分支预测测试及记分牌算法设计进展

童敢 黄家腾 赵琛然

摘要：本周的内容主要是对新的测试程序在之前的设计的处理器上运行，经过大量的尝试未能成功，报告总结了部分过程中的思考。另外对后续记分牌算法的引入做了简单的设计，将在下一阶段着手实现。

一、分支预测测试

（一）程序分析和修改

新的程序为 Boyer-Moore-Horspool 文本匹配算法的实现，主程序有两个，一个是 pbmsrch_small，一个是 pbmsrch_large，分别包含了较少（几十句）和较多（一千多句）需要匹配的文本内容，程序的功能就是在这些句子的列表中寻找是否存在某字符子串，然后输出匹配结果，通过 gcc 编译运行正常输出的结果如下：

```
"abb" is in "cabbie" ["abbie"]
"you" is in "your" ["your"]
"not" is not in "It isn't here"
"it" is in "But it is here" ["it is here"]
"dad" is in "hodad" ["dad"]
"yoo" is in "yoohoo" ["yoohoo"]
"hoo" is in "yoohoo" ["hoo"]
"oo" is in "yoohoo" ["oohoo"]
"oh" is in "yoohoo" ["ohoo"]
"xx" is not in "yoohoo"
"xx" is in "xx" ["xx"]
"x" is in "x" ["x"]
"x" is not in "."
"field" is in "In recent years, the field of photonic " ["field of photonic "]
"new" is in "crystals has found new" ["new"]
"row" is in "applications in the RF and microwave" ["rowave"]
"regime" is in "regime. A new type of metallic" ["regime. A new type of metallic"]
"boom" is not in "electromagnetic crystal has been"
"that" is in "developed that is having a" ["that is having a"]
"impact" is in "significant impact on the field of" ["impact on the field of"]
"and" is not in "antennas. It consists of a"
"zoom" is not in "conductive surface, covered with a"
"texture" is in "special texture which alters its" ["texture which alters its"]
"magnet" is in "electromagnetic properties. Made of solid" ["magnetic properties. Made of solid"]
"doom" is not in "metal, the structure"
"loom" is not in "conducts DC currents, but over a"
"freq" is in "particular frequency range it does" ["frequency range it does"]
"current" is in "not conduct AC currents. It does not" ["currents. It does not"]
"phase" is in "reverse the phase of reflected" ["phase of reflected"]
"images" is not in "waves, and the effective image currents"
"appears" is not in "appear in-phase, rather than"
"phase" is in "out-of-phase as they are on normal" ["phase as they are on normal"]
"conductor" is in "conductors. Furthermore, surface" ["conductors. Furthermore, surface"]
"wavez" is not in "waves do not propagate, and instead"
"normal" is not in "radiate efficiently into free"
"free" is not in "space. This new material, termed a"
"termed" is not in "high-impedance surface, provides"
"provide" is not in "a useful new ground plane for novel"
```

程序需要测试的部分如下：

```
for ( ;; )
      {
            while ( (i += skip[((uchar *)string)[i]]) < 0 )
                  ;                                 /* mighty fast inner loop */
            if (i < (LARGE - stringlen))
                  return NULL;
            i -= LARGE;
            j = patlen - 1;
            s = (char *)string + (i - j);
            while (--j >= 0 && s[j] == pat[j])
                  ;
            if ( j < 0 )                                        /* rdg 10/93 */
                  return s;                                     /* rdg 10/93 */
            if ( (i += skip2) >= 0 )                            /* rdg 10/93 */
                  return NULL;                                  /* rdg 10/93 */
```

```
    }
```

其中包含大量的判断（while 循环中的判断、if 判断），而在字符串匹配算法中，判断的结果不如上次的测试程序那么规律，每条不同的判断语句、每次判断的结果都可能不同，导致分支预测的准确度下降，我理解的分支测试的所有状况应该就是连续多次测试的结果序列的状态，比如考虑 4 个结果的序列，结果就有 16 种序列，但主要还是先尝试正确运行测试程序再考虑优化的事。

程序设计到部分系统函数，如 memcpy、strlen、printf、putchar，其中 memcpy 和 strlen 直接使用编译器静态链接即可，但 printf 和 putchar 属于输入输出函数，因此采取之前的方法，将 printf 相关的内容转化为向特定地址存储数据。程序中的 printf 涉及到了字符串的输出，但这不是这个程序的重点，因此只输出是否匹配成功即可，修改后的程序部分如下：

```
int result, count = 0;
      for (i = 0; find_strings[i]; i++)
      {
            count++;
            init_search(find_strings[i]);
            here = strsearch(search_strings[i]);
            if (here)
                result++;
      }
```

（二）程序的编译和运行

根据上面的分析，需要采用静态编译的方法来生成指令，所以需要加上-static 选项，但是在实际测试中发现，生成汇编代码的过程中-static 选项并没有生效，即依然是采取的动态链接的方式，这是一处疑惑的地方。

既然不能生成静态的汇编指令，那只能直接生成 ELF 文件了。生成 ELF 文件的指令为：riscv32-unknown-elf-gcc -march=rv32im -fno-stack-protector -fno-zero-initialized-in-bss -ffreestanding -fno-builtin -nostartfiles -mstrict-align -static pbmsrch_small.c -o small

程序主体包括大量的数据（字符串），在生成的 ELF 文件中对应 data 段，主程序部分对应 text 段，假设程序按照预期执行，那为了方便起见，可以将生成的二进制代码同时存入数据内存和指令内存。

