北京郵電大學

实验报告

题目：使用 MIPS 指令实现求两个数组的点积

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) 通过实验熟悉实验 1 和实验 2 的内容；
(2) 增强汇编语言的编程能力；
(3) 学会使用模拟器中的定向功能进行优化；
(4) 了解对代码进行优化的方法。

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

本实验要求自行编写一个计算两个向量点积的汇编程序，该程序要求可以实现求两个向量点积计算后的结果。向量点积的定义如下：

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

要求如下：

（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） 使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，与刚才的结果进行比较，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） 使用 TEQ $r0 $r0 结束程序。

实验报告要求：

（1） 实验目的；
（2） 实验原理；
（3） 向量点积程序代码清单及注释说明；
（4） 优化后的程序代码清单；
（5） 未优化代码和优化代码性能分析比较结果（文字和截图说明），并给出优化的原因。

四、实验记录及实验分析

向量点积用 C 语言代码可以表述为：

```c
int multi_func(int *a, int *b, int n) {
    int len = n, ans = 0;
    for(int i=0; i<len; i++)
        ans += a[i] * b[i];
    return ans;
}
```

接下来，我按照 MIPS 系统的规则，将这段代码翻译为如下所示的汇编代码：

```
.text
main:
ADDIU  $r4, $r0, a
ADDIU  $r5, $r0, b
ADDIU  $r6, $r0, n
BGEZAL $r0, multi_func
NOP
TEQ    $r0, $r0

multi_func:
LW     $r6, 0($r6)
ADD    $r8, $r0, $r0
ADD    $r2, $r0, $r0
loop:
LW     $r9, 0($r4)        # 读入向量元素
LW     $r10,0($r5)        # 读入向量元素
MUL    $r11,$r9, $r10     # 元素相乘
ADD    $r2, $r2, $r11     # 结果累加
ADDIU  $r4, $r4, 4        # 更新向量索引(+4字节)
ADDIU  $r5, $r5, 4        # 更新向量索引(+4字节)
ADDIU  $r8, $r8, 1        # 计数变量自增
BNE    $r8, $r6, loop     # 判断循环是否结束
JR     $r31

.data:
a:     # 向量a元素内容
.word  1,2,3,4,5,6,7,8,9,10
b:     # 向量b元素内容
.word  1,2,3,4,5,6,7,8,9,10
n:     # 向量a和b的元素数量
.word  10
```

在上面的代码中，参与点积运算的两个向量为 a 和 b，长度均为 10，元素均为 1 到 10 这是个数字。计算点积的主体为 multi_func，通过两个寄存器 r9 和 r10 分别记录两个向量对应位置的元素，通过 MUL 指令将乘法结果存放在寄存器 r11 中，并使用 ADD 指令实现“乘加+=”的效果，随后更新索引位置，并判断是否达到向量结束位置。在执行完 multi_func 功能后结束主函数 main。

