北京邮电大学计算机系统结构实验报告



实验名称:		使用 MIPS 指令实现求两个数组的点积
班	级:	2016211301
学	号:	2016211134
姓	名:	李智盛
指导教师:		/
实验日期:		2019年5月22日

0. 目录

0.	目录	. 1
1.	实验目的	_ 2
2.	实验平台	_ 2
3.	实验内容和步骤	_ 2
4.	向量点积程序代码清单及注释说明	. 6
5.	优化后的程序代码清单	. 7
6.	未优化代码和优化代码性能分析比较结果	7

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

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,

(注意: 不要使用浮点指令及浮点寄存器!! 使用 TEQ \$r0 \$r0 结束程序!!)

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

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:

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

这里在程序中自定义的向量数组如下:

```
# 数据段
```

.data

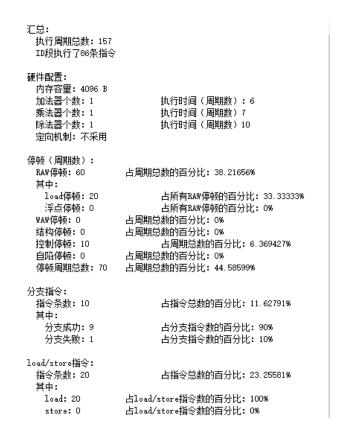
array1: .word 0,1,2,3,4,5,6,7,8,9 array2: .word 0,1,2,3,4,5,6,7,8,9

- (2) 启动 MIPSsim。
- (3) 载入自己编写的程序,观察流水线输出结果。



运行结果如下: R7(向量点积结果)=285

执行结果的统计窗口:



(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。

左边为未使用定向功能的上一次执行结果; 右边为使用了定向功能后的执行结果

汇总:

执行周期总数: 157 ID段执行了86条指令

硬件配置:

内存容**里: 4**096 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: 60 占周期总数的百分比: 38.21656%

其中:

10ad停顿: 20 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%

 WANY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: 10 占周期总数的百分比: 6.369427%

自陷停顿: 0 占周期总数的百分比: 0% 停顿周期总数: 70 占周期总数的百分比: 44.58599%

分支指令:

指令条数: 10 占指令总数的百分比: 11.62791%

其中:

分支成功: 9 占分支指令数的百分比: 90% 分支失败: 1 占分支指令数的百分比: 10%

load/store指令:

指令条数: 20 占指令总数的百分比: 23.25581%

其中:

load: 20 占load/store指令数的百分比: 100% store: 0 占load/store指令数的百分比: 0%

总的执行效率提高了 134%

RAW 的停顿周期数从 60 减少至 20

汇总:

执行周期总数: 117 ID段执行了86条指令

硬件配置:

内存容里: 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: 20 占周期总数的百分比: 17.09402%

其中:

load停顿: 10 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: 10 占周期总数的百分比: 8.547009%

自陷停顿: 0 占周期总数的百分比: 0%

停顿周期总数: 30 占周期总数的百分比: 25.64103%

分支指令:

指令条数: 10 占指令总数的百分比: 11.62791%

其中:

分支成功: 9 占分支指令数的百分比: 90% 分支失败: 1 占分支指令数的百分比: 10%

load/store指令:

指令条数: 20 占指令总数的百分比: 23.25581%

其中:

load: 20 占load/store指令数的百分比: 100% store: 0 占load/store指令数的百分比: 0%

(5) 采用静态调度方法重排指令序列,减少相关,优化程序

重排后的代码:

- # 采用静态调度优化程序
- # 重新组织代码顺序
- # 代码段

.text

main: #程序入口 main

ADDIU **\$r1**,**\$r0**,array1 # array1 的地址 ADDIU **\$r2**,**\$r0**,array2 # array2 的地址 ADDIU **\$r3**,**\$r0**,**10** # 变量控制循环出口

ADDIU \$r7,\$r0,0 # 初始化 r7

loop:

LW \$r4,0(\$r1) LW \$r5,0(\$r2)

ADDI **\$r1**,**\$r1**,**4** # 继续读取数组 array**1**

ADDI **\$r2,\$r2,4** # 继续读取数组 array2

MUL \$r6,\$r4,\$r5 # 优化:组织顺序 ADDI \$r3,\$r3,-1 # r3 控制程序出口 ADD \$r7,\$r7,\$r6 # 优化:组织顺序 BGTZ \$r3,loop # if r3>0: loop

