

计算机系统结构实验报告

学号： 20020203
姓名： 王思哲
专业： 计算机科学与技术

教师签字：

日期： 2022.12.10

目录

| | | |
|------------|--|-----------|
| 实验一 | 流水线中的相关..... | 3 |
| 一、 | 实验目的..... | 3 |
| 二、 | 实验内容与结果分析..... | 3 |
| | 1、以单步的方式执行 sum.s 和 test_for.s 两个程序，观察流水线执行情况以及寄存器、存储器的内容。..... | 3 |
| | 2、执行 test_for.s 程序，找出存在资源相关和导致资源相关的部件；记录资源相关引起的暂停时钟周期数，计算占总周期的百分比；论述资源相关对 CPU 性能的影响；讨论资源相关解决方法；..... | 5 |
| | 3、计算定向计数后性能提高的倍数..... | 6 |
| 实验二 | 循环展开与指令调度..... | 7 |
| 一、 | 实验目的..... | 7 |
| 二、 | 实验内容及结果分析..... | 7 |
| | 1、用 DLX 汇编语言编写代码文件，记录数据..... | 7 |
| | 2、采用指令调度技术解决流水线中的结构相关与数据相关..... | 8 |
| | 3、采用循环展开、寄存器换名以及指令调度提高性能..... | 10 |
| 实验三 | Cache 性能分析..... | 12 |
| 一、 | 实验目的..... | 12 |
| 二、 | 实验内容及结果分析..... | 12 |
| | 1、选用的测试程序及基本配置下的运行情况..... | 12 |
| | 2、改变 Cache 容量..... | 12 |
| | 3、改变 Cache 相联度..... | 14 |
| | 4、改变 Cache 块大小..... | 15 |
| | 5、采用 LRU 与随机法，在不同 Cache 容量和相联度下分别测试..... | 17 |

实验一 流水线中的相关

一、 实验目的

- 1、熟练掌握 WinMIPS64 模拟器的操作和使用，熟悉 MIPS 指令集结构及其特点；
- 2、加深对计算机流水线基本概念的理解；
- 3、进一步了解 MIPS 基本流水线各段的功能以及基本操作；
- 4、加深对数据相关、结构相关的理解，了解这两类相关对 CPU 性能的影响；
- 5、了解解决数据相关的方法，掌握如何使用定向技术来减少数据相关带来的暂停。

二、 实验内容与结果分析

- 1、以单步的方式执行 sum.s 和 test_for.s 两个程序，观察流水线执行情况以及寄存器、存储器的内容。

(1) sum.s

寄存器内容

| Registers | | | |
|-----------|------------------|-----|------------------|
| R0= | 0000000000000000 | F0= | 0000000.00000000 |
| R1= | 0000000000000000 | F1= | 0000000.00000000 |
| R2= | 0000000000000000 | F2= | 0000000.00000000 |
| R3= | 0000000000000012 | F3= | 0000000.00000000 |
| R4= | 000000000000000a | F4= | 0000000.00000000 |
| R5= | 0000000000000008 | F5= | 0000000.00000000 |
| R6= | 0000000000000000 | F6= | 0000000.00000000 |
| R7= | 0000000000000000 | F7= | 0000000.00000000 |
| R8= | 0000000000000000 | F8= | 0000000.00000000 |
| R9= | 0000000000000000 | F9= | 0000000.00000000 |

存储器内容

| Data | | | |
|------|------------------|-------------|--|
| 0000 | 000000000000000a | A: .word 10 | |
| 0008 | 0000000000000008 | B: .word 8 | |
| 0010 | 0000000000000012 | C: .word 0 | |
| 0018 | 0000000000000000 | | |
| 0020 | 0000000000000000 | | |
| 0028 | 0000000000000000 | | |
| 0030 | 0000000000000000 | | |
| 0038 | 0000000000000000 | | |
| 0040 | 0000000000000000 | | |
| 0048 | 0000000000000000 | | |
| 0050 | 0000000000000000 | | |
| 0058 | 0000000000000000 | | |

统计内容

| Statistics | |
|------------------------------------|--|
| Execution | |
| 13 Cycles | |
| 5 Instructions | |
| 2.600 Cycles Per Instruction (CPI) | |
| Stalls | |
| 4 RAW Stalls | |
| 0 WAW Stalls | |
| 0 WAR Stalls | |
| 0 Structural Stalls | |
| 0 Branch Taken Stalls | |
| 0 Branch Misprediction Stalls | |
| Code size | |
| 20 Bytes | |

