北京郵電大學

实验报告

题目：使用 MIPS 指令实现冒泡排序法

班　　级：　2020211310

学　　号：　2020211616

姓　　名：　付容天

学　　院：　计算机学院（国家示范性软件学院）

2023 年 5 月 8 日

一、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。

二、实验平台

实验平台采用指令级和流水线操作级模拟器 MIPSsim。

三、实验内容

要求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：

（1） 比较相邻的元素，如果第一个比第二个大，就交换这两个元素；
（2） 对每一对相邻元素作同样的工作，从开始第一对到结尾最后一对。在这一点，最后的元素应该会是最大的数；
（3） 针对所有的元素重复以上的步骤，除了最后一个；
（4） 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。
（5） 要求数组长度不得小于 10。

程序的执行过程为：

（1） 启动 MIPSsim；
（2） 载入自己编写的程序，观察流水线输出结果；
（3） 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；
（4） 采用静态调度方法重排指令序列，减少相关，优化程序；
（5） 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：

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

四、实验步骤及实验分析

我首先编写了冒泡排序的 C 语言代码，如下所示：

```
void bubble_func(int arr[], int len) {
    int i, j, temp;
    for (i = 0; i < len - 1; i++)
        for (j = 0; j < len - 1 - i; j++)
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
}
```

接下来据此写出朴素的汇编语言代码，如下所示：

```
.text
main:
ADDIU  $r4, $r0, a
ADDIU  $r5, $r0, n
LW     $r5, 0($r5)
BGEZAL $r0, bubble_func
NOP
TEQ    $r0, $r0

bubble_func:  # 冒泡排序函数
ADDIU  $r7, $r5, -1
BLEZ   $r7, exit
SLL    $r5, $r5, 2
ADDIU  $r8, $r4, 4
ADDU   $r6, $r4, $r5
loop:
ADDIU  $r2, $r8, 0
run:
LW     $r3,-4($r2)        # 读入元素
LW     $r4, 0($r2)        # 读入元素
SLT    $r5, $r4, $r3
BEQ    $r5, $r0, end      # 元素大小比较
swap:
SW     $r4,-4($r2)        # 交换元素
SW     $r3, 0($r2)        # 交换元素
end:
ADDIU  $r2, $r2, 4        # 元素索引更新(+4字节)
BNE    $r6, $r2, run      # 判断循环是否结束
```

```
ADDIU   $r7, $r7,-1
ADDIU   $r6, $r6,-4
BNE     $r7, $r0, loop    # 判断循环是否结束

exit:
JR      $r31

.data
a:      # 待排序数组a的元素内容
.word   11,10,9,8,7,6,5,4,3,2,1
n:      # 待排序数组a的元素数量
.word   11
```

