****

**计算机系统结构II**

**实验报告**

**院 系： 信息学部**

**学 号： 21071003**

**姓 名： 高立扬**

**同组人学号： 21071015**

**同组人姓名： 李博文**

**指导老师： 方 娟**

**完成时间：**

目录

[实验一 流水线相关 2](#_Toc13618)

[一、 实验目的 2](#_Toc23286)

[二、 实验平台 2](#_Toc19315)

[三、 实验内容与结果分析 2](#_Toc19213)

[1. 用WinMIPS64模拟器执行求和程序sum.s 2](#_Toc13759)

[2. 用WinMIPS64模拟器执行for循环程序test\_for.s 4](#_Toc25364)

[3. 执行 test\_for.s 程序 5](#_Toc21765)

[4. 计算定向技术前后性能提高的倍数 6](#_Toc2773)

[实验二 循环展开与指令调度 8](#_Toc6704)

[一、 实验目的 8](#_Toc7849)

[二、 实验平台 8](#_Toc27342)

[三、 实验内容及结果分析 8](#_Toc26142)

[1. DLX汇编语言编写代码文件 8](#_Toc20387)

[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 9](#_Toc20650)

[3. 采用循环展开、寄存器换名以及指令调度提高性能 10](#_Toc25857)

[实验三 Cache性能分析 12](#_Toc12581)

[一、 实验目的 12](#_Toc752)

[二、 实验平台 12](#_Toc13789)

[三、 实验内容及结果分析 12](#_Toc858)

[1.基本配置运行 12](#_Toc2691)

[2. 改变Cache容量 12](#_Toc32478)

[3. 改变Cache相联度 13](#_Toc14910)

[4. 改变Cache块大小 14](#_Toc23798)

[5. 采用 LRU 与随机法，在不同 Cache 容量和相联度下分别测试 14](#_Toc23471)

## 实验一 流水线相关

1. 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

1. 实验平台

WinMIPS64模拟器

1. 实验内容与结果分析
2. 用WinMIPS64模拟器执行求和程序sum.s

1.1单步执行

代码如图1.1

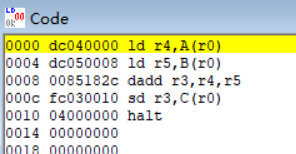


图1.1 sum.s指令代码

通过菜单栏的Execute中的Single Cycle单步执行，可以通过观察左下角的Pipeline图观察流水线的情况，如图1.2所示

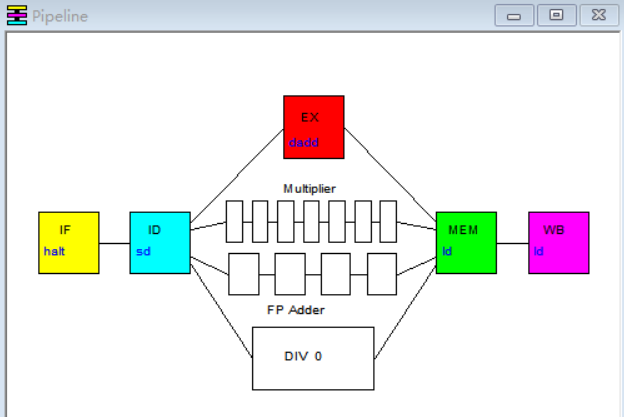


图1.2 sum.s Pipeline图

图1.2对应的是Cycles中的指令情况，如图1.3所示。

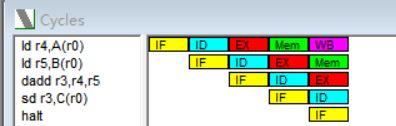


图1.3 sum.s 指令情况

当第一条指令运行到WB流水段时，即将数据写回给寄存器，对应的寄存器会标绿，如图所示。

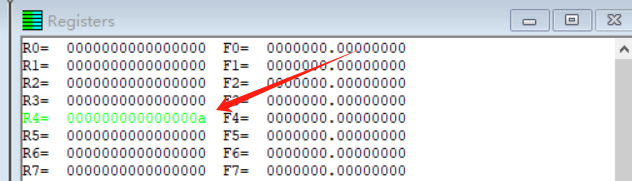


图1.4 sum.s 寄存器

1.2连续执行

不采用定向技术情况下的时空流水图：

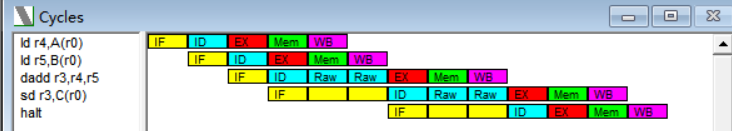


图1.5 sum.s时空流水图

如图1.5可以看出，在不采用定向技术的情况下，第三条指令“dadd”产生了数据相关，由于dadd r3,r4,r5需要将r4和r5的数据相加，因此需要等待前两条的指令完成才可以进行第三条指令的EX指令，因此产生了Raw。

