ni 实验一 SimpleScalar 介绍

在我们的《计算机系统结构》课程中，使用体系结构模拟器 SimpleScalar 作为我们的实验平台。

本实验内容包括：SimpleScalar 的体系结构，SimpleScalar 的各种部件或功能仿真，以及 SimpleScalar 的目录结构。通过本实验的学习，希望同学们能够对 SimpleScalar 有初步的认识，有助于后续实验的进行。

1.1 SimpleScalar 模拟器简介

SimpleScalar 模拟器是一个超标量、5 级流水的 RISC 体系结构模拟器，提供了从最简单到超标量乱序发射的不同的模拟程序。

SimpleScalar 工具集起源于 80 年代中后期由 Manoj Franklin 开发的模拟工具。1995 年夏，Steve Bennett 开发了 SimpleScalar x86 工具。SimpleScalar 模拟器由 Intel 公司微型计算机研究室的 Todd M. Austin 在 1996 年最终编写而成，美国麦迪逊威斯康星大学计算机学院的 Doug Burger 参与开发和文档整理工作，二人共同拥有该模拟器的版权。Austin 与 Bennett 两人都是威斯康星大学计算机学院的助手，由 Guri Sohi 教授指导研究工作。2001 年，加拿大 Queen 大学的 Naraig Manjikian 发布了一个 SimpleScalar2.0 的多处理器版本，称为 mp_simplesim。该版本中添加了用于线程创建与同步的运行时间库（run-time library），采用 MESI 协议作为 cache 一致性协议，并提供了一个 chache 一致性状态的查看工具。

SimpleScalar 模拟器在功能级上实现了执行驱动、解释执行，在行为级上实现了流水线模拟。该工具集提供了一个以 GCC 为主的编译器以及相关组件，能够产生基于 SimpleScalar 体系结构的目标代码，然后在 SimpleScalar 模拟器上运行。

运行模拟器时，主程序 main()做所有的初始化工作，并将二进制目标码载入内存，然后调用 sim_main()，sim_main()在每个模拟器中单独说明，预先译码整个正文段，加快模拟。然后开始目标程序的模拟。

1.2 SimpleScalar 仿真介绍

SimpleScalar 模拟器提供了从最简单到超标量乱序发射的不同的模拟程序。如 1.1 节所示，其中 sim-outorder 模拟器模拟的是一个超标量，5 级流水的 RISC 体系结构的 CPU 模型：分为取指(instruction fetch)、译码(decode)、执行(execute)、存储(store)和回写(write back)五个阶段，而乱序过程比五级流水线多了一个提交段(commit)。

以下是在 simplesim-3.0 中所有模拟器通用的参数：

-config	从文件中装载配置
-dumpconfig	将配置信息导出至文件
-h	显示帮助信息
-d	启用调试信息
-I	在 Dlite！调试器中开始执行（sim-fast 不支持）

-q		立即退出
-seed	<int>	随机数生成器的计时器种参数，默认为 1
-chkpt	<filename>	从<filename>中恢复 EIO trace 的执行
-redir:sim	<file path>	模拟器输出重定向至文件
-redir:prog	<file path>	模拟程序输出重定向至文件
-nice	<int>	模拟器调度优先级，默认为 0

SimpleScalar v3.0 提供的几个模拟程序如下：

（1）Sim-fast

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。

（2）Sim-safe

在工具集中，是最简单的最友好的模拟器，在 sim-fast 的基础上添加了 Dlite!调试支持，检查所有的指令错误，不讲究速度。

（3）Sim-bpred

实现一个分支预测(branch prediction，也称作跳转预测)分析器，可采用五种分支预测方式：nottaken, taken, bimod, 2lev, comb。

特有参数：

-bpred<string> # bimod # 分支预测器类型 {nottaken|taken|bimod|2lev|comb}

-bpred:bimod<int> #2048 # 二位预测器配置 (<size>)

-bpred:2lev<int list...> # 1 1024 8 0 # 两级预测器配置 (<l1size> <l2size> <hist_size> <xor>)

-bpred:comb<int> # 1024 # 复合预测器配置 (<meta_table_size>)

-bpred:ras<int> # 8 # 返回地址栈大小 (0 为无返回栈)

-bpred:btb<int list...> # 512 4 # BTB(branch target buffer) 配置 (<num_sets> <associativity>)

（4）Sim-cache & sim-cheetah

Sim-cache 实现 cache 模拟功能，为用户选择的 cache 和快表(TLB, translation lookaside buffer)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。另外，实现 cache 模拟功能的还有 sim-cheetah，能够有效地模拟全相联 cache，并能同时生成各种 cache set 数配置下的 cache 统计量。同样地，sim-cheetah 不会生成时间信息。