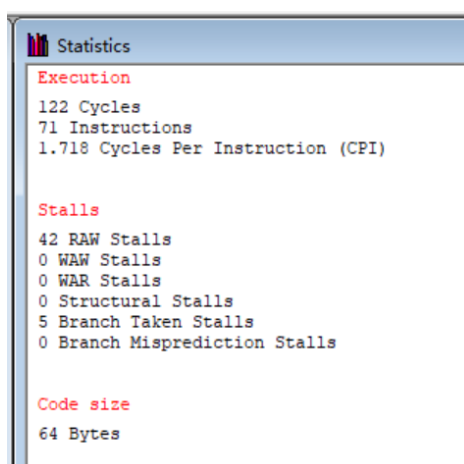
(2) test_for.s 寄存器

| Registers | | | |
|-----------|------------------|------|------------------|
| R0= | 0000000000000000 | F0= | 0000000.00000000 |
| R1= | 0000000000000030 | F1= | 0000000.00000000 |
| R2= | 0000000000000060 | F2= | 0000000.00000000 |
| R3= | 0000000000000090 | F3= | 0000000.00000000 |
| R4= | 0000000000000000 | F4= | 0000000.00000000 |
| R5= | 0000000000000007 | F5= | 0000000.00000000 |
| R6= | 0000000000000001 | F6= | 0000000.00000000 |
| R7= | 0000000000000006 | F7= | 0000000.00000000 |
| R8= | 0000000000000008 | F8= | 0000000.00000000 |
| R9= | 000000000000000e | F9= | 0000000.00000000 |
| R10= | 0000000000000000 | F10= | 0000000.00000000 |
| R11= | 0000000000000000 | F11= | 0000000.00000000 |
| R12= | 0000000000000000 | F12= | 0000000.00000000 |
| R13= | 0000000000000000 | F13= | 0000000.00000000 |
| R14= | 0000000000000000 | F14= | 0000000.00000000 |
| R15= | 0000000000000000 | F15= | 0000000.00000000 |
| R16= | 0000000000000000 | F16= | 0000000.00000000 |
| R17= | 0000000000000000 | F17= | 0000000.00000000 |
| R18= | 0000000000000000 | F18= | 0000000.00000000 |
| R19= | 0000000000000000 | F19= | 0000000.00000000 |
| R20= | 0000000000000000 | F20= | 0000000.00000000 |
| R21= | 0000000000000000 | F21= | 0000000.00000000 |
| R22= | 0000000000000000 | F22= | 0000000.00000000 |
| R23= | 0000000000000000 | F23= | 0000000.00000000 |
| R24= | 0000000000000000 | F24= | 0000000.00000000 |
| R25= | 0000000000000000 | F25= | 0000000.00000000 |
| R26= | 0000000000000000 | F26= | 0000000.00000000 |
| R27= | 0000000000000000 | F27= | 0000000.00000000 |
| R28= | 0000000000000000 | F28= | 0000000.00000000 |
| R29= | 0000000000000000 | F29= | 0000000.00000000 |
| R30= | 0000000000000000 | F30= | 0000000.00000000 |
| R31= | 0000000000000000 | F31= | 0000000.00000000 |

存储器

| Data | |
|------|---|
| 0000 | 0000000000000016 a: .space 48 |
| 0008 | 000000000000001a |
| 0010 | 000000000000001e |
| 0018 | 0000000000000022 |
| 0020 | 000000000000000a |
| 0028 | 000000000000000e |
| 0030 | 000000000000000a b: .word 10,11,12,13,0,1 |
| 0038 | 000000000000000b |
| 0040 | 000000000000000c |
| 0048 | 000000000000000d |
| 0050 | 0000000000000000 |
| 0058 | 0000000000000001 |
| 0060 | 0000000000000001 c: .word 1,2,3,4,5,6 |
| 0068 | 0000000000000002 |
| 0070 | 0000000000000003 |
| 0078 | 0000000000000004 |
| 0080 | 0000000000000005 |
| 0088 | 0000000000000006 |
| 0090 | 0000000000000000 |
| 0098 | 0000000000000000 |
| 00a0 | 0000000000000000 |
| 00a8 | 0000000000000000 |
| 00b0 | 0000000000000000 |

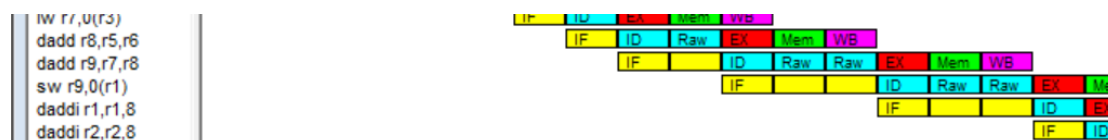
统计



2、执行 test_for.s 程序，找出存在资源相关和导致资源相关的部件；记录资源相关引起的暂停时钟周期数，计算占总周期的百分比；论述资源相关对 CPU 性能的影响；讨论资源相关解决方法；

(1) 找出资源相关和导致资源相关的部件

下图中 dadd r8,r5,r6 和 dadd r9,r7,r8 存在先写后读的数据相关，这会导致第二条指令在 id 阶段等待，直到第一条指令完成了写回阶段才会继续。而第二条指令在 id 阶段等待会占用 ID 资源，这就会引起与下一条指令的资源相关，他们都想要使用译码部件。同理，dadd r9,r7,r8 和 sw r9,0(r1) 也存在数据相关，同样会引起接下来的资源相关。



(2) 资源相关引起的暂停时钟周期数及百分比

| test_for.s | | |
|------------|--------------|-------------|
| 总时钟数 | 资源相关引起的暂停时钟数 | 占的百分比 |
| 122 | 42 | 0.344262295 |

(3) 资源相关对 cpu 性能的影响，以及解决方法

Cpu 性能可以用 cpi 来描述，cpi 是通过指令总周期数除以指令数来计算的。由于资源相关，会导致指令总周期数增大，而指令总数不变，这会导致 cpi 增大，

性能下降。

解决资源相关的方法有两种：一种是可以使指令停顿 n 拍后再进入流水线，从而错开等待周期；另一种是可以重复设置功能部件，避免资源相关的发生。

3、计算定向计数后性能提高的倍数

(1) sum.s

| sum.s | | | | | |
|---------|------|--------------|-------------|-----|--------|
| | 总时钟数 | 数据相关引起的暂停时钟数 | 占的百分比 | cpi | 性能提升倍数 |
| 不采用定向技术 | 13 | 4 | 0.307692308 | 2.6 | 1.3 |
| 采用定向技术 | 10 | 1 | 0.1 | 2 | |

(2) test_for.s

| test_for.s | | | | | |
|------------|------|--------------|-------------|-------|-------------|
| | 总时钟数 | 数据相关引起的暂停时钟数 | 占的百分比 | cpi | 性能提升倍数 |
| 不采用定向技术 | 122 | 42 | 0.344262295 | 1.718 | 1.418662263 |
| 采用定向技术 | 86 | 6 | 0.069767442 | 1.211 | |

实验二 循环展开与指令调度

一、 实验目的

- 1、加深对循环级并行性、指令调度技术、循环展开技术以及寄存器换名技术的理解；
- 2、熟悉用指令调度技术来解决流水线中的数据相关的方法；
- 3、了解循环展开、指令调度等技术对 CPU 性能的改进。

二、 实验内容及结果分析

1、用 DLX 汇编语言编写代码文件，记录数据

(1) 代码功能

实现的是如下功能的 C 语言程序：

```
01. for(int i = 0; i < 1000; i++){
02.     c[i] = a[i] * b;
03. }
```