寄存器内容如图1.6所示：



图1.6 sum.s寄存器

存储器内容如图1.7所示：

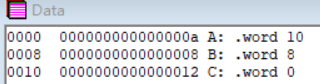


图1.7 sum.s存储器

statistics如图1.8所示：

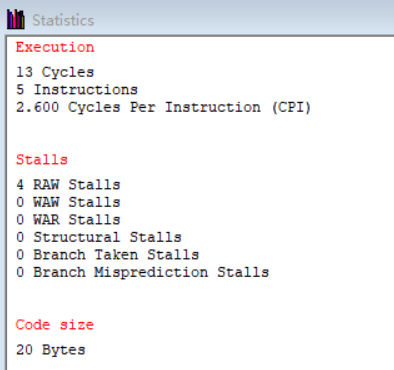


图1.8 sum.s statistics

1. 用WinMIPS64模拟器执行for循环程序test\_for.s

本程序包含了循环，不采用定向技术，直接通过run to执行此程序，模仿1.进行观察，结果如下

寄存器如图1.9所示：

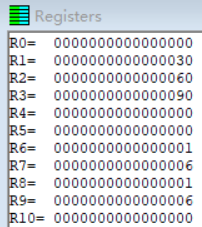


图1.9 test\_for.s寄存器

存储器如图1.10所示：

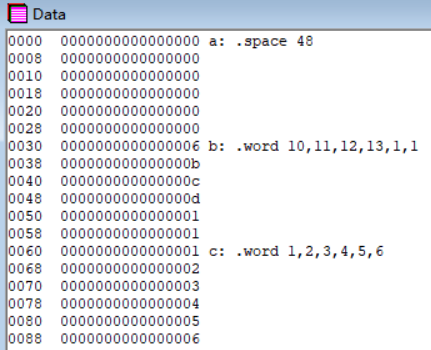


图1.10 test\_for.s存储器

statistics如图1.11所示：

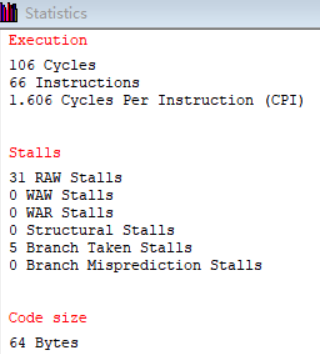


图1.11 test\_for.s statistics

1. 执行 test\_for.s 程序

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；

本环节不采用定向技术

3.1找出资源相关和导致资源相关的部件

下图1.12中，指令dadd r8,r5,r6和dmul r9,r7,r8存在先写后读的资源相关，这导致了后者在ID阶段等待，等待前者写入数据，直到前者写入完毕，后者才会结束ID阶段的Raw，进入乘法的七个周期。

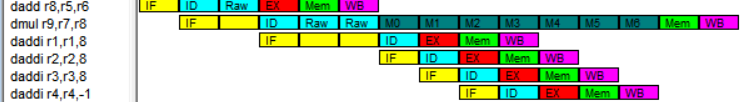


图1.12 test\_for.s流水线上的资源相关

3.2资源相关引起的暂停时钟周期数及百分比

statistics见上文图1.11所示，针对本问题的数据计算见下表1.1

表1.1 数据计算

|  |  |  |
| --- | --- | --- |
| test\_for.s | | |
| 总执行周期数 | 资源相关引起的暂停时钟周期数 | 占比 |
| 106 | 31 | 29.2% |

