实验 3：使用 MIPS 指令实现求两个数组的点积

于海鑫

2017211240

版本：8

更新：2020 年 4 月 16 日

1 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

2 实验平台

实验平台采用指令级和流水线操作级模拟器 `MIPSsim`。

3 实验内容和步骤

(1). 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。

向量的点积：假设有两个 n 维向量 a、b，则 a 与 b 的点积为：

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

两个向量元素使用数组进行数据存储，**要求向量的维度不得小于 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。

4 向量点积

4.1 代码

向量点积程序使用 C 程序描述是很简单的，代码如下：

```
int naive_prod(int *a, int *b, int *n) {
    int size = *n;
    int result = 0;
    for(int i = 0; i < size; i++) {
        result += a[i] * b[i];
    }
    return result;
}
```

我们要做的仅仅就是把这段代码翻译成 MIPS 汇编即可，结果如下（需要注意 MIPS 的调用约定）

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

naive_prod:
LW     $r6, 0($r6)
ADD    $r8, $r0, $r0
ADD    $r2, $r0, $r0
loop:
```

```
LW      $r9, 0($r4)
LW      $r10, 0($r5)
MUL     $r11, $r9, $r10
ADD     $r2, $r2, $r11
ADDIU   $r4, $r4, 4
ADDIU   $r5, $r5, 4
ADDIU   $r8, $r8, 1
BNE     $r8, $r6, loop
JR      $r31

.data
a:
.word 1,2,3,4,5,6,7,8,9,10,11
b:
.word 1,2,3,4,5,6,7,8,9,10,11
n:
.word 11
```

4.2 运行结果

4.2.1 未开启定向功能时

```
汇总：
  执行周期总数：178
  ID段执行了98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：66      占周期总数的百分比：37.07865%
  其中：
    load停顿：22       占所有RAW停顿的百分比：33.33333%
    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：13            占周期总数的百分比：7.303371%
    自陷停顿：0         占周期总数的百分比：0%
    停顿周期总数：79        占周期总数的百分比：44.38202%

  分支指令：
    指令条数：12            占指令总数的百分比：12.2449%
    其中：
      分支成功：12          占分支指令数的百分比：100%
      分支失败：1           占分支指令数的百分比：8.333333%

  load/store指令：
    指令条数：24            占指令总数的百分比：24.4898%
    其中：
      load：24          占load/store指令数的百分比：100%
      store：0          占load/store指令数的百分比：0%

  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：1.020408%
```

时钟周期图如下：

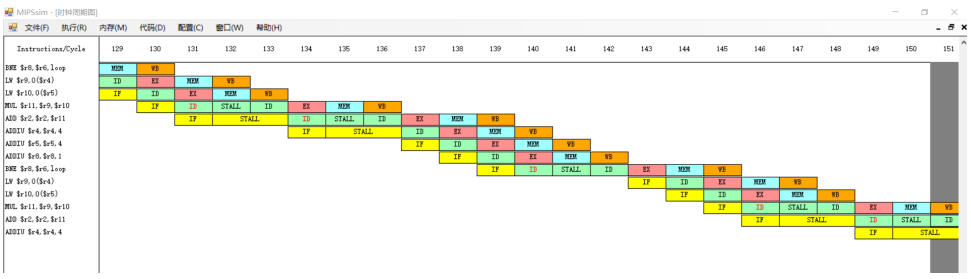

图 1: 时钟周期图

4.2.2 开启定向功能后

```
汇总：
  执行周期总数：134
  ID段执行了98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：22      占周期总数的百分比：16.41791%
  其中：
    load停顿：11       占所有RAW停顿的百分比：50%
    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：13         占周期总数的百分比：9.701492%
  自陷停顿：0      占周期总数的百分比：0%
  停顿周期总数：35     占周期总数的百分比：26.1194%

分支指令：
  指令条数：12         占指令总数的百分比：12.2449%
  其中：
    分支成功：12       占分支指令数的百分比：100%
    分支失败：1        占分支指令数的百分比：8.333333%

load/store指令：
  指令条数：24         占指令总数的百分比：24.4898%
  其中：
    load：24       占load/store指令数的百分比：100%
    store：0       占load/store指令数的百分比：0%