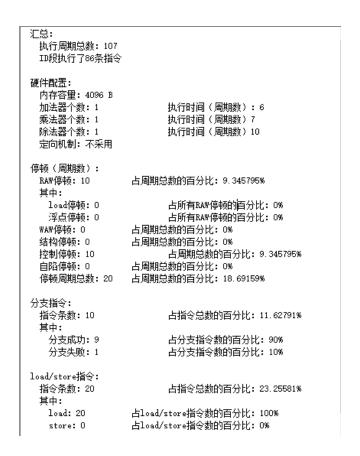
TEQ **\$r0**,**\$r0**

#数据段

.data

array1: .word 0,1,2,3,4,5,6,7,8,9 array2: .word 0,1,2,3,4,5,6,7,8,9

采用静态调度优化后的执行结果:





相比优化前(执行周期总数为157),效率有了相当大的提升

(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。 左边为优化后未使用定向功能的执行结果;右边为优化后使用定向功能的执行结果

汇总: 执行周期总数: 107 汇总: ID段执行了86条指令 ... 执行周期总数: 97 ID段执行了86条指令 硬件配置: 内存容量: 4096 B 硬件配置: 执行时间(周期数):6 加法器个数:1 内存容量: 4096 B 乘法器个数:1 执行时间(周期数)7 加法器个数:1 执行时间(周期数):6 除法器个数:1 执行时间(周期数)10 乘法器个数:1 执行时间(周期数)7 定向机制: 不采用 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): 占周期总数的百分比: 9.345795% RAW停顿: 10 停顿(周期数): 其中: RAW停顿: 0 占周期总数的百分比:0% load停顿: 0 占所有RAW停顿的百分比:0% 其中: 浮点停顿: 0 占所有RAW停顿的百分比: 0% load停顿: 0 占所有RAW停顿的百分比: 0% 占周期总数的百分比:0% WAW信顿: 0 浮点停顿: 0 占所有RAW停顿的百分比: 0% 结构停顿: 0 占周期总数的百分比:0% 占周期总数的百分比:0% ₩A₩停顿: O 控制停顿:10 占周期总数的百分比: 9.345795% 结构停顿: 0 占周期总数的百分比:0% 自陷停顿: 0 占周期总数的百分比:0% 占周期总数的百分比: 10.30928% 控制停顿: 10 占周期总数的百分比:0% 停顿周期总数:20 占周期总数的百分比: 18.69159% 白路停顿: 0 停顿周期总数:10 占周期总数的百分比: 10.30928% 分支指令: 占指令总数的百分比: 11.62791% 分支指令: 指令条数:10 指令条数:10 占指令总数的百分比: 11.62791% 其中: 其中: 分支成功:9 占分支指令数的百分比:90% 分支成功:9 占分支指令数的百分比:90% 分支失败:1 占分支指令数的百分比: 10% 分支失败:1 占分支指令数的百分比: 10% load/store指令: load/store指令: 指令条数:20 占指令总数的百分比: 23.25581% 指令条数:20 占指令总数的百分比: 23.25581% 其中: 其中: 占load/store指令数的百分比: 100% load: 20 load: 20 占load/store指令数的百分比: 100% store: 0 占load/store指令数的百分比: 0% store: 0 占load/store指令数的百分比: 0%

相比之下,采用定向功能后,执行效率提高了 110.3% RAW 停顿次数减少了 10 次

4. 向量点积程序代码清单及注释说明

```
# 代码段
.text
# 程序入口 main
main:
ADDIU $r1,$r0,array1 # array1 的地址
ADDIU $r2,$r0,array2 # array2 的地址
ADDIU $r3,$r0,10 # 变量控制循环出口
ADDIU $r7,$r0,0 # 初始化r7
loop:
LW $r4,0($r1)
LW $r5,0($r2)
MUL $r6,$r4,$r5
               # r7 存放点积结果
ADD $r7,$r7,$r6
               # 继续读取数组
ADDI $r1,$r1,4
ADDI $r2,$r2,4
```

```
ADDI $r3,$r3,-1 # r3 控制程序出口
BGTZ $r3,loop # if r3>0: Loop
TEQ $r0,$r0

# 数据段
.data
array1: .word 0,1,2,3,4,5,6,7,8,9
array2: .word 0,1,2,3,4,5,6,7,8,9
```

5. 优化后的程序代码清单

```
# 采用静态调度优化程序
# 重新组织代码顺序
# 代码段
.text
main: #程序入口 main
ADDIU $r1,$r0,array1 # array1的地址
ADDIU $r2,$r0,array2 # array2的地址
ADDIU $r3,$r0,10 # 变量控制循环出口
ADDIU $r7,$r0,0 # 初始化 r7
loop:
LW $r4,0($r1)
LW $r5,0($r2)
ADDI $r1,$r1,4 #继续读取数组 array1
ADDI $r2,$r2,4 # 继续读取数组 array2
MUL $r6,$r4,$r5 # 优化:组织顺序
ADDI $r3,$r3,-1 # r3 控制程序出口
ADD $r7,$r7,$r6 # 优化:组织顺序
BGTZ $r3,loop
             # if r3>0: loop
TEQ $r0,$r0
#数据段
.data
array1: .word 0,1,2,3,4,5,6,7,8,9
array2: .word 0,1,2,3,4,5,6,7,8,9
```