上面的汇编代码实现了对数组 a（长度为 11）的冒泡排序，实现思路与上述 C 语言代码相同。在未开启定向技术的时候，执行结果如下所示：

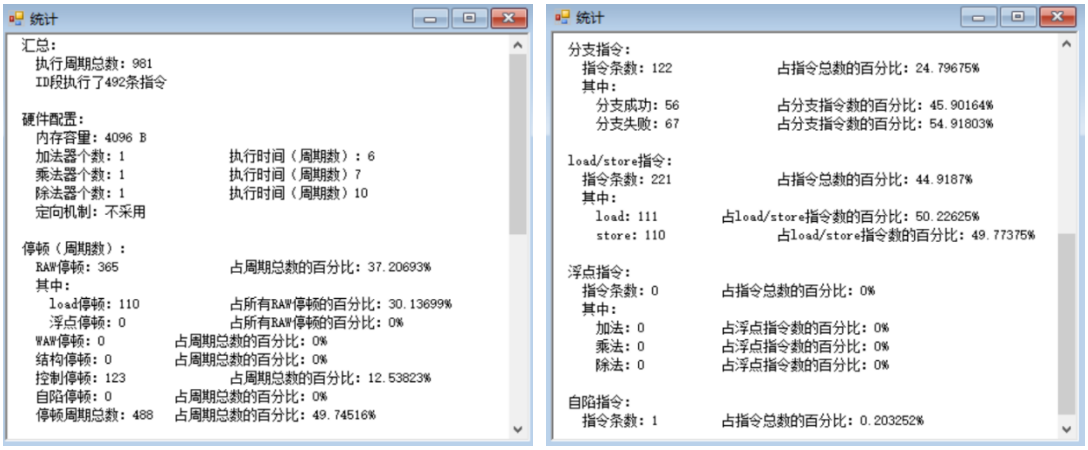

图 1：未开启定向技术时的执行结果（统计信息）

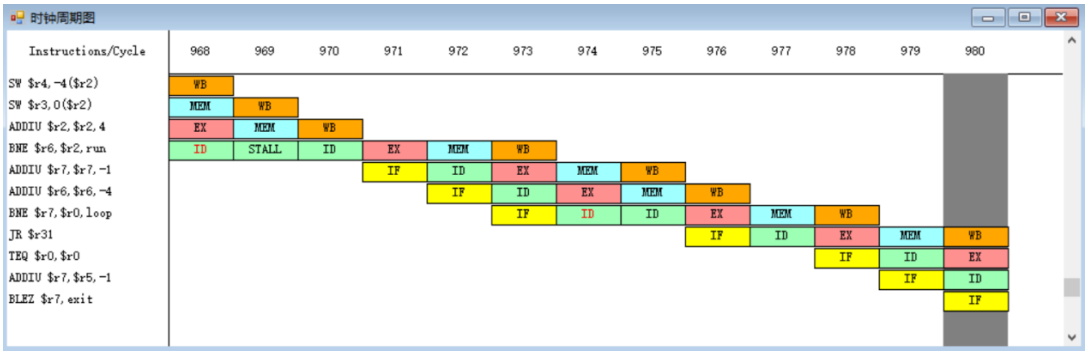

图 2：未开启定向技术时的执行结果（时钟周期图）

接下来我开启定向技术，执行得到如下结果：

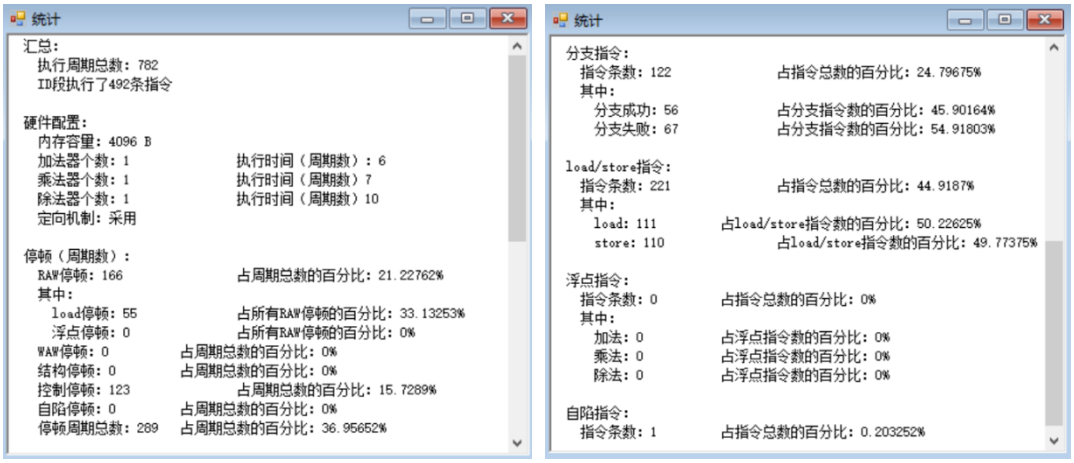

图 3：开启定向技术时的执行结果（统计信息）

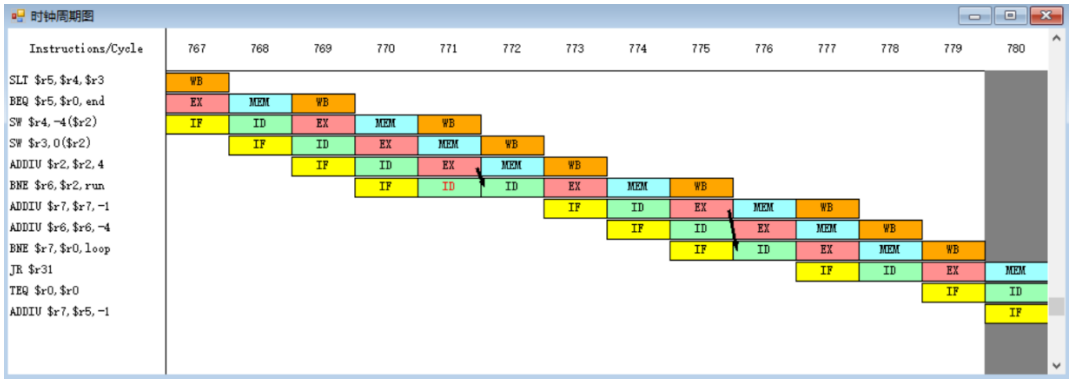

图 4：开启定向技术时的执行结果（时钟周期图）

可以发现，停顿周期占周期总数的百分比从约 45.74%降至约 36.95%，性能得到了较高的提升。

观察可知，上述汇编代码可供优化的地方并不多（这是因为冒泡排序中循环和判断较多，导致代码之间的“相关性”是比较强的），仅有的优化点在于 STL 指令处，优化后代码如下：

```
.text
main:
ADDIU   $r4, $r0, a
ADDIU   $r5, $r0, n
LW      $r5, 0($r5)
BGEZAL  $r0, better_bubble_func
NOP
TEQ     $r0, $r0

better_bubble_func:   # 改良后的冒泡排序函数
ADDIU   $r7, $r5, -1
BLEZ    $r7, exit
SLL     $r5, $r5, 2
```

```
ADDIU   $r8, $r4, 4
ADDU    $r6, $r4, $r5

loop:
ADDIU   $r2, $r8, 0
run:
LW      $r3,-4($r2)         # 读入元素
LW      $r4, 0($r2)         # 读入元素
