CS 349,2015 年夏季
优化流水线处理器的性能
分配日期:6 月 6 日,截止日期:6 月 21 日晚上 11:59

Harry Bovik (bovik@cs.cmu.edu) 是这项任务的带头人。

1 简介

在本实验中,您将了解流水线 Y86-64 处理器的设计和实现,优化它和基准程序以最大限度地提高性能。您可以对基准程序进行任何保留语义的转换,或对流水线处理器进行增强,或两者兼而有之。完成实验后,您将对影响程序性能的代码和硬件之间的交互有深刻的认识。

实验室分为三个部分,每个部分都有自己的手册。在 A 部分,您将编写一些简单的 Y86-64 程序并熟悉 Y86-64 工具。在 B 部分,您将使用新指令扩展 SEQ 模拟器。这两部分将为您准备 C 部分,这是实验室的核心,您将在其中优化 Y86-64 基准程序和处理器设计。

2 物流

您将独自在这个实验室工作。

对作业的任何澄清和修订都将发布在课程网页上。

3 讲义说明

SITE-SPECIFIC:在此插入一段说明学生应如何下载 archlab-handout.tar 文件。

1. 首先将文件 archlab-handout.tar 复制到您计划的（受保护的)目录中
做你的工作。

2. 然后给出命令:tar xvf archlab-handout.tar。这将导致以下文件解压到目录中:README、Makefile、sim.tar、archlab.pdf 和 simguide.pdf。

3. 接下来,输入命令 tar xvf sim.tar。这将创建目录 sim,其中包含 Y86-64 工具的个人副本。您将在此目录中完成所有工作。

4.最后,切换到sim目录并构建Y86-64工具:

```
unix> cd sim
unix> 清理;制作
```

4 A 部分

在这部分中,您将在目录 sim/misc 中工作。

您的任务是编写并模拟以下三个 Y86-64 程序。这些程序所需的行为由 examples.c 中的示例 C 函数定义。请务必在每个程序的开头将您的姓名和 ID 放在评论中。您可以通过首先使用程序YAS组装它们然后使用指令集模拟器YIS 运行它们来测试您的程序。

在所有 Y86-64 函数中,您应该遵循 x86-64 约定来传递函数参数、使用寄存器和使用堆栈。这包括保存和恢复您创建的任何被调用者保存寄存器
使用。

sum.ys:迭代求和链表元素

编写一个 Y86-64 程序 sum.ys,它迭代地对链表的元素求和。您的程序应该包含一些设置堆栈结构、调用函数然后停止的代码。在这种情况下,该函数应该是函数(求和列表)的 Y86-64 代码,它在功能上等同于图 1 中的 C 求和列表函数。使用以下三元素列表测试您的程序:

```
# 示例链表 .align 8 ele1:

        .quad 0x00a .quad
        ele2
ele2:
        .quad 0x0b0 .quad
        ele3
ele3:
        .quad 0xc00 .quad
        0
```

```
1 /* 链表元素 */ 2 typedef struct ELE { long val;结构
ELE *下一个； 5 } *list_ptr;
3个
4个


6个
7 /* sum_list - 对链表的元素求和 */ 8 long sum_list(list_ptr ls) 9 {

10              长值 = 0； while (ls)
11              { val += ls->val; ls = ls-
12                      >下一个；
13
14
15              } 返回值；
16 }
17
18 /* rsum_list - sum_list 的递归版本 */ 19 long rsum_list(list_ptr ls) 20 {

21              如果 (!ls) 返回
22                      0； else { long
23              val = ls-
24                      >val;长时间休息 = rsum_list(ls-
25                      >next);返回值 + 休息；
26
27              }
28 }
29
30 /* copy_block - 将 src 复制到 dest 并返回 src 的异或校验和 */ 31 long copy_block(long *src, long *dest, long len) 32 {

33              长结果= 0； while (len > 0)
34              { long val = *src++; *目标++ =
35                      val;结果 ^= val;伦--;
36
37
38
39
40              } 返回结果；
41 }
```