3.3资源相关对 CPU 性能的影响，以及解决方法

·资源相关对CPU性能的有以下几个影响：

①资源相关会可能会导致错误的执行结果。

②流水线可能因此出现停顿，不能在指定时钟周期内执行完成，增大了实际执行时钟周期，降低效率和实际加速比，增大了CPI，降低了CPU性能。

·解决资源相关方法：

①使指令停顿n拍后再进入流水线，使该条指令进入后不再产生资源相关。

②重复设置资源相关的功能部件，使得不同指令使用不同的功能部件执行相同操作，从而避免资源相关的发生。

1. 计算定向技术前后性能提高的倍数

通过CPU执行用户程序所用时间衡量CPU性能，即：

根据采用定向技术前后的统计数据记录数据相关引起的暂停时钟周期数以及程序执行的总时钟周期数，计算暂停时钟周期数占总执行周期数的百分比，并计算采用定向技术后性能提高的倍数。由于未知，且采用定向技术前后的固定，所以性能的提升比例可以通过计算比值得到。

4.1 sum.s

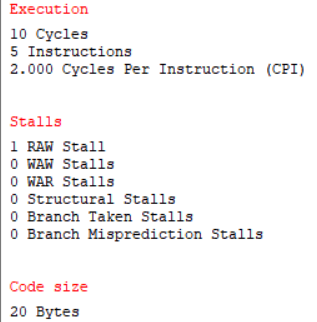
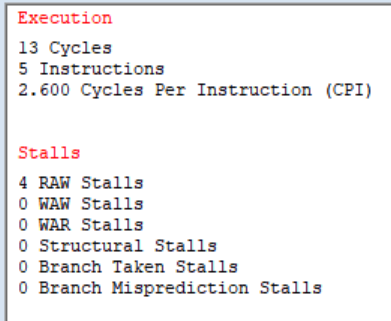


图1.14 sum.s未采用定向技术 图1.15 sum.s采用定向技术

表2.计算CPU性能

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| sum.s | | | | | | | |
|  | 总时钟数 | 指令总数 | 数据相关引起的暂停时钟数 | 所占百分比 | CPI |  | 性能提升倍数 |
| 不采用定向技术 | 13 | 5 | 4 | 30.8% | 2.600 | 13Tc | 1.30 |
| 采用定向技术 | 10 | 5 | 1 | 10.0% | 2.000 | 10Tc |

4.2 test\_for.s

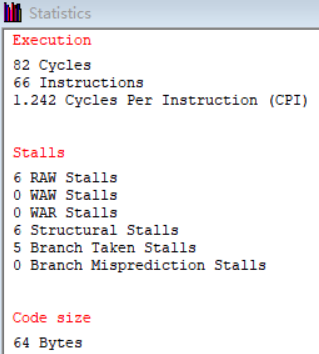
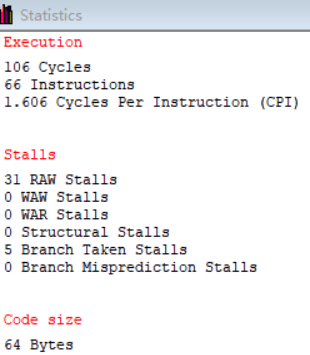


图1.16 test\_for.s未采用定向技术 图1.17 test\_for.s采用定向技术

表2.计算CPU性能

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| test\_for.s | | | | | | | |
|  | 总时钟数 | 指令总数 | 数据相关引起的暂停时钟数 | 所占百分比 | CPI |  | 性能提升倍数 |
| 不采用定向技术 | 106 | 66 | 31 | 29.2% | 1.606 | 106Tc | 1.29 |
| 采用定向技术 | 82 | 66 | 6 | 7.3% | 1.242 | 82Tc |

## 实验二 循环展开与指令调度

1. 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。

1. 实验平台

WinMIPS64模拟器

1. 实验内容及结果分析

1. DLX汇编语言编写代码文件

由于加法﹑乘法﹑除法部件各有2个，延迟时间都是3个时钟周期，这里以乘法为例，循环次数设置为四次（即4的整数次循环）。在WINMIPS64中设置乘法计算周期为3，采用定向技术，进行如下实验

1.1代码实现

程序中包括数据相关与结构相关，程序代码如下：

.data

A: .word 12,16,20,8

B: .word 4,2,5,1

C: .space 64

.text

main:

daddi r2,r0,4;设置循环次数

Loop:

ld r3,A(r1);读取A[i]

ld r4,B(r1);读取B[i]

dmul r5,r3,r4;运算

sd r5,C(r1);存储C[i]

daddi r1,r1,8;数组下标推进

daddi r2, r2, -1;i--

bnez r2,Loop

halt

转换为c语言代码如下：

int A[4]={12,16,20,8},B[4]={4,2,5,1},C[4];

for(int i=0;i<4;i++){

C[i]=A[i]\*B[i];

