北京郵電大學

实验五：
指令调度与延迟分支

学院：计算机学院（国家示范性软件学院）

专业：计算机科学与技术

班级：2022211305

学号：2022211683

姓名：张晨阳

2025 年 5 月 30 号

目录

1. 实验目的

（1） 加深对指令调度技术的理解。

（2） 加深对延迟分支技术的理解。

（3） 熟练账务用指令调度技术解决流水线中的数据冲突的方法。

（4） 进一步理解指令调度技术对 CPU 性能的改进。

（5） 进一步理解延迟分支技术对 CPU 性能的改进。

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. 实验内容及分析

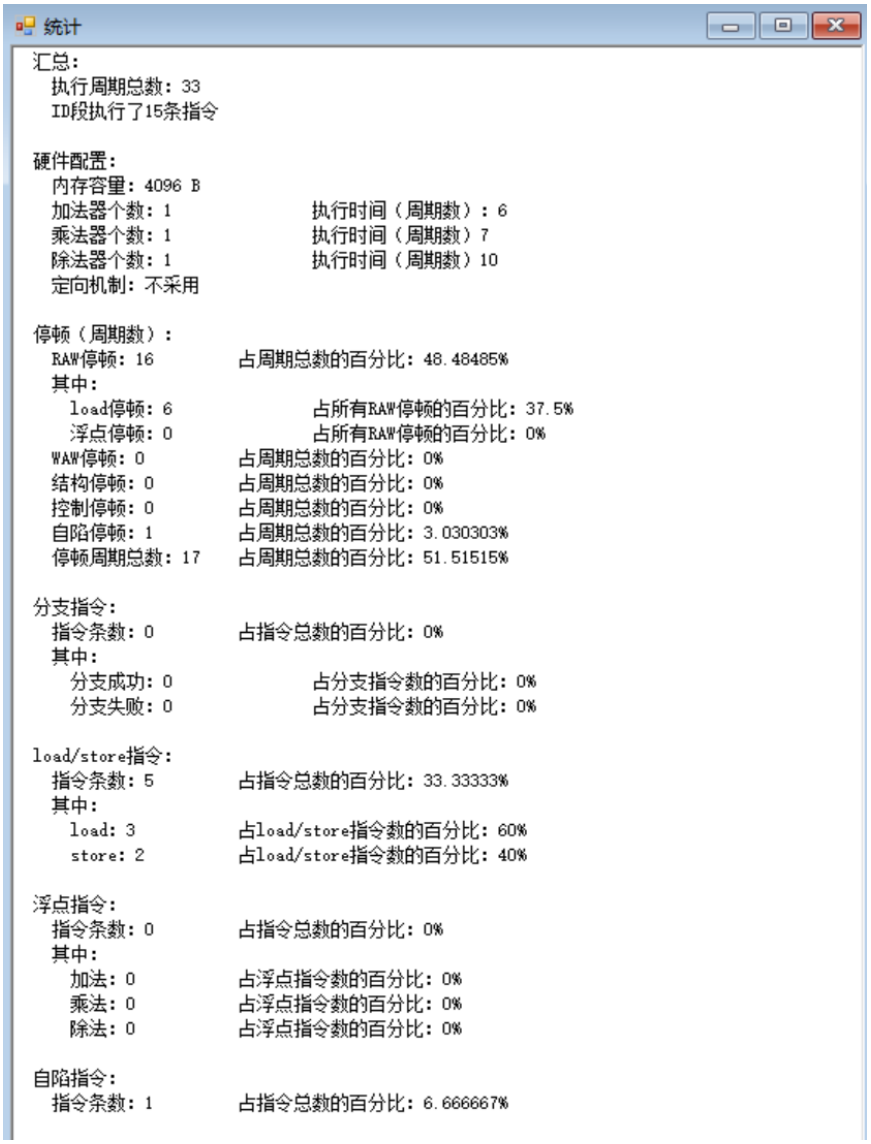
3.1. 用指令调度技术解决流水线中的结构冲突与数据冲突

1）启动 MIPSsim。

2）用 MIPSsim 的“文件”->“载入程序”选项来加载 schedule.s（在模拟器所在文件夹下的“样例程序”文件夹中）。

3）关闭定向功能，这是通过“配置“->”定向“选项来实现的。