图 1： Y86-64 解决方案函数的 C 版本。参见 sim/misc/examples.c

rsum.ys:递归求和链表元素

编写一个 Y86-64 程序 rsum.ys,递归地对链表的元素求和。此代码应类似于 sum.ys 中的代码,不同之处在于它应使用函数 rsum list 递归地对数字列表求和,如图 1 中的 C 函数 rsum list 所示。使用相同的三个测试程序 -用于测试 list.ys 的元素列表。

copy.ys:将源块复制到目标块

编写一个程序 (copy.ys),将一个单词块从内存的一个部分复制到另一个（非重叠区域)内存区域,计算所有复制单词的校验和 (Xor)。

您的程序应该包含设置堆栈帧、调用函数复制块然后停止的代码。该函数在功能上应该等同于图 Figure 1 中所示的 C 函数复制块。使用以下三元素源块和目标块测试您的程序：

```
.align 8 # 源块

来源：
        .quad 0x00a .quad
        0x0b0 .quad 0xc00

# 目标块
手：
        .quad 0x111 .quad
        0x222 .quad 0x333
```

5 B 部分

在这部分中,您将在目录 sim/seq 中工作。

你在 B 部分的任务是扩展 SEQ 处理器以支持 iaddq,如家庭作业问题 4.51 和 4.52 中所述。要添加此指令,您将修改文件 seq-full.hcl,它实现了 CS:APP3e 教科书中描述的 SEQ 版本。此外,它还包含您的解决方案所需的一些常量的声明。

您的 HCL 文件必须以包含以下信息的标头注释开头：

- 您的姓名和身份证。
- iaddq 指令所需计算的描述。使用 CS:APP3e 文本中图 4.18 中对 irmovq 和 OPq 的描述作为指南。

构建和测试您的解决方案

修改完 seq-full.hcl 文件后,您将需要基于此 HCL 文件构建一个新的 SEQ 模拟器 (ssim) 实例,然后对其进行测试:

- 构建一个新的模拟器。您可以使用 make 构建一个新的 SEQ 模拟器:

  ```
  unix> 使 VERSION=full
  ```

  这将构建一个使用您在 seq-full.hcl 中指定的控制逻辑的 ssim 版本。为了节省输入,您可以在 Makefile 中指定 VERSION=full。

- 在一个简单的Y86-64 程序上测试您的解决方案。对于您的初始测试,我们建议在 TTY 模式下运行简单的程序,例如 asumi.yo（测试 iaddq）,将结果与 ISA 模拟进行比较:

  ```
  unix> ./ssim -t ../y86-code/asumi.yo
  ```

  如果 ISA 测试失败,那么您应该通过单步执行模拟器来调试您的实现
  GUI mode:

  ```
  unix> ./ssim -g ../y86-code/asumi.yo
  ```

- 使用基准程序重新测试您的解决方案。一旦您的模拟器能够正确执行小程序,您就可以在 ../y86-code 中的 Y86-64 基准程序上自动测试它:

  ```
  unix> (cd ../y86-code; make testsim)
  ```

  这将在基准程序上运行 ssim,并通过将生成的处理器状态与高级 ISA 模拟的状态进行比较来检查正确性。请注意,这些程序都不会测试添加的指令。您只是确保您的解决方案没有为原始指令注入错误。有关详细信息,请参阅文件 ../y86-code/README 文件。

- 执行回归测试。一旦可以正确执行基准测试程序,就应该在 ../ptest 中运行大量的回归测试。要测试除 iaddq 之外的所有内容并离开:

  ```
  unix> (cd ../ptest; make SIM=../seq/ssim)
  ```

  测试 iaddq 的实现:

  ```
  unix> (cd ../ptest; make SIM=../seq/ssim TFLAGS=-i)
  ```

有关 SEQ 模拟器的更多信息,请参阅讲义 CS:APP3e Y86-64 处理器模拟器指南 (simguide.pdf)。

```
1 /* *
2     ncopy - 将 src 复制到 dst,返回 src 数组中包含的正整数 * 的数量。 4 */ 5 word_t ncopy(word_t *src, word_t *dst, word_t len)
3     6 {
4
5
6
7         word_t 计数 = 0； word_t 值；
8
9
10        while (len > 0) { val = *src++;
11                *dst++ = val;如果 (val >
12                0) 计数++;伦--;
13
14
15
16
17        } 返回计数；
18 }
```