}

1.2数据记录

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| test3.s | | | | | | |
|  | 总时钟数 | 数据相关数 | 资源相关数 | 指令总数 | cpi | CPU性能 |
| 原程序 | 53 | 16 | 4 | 30 | 1.767 | 53.01Tc |

1.3发生冲突的指令集合

通过观察statistics发现每一次循环有四次数据相关和一次资源相关，Cycles一次循环情况如下图2.1所示

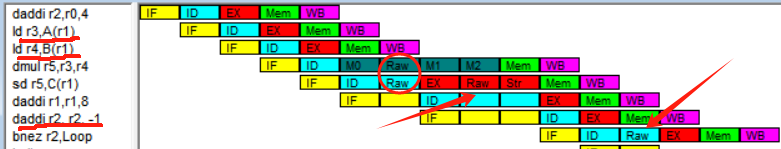


图2.1 Cycles

通过观察上图，发现两个ld指令与dmul数据相关，即dmul r5,r3,r4需要等待两个ld指令先对r3,r4进行写入操作。sd指令与dmul数据相关，sd需要等待dmul运算完毕给r5赋值，sd才能将r5的内容存入存储器。sd和其下一行daddi指令资源相关，二者都要访问r1。bnez和倒数第二行addi指令数据相关。

1.4存储器内容

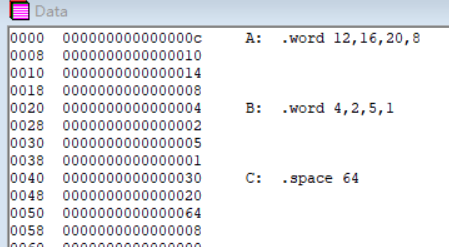


图2.2 存储器图

1.5寄存器内容

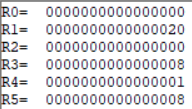


图2.3 寄存器图

1. 采用指令调度技术解决流水线中的结构相关与数据相关

2.1分析及实现思路

根据上文列出的发生冲突的指令组合，使用指令调度技术减少数据冲突以及资源冲突。即通过重排指令顺序避免指令流水线停顿，从而提高在计算机上指令流水线的性能。调整daddi指令的执行顺序，从而避免dmul指令与ld指令之间的数据相关，以及避免bnez指令和daddi指令之间的数据相关问题。由于指令功能简单，无法避免sd与daddi之间的资源相关问题。

2.2修改后代码

.data

A: .word 12,16,20,8

B: .word 4,2,5,1

C: .space 64

.text

main:

daddi r2,r0,4;设置循环次数

Loop:

ld r3,A(r1)

ld r4,B(r1)

daddi r2, r2, -1;i--

dmul r5,r3,r4

daddi r1,r1,8

sd r5,C(r1);无法避免

bnez r2,Loop

halt

2.3 性能计算

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 指令调度技术 | | | | | | | |
|  | 总时钟数 | 数据相关数 | 结构相关数 | 指令总数 | cpi | CPU性能 | 性能提升倍数 |
| 原程序 | 53 | 16 | 4 | 30 | 1.767 | 53.01Tc | 1.290 |
| 指令调度 | 41 | 0 | 4 | 30 | 1.367 | 41.01Tc |

2.4 结论

比较调度前和调度后的性能。论述指令调度对于提高CPU性能的意义

通过指令调度，CPU 的性能明显得到提高。指令调度能够消除部分的数据相关、结构相关，从而减少了 CPU 空转的次数，实现 CPU 性能的提升的目的。

1. 采用循环展开、寄存器换名以及指令调度提高性能

3.1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，从而进一步提高性能。

3.2修改后代码

.data

A: .word 12,16,20,8

B: .word 4,2,5,1

C: .space 64

.text

main:

Loop:

ld r3,A(r1)

ld r4,B(r1)

daddi r1,r1,8

dmul r5,r3,r4

ld r3,A(r1)

ld r4,B(r1)

sd r5,C(r1) ;C[1]存储A[0]\*B[0],这样保证了无数据相关,但是仍无法避免资源相关

daddi r1,r1,8

dmul r5,r3,r4

ld r3,A(r1)

ld r4,B(r1)

sd r5,C(r1)

daddi r1,r1,8

dmul r5,r3,r4

ld r3,A(r1)

ld r4,B(r1)

sd r5,C(r1)

dmul r5,r3,r4

daddi r1,r1,8

sd r5,C(r1)