4）执行所载入的程序，通过查看统计数据和时钟周期图，找出并记录程序执行过程中各种冲突发生的次数，发生冲突的指令组合以及程序执行的总时钟周期数。



RAW 停顿：16，自陷停顿：1，执行周期总数：33。

发生 RAW 冲突的指令组合：

ADDIU $r1, $r0, A	LW $r2, 0($r1)
LW $r2, 0($r1)	ADD $r4, $r0, $r2
ADD $r4, $r0, $r2	SW $r4, 0($r1)
LW $r6, 4($r1)	ADD $r8, $r6, $r1
MUL $r12, $r10, $r1	ADD $r16, $r12, $r1
ADD $r16, $r12, $r1	ADD $r18, $r16, $r1
ADD $r18, $r16, $r1	SW $r18, 16($r1)
LW $r20, 8($r1)	MUL $r22, $r20, $r14

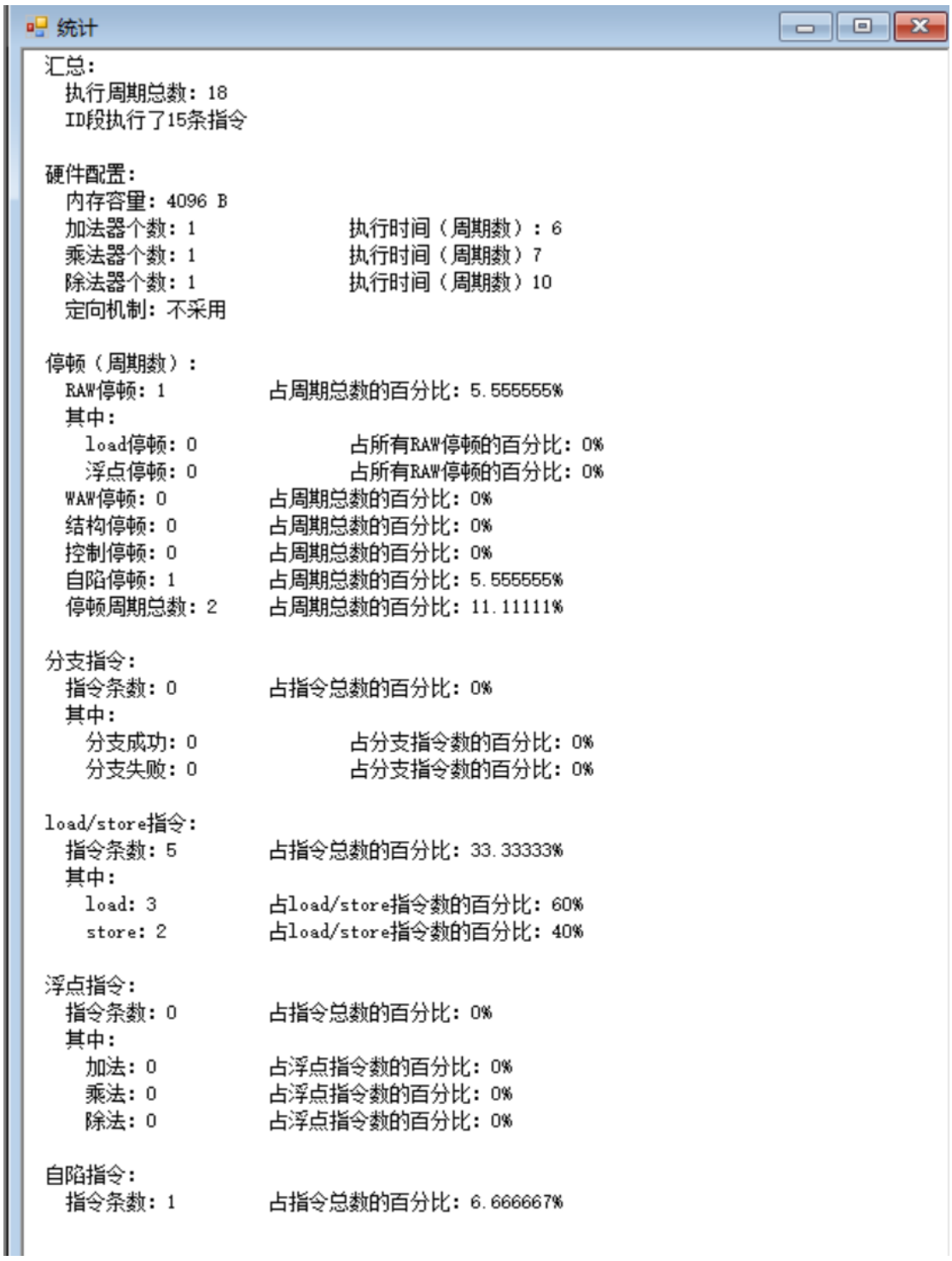
5）自己采用调度技术对程序进行指令调度，消除冲突（自己修改源程序）。将调度（修改）后的程序重新命名为 after-schedule.s。（注意：调度方法灵活多样，在保证程序正确性的前提下自己随意调度，尽量减少冲突即可，不要求要达到最优。）