特有参数：

Sim-cache:

-cache:dl1 <string># dl1:256:32:1:l # l1 数据 cache 配置, i.e., {<config>|none}

-cache:dl2 <string># ul2:1024:64:4:l # l2 数据 cache 配置, i.e., {<config>|none}

-cache:il1 <string># il1:256:32:1:l # l1 指令 cache 配置, i.e., {<config>|dl1|dl2|none}
-cache:il2 <string> # dl2 # l2 指令 cache 配置, i.e., {<config>|dl2|none}
-tlb:itlb <string> # itlb:16:4096:4:l # 指令 TLB 配置, i.e., {<config>|none}
-tlb:dtlb <string> # dtlb:32:4096:4:l # 数据 TLB 配置, i.e., {<config>|none}
-flush <true|false> # false # 系统调用时是否清空 cache
-cache:icompress <true|false># false # 将 64 位指令地址转换为 32-位等价地址
-pcstat <string list...> # <null> # 文本地址统计量 (可采用多个)
Sim-cheetah:
-refs <string> #data # 需要分析的引用流, i.e., {none|inst|data|unified}
-R <string> #lru # 替换策略, i.e., lru or opt
-C <string> # sa # cache 配置, i.e., fa（全关联）, sa（组关联）, or dm（直接关联）
-a <int> # 7 # 最小组数(以 2 为底的对数,直接映射时为 line 大小)
-b <int> # 14 # 最大组数(以 2 为底的对数,直接映射时为 line 大小)
-l <int> # 4 # cache line 大小（以 2 为底的对数）
-n <int># 1 # 需分析的最大关联度（以 2 为底的对数）
-in <int> # 512 # 模拟全关联 cache 时的 cache 大小间隔
-M <int># 524288 # 最大 cache 大小
-c <int> #16 # 供直接映射分析的 cache 大小（以 2 为底的对数）

（5）Sim-eio

这个模拟器支持生成外部输入/输出跟踪（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。

特有参数：

-fastfwd <int># 0 # 跟踪开始前跳过的指令数
-trace <string># <null> # EIO 跟踪文件的输出文件名
-perdump <string list...> # <null> # 每 n 条指令设置周期断点: <base fname> <interval>
-dump <string list...> #<null> # 指明断点文件与触发器: <fname> <range>

（6）Sim-outorder

最完整的工具，前文中已提到。支持按序（in-order）和乱序（out-order）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。

特有参数：

-fetch:ifqsize <int>#4 # 取值队列大小 (in insts)
-fetch:mplat <int>#3 # 额外的分支误预测延时
-fetch:speed <int># 1 # 执行核心相关的机器前端速度
-decode:width<int># 4 # 指令解码带宽 (insts/cycle)
-issue:width <int> #4 # 指令发射带宽 (insts/cycle)
-issue:inorder <true|false># false # 以按序发射运行流水线
-issue:wrongpath <true|false># true # 在错误执行路径下发射指令

```
-commit:width      <int># 4          # 指令提交带宽 (insts/cycle)
-ruu:size          <int># 16         # 寄存器更新单元 (RUU) 大小
-lsq:size          <int>#8           # load/store 队列 (LSQ) 大小
-mem:lat           <int list...># 18 2  # 主存访问延时 (<first_chunk> <inter_chunk>)
-mem:width         <int> #8          # 主存访问总线带宽 (in bytes)
-tlb:itlb          <string># itlb:16:4096:4:l    # 指令 TLB 配置, i.e., {<config>|none}
-tlb:dtlb          <string># dtlb:32:4096:4:l    # 数据 TLB 配置, i.e., {<config>|none}
-tlb:lat           <int># 30         # 指令/数据 TLB 未命中延时 (in cycles)
-res:ialu          <int># 4          # 整型 ALU 总可用数
-res:imult         <int>#1           # 整型乘法器/除法器总可用数
-res:memport       <int> #2          # 存储系统端口总可用数 (to CPU)
-res:fpalu         <int>#4           # 浮点型 ALU 总可用数
-res:fpmult        <int>#1           # 浮点型乘法器/除法器总可用数
-bugcompat         <true|false># false # 在后退兼容 bug 模式下操作（仅供测试）
```

（7）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，决定所要模拟的统计量，如 instruction classes and addresses、text symbols、memory accesses、branches and data segment symbols 等，以方便使用者整理收集数据材料。