根据实验要求，我首先在不开启定向功能的条件下执行了这段代码，得到下面的结果：

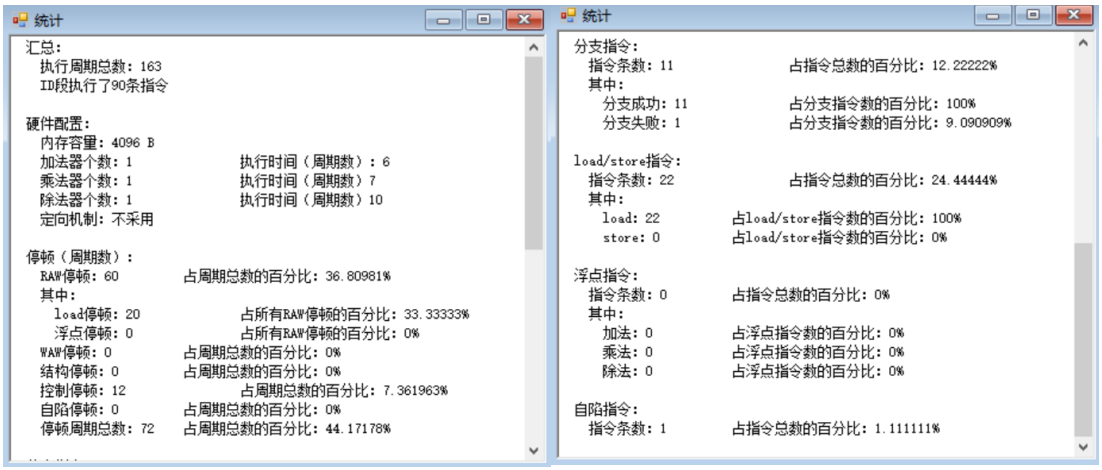

图 1：不开启定向功能时的运行结果（统计信息）

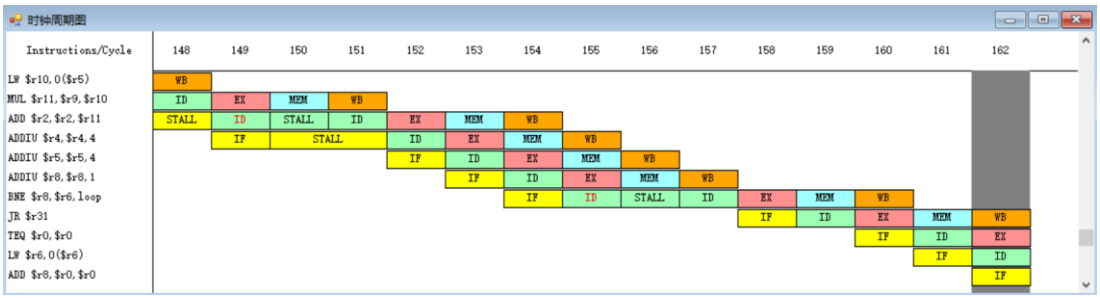

图 2：不开启定向功能时的运行结果（时钟周期图）

然后，按照实验要求，我打开了定向功能，得到了如下的结果：

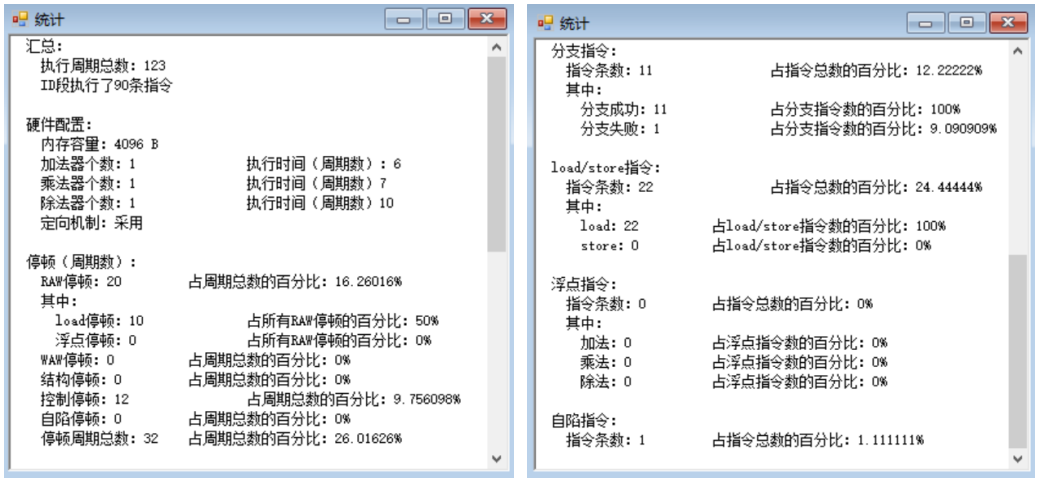

图 3：开启定向功能时的运行结果（统计信息）

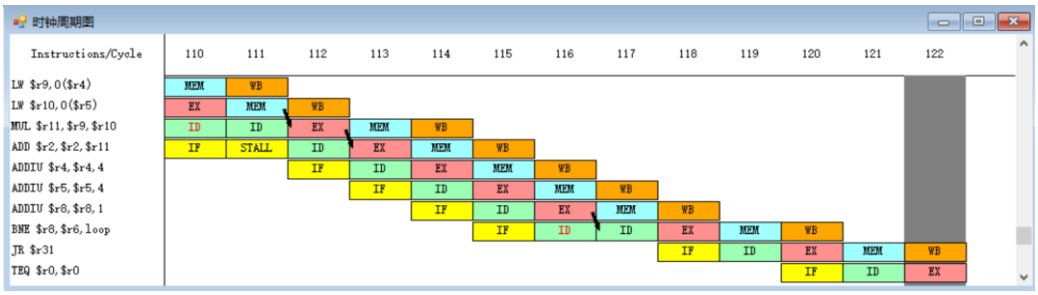

图 4：开启定向功能时的运行结果（时钟周期图）

可以看到，停顿周期总数占时钟周期的百分比从约 44.17%降到了约 26.02%，性能有着巨大的提升。

不难发现，在上述的汇编代码中存在数据相关，这也正是我对其进行优化的主要思路，尽可能避开这些数据冲突，得到了下面的代码：

```
.text
main:
ADDIU  $r4, $r0, a
ADDIU  $r5, $r0, b
ADDIU  $r6, $r0, n
BGEZAL $r0, better_func
NOP
TEQ    $r0, $r0

better_func:
LW     $r6, 0($r6)
ADD    $r8, $r0, $r0
ADD    $r2, $r0, $r0
loop:
LW     $r9, 0($r4)        # 读入向量元素
LW     $r10,0($r5)        # 读入向量元素
ADDIU  $r4, $r4, 4        # 更新向量索引(+4字节)
ADDIU  $r5, $r5, 4        # 更新向量索引(+4字节)
MUL    $r11,$r9, $r10     # 元素相乘
ADDIU  $r8, $r8, 1        # 计数变量自增
ADD    $r2, $r2, $r11     # 结果累加
BNE    $r8, $r6, loop     # 判断循环是否结束
JR     $r31

.data
a:     # 向量a元素内容
.word  1,2,3,4,5,6,7,8,9,10
b:     # 向量b元素内容
.word  1,2,3,4,5,6,7,8,9,10
n:     # 向量a和b的元素数量
```

```
.word   10
```

