实验 4 使用 MIPS 指令实现冒泡排序法

张梓良
2021212484

日期：2024 年 5 月 22 日

1 实验目的

1. 掌握静态调度方法
2. 增强汇编语言编程能力
3. 学会使用模拟器中的定向功能进行优化

2 实验平台

指令级和流水线操作级模拟器 MIPSsim。

3 实验内容

1. 自行编写一个实现冒泡排序的汇编程序，该程序要求可以实现对一维整数数组进行冒泡排序。冒泡排序算法的运作如下:
   (a) 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
   (b) 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
   (c) 针对所有的元素重复以上的步骤，除了最后一个。
   (d) 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。
   要求数组长度不得小于 10。
2. 启动 MIPSsim。
3. 载入自己编写的程序，观察流水线输出结果。
4. 使用定向功能再次执行代码，与刚才执行结果进行比较，观察执行效率的不同。
5. 采用静态调度方法重排指令序列，减少相关，优化程序。
6. 对优化后的程序使用定向功能执行，与刚才执行结果进行比较，观察执行效率的不同。

注意：不要使用浮点指令及浮点寄存器；整数减勿使用 SUB 指令，请使用 DSUB 指令代替。

4 实验步骤及分析

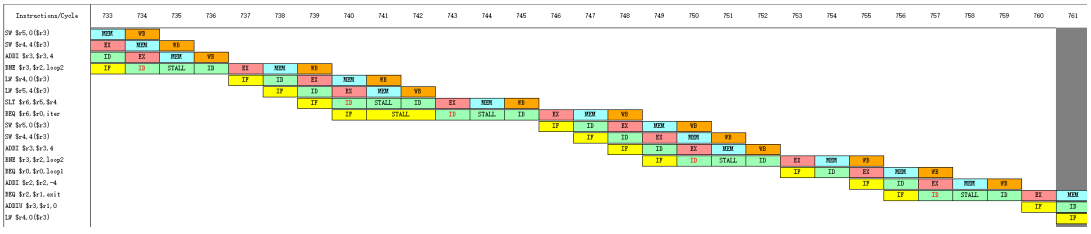
自行编写一个实现冒泡排序的汇编程序：

```
.text
main:
ADDIU   $r1, $r0, a # 取a地址
ADDIU   $r2, $r0, n # 取n地址
LW      $r2, 0($r2) # 取n
BGEZAL  $r0, bubble # 调用bubble sort
NOP
TEQ     $r0, $r0

bubble:
SLL     $r2, $r2, 2 # n<<=2
ADD     $r2, $r2, $r1   # 取a[n]地址

loop1:
ADDI    $r2, $r2, -4 # n--
BEQ     $r2, $r1, exit   # n==1跳转到exit
ADDIU   $r3, $r1, 0 # 取a[1]地址

loop2:
LW      $r4, 0($r3) # 取a[j]
LW      $r5, 4($r3) # 取a[j+1]
SLT     $r6, $r5, $r4   # 判断a[j+1]<a[j]
BEQ     $r6, $r0, iter  # a[j+1]>=a[j]跳转iter
SW      $r5, 0($r3) # 交换a[j]和a[j+1]
SW      $r4, 4($r3)

iter:
ADDI    $r3, $r3, 4 # j++
BNE     $r3, $r2, loop2 # j!=n跳转到loop2
BEQ     $r0, $r0, loop1 # 跳转到loop1

exit:
JR     $r31

.data
a:
.word 10,9,8,7,6,5,4,3,2,1
n:
.word 10
```

载入自己编写的程序，观察流水线输出结果。

```
  汇总:
    执行周期总数: 833
    ID段执行了407条指令

  硬件配置:
    内存容量: 4096 B
```