特有参数:

```
-all               <true|false># false     # 启用所有统计量选项
-iclass            <true|false># false     # 启用指令类型统计
-iprof             <true|false># false     # 启用指令统计
-brprof            <true|false># false     # 启用分支类型统计
-amprof            <true|false># false     # 启用地址模式统计
-segprof           <true|false># false     # 启用 load/store 地址段统计
-tsymprof          <true|false># false     # 启用文本记号统计
-taddrprof         <true|false># false     # 启用文本地址统计
-dsymprof          <true|false># false     # 启用数据记号统计
-internal          <true|false># false     # 符号统计时包含编译器内部记号
-pcstat            <string list...># <null>  # 文本地址统计量(可采用多个)
```

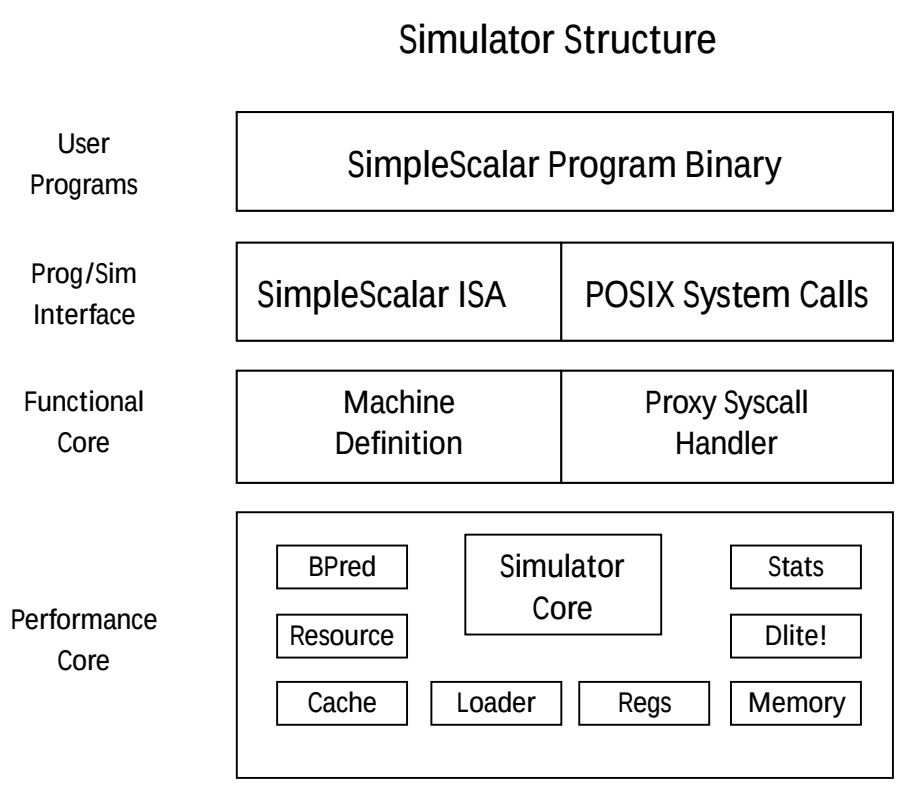

图 1.1 SimpleScalar 的模拟器结构

1.3 SimpleScalar 的目录结构

如图 1.1 所示，Simplesim-3.0 模拟器包(含 cheetah)包括以下几部分内容：

（1）模拟器模块（simulator modules），包括：sim-bpred.c，sim-cache.c，sim-cheetah.c，sim-eio.c，sim-fast.c，sim-outorder.c，sim-profile.c，及 sim-safe.c，其功能已在 1.2 节中介绍完毕；

（2）模拟组件（simulation components），包括：

bpred.[h，c] 分支（跳转）预测器；
cache.[h，c] cache 模块；
eventq.[h，c] 事件队列模块；
libcheetah/ Cheetah cache 模拟器支持库；
ptrace.[h，c] 流水线轨迹模块；
res.[h，c] 资源管理器模块；
sim.h 模拟器主体代码接口定义；
textprof.pl 代码段属性视图 (Perl Script)；
pipeview.pl 流水线轨迹视图 (Perl script)；

（3）系统组件（system components），包括：

dlite.[h，c] DLite!, 轻量级调试程序；
eio.[h，c] 外部 I/O 追踪模块；
loader.[h，c] 程序载入器；
memory.[h，c] 平面存储器空间模块；

regs.[h，c] 寄存器模块；
machine.[h，c] 目标机及 ISA 依赖的例行程序；
machine.def SimpleScalar ISA 定义；
symbol.[h，c] 符号表模块；
syscall.[h，c] 代理系统调用实现；
（4）辅助模块（“very useful” modules），包括：
eval.[h，c] 一般表达式验证程序；
libexo/ EXO(-skeletal) 持久性数据结构库（由 EIO trace 使用）；
misc.[h，c] 提供一些提示信息函数；
options.[h，c] 选项包；
range.[h，c] 程序范围表达式包；
stats.[h，c] 统计量包；
（5）构建组件（building components），包括：
Makefile 顶层的构建文件；
tests/ 独立自验证大小端模式测试包；
endian.[h，c] 大小端模式探测；
main.c 主执行代码；
sysprobe.c 系统探测，构建过程中使用；
version.h 模拟器版本验证。