(2) 代码实现

```
1 .data ;数据段
2 a: .word 1,2,3,4,5,6,7,8
3 b: .word 2
4 c: .space 64
5
6 ; for(int i = 0; i < 8; i++) c[i] = a[i] * b;
7
8 .text ;代码段
9 daddi r1,r0,a ; r1 --- &a[i]
10 daddi r2,r0,c ; r2 --- &c[i]
11 daddi r3,r0,8 ; cnt
12 l.d f1,b(r0) ; f1 --- 常数
13 cvt.d.l f1,f1 ; 转浮点
14
15 Loop:
16 l.d f5,0(r1) ; 读浮点
17 l.d f6,0(r2)
18 mul.d f6,f5,f1 ; 浮点乘
19 s.d f6,0(r2) ; 存浮点
20 daddi r1,r1,8 ; 指针后移
21 daddi r2,r2,8
22 daddi r3,r3,-1 ; 更新计数器
23 bnez r3,Loop ; 不为0就跳转
24
25 halt
```

(3) 数据记录

| exp2-1 | | | | | | |
|--------|------|-------|-------|------|-------|--------|
| | 总时钟数 | 数据相关数 | 结构相关数 | 指令总数 | cpi | 性能提升倍数 |
| 原程序 | 139 | 58 | 58 | 70 | 1.986 | - |

(4) 发生冲突的指令组合

- a.
l.d f1, b(r0)
cvt.d.l f1, f1 数据相关

l.d f5, 0(r1) 导致资源相关 (ID)

b.

l.d f5, 0(r1)

mul.d f6, f5, f1 数据相关

s.d f6, 0(r2) 数据相关

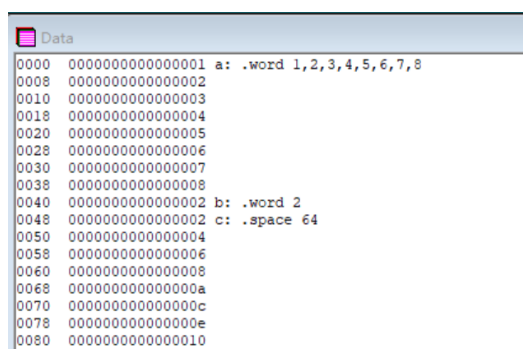
daddi r1, r1, 8 导致的资源相关 (ID)

c.

daddi r3, r3, -1

bnez r3, Loop 数据相关

(5) 存储器内容



| | | |
|------|------------------|--------------------------|
| 0000 | 0000000000000001 | a: .word 1,2,3,4,5,6,7,8 |
| 0008 | 0000000000000002 | |
| 0010 | 0000000000000003 | |
| 0018 | 0000000000000004 | |
| 0020 | 0000000000000005 | |
| 0028 | 0000000000000006 | |
| 0030 | 0000000000000007 | |
| 0038 | 0000000000000008 | |
| 0040 | 0000000000000002 | b: .word 2 |
| 0048 | 0000000000000000 | c: .space 64 |
| 0050 | 0000000000000004 | |
| 0058 | 0000000000000006 | |
| 0060 | 0000000000000008 | |
| 0068 | 000000000000000a | |
| 0070 | 000000000000000c | |
| 0078 | 000000000000000e | |
| 0080 | 0000000000000010 | |

2、采用指令调度技术解决流水线中的结构相关与数据相关

(1) 分析及实现思路

根据上文列出的发生冲突的指令组合，进行消解。将 l.d 指令和 cvt 指令使用 daddi 指令隔开，消除数据相关。通过 daddi 指令将 l.d 指令和 mul.d 指令以及 s.d 指令隔开，消除数据相关。通过改变 daddi 和 bnez 的位置，消除 r3 的数据相关。同时也会消除由于数据相关导致的资源相关问题。

(2) 代码实现

```

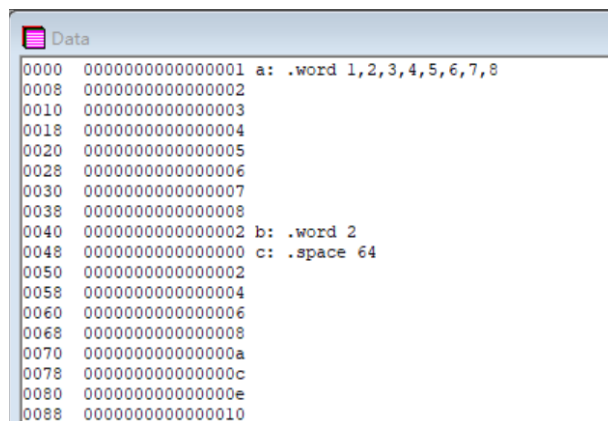
1  .data                ;数据段
2  a: .word 1,2,3,4,5,6,7,8
3  b: .word 2
4  c: .space 64
5
6  ; for(int i = 0; i < 8; i++)  c[i] = a[i] * b;
7
8  .text                ;代码段
9  l.d f1,b(r0)         ; f1 --- 常数 解决l.d cvt.d.l数据相关
10 daddi r1,r0,a         ; r1 --- &a[i]
11 daddi r2,r0,c         ; r2 --- &c[i]
12 daddi r3,r0,8         ; cnt
13
14 cvt.d.l f1,f1
15
16 Loop:
17   l.d f5,0(r1)
18   l.d f6,0(r2)
19   daddi r1,r1,8        ; 解决l.d mul.d数据相关
20   mul.d f6,f5,f1
21   daddi r2,r2,8        ; 解决mul.d s.d数据相关
22   daddi r3,r3,-1
23   s.d f6,0(r2)
24   bnez r3,Loop
25
26 halt

```

(3) 数据记录

| exp2-1-before.s & exp2-1-after.s | | | | | | |
|----------------------------------|------|-------|-------|------|-------|-------------|
| | 总时钟数 | 数据相关数 | 结构相关数 | 指令总数 | cpi | 性能提升倍数 |
| 原程序 | 139 | 58 | 58 | 70 | 1.986 | 1.432900433 |
| 指令调度 | 97 | 8 | 8 | 70 | 1.386 | |