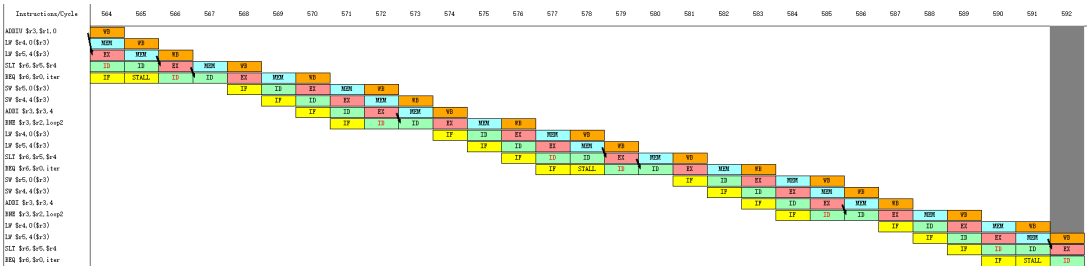
```
  加法器个数: 1    执行时间（周期数）: 6
  乘法器个数: 1    执行时间（周期数）7
  除法器个数: 1    执行时间（周期数）10
  定向机制: 不采用

停顿（周期数）:
  RAW停顿: 314     占周期总数的百分比: 37.69508%
  其中:
    load停顿: 90     占所有RAW停顿的百分比: 28.66242%
    浮点停顿: 0    占所有RAW停顿的百分比: 0%
  WAW停顿: 0     占周期总数的百分比: 0%
  结构停顿: 0    占周期总数的百分比: 0%
  控制停顿: 111    占周期总数的百分比: 13.32533%
  自陷停顿: 0    占周期总数的百分比: 0%
  停顿周期总数: 425 占周期总数的百分比: 51.02041%

分支指令:
  指令条数: 110    占指令总数的百分比: 27.02703%
  其中:
    分支成功: 48     占分支指令数的百分比: 43.63636%
    分支失败: 63     占分支指令数的百分比: 57.27273%

load/store指令:
  指令条数: 181    占指令总数的百分比: 44.47174%
  其中:
    load: 91     占load/store指令数的百分比: 50.27624%
    store: 90    占load/store指令数的百分比: 49.72376%

浮点指令:
  指令条数: 0    占指令总数的百分比: 0%
  其中:
    加法: 0    占浮点指令数的百分比: 0%
    乘法: 0    占浮点指令数的百分比: 0%
    除法: 0    占浮点指令数的百分比: 0%

自陷指令:
  指令条数: 1    占指令总数的百分比: 0.2457002%
```



其中主要是 RAW 冲突。

使用定向功能再次执行代码，与刚才执行结果进行比较，观察执行效率的不同。

```
汇总:
  执行周期总数: 664
  ID段执行了407条指令

硬件配置:
  内存容量: 4096 B
  加法器个数: 1   执行时间 (周期数) : 6
  乘法器个数: 1   执行时间 (周期数) 7
  除法器个数: 1   执行时间 (周期数) 10
  定向机制: 采用

停顿 (周期数) :
  RAW停顿: 145    占周期总数的百分比: 21.83735%
  其中:
    load停顿: 45    占所有RAW停顿的百分比: 31.03448%
    浮点停顿: 0   占所有RAW停顿的百分比: 0%
  WAW停顿: 0    占周期总数的百分比: 0%
  结构停顿: 0   占周期总数的百分比: 0%
  控制停顿: 111   占周期总数的百分比: 16.71687%
  自陷停顿: 0   占周期总数的百分比: 0%
  停顿周期总数: 256 占周期总数的百分比: 38.55422%

分支指令:
  指令条数: 110   占指令总数的百分比: 27.02703%
  其中:
    分支成功: 48    占分支指令数的百分比: 43.63636%
    分支失败: 63    占分支指令数的百分比: 57.27273%

load/store指令:
  指令条数: 181   占指令总数的百分比: 44.47174%
  其中:
    load: 91    占load/store指令数的百分比: 50.27624%
    store: 90   占load/store指令数的百分比: 49.72376%

浮点指令:
  指令条数: 0   占指令总数的百分比: 0%
  其中:
    加法: 0   占浮点指令数的百分比: 0%
    乘法: 0   占浮点指令数的百分比: 0%
    除法: 0   占浮点指令数的百分比: 0%

自陷指令:
  指令条数: 1   占指令总数的百分比: 0.2457002%
```