实验二 SimpleScalar 的安装与配置

2.1 实验摘要

1. 安装 VMware Workstation 虚拟机；
2. 安装 Linux 操作系统；
3. 安装 SimpleScalar PISA 版；
4. 安装 SimpleScalar 扩展——多线程处理器模拟器 mp_simplesim。

2.2 实验目的

安装实验平台并熟悉实验环境，为今后的学习做准备。

2.3 实验步骤

2.3.1 VMware Workstation 的安装

VMware Workstation 虚拟机网上资源较多，最新版本为 9.0 版。不少同学已经安装过老版本的 VMware，不影响实验。这里不再赘述。

2.3.2 Linux 的安装

下载 Ubuntu 镜像(iso)文件，建议使用 Ubuntu 9.04。在 VMware Workstation 的 file->new->Virtual Machine 下进行安装：

(1) 选择 Typical；
(2) Installer disc image file (iso)，选择 Ubuntu 镜像文件的路径；
(3) 设置用户名、密码；
(4) 设置虚拟机存储位置；
(5) 最大磁盘空间，可根据自身硬盘大小选择，建议 10G-20G；选择单一文件；
(6) 完成。

完成 Linux 安装后，请启动 Linux 进行熟悉，并适当了解 Linux 终端的指令集。

2.3.3 SimpleScalar 的安装

以 Ubuntu 9.04 系统下为例：

可使用如下 shell 脚本进行安装（也可在终端中单句执行以下命令）：

```
export NAME=Simplescalar
export PACKAGE=simplescalar
export TOOL=simpletools-2v0
```

```
export UTIL=simpleutils
export SIM=simplesim

# Update Ubuntu Software Package
sudo apt-get update

# Getting required applications
sudo apt-get install flex-old bison build-essential

# Create Simplescalar Directory
mkdir $NAME
cd $NAME

# Getting simplescalar tar file
wget http://csrl.unt.edu/downloads/$PACKAGE.tgz

# Extraction
tar xvfz $PACKAGE.tgz
export CC="gcc"

# Setting up installation
export HOST=i686-unknown-linux
export TARGET=sslittle-na-sstrix
export IDIR=~/$NAME

# Build Simplescalar tools
cd ~/$NAME
tar xvfz $TOOL.tgz
rm -rf gcc-2.6.3

# Build Simplescalar utils
cd ~/$NAME
tar xvfz $UTIL-990811.tar.gz
cd $UTIL-990811
./configure --host=$HOST --target=$TARGET --with-gnu-as --with-gnu-ld --prefix=$IDIR
make CC=gcc
sudo make install CC=gcc

# Build Simplescalar
cd ~/$NAME
tar xvfz $SIM-3v0d.tgz
cd $SIM-3.0
make config-pisa
make CC=gcc
```

```
# Build Compiler
cd ~/$NAME
tar xvfz gcc-2.7.2.3.ss.tar.gz
cd ~/$NAME/gcc-2.7.2.3
export PATH=$PATH:$IDIR/simpleutils-990811/sslittle-na-sstrix/bin
./configure --host=$HOST --target=$TARGET --with-gnu-as --with-gnu-ld --prefix=$IDIR

# Fix file errors
make LANGUAGES="c c++" CFLASS=-O3 CC="gcc"
sed -i 's/return \"FIXME\\n/return \"FIXME\\n\\/g' ~/$NAME/gcc-2.7.2.3/insn-output.c
make LANGUAGES="c c++" CFLASS=-O3 CC="gcc"
wget http://www.ict.kth.se/courses/IS2202/ar
wget http://www.ict.kth.se/courses/IS2202/ranlib
chmod 700 ar
chmod 700 ranlib
sudo cp ar $IDIR/sslittle-na-sstrix/bin/ar
sudo cp ranlib $IDIR/sslittle-na-sstrix/bin/ranlib
rm ar
rm ranlib
chmod +w ~/$NAME/gcc-2.7.2.3/obstack.h
sed -i 's/next_free)++/next_free++)/g' ~/$NAME/gcc-2.7.2.3/obstack.h
make LANGUAGES="c c++" CFLASS=-O3 CC="gcc"
sed -i '98i\

#define BITS_PER_UNIT 8
' ~/$NAME/gcc-2.7.2.3/libgcc2.c

#make LANGUAGES="c c++" CFLASS=-O3 CC="gcc"
cp $IDIR/gcc-2.7.2.3/patched/sys/cdefs.h $IDIR/sslittle-na-sstrix/include/sys/cdefs.h
make LANGUAGES="c c++" CFLASS=-O3 CC="gcc"
make enquire CC=gcc
sudo make install LANGUAGES="c c++" CFLASS=-O3 CC="gcc" PATH=$PATH:~/$NAME/bin

# Done!
exit 0
```