halt

将循环展开，导致和循环有关的寄存器和运算指令都被去除，使得指令需要重新调度。采用牺牲C[0]来进行调度，如代码注释所示。资源相关仍旧无法避免

3.3数据记录

循环展开后的统计数据如图2.4所示，根据这些数据可以得到循环展开后的CPU性能提升倍数

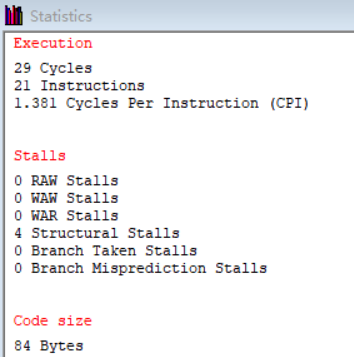


图2.4 统计信息图

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 循环展开 | | | | | | | |
|  | 总时钟数 | 数据相关数 | 结构相关数 | 指令总数 | cpi | CPU性能 | 性能提升倍数 |
| 原程序 | 53 | 16 | 4 | 30 | 1.767 | 53.01Tc | 1.828 |
| 循环展开 | 29 | 0 | 4 | 21 | 1.381 | 29.00Tc |

3.4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，能够减少无关命令、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

1. 实验目的

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。

1. 实验平台

SimpleScalar 模拟器

1. 实验内容及结果分析

1.基本配置运行

选用程序test-math 指令:./simplesim-3.0/sim-cache test-math，默认参数为dl1:256:32:1:l，运行结果如图3.1所示，misses是总失效次数，replacements是容量失效和冲突失效总次数

初步运行结果：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 总访问次数 | 总失效次数 | 容量失效&冲突失效 | 强制性失效 | 失效率 |
| 默认参数 | 57466 | 804 | 548 | 256 | 0.014 |

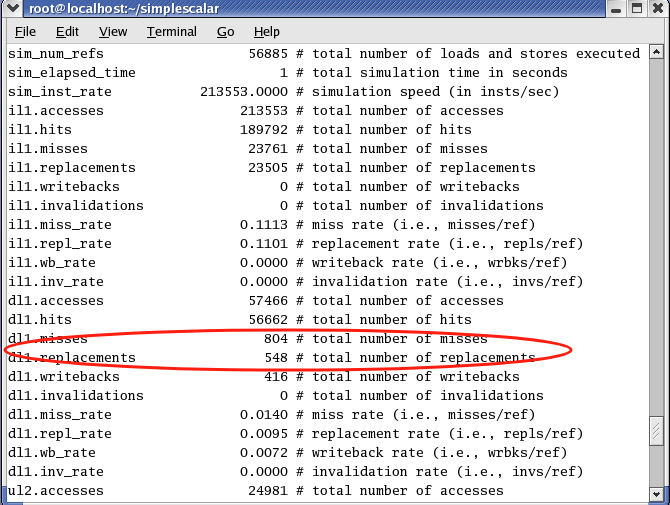


图3.1 运行结果

1. 改变Cache容量

改变Cache的容量，实验结果如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 总访问次数 | 总失效次数 | 容量失效&冲突失效 | 强制性失效 | 失效率 |
| 16:32:1:l | 57466 | 7708 | 7692 | 16 | 0.1341 |
| 32:32:1:l | 4102 | 4070 | 32 | 0.0714 |
| 64:32:1:l | 2030 | 1966 | 64 | 0.0353 |
| 128:32:1:l | 1209 | 1081 | 128 | 0.0210 |
| 1024:32:1:l | 542 | 13 | 529 | 0.0094 |
| 2048:32:1:l | 542 | 11 | 531 | 0.0094 |
| 4096:32:1:l | 541 | 0 | 541 | 0.0094 |

图3.2

如图3.2所示，随着Cache容量提高：失效率下降，但是并不会无限下降，组数在1024及以上，失效率停留在0.94%；容量失效&冲突失效不断下降到0，强制性失效相对提升