(4) 存储器内容



```

Data
0000 0000000000000001 a: .word 1,2,3,4,5,6,7,8
0008 0000000000000002
0010 0000000000000003
0018 0000000000000004
0020 0000000000000005
0028 0000000000000006
0030 0000000000000007
0038 0000000000000008
0040 0000000000000002 b: .word 2
0048 0000000000000000 c: .space 64
0050 0000000000000002
0058 0000000000000004
0060 0000000000000006
0068 0000000000000008
0070 000000000000000a
0078 000000000000000c
0080 000000000000000e
0088 0000000000000010

```

(5) 对比指令调度前后的性能，及对于提高 CPU 性能的意义

通过指令调度，CPU 性能提升。指令调度可以消除部分的数据相关、结构相关，减少 CPU 空转的次数，从而达到 CPU 性能的提升。

3、采用循环展开、寄存器换名以及指令调度提高性能

(1) 分析及实现思路

通过循环展开，减少重复使用寄存器的方法，使得每一个循环中，指令并行性提高。再在新的指令结构上进行指令调度，进一步提高性能。数据相关的消除在代码中已经标注。

(2) 代码实现

```

1  .data          ;数据段
2  a: .word 1,2,3,4,5,6,7,8
3  b: .word 2
4  c: .space 64
5
6  ; for(int i = 0; i < 8; i ++ )  c[i] = a[i] * b;
7
8  .text          ;代码段
9  l.d f1,b(r0)   ; f1 --- 常数 解决l.d cvt.d.l数据相关
10 daddi r1,r0,a   ; r1 --- &a[i] a数组指针
11 daddi r2,r0,c   ; r2 --- &c[i] b数组指针
12 daddi r3,r0,8   ; cnt 记数
13 cvt.d.l f1,f1   ; 转浮点
14
15 Loop:
16     l.d f5,0(r1) ; 读取浮点,填补空转
17     l.d f6,0(r2)
18     l.d f7,8(r1)
19     l.d f8,8(r2)
20     l.d f9,16(r1)
21     l.d f10,16(r2)
22     l.d f11,24(r1)
23     l.d f12,24(r2)
24
25     mul.d f6,f5,f1 ; 浮点乘,填补空转
26     mul.d f8,f7,f1
27     mul.d f10,f9,f1
28     mul.d f12,f11,f1
29
30     daddi r3,r3,-4 ; 填补空转,同时还解决了r3数据相关
31
32     s.d f6,0(r2)   ; 存浮点
33     s.d f8,8(r2)
34     s.d f10,16(r2)
35     s.d f12,24(r2)
36
37     daddi r1,r1,32 ; 用两个daddi填补空转
38     daddi r2,r2,32
39     bnez r3,Loop
40
41 halt

```

(3) 数据记录

| exp2-1-before.s & exp2-2.s | | | | | | |
|----------------------------|------|-------|-------|------|-------|-------------|
| | 总时钟数 | 数据相关数 | 结构相关数 | 指令总数 | cpi | 性能提升倍数 |
| 原程序 | 139 | 58 | 58 | 70 | 1.986 | 1.660535117 |
| 循环展开+指令调度 | 55 | 0 | 4 | 46 | 1.196 | |

(4) 存储器内容

```

Data
0000 0000000000000001 a: .word 1,2,3,4,5,6,7,8
0008 0000000000000002
0010 0000000000000003
0018 0000000000000004
0020 0000000000000005
0028 0000000000000006
0030 0000000000000007
0038 0000000000000008
0040 0000000000000002 b: .word 2
0048 0000000000000002 c: .space 64
0050 0000000000000004
0058 0000000000000006
0060 0000000000000008
0068 000000000000000a
0070 000000000000000c
0078 000000000000000e
0080 0000000000000010

```

（5） 比较循环展开、指令调度前后的性能

由于具有循环的操作，因此会出现大量的无关指令，使得空转增加。同时循环也会导致空转的增加。因此仅仅使用指令调度，CPU 的执行效率并不高。

对于此，使用循环展开加指令调度的方式，减少无关命令、循环引起的空转次数的产生。这样一来，指令级的并行性提高，CPU 性能、执行效率进一步提高。

实验三 Cache 性能分析

一、实验目的

- 1、加深对 Cache 的基本概念、基本组织结构以及基本工作原理的理解；
- 2、了解 Cache 的容量、相联度、块大小对 Cache 性能的影响；
- 3、掌握降低 Cache 失效率的各种方法，以及这些方法对 Cache 性能提高的好处；
- 4、理解 Cache 失效的产生原因以及 Cache 的三种失效；
- 5、理解 LRU 与随机法的基本思想，及它们对 Cache 性能的影响。

二、实验内容及结果分析

1、选用的测试程序及基本配置下的运行情况

选用的 4 个程序分别是：test-math、test-fmath、vortex.ss、mcf00.02unroll.gcc.100M.ss

| 基本配置 | | | | | |
|----------------------------|-------|-------|-------------|---------|--------|
| 程序 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| test-math | 57466 | 804 | 548 | 256 | 0.014 |
| test-fmath | 16639 | 578 | 322 | 256 | 0.0347 |
| vortex.ss | 17010 | 746 | 490 | 256 | 0.0439 |
| mcf00.02unroll.gcc.100M.ss | 3801 | 451 | 195 | 256 | 0.1187 |