在一独立文件夹下，新建 hello.c，内容为：

```
#include<stdio.h>
main()
{
    Printf(“Hello World!\n”);
}
```

然后用如下命令编译：

~/$NAME/bin/sslittle-na-sstrix-gcc –o hello hello.c
生成文件 hello，可用如下方式运行：
~/$NAME/simplesim-3.0/sim-safe hello
如果输出"Hello World!"，说明安装成功！

2.3.4 mp_simplesim 的安装

注：mp_simplesim 需另行下载。

1.移动到 SimpleScalar 安装目录:
$ cd simplescalar

2.将 mp_simplesim.tar.gz 解压至该目录：
$ tar xvzf mp_simplesim.tar.gz

3.进入 mp_simplesim 目录:
$ cd mp_simplesim

4.打开 Makefile 文件，将 SS_BIN_PATH 设置为交叉编译器的路径：
SS_BIN_PATH = /home/student/simplescalar/sslittle-na-sstrix/bin

5.编译:
$ make

遇到错误提示：
syscall.c:102:23: 错误： bsd/sgtty.h：没有该文件或目录

将 syscall.c 第 102 行中替换为：
#include <sgtty.h>
并保存。

6.继续运行 make，出现新的错误：
syscall.c:823: 错误： 'TIOCGETP'未声明(在此函数内第一次使用)
syscall.c:823: 错误： (即使在一个函数内多次出现，每个未声明的标识符在其
syscall.c:823: 错误： 所在的函数内也只报告一次。)
syscall.c:826: 错误： 'TIOCSETP'未声明(在此函数内第一次使用)

将 syscall.c 的 823－831 行由：

```
/* #if !defined(__CYGWIN32__) */
      case SS_IOCTL_TIOCGETP:
        local_req = TIOCGETP;
        break;
      case SS_IOCTL_TIOCSETP:
        local_req = TIOCSETP;
        break;
```

```
        case SS_IOCTL_TCGETP:
          local_req = TIOCGETP;
          break;
    /* #endif */
```

替换为：

```
    #ifdef TIOCGETP
        case SS_IOCTL_TIOCGETP:
          local_req = TIOCGETP;
          break;
    #endif
    #ifdef TIOCSETP
        case SS_IOCTL_TIOCSETP:
          local_req = TIOCSETP;
          break;
    #endif
    #ifdef TIOCGETP
        case SS_IOCTL_TCGETP:
          local_req = TIOCGETP;
          break;
    #endif
```

并保存。

7.继续运行 make，出现新的错误：

```
#-----------------------------------------------------
#
# Linking objects to produce 'sim-mpfast' simulator
#
#-----------------------------------------------------
gcc -o sim-mpfast `./sysprobe -flags` -DDEBUG -DGRAPHICS -O3      sim-mpfast.o \
main.o syscall.o memory.o regs.o loader.o ss.o endian.o symbol.o eval.o options.o stats.o
range.o misc.o -lm -lX11 `./sysprobe -libs`
/usr/bin/ld: cannot find –lx11
collect2: ld 返回 1
make: *** [sim-mpfast] 错误 1
```

解决方法：安装缺失的软件包 libx11-dev：

```
$ sudo apt-get install libx11-dev
```

8.继续运行 make，出现新的错误：

```
#-----------------------------------------------------
#
# Linking objects to produce 'sim-mpfast' simulator
#
```

#----------

gcc -o sim-mpfast `./sysprobe -flags` -DDEBUG -DGRAPHICS -O3 sim-mpfast.o \

main.o syscall.o memory.o regs.o loader.o ss.o endian.o symbol.o eval.o options.o stats.o range.o misc.o -lm -lX11 `./sysprobe -libs`