修改后的代码如下：

```
.text
main:
ADDIU $r1,$r0,A
MUL $r24,$r26,$r14
LW $r2,0($r1)
MUL $r12,$r10,$r1
LW $r6,4($r1)
LW $r20,8($r1)
ADD $r16,$r12,$r1
ADD $r4,$r0,$r2
ADD $r8,$r6,$r1
ADD $r18,$r16,$r1
SW $r4,0($r1)
MUL $r22,$r20,$r14
SW $r18,16($r1)
TEQ $r0,$r0


.data
A:
.word 4,6,8
```

6）载入 after-schedule.s，执行该程序，观察程序在流水线中的执行情况，记录程序执行的总时钟周期数。



执行周期总数：18。

停顿周期占比也下降到 11.11%

7）比较调度前和调度后的性能，论述指令调度对提高 CPU 性能的作用。

调度后的性能提升了 33/18=1.83 倍。

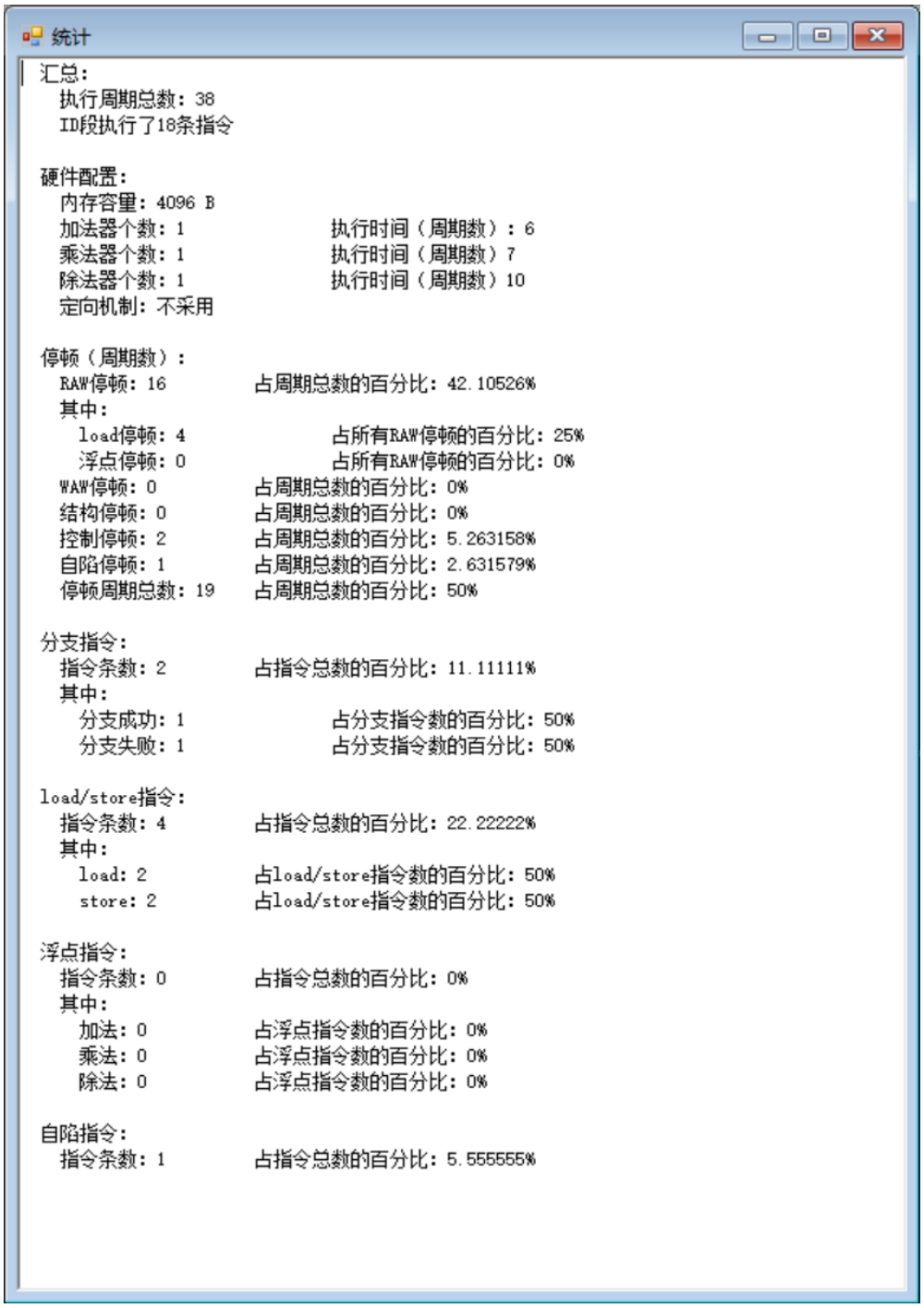
说明指令调度可以消除部分的数据冲突，通过使用指令调度提高了 CPU 的使用率，大大减少了指令冲突的次数，提高了 CPU 性能。