1. 改变Cache相联度

改变相联度，为了确保容量不变，因此需要改变组数。一共进行两组实验，结果如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 总访问次数 | 总失效次数 | 容量失效&冲突失效 | 强制性失效 | 失效率 |
| 1024:32:1:l | 57466 | 542 | 13 | 529 | 0.0094 |
| 512:32:2:l | 541 | 2 | 539 | 0.0094 |
| 256:32:4:l | 541 | 1 | 540 | 0.0094 |
| 128:32:8:l | 541 | 0 | 541 | 0.0094 |
| 16:32:64:l | 541 | 0 | 541 | 0.0094 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 总访问次数 | 总失效次数 | 容量失效&冲突失效 | 强制性失效 | 失效率 |
| 1024:16:1:l | 57466 | 1034 | 204 | 830 | 0.0180 |
| 512:16:2:l | 1029 | 180 | 849 | 0.0179 |
| 256:16:4:l | 1018 | 79 | 939 | 0.0177 |
| 128:16:8:l | 1015 | 44 | 971 | 0.0177 |
| 16:16:64:l | 1014 | 13 | 1001 | 0.0176 |

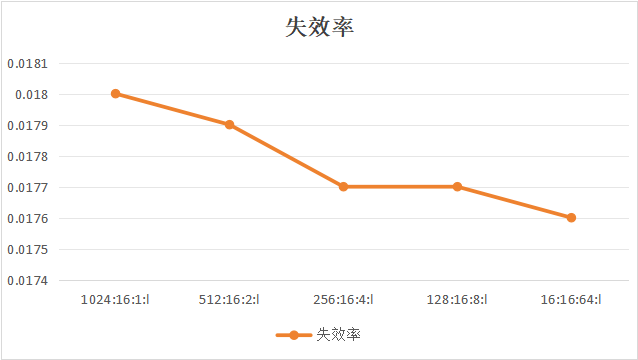
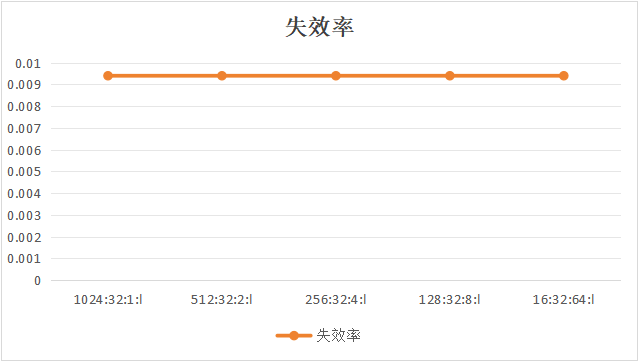


图3.3 第一组 图3.4 第二组

结合2.中结果，分析上述结果，发现当组内块数较大，满足程序运转需求时，提升Cache相联度不会改变失效率，会略微降低容量失效&冲突失效。而当组内块数不足时，变为第二组的情况，随着相联度提升，容量失效&冲突失效降低，强制性失效相对上升，但是失效率仅略微下降，下降步幅为0.01%

1. 改变Cache块大小

改变Cache块大小，为了确保容量不变，因此需要改变组数。结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 总访问次数 | 总失效次数 | 容量失效&冲突失效 | 强制性失效 | 失效率 |
| 1024:8:1:l | 57466 | 2160 | 1136 | 1024 | 0.0376 |
| 512:16:1:l | 1201 | 689 | 512 | 0.0209 |
| 256:32:1:l | 804 | 548 | 256 | 0.0140 |
| 128:64:1:l | 623 | 495 | 128 | 0.0108 |
| 16:512:1:l | 1 | 1 | 0 | 1 | 1 |

图3.5 改变块大小

随着块增大，失效率显著下降，容量失效&冲突失效下降，强制性失效相对上升。当块大小足够大的时候，发生一次强制性失效后，停止访问，代表了错误发生，因此绘图时没有计入该列数据。

1. 采用 LRU 与随机法，在不同 Cache 容量和相联度下分别测试
   1. 改算法，改容量

结果如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 总访问次数 | 总失效次数 | 容量失效&冲突失效 | 强制性失效 | 失效率 |
| 16:16:2:l | 57466 | 4172 | 4140 | 32 | 0.0726 |
| 16:16:2:r | 4758 | 4726 | 32 | 0.0828 |
| 32:16:2:l | 2889 | 2825 | 64 | 0.0503 |
| 32:16:2:r | 3058 | 2090 | 68 | 0.0532 |
| 64:16:2:l | 1608 | 1480 | 128 | 0.0280 |
| 64:16:2:r | 1671 | 1543 | 128 | 0.0291 |
| 128:16:2:l | 1278 | 1022 | 256 | 0.0222 |
| 128:16:2:r | 1340 | 1087 | 253 | 0.0233 |
| 1024:16:2:l | 1011 | 2 | 1009 | 0.0176 |
| 1024:16:2:r | 1025 | 102 | 923 | 0.0178 |
| 2048:16:2:l | 1011 | 0 | 1011 | 0.0176 |
| 2048:16:2:r | 1011 | 8 | 1003 | 0.0176 |