BLT     $r4, $r3, end       # 元素大小比较
swap:
SW      $r4,-4($r2)         # 交换元素
SW      $r3, 0($r2)         # 交换元素
end:
ADDIU   $r2, $r2, 4         # 元素索引更新(+4字节)
BNE     $r6, $r2, run       # 判断循环是否结束
ADDIU   $r7, $r7,-1
ADDIU   $r6, $r6,-4
BNE     $r7, $r0, loop      # 判断循环是否结束
exit:
JR      $r31

.data
a:      # 待排序数组a的元素内容
.word   11,10,9,8,7,6,5,4,3,2,1
n:      # 待排序数组a的元素数量
.word   11
```

在启用定向技术的前提下执行这段优化后的代码，得到下面的结果：

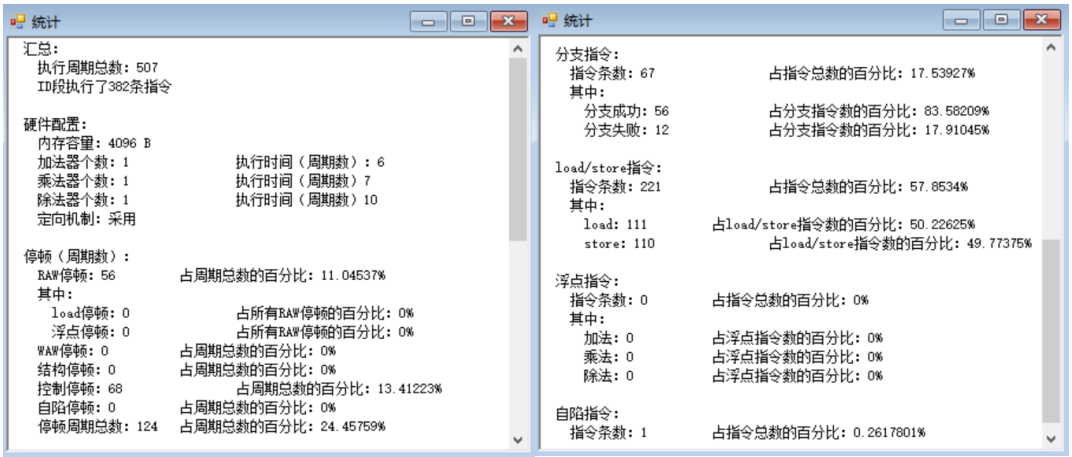

图 5：优化且开启定向技术时的执行结果（统计信息）

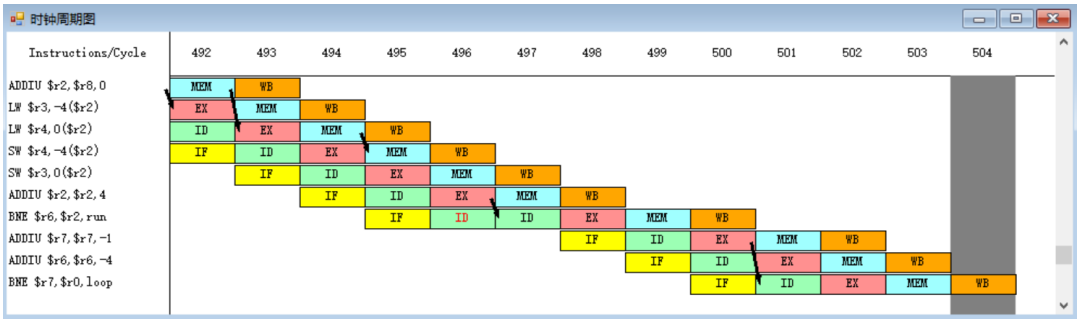

图 6：优化且开启定向技术时的执行结果（时钟周期图）

可以看到，停顿周期占总执行周期的百分比进一步降至约 24.45%，性能得到了进一步的提升！与仅仅采用定向技术的程序执行结果相比，效率大约是之前的 782/507=1.54 倍；而与最朴素的汇编代码相比，效率大约是之前的 981/507=1.93 倍，可见性能提升非常明显！

五、实验总结

在本次实验中，我回顾了冒泡排序的基本知识，并用汇编代码对其进行了实现，然后先后使用定向技术和代码级优化提高了程序的执行效率，圆满完成了实验四既定的实验任务。我还加深了对执行中 MIPS 程序行为的理解。我对实验四中程序的执行过程分析得很细致，弄清楚了每一处细节，收获颇丰！