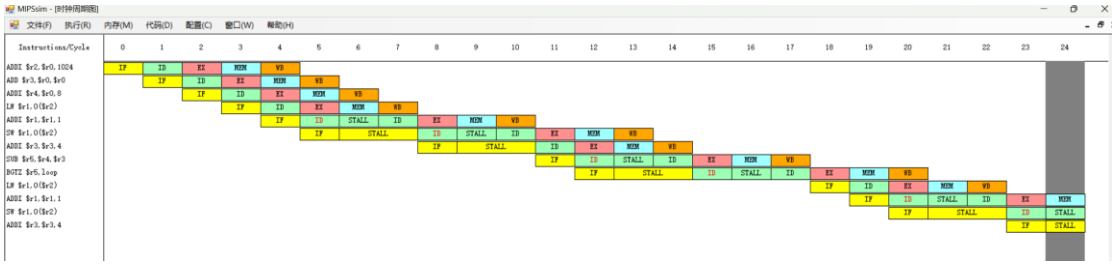
3.2. 用延迟分支技术减少分支指令对性能的影响

1）在 MIPSsim 中载入 branch.s 样例程序（在本模拟器目录的“样例程序”文件夹中。

2）关闭延迟分支功能。这是通过在“配置”->“延迟槽”选项来实现的。

3）执行该程序，观察并记录发生分支延迟的时刻，记录该程序执行的总时钟周期数。





分支指令是 BGTZ $r5,loop 。当这条指令执行时，下一条指令 ADD $r7,$r0,$r6 可能已经在流水线中了。所以，即使 $r5 的值大于 0，CPU 也需要执行 ADD $r7,$r0,$r6 这条指令，然后才能跳转到 loop 。

这就是发生分支延迟的时刻。

发生分支延迟的时刻：15，执行周期总数：38.