2、改变 Cache 容量

(1) 思路

改变 Cache 容量，即保证在 Cache 块大小、组内块数、替换算法不变的情况下，通过增加组数，来达到改变的效果。

本次实验中，固定块大小为 32bytes，组内块数为 1，替换算法为 LRU，并以组数 16 为基准参照组进行实验。

(2) 实验结果记录

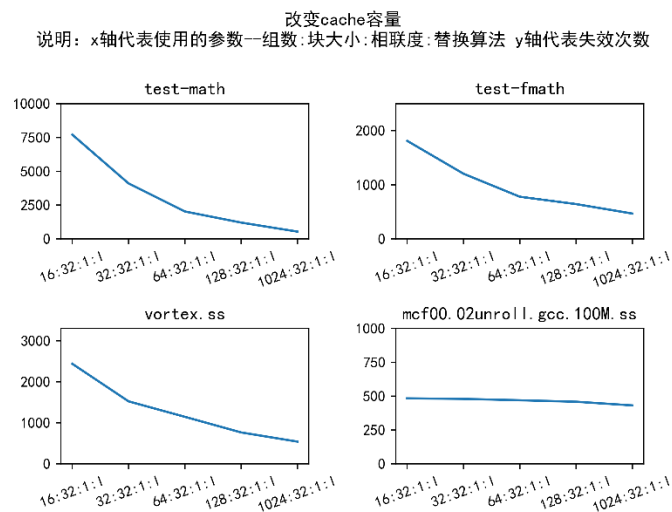
| test-math 改变cache容量 | | | | | |
|---------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 16:32:1:1 | 57466 | 7708 | 7692 | 16 | 0.1341 |
| 32:32:1:1 | 57466 | 4102 | 4070 | 32 | 0.0714 |
| 64:32:1:1 | 57466 | 2030 | 1966 | 64 | 0.0353 |
| 128:32:1:1 | 57466 | 1209 | 1081 | 128 | 0.021 |
| 1024:32:1:1 | 57466 | 542 | 13 | 529 | 0.0094 |

| test-fmath 改变cache容量 | | | | | |
|----------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 16:32:1:1 | 16639 | 1812 | 1796 | 16 | 0.1089 |
| 32:32:1:1 | 16639 | 1205 | 1173 | 32 | 0.0724 |
| 64:32:1:1 | 16639 | 780 | 716 | 64 | 0.0469 |
| 128:32:1:1 | 16639 | 644 | 516 | 128 | 0.0387 |
| 1024:32:1:1 | 16639 | 469 | 1 | 468 | 0.0282 |

| vortex.ss 改变cache容量 | | | | | |
|---------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 16:32:1:1 | 17010 | 2440 | 2424 | 16 | 0.1434 |
| 32:32:1:1 | 17010 | 1524 | 1492 | 32 | 0.0896 |
| 64:32:1:1 | 17010 | 1147 | 1083 | 64 | 0.0674 |
| 128:32:1:1 | 17010 | 766 | 638 | 128 | 0.045 |
| 1024:32:1:1 | 17010 | 542 | 102 | 440 | 0.0319 |

| mcf00.02unroll.gcc.100M.ss 改变cache容量 | | | | | |
|--------------------------------------|------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 16:32:1:1 | 3801 | 484 | 468 | 16 | 0.1273 |
| 32:32:1:1 | 3801 | 480 | 448 | 32 | 0.1263 |
| 64:32:1:1 | 3801 | 470 | 406 | 64 | 0.1237 |
| 128:32:1:1 | 3801 | 459 | 331 | 128 | 0.1208 |
| 1024:32:1:1 | 3801 | 432 | 19 | 413 | 0.1137 |

(3) 拐点寻找



| 拐点出现位置 | | | | |
|--------|-----------|------------|-----------|----------------------------|
| 程序 | test-math | test-fmath | vortex.ss | mcd00.02unroll.gcc.100M.ss |
| 拐点 | 组数64的左右 | 组数64的左右 | 组数32的左右 | 无 |

(4) 分析 Cache 容量对 Cache 性能的影响

随着 Cache 容量增大, Cache 总失效次数降低, 失效率降低。同时, 容量失效和冲突失效随着 Cache 容量的增加不断减少, 而强制性失效次数则不断的增加。

从总失效次数的降低趋势可以看出, 前三个程序对于容量的变化较为敏感, 即运行可能需要的空间更多, 而最后一个程序对容量变化不敏感, 说明空间已经够用。

3、改变 Cache 相联度

(1) 思路

改变 Cache 相联度，需要保证 Cache 的容量不变。因此需要在改变相联度的同时，改变组数，并保证块大小固定、替换算法固定。

本次实验中，以 1024 组，块大小 16bytes，组内 1 块，LRU 替换算法为基准。

(2) 实验结果记录

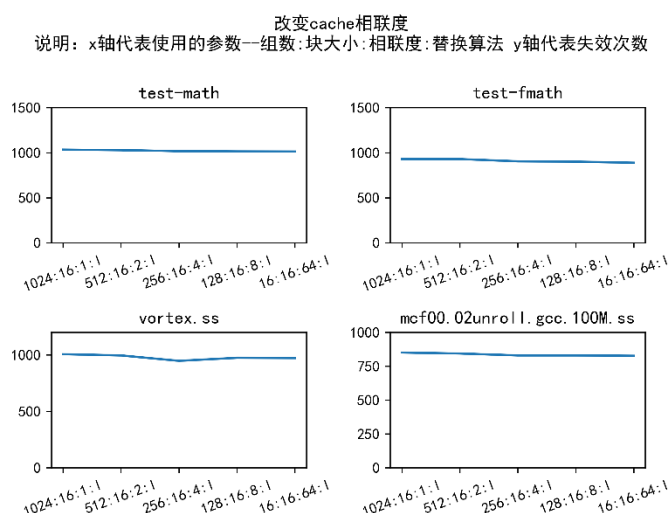
| test-math 改变cache相联度 | | | | | |
|----------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:16:1:1 | 57466 | 1034 | 204 | 830 | 0.018 |
| 512:16:2:1 | 57466 | 1029 | 180 | 849 | 0.0179 |
| 256:16:4:1 | 57466 | 1018 | 79 | 939 | 0.0177 |
| 128:16:8:1 | 57466 | 1015 | 44 | 971 | 0.0177 |
| 16:16:64:1 | 57466 | 1014 | 13 | 1001 | 0.0176 |