/usr/bin/ld: cannot find -lbsd

collect2: ld 返回 1

make: *** [sim-mpfast] 错误 1

解决方法：安装缺失的软件包 libbsd-dev：

$ sudo apt-get install libbsd-dev

9.继续运行 make，出现新的错误：

#----------

#

Linking objects to produce 'sim-mpfast' simulator

#

#----------

gcc -o sim-mpfast `./sysprobe -flags` -DDEBUG -DGRAPHICS -O3 sim-mpfast.o \

main.o syscall.o memory.o regs.o loader.o ss.o endian.o symbol.o eval.o options.o stats.o range.o misc.o -lm -lX11 `./sysprobe -libs`

/usr/bin/ld: errno: TLS definition in /lib/libc.so.6 section .tbss mismatches non-TLS reference in eval.o

/lib/libc.so.6: could not read symbols: Bad value

collect2: ld 返回 1

make: *** [sim-mpfast] 错误 1

打开文件 eval.c，64－66 行内容为：

#if defined(__CYGWIN32__)

#include <errno.h>

#endif

删除或注释掉 64 及 66 行，保留:

#include <errno.h>

并保存。

10.继续运行 make，出现新的错误：

#----------

#

Linking objects to produce 'sim-mpfast' simulator

#

#----------

gcc -o sim-mpfast `./sysprobe -flags` -DDEBUG -DGRAPHICS -O3 sim-mpfast.o \

main.o syscall.o memory.o regs.o loader.o ss.o endian.o symbol.o eval.o options.o stats.o range.o misc.o -lm -lX11 `./sysprobe -libs`

/usr/bin/ld: errno: TLS definition in /lib/libc.so.6 section .tbss mismatches non-TLS reference in range.o

/lib/libc.so.6: could not read symbols: Bad value

collect2: ld 返回 1

make: *** [sim-mpfast] 错误 1

打开文件 range.c，64－66 行内容为：

```
#if defined(__CYGWIN32__)
#include <errno.h>
#endif
```

删除或注释掉 64 及 66 行，保留:

```
#include <errno.h>
```

并保存。

11.为了避免出现名为”binary endian does not match host endian”的错误，

将 loader.c 第 474 行的”endian_host_word_order”替换为

”endian_host_byte_order”，保存。

12.为了修正一处幂运算错误，将 sim-mpcache.c 第 134 行：

```
#define log2(x) ((int)(log(x)/log(2)))
```

替换为：

```
/* return log of a number to the base 2 */
int
log2(int n)
{
   int power = 0;
  if (n <= 0 || (n & (n-1)) != 0)
     panic("log2() only works for positive power of two values");
  while (n >>= 1)
    power++;
  return power;
}
```