图 2： ncopy 函数的 C 版本。参见 sim/pipe/ncopy.c。

6 C 部分

在这部分中,您将在目录 sim/pipe 中工作。

图 2 中的 ncopy 函数将一个 len 元素的整数数组 src 复制到一个非重叠的 dst 中,重新计算 src 中包含的正整数的个数。图 3 显示了 ncopy 的基线 Y86-64 版本。文件 pipe-full.hcl 包含 PIPE 的 HCL 代码副本,以及常量值 IIADDQ 的声明。

您在 C 部分的任务是修改 ncopy.ys 和 pipe-full.hcl,目的是使 ncopy.ys 尽可能快地运行。

您将提交两个文件:pipe-full.hcl 和 ncopy.ys。每个文件都应以带有以下信息的标题注释开头：

- 您的姓名和身份证。
- 代码的高级描述。在每种情况下,请描述您修改代码的方式和原因。

编码规则

您可以随意进行任何修改,但有以下限制：

- 您的ncopy.ys 函数必须适用于任意大小的数组。您可能想通过简单地编写 64 条复制指令来硬连接 64 元素数组的解决方案,但这是一个坏主意,因为我们将根据其在任意数组上的性能对您的解决方案进行分级。

```
1 ################################################ ################## 2 # ncopy.ys - 将 len 个单词的 src 块复制到 dst。

3 # 返回src中包含的肯定词（>0)的个数。 4 # 5 # 在此填写您的姓名和 ID。 6 # 7 # 描述您修改基线代码的方式和原因。 8 # 9

################################################

###################

10 # 不要修改这部分11 # 函数序言。 12 # %rdi = src, %rsi = dst,
%rdx = len

13副本：
14
15 ################################################## ##################
16 # 你可以修改这部分
17                  # 循环头 xorq %rax,
18                  %rax andq %rdx,%rdx
19                  jle Done
20                                                  到完成：
21
22循环:mrmovq (%rdi), %r10 rmmovq %r10, (%rsi) andq %r10,
23                  %r10 jle Npos irmovq $1, %r10 addq
24                  %r10, %rax 28 Npos: irmovq
25                  $1, %r10 subq
26                  % r10 , %rdx irmovq $8, %r10
27                  addq %r10, %rdi # src++ # dst+
+ addq %r10, %rsi # len > 0? andq %rdx,%rdx # 如果是
29                  这样,转到 Loop: jg Loop 35
30
31                  ################################
32
33                  数组尾################################ 36 # 不要修改以下代码段37 # 函
34


38完成：
39                  正确的
40 ################################################## ##################
41 # 在函数末尾保留以下标签
42结束：
```

计数 = 0; # 长度 <= 0? # 如果是这样,转

从 src 读取 val... # ... 并将其存储到 dst # val <= 0? # 如果是这样,转到 Npos：

计数++

只是 -

图 3： ncopy 函数的基线 Y86-64 版本。请参见 sim/pipe/ncopy.ys。

- 您的ncopy.ys 函数必须与YIS 一起正确运行。正确地,我们的意思是它必须正确地
  复制 src 块并返回（在 %rax 中)正确数量的正整数。
- ncopy 文件的汇编版本不得超过 1000 字节长。您可以使用提供的脚本 check-len.pl 检查任何嵌入了 ncopy 函数的程序的长度：

  unix> ./check-len.pl < ncopy.yo

- 您的 pipe-full.hcl 实现必须通过 ../y86-code 和 ../ptest 中的回归测试
  （没有测试 iaddq 的 -i 标志）。

除此之外,如果您认为有帮助,您可以自由实施 iaddq 指令。您可以对 ncopy.ys 函数进行任何保留语义的转换,例如重新排序指令、用单个指令替换指令组、删除一些指令以及添加其他指令。您可能会发现阅读 CS:APP3e 的第 5.8 节中有关循环展开的内容很有用。

构建和运行您的解决方案

为了测试您的解决方案,您需要构建一个调用 ncopy 函数的驱动程序。我们为您提供了 gen-driver.pl 程序,它可以为任意大小的输入数组生成驱动程序。例如,键入

unix> 制作驱动程序

将构建以下两个有用的驱动程序：

- sdriver.yo：一个小型驱动程序,用于在具有 4 个元素的小型数组上测试 ncopy 函数。
  如果您的解决方案是正确的,那么此程序将在复制 src 数组后在寄存器 %rax 中的值为 2 时停止。