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：1.020408%
```

时钟周期图如下：

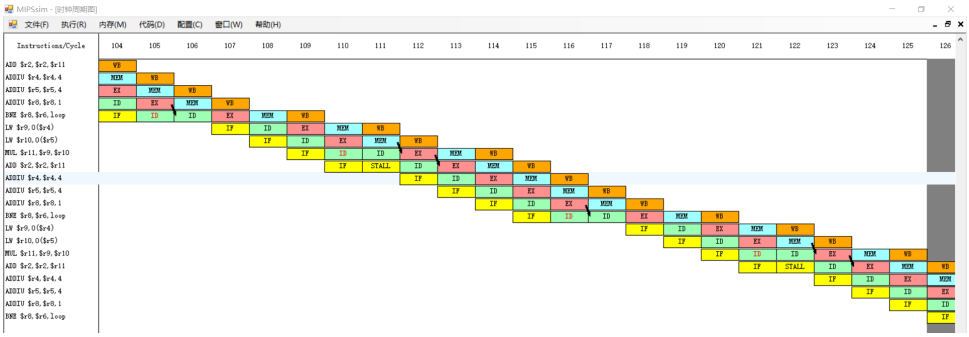

图 2: 时钟周期图

5 优化后的向量点积

5.1 代码

通过发现原始代码中的数据相关并将之规避掉，我们得到了如下的代码：

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

prod:
LW      $r6, 0($r6)
ADD     $r8, $r0, $r0
ADD     $r2, $r0, $r0
loop:
LW      $r9, 0($r4)
LW      $r10, 0($r5)
```

```
ADDIU  $r4, $r4, 4
ADDIU  $r5, $r5, 4
MUL    $r11, $r9, $r10
ADDIU  $r8, $r8, 1
ADD    $r2, $r2, $r11
BNE    $r8, $r6, loop
JR     $r31

.data
a:
.word 1,2,3,4,5,6,7,8,9,10,11
b:
.word 1,2,3,4,5,6,7,8,9,10,11
n:
.word 11
```

5.2 运行结果

```
汇总：
  执行周期总数：112
  ID段执行了98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：0       占周期总数的百分比：0%
  其中：
    load停顿：0        占所有RAW停顿的百分比：0%
    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：13         占周期总数的百分比：11.60714%
  自陷停顿：0      占周期总数的百分比：0%
```

```
    停顿周期总数：13      占周期总数的百分比：11.60714%

  分支指令：
    指令条数：12          占指令总数的百分比：12.2449%
    其中：
      分支成功：12        占分支指令数的百分比：100%
      分支失败：1         占分支指令数的百分比：8.333333%

  load/store指令：
    指令条数：24          占指令总数的百分比：24.4898%
    其中：
      load：24        占load/store指令数的百分比：100%
      store：0        占load/store指令数的百分比：0%

  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：1.020408%
```

时钟周期图如下：

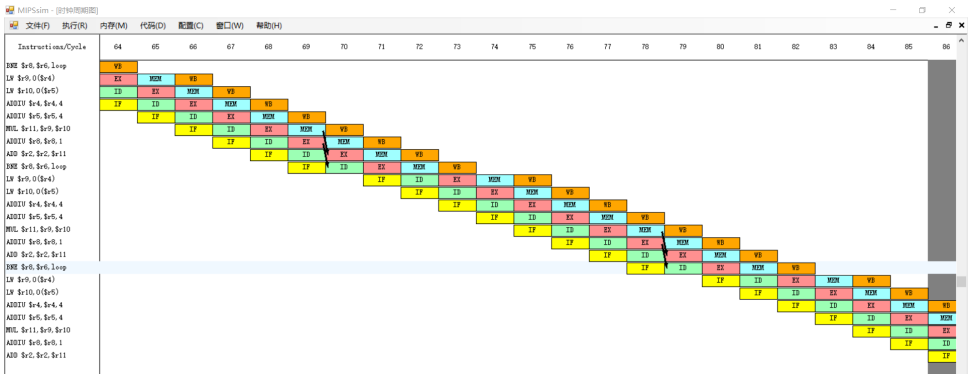

图 3: 时钟周期图

与之前的代码相比，效率大约是之前的 134/112 = 1.19 倍。

6 实验中的问题与心得

本次实验主要是人肉模拟编译器做一些简单的优化，本次实验中遇到的问题有一部分指令没有在现有的 `MIPSsim` 模拟器上实现，比如实际编译器经常产生的 `BAL` 指令，推测是模拟器实现者水平有限，不过这类问题可以通过更换别的指令替代来解决，不是很影响实现。

本次的心得大约是通过一些静态优化，我们也可以很好的进一步榨取 CPU 的性能。