并保存。

13.现在，执行：

```
$ make
$ make sim-tests
```

将会生成包含 debug、warning 信息及 statistics 的输出结果，说明执行通过。

14.运行

$ ./sim-mpcache -graphics -speed 1 tests/dotbar.ss -p8 -n10000

如果执行成功，则能够看到图表化的 cache 命中情况：（见下图）

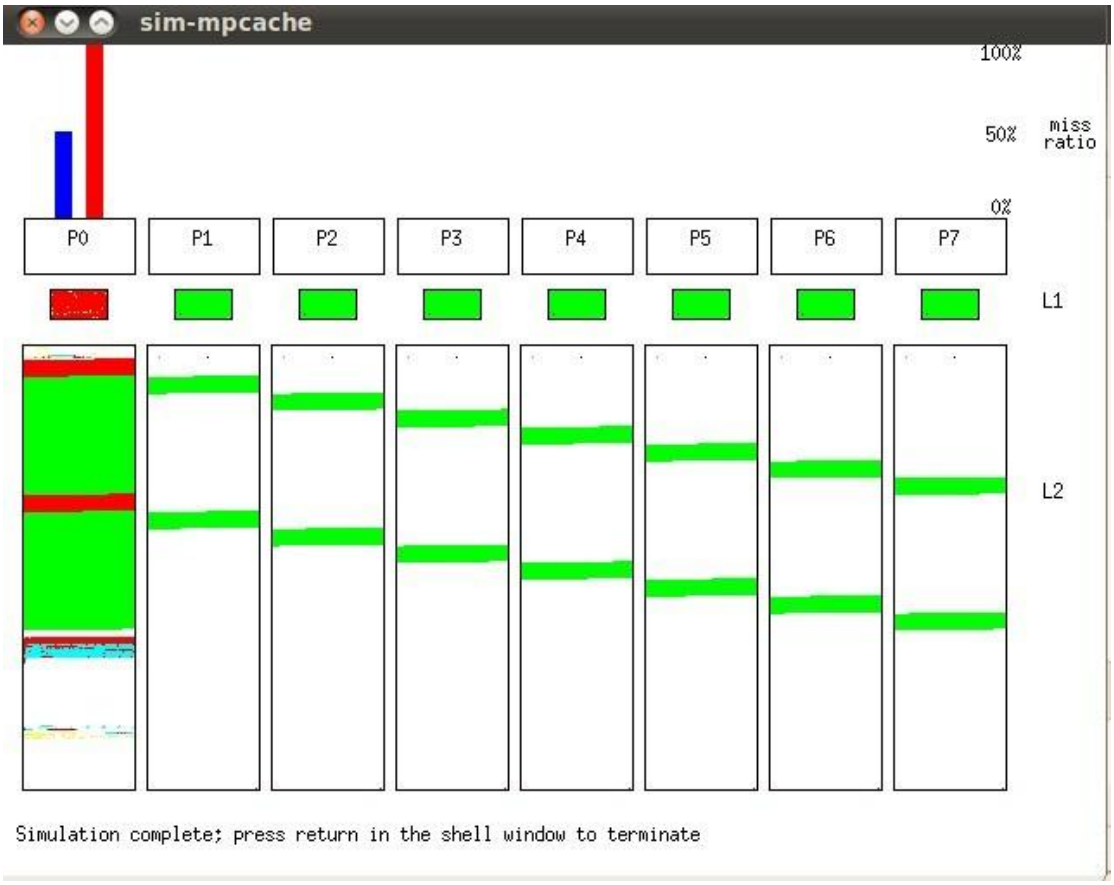

图 2.1 cache 命中情况

其中每个像素的颜色代表着缓存行(cache line)的状态：

红：已修改

绿：共享

黄：干净且独占

黑：不可用

白：未使用

青（仅 L2）：L1 一缓存行数据已修改，且尚未写回 L2

当模拟完成后，终端中会提示：

Simulation complete; press return to terminate

在终端输入 return 即可退出执行。

注：(1)拖拽窗口时，注意不要使窗口挡住图形界面显示，或最小化图形窗口，否则被遮挡部分会消失！

(2)推荐截图时手动选择边框。

2.4 SimpleScalar 主要部件

一套功能完整的 SimpleScalar 组件包括：

（1）simpletools 工具包（simpletools-2v0.tgz）：包含了重定目标至 SimpleScalar 体系结构的旧版本交叉编译器 gcc-2.6.3，库源代码 glibc-1.0.9 及 FORTRAN-C 语言解释器 f2c-1994.09.27，仅用于构建 SimpleScalar 基准测试程序二进制文件；

（2）simpleutils 实用程序包（simpleutils-990811.tar.gz）：包含了重定目标至 SimpleScalar 体系结构的 GNU binutils，仅用于编译 SimpleScalar 基准测试程序二进制文件；

（3）simplesim 模拟器包（simplesim-3v0d-with-cheetah.tar.gz，mp_simplesim.tar.gz）：含模拟器源代码，指令集定义宏，及测试程序代码与二进制文件；

（4）gcc 交叉编译器包（gcc-2.7.2.3.ss.tar.gz）：gcc 交叉编译器的另一版本，可用于替换 gcc-2.6.3，以编译、构建出较好的 benchmark。

其中最为重要的是 simplesim 模拟器包。simplesim-3.0 中各文件的功能已在前文中描述。至于 mp_simplesim 则与 simplesim-2.0 类似，在这里不再一一阐述。

2.5 关于 Mibench-auotomotive 基准测试包的简要说明

Mibench (version 1.0)是一款免费的商用典型嵌入式 benchmark 组件。

Automotiveand Industrial Control：此 Benchmarks 主要应用於嵌入式控制系统中，典型的应用如安全气囊控制器(air bag controllers)，引擎效能监视器(engineperformance monitor)和感应系统(sensor systems)，其主要工作为基本数学运算，bit manipulation，data input/output 和 simple data organization，因此在此分类中挑选 basicmath，bitcount，quicksort，susan 等演算法作为 Benchmark。

Network：嵌入式处理器在网络设备如交换机(switches)，路由器中，其主要用作 shortest path calculations，tree and table lookups 和 datainput/output。在 Benchmark 选择上，挑选具有查找图最短路径和创建与查找 Patricia 字典树数据结构的算法。而 CRC32，Sha，Blowfish 因与 Security 和 Telecommunications 有相关，所以将这三者移出 Network 的测试范围中。

Security：随著网络的发展，资讯安全逐渐显其重要性，在此种 Benchmarks 包含各种常见的加解密演算法及 hash 演算法，如 Pretty Good Privacy(PGP)，rijndael encrypt/decrypt，blowfish encrypt/decrypt 等。

