848
Dl1.miss_rate 0.2032

修改块大小为8 –cache:dl1 dl1:4096:8:1:l
Dl1.accessses 4173
Dl1.hits 2509
Dl1.misses 1664
Dl1.miss_rate 0.3988

修改块大小为64 –cache:dl1 dl1:4096:64:1:l
Dl1.accessses 4173
Dl1.hits 3949
Dl1.misses 224
Dl1.miss_rate 0.0537
实验结论：通过以上4个测试程序的测试，能发现，块大小减小，则会导致失效率明显升高，反之，当块变大时，失效率会降低。但块的大小不能超过128.

4.2.3相联度及块置换策略参数的测试
测试方法同上，但实验结果显示，这两个参数在4个测试程序下对失效率都没有影响。（我们认为理论上应该是会有影响的，但实验结果却没有任何影响，不知是不是实验设计的问题，还望老师指正）。

5. 总结
通过这几个实验，我们对simplescalar的几个模拟工具及整个模拟过程有了更深一步的了解，同时再工具的使用方面也得到一定的加强。不过在报告中可能还有许多不对的地方，还望老师给与指正。