| test-fmath 改变cache相联度 | | | | | |
|-----------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:16:1:1 | 16639 | 930 | 127 | 803 | 0.0559 |
| 512:16:2:1 | 16639 | 930 | 119 | 811 | 0.0559 |
| 256:16:4:1 | 16639 | 905 | 32 | 873 | 0.0544 |
| 128:16:8:1 | 16639 | 901 | 10 | 891 | 0.0541 |
| 16:16:64:1 | 16639 | 889 | 0 | 889 | 0.054 |

| vortex.ss 改变cache相联度 | | | | | |
|----------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:16:1:1 | 17010 | 1008 | 204 | 804 | 0.0593 |
| 512:16:2:1 | 17010 | 997 | 174 | 823 | 0.0586 |
| 256:16:4:1 | 17010 | 949 | 58 | 891 | 0.0576 |
| 128:16:8:1 | 17010 | 976 | 23 | 953 | 0.0574 |
| 16:16:64:1 | 17010 | 974 | 3 | 971 | 0.0573 |

| mcf00.02unroll.gcc.100M.ss 改变cache相联度 | | | | | |
|---------------------------------------|------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:16:1:1 | 3801 | 852 | 75 | 777 | 0.2242 |
| 512:16:2:1 | 3801 | 845 | 54 | 791 | 0.2223 |
| 256:16:4:1 | 3801 | 831 | 13 | 818 | 0.2186 |
| 128:16:8:1 | 3801 | 831 | 12 | 819 | 0.2186 |
| 16:16:64:1 | 3801 | 828 | 0 | 828 | 0.2178 |

(3) 拐点寻找



| 拐点出现位置 | | | | |
|--------|-----------|------------|-----------|----------------------------|
| 程序 | test-math | test-fmath | vortex.ss | mcd00.02unroll gcc.100M.ss |
| 拐点 | 无 | 无 | 无 | 无 |

(4) 分析 Cache 相联度对 Cache 性能的影响

Cache 相联度的增加，总失效次数小幅度减少，失效率缓慢降低。同时，容量失效与冲突失效次数减少，强制性失效次数增加。总的来说，改变相联度对于失效率的影响并不明显。

改变相联度，均无明显拐点出现，说明可能四个程序的数据聚集性较好，不会分布的很分散。

4、改变 Cache 块大小

(1) 思路

改变 Cache 块大小，同样需要保证 Cache 总容量不变，因此同样使用改变组数的策略，来保证控制变量。

本次实验中，使用 1024 组，块大小 8bytes，组内 1 块，LRU 替换算法作为基准。

(2) 实验结果记录

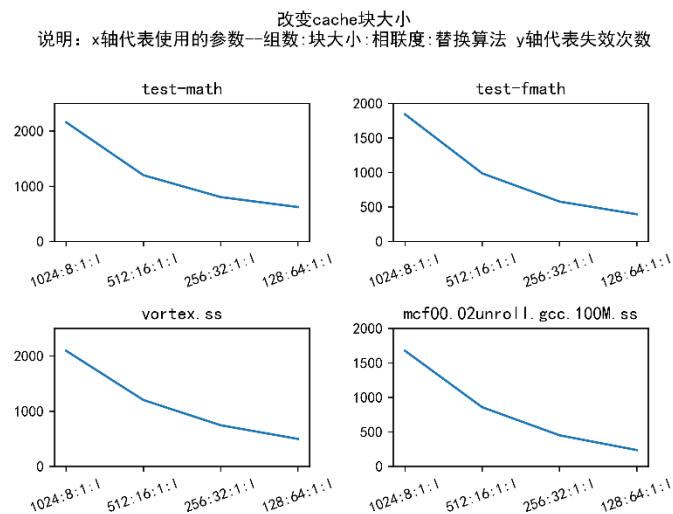
| test-math 改变cache块大小 | | | | | |
|----------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:8:1:1 | 57466 | 2160 | 1136 | 1024 | 0.0376 |
| 512:16:1:1 | 57466 | 1201 | 689 | 512 | 0.0209 |
| 256:32:1:1 | 57466 | 804 | 548 | 256 | 0.014 |
| 128:64:1:1 | 57466 | 623 | 495 | 128 | 0.0108 |
| 16:512:1:1 | 1 | 1 | 0 | 1 | 1 |

| test-fmath 改变cache块大小 | | | | | |
|-----------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:8:1:1 | 16639 | 1844 | 820 | 1024 | 0.1108 |
| 512:16:1:1 | 16639 | 987 | 475 | 512 | 0.0593 |
| 256:32:1:1 | 16639 | 578 | 322 | 256 | 0.0347 |
| 128:64:1:1 | 16639 | 394 | 266 | 128 | 0.0237 |
| 16:512:1:1 | 1 | 1 | 0 | 1 | 1 |

| vortex.ss 改变cache块大小 | | | | | |
|----------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:8:1:1 | 17010 | 2096 | 1072 | 1024 | 0.1232 |
| 512:16:1:1 | 17010 | 1203 | 691 | 512 | 0.0707 |
| 256:32:1:1 | 17010 | 746 | 490 | 256 | 0.0439 |
| 128:64:1:1 | 17010 | 498 | 370 | 128 | 0.0293 |
| 16:512:1:1 | 1 | 1 | 0 | 1 | 1 |

| mcf00.02unroll.gcc.100M.ss 改变cache块大小 | | | | | |
|---------------------------------------|------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:8:1:1 | 3801 | 1677 | 653 | 1024 | 0.4412 |
| 512:16:1:1 | 3801 | 858 | 346 | 512 | 0.2257 |
| 256:32:1:1 | 3801 | 451 | 195 | 256 | 0.1187 |
| 128:64:1:1 | 3801 | 237 | 109 | 128 | 0.0624 |
| 16:512:1:1 | 1 | 1 | 0 | 1 | 1 |

(3) 拐点寻找



| 拐点出现位置 | | | | |
|--------|-----------|------------|-----------|----------------------------|
| 程序 | test-math | test-fmath | vortex.ss | mcd00.02unroll.gcc.100M.ss |
| 拐点 | 块大小16的左右 | 块大小16的左右 | 块大小16的左右 | 块大小16的左右 |