Comsumer：此类常见的设备为扫描机，位相机，PDA，其需要作多媒体，影像编解码(jpeg，tiff)，MP3 编解码，html typesetting 等工作，因此选择 jpegenc(dec)，lame，mad，tiff，typesetting 等算法作为 Benchmark。

Office automation：此类常见的设备为印表机，传真机及文书处理器，其主要作办公相关工作，因此选择 ghost-script，string-search 等演算法作为 Benchmark。

Telecommunications：随著网络的发展，无线通讯技术多已整合于可携性的消费性电子产品中。由于该技术的重要性，在 Mibench 中也包含了通信相关的 Benchmarks，其主要包含声音的编解码，frequency analysis 和 checksum 算法，如 FFT，GSM，CRC32 等。

以上的 Benchmark 中，我们挑选 automotive benchmark，包括 basicmath，bitcount，quicksor，susan 等 benchmarks，以供模拟分析使用，用于后续的实验扩展中。

附录 64 位 Ubuntu 上安装 SimpleScalar

由于不少同学的机器使用 64 位系统，故附录 64 位 Ubuntu 上 SimpleScalar 的参考。

准备必要文件：

Simpletools-2v0.tgz
Simplesim-3v0e.tar.gz
Simpleutils-990811.tar.gz
Gcc-2.7.2.3.ss.tar.gz

安装过程：

1. 环境设置：

$ uname -a：查看内核版本和机器类型。
根据结果设定如下：HOST=i686-pc-linux or HOST=i386-pc-linux
导出环境变量：

```
$ export HOST=i386-pc-linux
$ export IDIR=/home/YOUR_USER_NAME/simplescalar
$ export TARGET=sslittle-na-sstrix
```

确保安装以下包 ：

flex
bison
build-essential

$ sudo apt-get install <PACKAGE_NAME> 使用该语句进行安装。

2. 安装 Simple tools：

解压文件并移除旧版本的 gcc，执行如下命令：

```
$ cd $IDIR
$ tar xzvf simpletools-2v0.tgz
$ rm –rf gcc-2.6.3
```

3. 安装 SimpleUtils：

解压：

```
$ tar xzvf simpleutils-990811.tar.gz
$ cd simpleutils-990811
```

修正一处错误代码：

在 ld 目录下找到 ldlex.l 文件，将所有的 yy_current_buffer 修改为 YY_CURRENT_BUFFER

安装：

```
$ ./configure -host=$HOST -target=$TARGET -with-gnu-as -with-gnu-ld -prefix=$IDIR
$ make
$ make install
```

4. 安装 Simulator：
执行以下命令
$ cd $IDIR
$ tar xzvf simplesim-3v0e.tgz
$ cd simplesim-3.0
$ make config-pisa 或者 make config-alpha（？）
$ make

5. 安装 gcc 交叉编译器：
请仔细按照步骤执行：
解压：
$ cd $IDIR
$ tar xzvf gcc-2.7.2.3.ss.tar.gz

配置：
$ cd gcc-2.7.2.3
$ export PATH=$PATH:/home/YOUR_USER_NAME/simplescalar/sslittle-na-sstrix/bin
$ ./configure -host=$HOST -target=$TARGET -with-gnu-as -with-gnu-ld -prefix=$IDIR

注意：修改 Makefile 第 130 行，在后面添加 -I/usr/include （-I 大写的 i）

修改文件 protoize.c 第 60 行：
将#include <varargs.h> 替换为 #include <stdarg.h>

修改文件 obstack.h 第 341 行：
将*((void **)__o->next_free)++=((void *)datum);
替换为*((void **)__o->next_free++)=((void *)datum);

复制补丁文件：
$ cp ./patched/sys/cdefs.h ../sslittle-na-sstrix/include/sys/cdefs.h
$ cp ../sslittle-na-sstrix/lib/libc.a ../lib/
$ cp ../sslittle-na-sstrix/lib/crt0.o ../lib/

下载文件 ar-ranlib.tar.gz，解压将它的内容放到$IDIR/sslittle-na-sstrix/bin 中。
并且赋予这些文件可执行和可写权限：
chmod +w <filename>
chmod +x <filename>

$ make，解决出现的错误。
编辑文件 insn-output.c
在 line 675, 750 and 823 之后增加 “\”

$ make 出错：

在 objc/sendmsg.c, 35 行处增加：#define STRUCT_VALUE 0

```
$ make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
```

修改 cxxmain.c 文件，删掉 2978-2979 行，即：

```
char * malloc ();
char * realloc ();
```

继续执行，应该没错了：

```
$ make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
$ make install LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
```

到此，模拟器工具和编译器都安装好了。

测试程序请参考 2.3.3 节最后的 hello world 程序。