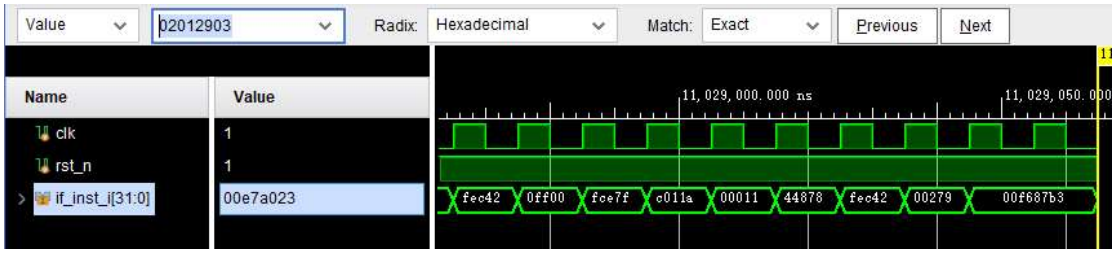
由于需要确定指令入口，所以需要知道 text 段的位置，采用下面的指令查看 section 信息：riscv32-unknown-elf-readelf -S small，结果如下：

```
There are 9 section headers, starting at offset 0x123c:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00010074 000074 000510 00  AX  0   0  4
  [ 2] .rodata           PROGBITS        00010584 000584 0009c8 00   A  0   0  4
  [ 3] .bss              NOBITS          00011000 001000 000408 00  WA  0   0  4
  [ 4] .comment          PROGBITS        00000000 000f4c 000011 01  MS  0   0  1
  [ 5] .riscv.attributes RISCV_ATTRIBUTE 00000000 000f5d 000021 00      0   0  1
  [ 6] .symtab           SYMTAB          00000000 000f80 0001b0 10      7  13  4
  [ 7] .strtab           STRTAB          00000000 001130 0000c0 00      0   0  1
  [ 8] .shstrtab         STRTAB          00000000 0011f0 000049 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

可以看到代码部分在 elf 文件中的偏移量为 74，因此程序从 74 开始运行即可。接下来将二进制文件转化为十六进制文本数据，这里使用了 python 来实现，代码如下：

```
import os
import binascii

if __name__ == '__main__':
    data = ''
    filepath = 'small'
    binfile = open(filepath,'rb')
    size = os.path.getsize(filepath)
    for i in range(int(size/4)):
        temp = str(binascii.b2a_hex(binfile.read(1)))
        temp = str(binascii.b2a_hex(binfile.read(1))) + temp
        temp = str(binascii.b2a_hex(binfile.read(1))) + temp
        temp = str(binascii.b2a_hex(binfile.read(1))) + temp
        data += temp+'\n'
    binfile.close()
    filepath = 'small.data'
    datafile = open(filepath, 'w')
    datafile.write(data)
    datafile.close()
```

后面就是对处理器设计加载程序和数据的部分进行修改，在此不做赘述。之后将程序运行的起始地址更改为 32’h00000074，然后重新仿真。但是并没有正确执行完程序，将待匹配的字符串删除到只剩一个也不行，运行了很多很多时钟周期也没能正常运行的 main 函数的返回语句，花了很多时间排查也毫无头绪，这个程序的运行只能以失败告终。



二、记分牌算法实现进度

（一）记分牌算法

引入记分牌算法是为了实现指令在处理器中的动态调度，减少流水线中因为数据相关而导致的 Stall，比如下面的代码：

```
div      f0, f2, f4
add      f10, f0, f8
sub      f8, f8, f14
```

add 指令需要 f0 作为操作数输入，而这就需要 div 指令先执行完毕，而在我们设计的流水线中除法需要执行 33 个周期，需要执行完毕才能继续执行 add 指令，而 sub 指令时和 div 指令无关的，如果可以在执行 div 指令的同时执行 sub 指令的话，将会一定程度上节约时钟周期数。

记分牌算法就是这样的动态调度算法，每条指令会经过记分牌，其中构造了数据相关性的记录，这一步对应于流水线中的 issue 阶段，取代 id 阶段的一部分；然后记分牌确定指令能否执行，监测硬件中的状态改变并在允许的时机读取操作数，这一部分对应 read oprand 阶段；最后记分牌还会控制指令合适将结果写回寄存器。综上所述，需要对之前的流水线进行修改。

（二）流水线的修改

需要将 ID 阶段拆分为两个阶段：发射（is）阶段和读操作数（ro）阶段，主要的四个阶段如下：

1. IS 阶段：如果该指令的功能单元是空闲的并且没有其他活动指令有相同的目的寄存器，则记分牌将指令发射到功能单元并更新其内部数据结构，这一阶段可以避免 WAW 冒险和结构冒险；

2. RO 阶段：记分牌监测源操作数的可用性，如果没有更早发射的指令会写入该源寄存器，则源操作数可用，当源操作数可用时，数据发送到功能单元并执行，这一部分可以解决 RAW 冒险；

3. EX 阶段：接收到源操作数后开始执行，准备好结果后告知计分板已经完成执行；

4. WB 阶段：一旦记分牌发现功能单元完成执行，则会检查 WAR 冒险并在必要时停止完成指令。

（三）待完成的工作

1. 流水线修改：由于 ID 阶段拆分为两个阶段，因此需要对整个流水线进行修改，这一部分工作较为繁琐；

2. EX 阶段的拆分：在前面的实现中，将所有的运算功能单元都放在了一个模块，但记分牌是对多个功能单元的利用，因此要将功能单元进行拆分，简单的功能单元可以多安排几个，但其实可以只拆分成多个整数运算单元（除法除外）和整数除法单元；

3. 记分牌的实现：记分牌相当于整个流水线的控制阶段，如果存在 Stall 也可以通过记分牌进行控制，因此可以替代之前设计中的 Ctrl 模块，这一部分涉及到的内容很多，工作量较大。

总得来说，还有一周的时间，任务很重，后面的修改还需要细致规划，多花时间去实现。