(4) 分析 Cache 块大小对于 Cache 性能的影响

随着 Cache 块大小的增加,总失效次数降低,容量失效与冲突失效次数降低,强制性失效次数降低。当块大小非常大时,总失效次数变为 1 次。是因为该程序在内存中存放的位置的数据聚集性较好,使得当数据块增大时,这些数据落入这个块的机会将显著增大。因此,只需 1 次强制性失效即可满足程序要求。

改变块大小,总失效次数出现明显减少。说明这四个程序的数据聚集性较好,

当块大小逐渐增大时,可能刚好落在一个块内的数据更多,使得总失效次数减少。由于当块大小特别大时,总失效次数为1,代表了错误的出现,因此绘图时舍去了这一组数据。

5、采用 LRU 与随机法,在不同 Cache 容量和相联度下分别测试

(1) 思路

Cache 的组内块数最小应为 2,否则替换算法之间将无法体现差别。其余配置遵循上文对于 Cache 容量、相联度的约定。实验时,改变替换算法为 LRU 和随机法即可。

(2) 不同 Cache 容量下、不同替换算法的实验结果与记录

| test-math 改变cache容量、替换算法 | | | | | |
|--------------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 16:32:2:l | 57466 | 2838 | 2806 | 32 | 0.0494 |
| 16:32:2:r | 57466 | 3137 | 3105 | 32 | 0.0546 |
| 32:32:2:l | 57466 | 1394 | 1330 | 64 | 0.0243 |
| 32:32:2:r | 57466 | 1427 | 1363 | 64 | 0.0248 |
| 64:32:2:l | 57466 | 839 | 711 | 128 | 0.0146 |
| 64:32:2:r | 57466 | 882 | 754 | 128 | 0.0153 |
| 128:32:2:l | 57466 | 668 | 412 | 256 | 0.0116 |
| 128:32:2:r | 57466 | 693 | 454 | 239 | 0.0121 |
| 1024:32:2:l | 57466 | 541 | 0 | 541 | 0.0094 |
| 1024:32:2:r | 57466 | 541 | 7 | 534 | 0.0094 |

| test-fmath 改变cache容量、替换算法 | | | | | |
|---------------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 16:32:2:l | 16639 | 932 | 900 | 32 | 0.056 |
| 16:32:2:r | 16639 | 969 | 937 | 32 | 0.0582 |
| 32:32:2:l | 16639 | 583 | 519 | 64 | 0.035 |
| 32:32:2:r | 16639 | 631 | 567 | 64 | 0.0379 |
| 64:32:2:l | 16639 | 511 | 383 | 128 | 0.0307 |
| 64:32:2:r | 16639 | 524 | 396 | 128 | 0.0315 |
| 128:32:2:l | 16639 | 501 | 245 | 256 | 0.0301 |
| 128:32:2:r | 16639 | 506 | 273 | 233 | 0.0304 |
| 1024:32:2:l | 16639 | 469 | 0 | 469 | 0.0282 |
| 1024:32:2:r | 16639 | 469 | 0 | 469 | 0.0282 |

| vortex.ss 改变cache容量、替换算法 | | | | | |
|--------------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 16:32:2:l | 17010 | 1366 | 1334 | 32 | 0.0803 |
| 16:32:2:r | 17010 | 1472 | 1440 | 32 | 0.0865 |
| 32:32:2:l | 17010 | 817 | 753 | 64 | 0.048 |
| 32:32:2:r | 17010 | 830 | 766 | 64 | 0.0488 |
| 64:32:2:l | 17010 | 655 | 527 | 128 | 0.0385 |
| 64:32:2:r | 17010 | 672 | 545 | 127 | 0.0395 |
| 128:32:2:l | 17010 | 545 | 289 | 256 | 0.032 |
| 128:32:2:r | 17010 | 557 | 321 | 236 | 0.0327 |
| 1024:32:2:l | 17010 | 508 | 1 | 507 | 0.0299 |
| 1024:32:2:r | 17010 | 513 | 36 | 477 | 0.0302 |

| mcf00.02unroll.gcc.100M.ss 改变cache容量、替换算法 | | | | | |
|---|------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 16:32:2:l | 3801 | 456 | 424 | 32 | 0.12 |
| 16:32:2:r | 3801 | 459 | 427 | 32 | 0.1208 |
| 32:32:2:l | 3801 | 456 | 392 | 64 | 0.12 |
| 32:32:2:r | 3801 | 459 | 395 | 64 | 0.1208 |
| 64:32:2:l | 3801 | 450 | 322 | 128 | 0.1184 |
| 64:32:2:r | 3801 | 455 | 329 | 126 | 0.1197 |
| 128:32:2:l | 3801 | 449 | 193 | 256 | 0.1181 |
| 128:32:2:r | 3801 | 452 | 225 | 227 | 0.1189 |
| 1024:32:2:l | 3801 | 424 | 0 | 424 | 0.1115 |
| 1024:32:2:r | 3801 | 425 | 3 | 422 | 0.1118 |