图3.6 改算法，改容量

分析结果发现，两种算法通过更改容量，在失效率上差距不大；而LRU算法的总失效次数少于随机法，但是随着容量增大，二者总失效次数趋近于相等；二者的容量失效&冲突失效占比和强制性失效占比相似

* 1. 改算法，改相联度

结果如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 总访问次数 | 总失效次数 | 容量失效&冲突失效 | 强制性失效 | 失效率 |
| 1024:16:1:l | 57466 | 1034 | 204 | 830 | 0.018 |
| 1024:16:1:r | 1034 | 204 | 830 | 0.018 |
| 512:16:2:l | 1029 | 180 | 849 | 0.0179 |
| 512:16:2:r | 1050 | 312 | 738 | 0.0183 |
| 256:16:4:l | 1018 | 79 | 939 | 0.0177 |
| 256:16:4:r | 1057 | 346 | 711 | 0.0184 |
| 128:16:8:l | 1015 | 44 | 971 | 0.0177 |
| 128:16:8:r | 1048 | 370 | 678 | 0.0182 |
| 16:16:64:l | 1014 | 13 | 1001 | 0.0176 |
| 16:16:64:r | 1052 | 376 | 676 | 0.0183 |

图3.7 改算法，改相联度

分析结果发现，LRU和随机法：

①相联度为1时算法性能无差别

②随着相联度的提升，前者的总失效次数下降，容量失效&冲突失效下降，强制性失效相对提升；而随机法则反之

## 实验四 华为云实验——HelloWorld程序实验

1. 实验目的
2. 加深对集群并行计算及开发的理解。
3. 掌握对MPI库的基本使用、编写简单并行计算程序。
4. 掌握MPI库在集群并行计算各阶段对程序进行什么处理。
5. 掌握集群之间配置密钥的方法。

（5）运行程序确保环境的正确安装，为后续实验做准备。

1. 实验平台

华为鲲鹏云主机、ubuntu20.04系统。

1. 实验内容及结果分析
2. 根据指导书的提示，进行HelloWorld源码的创建，Makefile的编写和hostfile的配置，如图1所示，Makefile的代码在粘贴时会丢掉第五行和第七行的Tab，需要注意补全。在进行hostfile配置的时候，如果按照指导书写，可能会出现报错：

[mpiexec@tracycomputer] Sending Ctrl-C to processes as requested

[mpiexec@tracycomputer] Press Ctrl-C again to force abort

[mpiexec@tracycomputer] HYDU\_sock\_write (utils/sock/sock.c:256): write error (Bad file descriptor)

[mpiexec@tracycomputer] HYD\_pmcd\_pmiserv\_send\_signal (pm/pmiserv/pmiserv\_cb.c:178): unable to write data to proxy

[mpiexec@tracycomputer] ui\_cmd\_cb (pm/pmiserv/pmiserv\_pmci.c:77): unable to send signal downstream

[mpiexec@tracycomputer] HYDT\_dmxu\_poll\_wait\_for\_event (tools/demux/demux\_poll.c:77): callback returned error status

[mpiexec@tracycomputer] HYD\_pmci\_wait\_for\_completion (pm/pmiserv/pmiserv\_pmci.c:196): error waiting for event

[mpiexec@tracycomputer] main (ui/mpich/mpiexec.c:336): process manager error waiting for completion

这时候将tracycomputer和konghancomputer换成对应的ip地址即可



图1. 配置

1. 输入如下的指令，查看运行结果如图2-3

mpiexec -n 4 -f /home/huawei/bjut21071003/hello/hostfile /home/huawei/bjut21071003/hello/mpi\_hello\_world