定向功能消除了部分数据冲突，但是当相邻的指令发生 RAW 冲突时，仍然会停顿一个周期，性能提升了 **833/664=1.25** 倍。

采用静态调度方法重排指令序列，在上述**造成数据冲突的指令之间插入无关指令形成延迟槽，消除数据冲突**，优化程序。

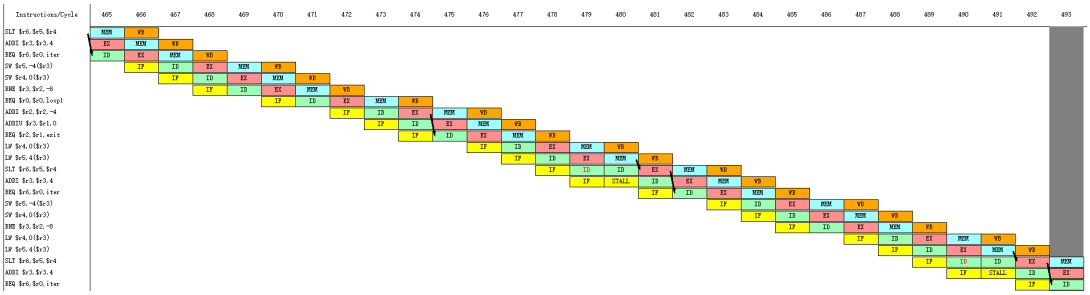
```
1  .text
2  main:
3  ADDIU   $r1, $r0, a # 取a地址
4  ADDIU   $r2, $r0, n # 取n地址
5  LW      $r2, 0($r2) # 取n
6  BGEZAL  $r0, bubble # 调用bubble sort
7  NOP
8  TEQ     $r0, $r0
9
10 bubble:
11 SLL     $r2, $r2, 2 # n<<=2
12 ADD     $r2, $r2, $r1   # 取a[n]地址
13
14 loop1:
15 ADDI    $r2, $r2, -4 # n--
16 ADDIU   $r3, $r1, 0 # 取a[1]地址
17 BEQ     $r2, $r1, exit   # n==1跳转到exit
18
19 loop2:
20 LW      $r4, 0($r3) # 取a[j]
21 LW      $r5, 4($r3) # 取a[j+1]
22 SLT     $r6, $r5, $r4   # 判断a[j+1]<a[j]
23 ADDI    $r3, $r3, 4 # j++
24 BEQ     $r6, $r0, iter  # a[j+1]>=a[j]跳转iter
25 SW      $r5, -4($r3) # 交换a[j]和a[j+1]
26 SW      $r4, 0($r3)
27
28 iter:
29 BNE     $r3, $r2, loop2 # j!=n跳转到loop2
30 BEQ     $r0, $r0, loop1 # 跳转到loop1
31
32 exit:
33 JR     $r31
34
```

```
.data
a:
.word 10,9,8,7,6,5,4,3,2,1
n:
.word 10
```

对优化后的程序使用定向功能执行，与刚才执行结果进行比较，观察执行效率的不同。

```
  汇总:
    执行周期总数: 565
    ID段执行了408条指令

  硬件配置:
    内存容量: 4096 B
    加法器个数: 1   执行时间（周期数）: 6
    乘法器个数: 1   执行时间（周期数）7
    除法器个数: 1   执行时间（周期数）10
    定向机制: 采用

  停顿（周期数）:
    RAW停顿: 45   占周期总数的百分比: 7.964602%
    其中:
      load停顿: 45    占所有RAW停顿的百分比: 100%
      浮点停顿: 0   占所有RAW停顿的百分比: 0%
    WAW停顿: 0    占周期总数的百分比: 0%
    结构停顿: 0   占周期总数的百分比: 0%
    控制停顿: 111   占周期总数的百分比: 19.64602%
    自陷停顿: 0   占周期总数的百分比: 0%
    停顿周期总数: 156 占周期总数的百分比: 27.61062%

  分支指令:
    指令条数: 110   占指令总数的百分比: 26.96078%
    其中:
      分支成功: 48    占分支指令数的百分比: 43.63636%
      分支失败: 63    占分支指令数的百分比: 57.27273%

  load/store指令:
    指令条数: 181   占指令总数的百分比: 44.36274%
    其中:
      load: 91    占load/store指令数的百分比: 50.27624%
      store: 90   占load/store指令数的百分比: 49.72376%

  浮点指令:
    指令条数: 0   占指令总数的百分比: 0%
    其中:
      加法: 0   占浮点指令数的百分比: 0%
      乘法: 0   占浮点指令数的百分比: 0%
      除法: 0   占浮点指令数的百分比: 0%
```

```
41
42    自陷指令:
43      指令条数: 1    占指令总数的百分比: 0.245098%
```



消除了绝大部分数据冲突，性能提升了 **664/565=1.18** 倍。