- ldriver.yo：一个大型驱动程序,用于在包含 63 个元素的较大数组上测试 ncopy 函数。如果您的解决方案是正确的,则此程序将在复制 src 数组后停止,寄存器 %rax 中的值为 31 (0x1f)。

每次修改 ncopy.ys 程序时,您都可以通过键入以下内容重建驱动程序

unix> 制作驱动程序

每次修改 pipe-full.hcl 文件时,您都可以通过键入来重建模拟器

unix> 使 psim 版本=完整

如果要重建模拟器和驱动程序,请键入

unix> 使 VERSION=full

要在 GUI 模式下在小型 4 元素数组上测试您的解决方案,请键入

unix> ./psim -g sdriver.yo

要在更大的 63 元素数组上测试您的解决方案,请键入

unix> ./psim -g ldriver.yo

一旦您的模拟器在这两个块长度上正确运行您的 ncopy.ys 版本,您将需要执行以下附加测试:

- 在ISA 模拟器上测试您的驱动程序文件。确保你的 ncopy.ys 函数有效
  与 YIS 相处融洽:

  unix> make drivers unix> ../
  misc/yis sdriver.yo

- 使用ISA 模拟器在一系列块长度上测试您的代码。 Perl 脚本 correctness.pl 生成块长度从 0 到某个限制(默认 65)以及更大尺寸的驱动程序文件。
  它模拟它们(默认情况下使用YIS),并检查结果。它生成一份报告,显示每个块长度的状态:

  unix> ./correctness.pl

  此脚本生成测试程序,其中每次运行的结果计数随机变化,因此它提供了比标准驱动程序更严格的测试。

  如果您得到某个长度 K 的错误结果,您可以生成该长度的驱动程序文件,其中包括检查代码,并且结果随机变化:

  unix> ./gen-driver.pl -f ncopy.ys -n K -rc > driver.ys unix> make driver.yo unix> ../misc/yis
  driver.yo

  该程序将以具有以下值的寄存器 %rax 结束:

  0xaaaa :所有测试通过。
  0xbbbb : 计数不正确

  0xcccc :函数 ncopy 的长度超过 1000 个字节。 0xdddd : 一些源数据没有

  复制到它的目的地。 0xeeee :目标区域损坏之前或之后的某个单词。

- 在基准程序上测试您的管道模拟器。一旦您的模拟器能够正确执行 sdriver.ys 和 ldriver.ys,您应该针对 ../y86-code 中的 Y86-64 基准程序对其进行测试:

  unix> (cd ../y86-code; make testpsim)

这将在基准程序上运行 psim,并将结果与 YIS 进行比较。

- 使用广泛的回归测试测试您的管道模拟器。一旦您可以正确执行基准程序,那么您应该使用 ../ptest 中的回归测试来检查它。例如,如果您的解决方案实现了 iaddq 指令,则

```
unix> (cd ../ptest; make SIM=../pipe/psim TFLAGS=-i)
```

- 使用管道模拟器在一系列块长度上测试您的代码。最后,您可以运行
  您之前使用 ISA 模拟器在管道模拟器上进行的相同代码测试

```
unix> ./correctness.pl -p
```

7 评价

实验室值 190 分:A 部分 30 分,B 部分 60 分,C 部分 100 分。

甲部

A部分30分,每个Y86-64解题方案10分。将评估每个解决方案程序的正确性,包括堆栈和寄存器的正确处理,以及与 examples.c 中的示例 C 函数的功能等效性。

如果评分者没有发现任何错误,程序 sum.ys 和 rsum.ys 将被认为是正确的,并且它们各自的 sum list 和 rsum list 函数在寄存器 %rax 中返回总和 0xcba。

如果评分者没有发现任何错误,程序 copy.ys 将被认为是正确的,并且复制块函数返回寄存器 %rax 中的和 0xcba,将三个 64 位值 0x00a、0x0b 和 0xc 复制到 24从地址 dest 开始的字节,并且不会破坏其他内存位置。

B部分

实验室的这一部分价值 35 分:

- iaddq 指令所需计算的描述得 10 分。
- 通过 y86 代码中的基准回归测试得 10 分,以验证您的模拟器仍然
  正确执行基准套件。
- 通过 iaddq 的 ptest 回归测试得 15 分。