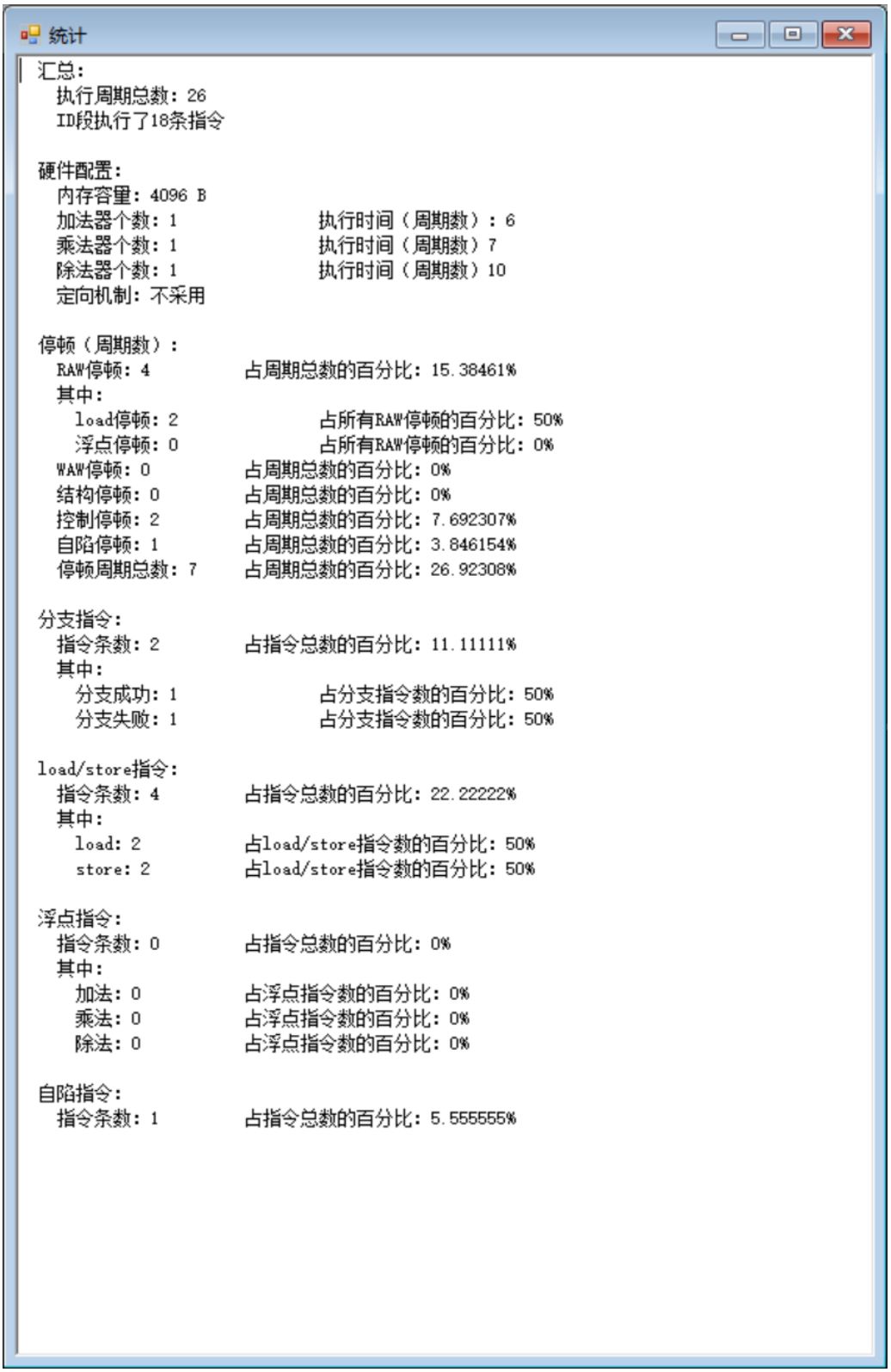
4）假设延迟槽为一个，自己对 branch.s 程序进行指令调度（自己修改源程序），将调度后的程序重新命名为 delayed-branch.s。

修改后的程序如下：

```
.text
main:
ADDI  $r2,$r0,1024
ADD   $r3,$r0,$r0
ADDI  $r4,$r0,8
loop:
LW    $r1,0($r2)
ADDI  $r3,$r3,4
ADDI  $r1,$r1,1
SUB   $r5,$r4,$r3
SW    $r1,0($r2)
BGTZ  $r5,loop

ADD   $r7,$r0,$r6
TEQ   $r0,$r0
```

5）载入 delayed-branch.s，打开延迟分支功能，执行该程序，观察其时钟周期图，记录程序执行的总时钟周期数。



执行周期总数：26。

4. 实验结论

从给出的两次运行结果来看，我们可以看到以下几点差异：

1. 执行周期总数：第一次运行的执行周期总数为 38，而第二次运行的执行周期总数为 26。

2. 停顿（周期数）：第一次运行的停顿周期总数为 19，占周期总数的百分比为 50%，而第二次运行的停顿周期总数为 7，占周期总数的百分比为 26%。这表明第二次运行的程序在执行过程中的停顿时间更少，性能更高。

3. 分支指令、load/store 指令、浮点指令和自陷指令的执行情况在两次运行中基本相同。

综上所述，延迟分支技术是由编译器通过重排指令序列，在分支指令后紧跟一条或几条延迟槽指令，不管分支是否成功，都顺序执行延迟槽中的指令，从而逻辑上“延长”分支指令的执行时间，减少甚至消除了控制停顿，从而提高 CPU 性能。

通过本次实验，我对 MIPS 汇编语言的编程、流水线模拟以及程序优化有了更深刻的理解。整个实验过程不仅帮助我掌握了指令调度与延迟分支，还使我认识到程序执行效率和优化的重要性。

总之，本次实验不仅提升了我的汇编编程能力，还让我认识到优化对程序性能的影响。在今后的学习中，我将继续深入研究程序优化，力求在设计和开发过程中最大化地提高程序的运行效率。