6. 未优化代码和优化代码性能分析比较结果

文字加截图说明, 并给出优化的原因

未优化代码:

```
优化后:
```

```
# 代码段
.text
main: # 程序入口 main
ADDIU $r1,$r0,array1 # array1的地址
ADDIU $r2,$r0,array2 # array2的地址
ADDIU $r3,$r0,10 # 变量控制循环出口
ADDIU $r7,$r0,0 # 初始化r7
loop:
LW $r4,0($r1)
LW $r5,0($r2)
MUL $r6,$r4,$r5
ADD $r7,$r7,$r6
                # r7存放点积结果
ADDI $r1,$r1,4
                # 继续读取数组
ADDI $r2,$r2,4
ADDI $r3,$r3,-1
                # r3 控制程序出口
BGTZ $r3,loop
                 # if r3>0: loop
TEQ $r0,$r0
# 数据段
.data
array1: .word 0,1,2,3,4,5,6,7,8,9
array2: .word 0,1,2,3,4,5,6,7,8,9
```

```
# 采用静态调度优化程序
# 重新组织代码顺序
 # 代码段
 .text
       # 程序入口 main
main:
ADDIU $r1,$r0,array1 # array1的地址
ADDIU $r2,$r0,array2 # array2的地址
ADDIU $r3,$r0,10 # 变量控制循环出口
ADDIU $r7,$r0,0 # 初始化r7
loop:
LW $r4,0($r1)
 LW $r5,0($r2)
                # 继续读取数组array1
ADDI $r1,$r1,4
                # 继续读取数组array2
ADDI $r2,$r2,4
MUL $r6,$r4,$r5
                # 优化:组织顺序
ADDI $r3,$r3,-1
                # r3 控制程序出口
 ADD $r7,$r7,$r6
                # 优化: r7存放点积结果
 BGTZ $r3,loop
                # if r3>0: loop
 TEQ $r0,$r0
# 数据段
 .data
 array1: .word 0,1,2,3,4,5,6,7,8,9
array2: .word 0,1,2,3,4,5,6,7,8,9
```

执行结果比较:

```
执行周期兑数: 157
 ID段执行了86条指令
硬件配置:
 内存容量: 4096 B
                     执行时间(周期数):6
 加法器个数:1
                    执行时间(周期数)7
执行时间(周期数)10
 乘法器个数:1
 除法器个数:1
 定向机制: 不采用
停顿(周期数):
 RAW停顿: 60
               占周期总数的百分比: 38.21656%
 其中:
load停顿: 20
                     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%
 wムW位はお・ ೧
               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%
                    占周期总数的百分比: 6.369427%
 控制停顿: 10
               占周期总数的百分比:0%
 白路停顿: 0
               占周期总数的百分比: 44.58599%
 停顿周期总数:70
分支指令:
                    占指令总数的百分比: 11.62791%
 指令条数:10
 其中:
  分支成功: 9
                     占分支指令数的百分比: 90%
  分支失败: 1
                     占分支指令数的百分比: 10%
load/store指令:
                     占指令总数的百分比: 23.25581%
 指今条数:20
 其中:
  load: 20
               占load/store指令数的百分比: 100%
  store: 0
               占load/store指令数的百分比: 0%
```

```
....
执行周期总数: 107
 ID段执行了86条指令
硬件配置:
 内存容量: 4096 B
                    执行时间(周期数):6
 加法器个数:1
 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: 10
              占周期总数的百分比: 9.345795%
 其中:
  load信頼: 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%
 ₩A₩停顿: O
               占周期总数的百分比:0%
 结构信顿: 0
              占周期总数的百分比:0%
 控制停顿: 10
                    占周期总数的百分比: 9.345795%
               占周期总数的百分比:0%
 停顿周期总数:20
              占周期总数的百分比: 18.69159%
分支指令:
 指令条数: 10
                    占指令总数的百分比: 11.62791%
 其中:
  .
分支成功: 9
                    占分支指令数的百分比: 90%
  分支失败:1
                    占分支指令数的百分比: 10%
load/store指令:
 指令条数:20
                    占指令总数的百分比: 23.25581%
 其中:
  load: 20
               占load/store指令数的百分比: 100%
  store: 0
              占load/store指令数的百分比: 0%
```

优化前: 执行周期总数 157, RAW 停顿次数 60

优化后: 执行周期总数 107, RAW 停顿次数 10

采用静态调度的方法减少了数据相关所造成的 RAW 停顿;

因为分支指令并未进行优化、控制冲突造成的控制停顿次数没有发生变化。