在上面的代码中，我将原本紧密相连的两条 LW 和 MUL 指令拆开，在执行完两条 LW 指令后先进行索引更新（即两条 ADDIU 指令），然后再执行 MUL 指令，有效处理了数据相关。打开定向功能，得到如下执行结果：

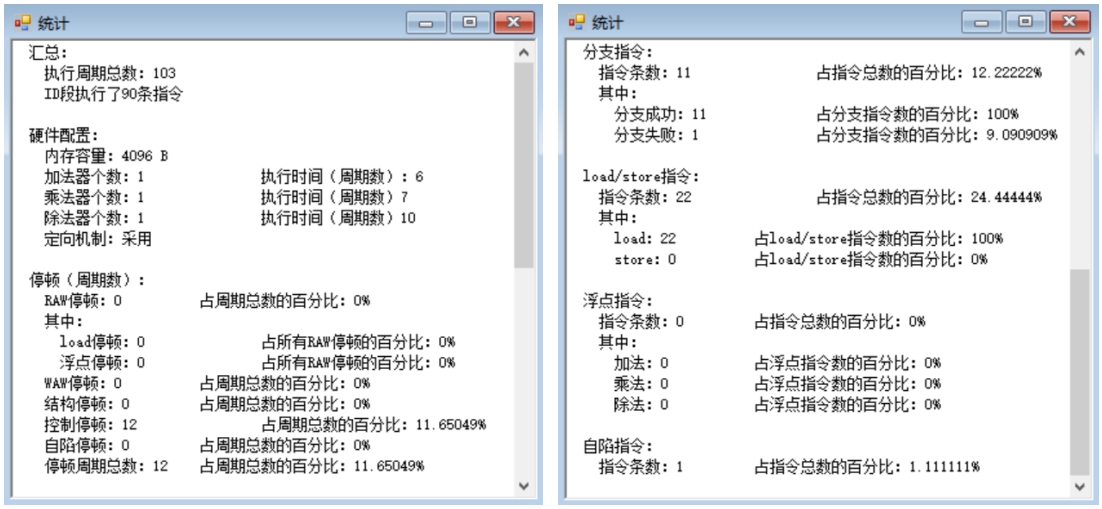

图 5：优化且开启定向时的执行结果（统计信息）

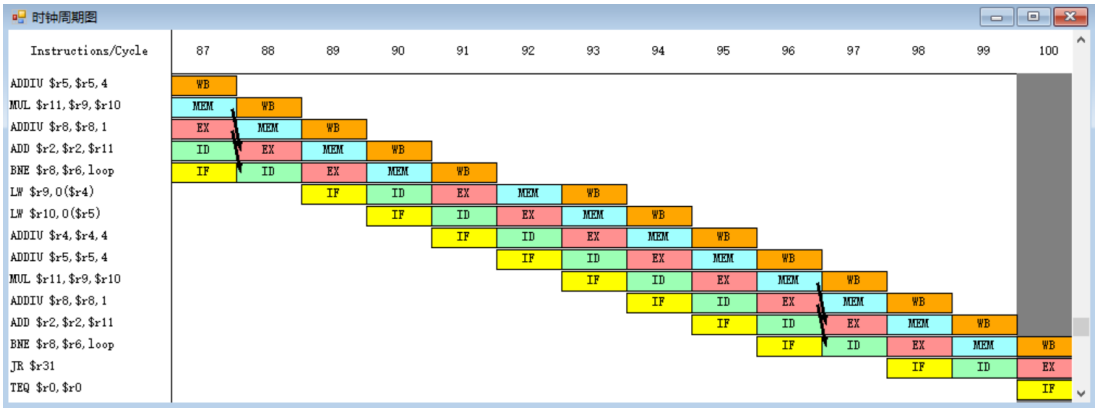

图 6：优化且开启定向时的执行结果（时钟周期图）

可以看到，停顿周期总数占周期总数的百分比进一步下降至了约 11.65%，性能得到了进一步的提升。与最开始的程序相比，效率大概是 163/103=1.58 倍，与仅仅开启定向的程序相比，效率大概是 123/103=1.19 倍。

从三个执行过程的时钟周期图可以发现，性能提升的原因有两点：

（1） 定向技术的使用优化了数据的流动过程；

（2） 数据冲突的避免减少了执行过程中的“空泡（bubble）”。

五、实验总结

在本次实验中，我完成了向量点积功能的汇编代码编写，并进行了两方面的优化：（1）定向技术，（2）数据冲突避免。我复习了 MIPSsim 模拟器的基本使用方法，并学习了模拟器所支持的指令，还加深了对执行中 MIPS 程序行为的理解。我对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！