(3) 不同 Cache 相联度下、不同替换算法的实验结果与记录

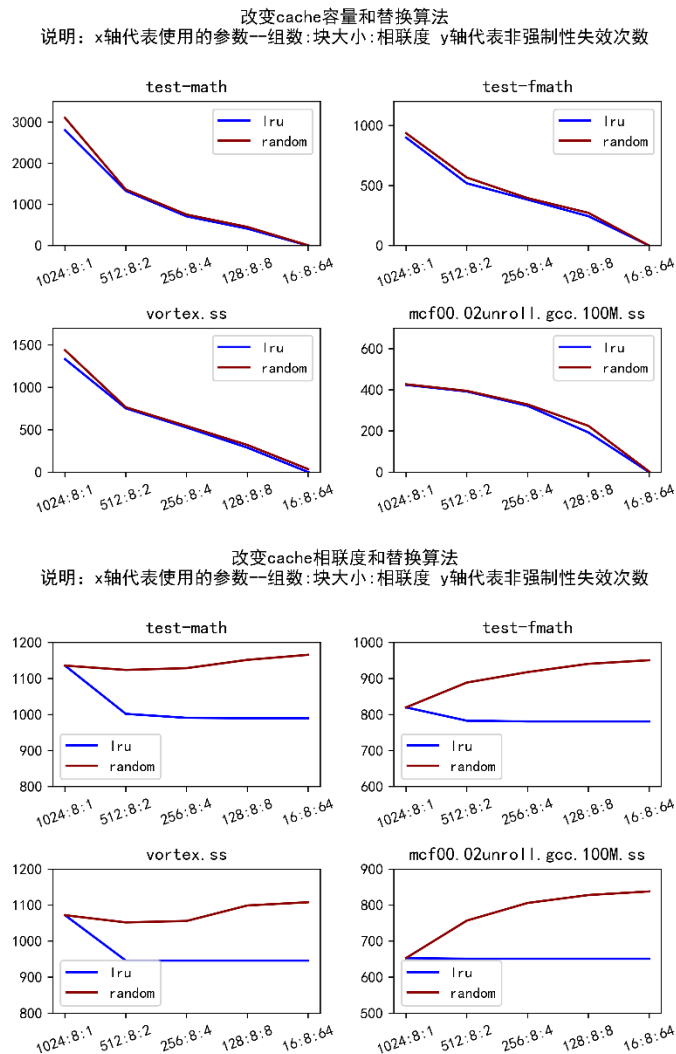
| test-math 改变cache相联度、替换算法 | | | | | |
|---------------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:8:1:l | 57466 | 2160 | 1136 | 1024 | 0.0376 |
| 1024:8:1:r | 57466 | 2160 | 1136 | 1024 | 0.0376 |
| 512:8:2:l | 57466 | 2026 | 1002 | 1024 | 0.0353 |
| 512:8:2:r | 57466 | 2063 | 1124 | 939 | 0.0359 |
| 256:8:4:l | 57466 | 2015 | 991 | 1024 | 0.0351 |
| 256:8:4:r | 57466 | 2044 | 1129 | 915 | 0.0356 |
| 128:8:8:l | 57466 | 2014 | 990 | 1024 | 0.035 |
| 128:8:8:r | 57466 | 2046 | 1152 | 894 | 0.0356 |
| 16:8:64:l | 57466 | 2014 | 990 | 1024 | 0.035 |
| 16:8:64:r | 57466 | 2064 | 1166 | 898 | 0.0359 |

| test-fmath 改变cache相联度、替换算法 | | | | | |
|----------------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:8:1:l | 16639 | 1844 | 820 | 1024 | 0.1108 |
| 1024:8:1:r | 16639 | 1844 | 820 | 1024 | 0.1108 |
| 512:8:2:l | 16639 | 1807 | 783 | 1024 | 0.1086 |
| 512:8:2:r | 16639 | 1817 | 889 | 928 | 0.1092 |
| 256:8:4:l | 16639 | 1805 | 781 | 1024 | 0.1085 |
| 256:8:4:r | 16639 | 1807 | 918 | 889 | 0.1086 |
| 128:8:8:l | 16639 | 1805 | 781 | 1024 | 0.1085 |
| 128:8:8:r | 16639 | 1803 | 941 | 862 | 0.1084 |
| 16:8:64:l | 16639 | 1805 | 781 | 1024 | 0.1085 |
| 16:8:64:r | 16639 | 1807 | 951 | 856 | 0.1086 |

| vortex.ss 改变cache相联度、替换算法 | | | | | |
|---------------------------|-------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:8:1:l | 17010 | 2096 | 1072 | 1024 | 0.1232 |
| 1024:8:1:r | 17010 | 2096 | 1072 | 1024 | 0.1232 |
| 512:8:2:l | 17010 | 1970 | 946 | 1024 | 0.1158 |
| 512:8:2:r | 17010 | 1991 | 1052 | 939 | 0.117 |
| 256:8:4:l | 17010 | 1970 | 946 | 1024 | 0.1158 |
| 256:8:4:r | 17010 | 1970 | 1056 | 914 | 0.1158 |
| 128:8:8:l | 17010 | 1970 | 946 | 1024 | 0.1158 |
| 128:8:8:r | 17010 | 1985 | 1099 | 886 | 0.1167 |
| 16:8:64:l | 17010 | 1970 | 946 | 1024 | 0.1158 |
| 16:8:64:r | 17010 | 1987 | 1108 | 879 | 0.1168 |

| mcf00.02unroll.gcc.100M.ss 改变cache相联度、替换算法 | | | | | |
|--|------|-------|-------------|---------|--------|
| 参数 | 访问次数 | 总失效次数 | 容量失效与冲突失效次数 | 强制性失效次数 | 失效率 |
| 1024:8:1:l | 3801 | 1677 | 653 | 1024 | 0.4412 |
| 1024:8:1:r | 3801 | 1677 | 653 | 1024 | 0.4412 |
| 512:8:2:l | 3801 | 1675 | 651 | 1024 | 0.4407 |
| 512:8:2:r | 3801 | 1672 | 757 | 915 | 0.4399 |
| 256:8:4:l | 3801 | 1675 | 651 | 1024 | 0.4407 |
| 256:8:4:r | 3801 | 1672 | 806 | 866 | 0.4399 |
| 128:8:8:l | 3801 | 1675 | 651 | 1024 | 0.4407 |
| 128:8:8:r | 3801 | 1668 | 828 | 840 | 0.4388 |
| 16:8:64:l | 3801 | 1675 | 651 | 1024 | 0.4407 |
| 16:8:64:r | 3801 | 1667 | 838 | 829 | 0.4386 |

(4) 分析不同的替换算法对 Cache 性能的影响



图中是非强制性失效的次数在不同容量/相联度、替换算法下的变化趋势。

使用 LRU 算法与随机替换算法，在总失效次数上差别不大。但是在容量失效和冲突失效上，即非强制性失效上，使用 LRU 算法的失效次数会明显少于随机替换算法。在强制性失效上使用 LRU 算法的失效次数会明显多于随机替换算法。