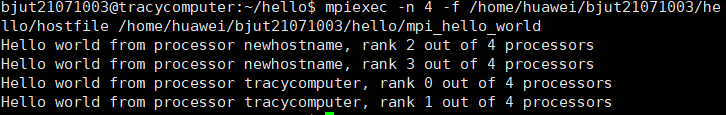


图2. 运行结果（konghan）

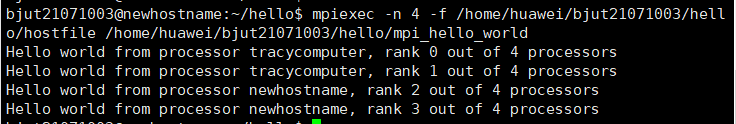


图3. 运行结果（tracy）

## 实验五 华为云实验——快速排序算法实验

1. 实验目的

（1）加深对集群并行计算及开发的理解。

（2）掌握对MPI库的基本使用、编写简单并行计算程序。

（3）掌握MPI库在集群并行计算各阶段对程序进行什么处理。

（4）掌握在集群上运行快速排序的方法，结合MPI库进行编写并行计算环境下的快速排序。

1. 实验平台

华为鲲鹏云主机、ubuntu20.04系统。

1. 实验内容及结果分析
2. 根据指导书的提示，进行quicksort源码的创建，Makefile的编写、hostfile的配置和脚本编写，如图1所示，Makefile的代码在粘贴时会丢掉第五行和第七行的Tab，需要注意补全；run.sh的Tab也会丢失，注意补全



图1. 配置

2.执行命令，查看结果如图2-3所示

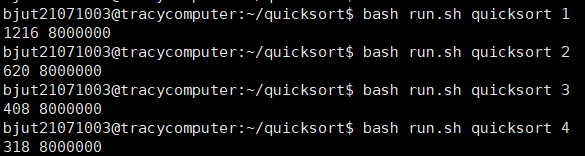


图2. 运行结果（konghan ）

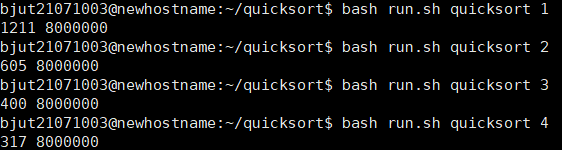


图3.运行结果（tracy）

## 华为云实验——思考题

1. 简述HelloWorld程序执行过程。

①初始化MPI环境

②获取进程数目

③获取进程的rank，在MPI中,进程之间的通信是通过发送和接收消息来实现的。发送和接收操作通常需要指定源进程和目标进程的Rank。通过Rank,可以明确指定通信的发送方和接收方,从而实现进程间的数据交换和同步。

④获得进程名

⑤打印Hello World

⑥结束MPI环境

1. g++中的-O1、-O2、-O3的区别是？

①-O1： 对程序做部分编译优化，对于大函数,优化编译占用稍微多的时间和相当大的内存。使用本项优化，编译器会尝试减小生成代码的尺寸，以及缩短执行时间，但并不执行需要占用大量编译时间的优化。

②-O2： 是比O1更高级的选项，进行更多的优化。Gcc将执行几乎所有的不包含时间和空间折中的优化。当设置O2选项时，编译器并不进行循环打开（）loop unrolling以及函数内联。与O1比较而言，O2优化增加了编译时间的基础上，提高了生成代码的执行效率。

③-O3：更进一步优化

3. 分别解释实验中所编写的makefile文件与hostfile文件的含义。

①makefile定义了一系列的规则来指定哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作

②hostfile指定了MPI节点，使得程序能够“知道”在哪个服务器上跑代码

4. ARM处理器的寻址方式有哪些？分别做简要介绍。

ARM处理器支持九种寻址方式：立即数寻址、寄存器寻址、寄存器偏移寻址、寄存器间接寻址、基址变址寻址、多寄存器寻址、相对寻址、堆栈寻址和块拷贝寻址。

立即数寻址是将指令中的数值直接作为操作数的地址；

寄存器寻址是以寄存器的值作为操作数的地址，即操作数存放在寄存器中；

寄存器偏移寻址是以寄存器的值加上偏移量得到的地址作为操作数的地址，其中偏移量可正可负；

寄存器间接寻址是以寄存器的值作为操作数的地址的地址，即操作数的地址存放在寄存器中；

基址变址寻址是以基址加上变址得到的地址作为操作数的地址，其中基址和变址可以是寄存器的值或立即数；

多寄存器寻址是指一次可以处理多个寄存器的操作数，这些寄存器的值作为操作数的地址；

相对寻址是由程序计数器PC提供基准地址，指令中的地址码字段作为偏移量，两者相加后得到有效地址；

堆栈寻址是指以堆栈指针SP所指向的内存区域为基准地址，加上或减去一个偏移量得到操作数的地址；

块拷贝寻址是将一块连续的内存区域复制到另一块连续的内存区域，源地址和目的地址都是基址加上偏移量得到的。