C部分

实验室的这一部分得分为 100 分:如果您的ncopy.ys代码或修改后的模拟器未能通过前面描述的任何测试,您将不会获得任何分数。

- 对于您在ncopy.ys 和pipe-full.hcl 的标题中的描述以及这些实现的质量,各20 分。

- 60 分表现。要获得此处的荣誉,您的解决方案必须是正确的,如前所述。
  也就是说,ncopy 与YIS 一起正确运行,并且 pipe-full.hcl 通过了 y86-code 和 ptest 中的所有测试。

  我们将以每个元素的周期数 (CPE) 为单位来表达您的函数的性能。也就是说,如果模拟代码需要 C 周期来复制一个包含 N 个元素的块,那么 CPE 就是 C/N。 PIPE 模拟器显示完成程序所需的循环总数。在具有大型 63 元素阵列的标准 PIPE 模拟器上运行的 ncopy 函数的基线版本需要 897 个周期来复制 63 个元素,CPE 为 897/63 = 14.24。

  由于一些循环用于设置对 ncopy 的调用以及在 ncopy 内设置循环,您会发现对于不同的块长度,您将获得不同的 CPE 值（通常 CPE 会随着 N 的增加而下降）。因此,我们将通过计算 1 到 64 个元素的块的 CPE 平均值来评估您的函数的性能。您可以使用管道目录中的 Perl 脚本 benchmark.pl 在一定范围的块长度上运行 ncopy.ys 代码的模拟并计算平均 CPE。只需运行命令

  unix> ./benchmark.pl

  看看会发生什么。例如,ncopy 函数的基线版本的 CPE 值介于 29.00 和 14.27 之间,平均值为 15.18。请注意,此 Perl 脚本不会检查答案的正确性。为此使用脚本 correctness.pl。

  您应该能够达到低于 9.00 的平均 CPE。我们最好的版本平均为 7.48。如果您的平均 CPE 是 c,那么您在这部分实验中的分数 S 将是：

$$\text{小号} = \begin{cases} 0 & c > 10.5 \\ 20 \cdot (10.5 - c), & 7.50 \leqslant c \leqslant 10.50 \\ 60 & c < 7.50 \end{cases}$$

  默认情况下,benchmark.pl 和 correctness.pl 编译和测试 ncopy.ys。使用 -f 参数指定不同的文件名。 -h 标志给出了命令行参数的完整列表。

8 递交说明

SITE-SPECIFIC：插入说明,解释学生应如何提交实验室的三个部分。这是我们在 CMU 使用的描述。

- 您将提交三组文件：
  - A 部分：sum.ys、rsum.ys 和 copy.ys。
  - B 部分：seq-full.hcl。

– C 部分:ncopy.ys 和 pipe-full.hcl。

- 确保在每个提交文件顶部的注释中包含您的姓名和ID。
- 要提交第 X 部分的文件,请转到您的 archlab-handout 目录并键入:

  unix> make handin-partX TEAM=团队名称

  其中 X 是 a、b 或 c,teamname 是您的 ID。例如,提交 A 部分:

  unix> make handin-parta TEAM=团队名称

- 提交后,如果您发现错误并想提交修改后的副本,请键入

  unix make handin-partX TEAM=团队名称 VERSION=2

  每次提交时不断增加版本号。

- 您可以通过查看来验证您的提交

  CLASSDIR/archlab/handin-partX

  您在此目录中具有列出和插入权限,但没有读取或写入权限。

9 条提示

- 按照设计,sdriver.yo 和ldriver.yo 都足够小,可以在GUI 模式下进行调试。我们发现在 GUI 模式下调试最容易,因此建议您使用它。
- 如果您在 Unix 服务器上以 GUI 模式运行,请确保您已经初始化了 DISPLAY
  环境变量:

  unix> setenv 显示 myhost.edu:0

- 对于某些 X 服务器,当您运行 psim 时,“程序代码”窗口开始时是一个关闭的图标
  或 GUI 模式下的 ssim。只需单击图标即可展开窗口。
- 对于某些基于 Microsoft Windows 的 X 服务器,“内存内容”窗口不会自动显示
  cally 自行调整大小。您需要手动调整窗口大小。
- 如果您要求 psim 和 ssim 模拟器执行文件,它们将以分段错误终止
  那不是有效的 Y86-64 目标文件。