

哈尔滨工业大学

实验报告

实验（六）

题 目 Cachelab

高速缓冲器模拟

专 业 计算机科学与技术

学 号 L170300901

班 级 170300901

学 生 卢兑琬

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 12.2

计算机科学与技术学院

目 录

第 1 章 实验基本信息	- 3 -
1.1 实验目的.....	- 3 -
1.2 实验环境与工具.....	- 3 -
1.2.1 硬件环境.....	- 3 -
1.2.2 软件环境.....	- 3 -
1.2.3 开发工具.....	- 3 -
1.3 实验预习.....	- 3 -
第 2 章 实验预习	- 5 -
2.1 画出存储器层级结构，标识容量价格速度等指标变化（5 分）	- 5 -
2.2 用 CPUZ 等查看你的计算机 CACHE 各参数，写出各级 CACHE 的 C S E B S E B（5 分）	- 5 -
2.3 写出各类 CACHE 的读策略与写策略（5 分）	- 7 -
2.4 写出用 GPROF 进行性能分析的方法（5 分）	- 8 -
2.5 写出用 VALGRIND 进行性能分析的方法（5 分）	- 9 -
第 3 章 CACHE 模拟与测试	- 10 -
3.1 CACHE 模拟器设计	- 10 -
3.2 矩阵转置设计.....	- 13 -
第 4 章 总结	- 16 -
4.1 请总结本次实验的收获.....	- 16 -
4.2 请给出对本次实验内容的建议.....	- 16 -
参考文献	- 17 -

第 1 章 实验基本信息

1.1 实验目的

理解现代计算机系统存储器层级结构
掌握 Cache 的功能结构与访问控制策略
培养 Linux 下的性能测试方法与技巧
深入理解 Cache 组成结构对 C 程序性能的影响

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU; 2GHz; 2G RAM; 256GHD Disk

1.2.2 软件环境

Windows10 64 位; Ubuntu 16.04 LTS 64 位

1.2.3 开发工具

Visual Studio 2010 64 位以上; TestStudio; Gprof; Valgrind 等

1.3 实验预习

上实验课前, 必须认真预习实验指导书 (PPT 或 PDF)

了解实验的目的、实验环境与软硬件工具、实验操作步骤, 复习与实验有关的理论知识。

画出存储器的层级结构, 标识其容量价格速度等指标变化

用 CPUZ 等查看你的计算机 Cache 各参数, 写出 C S E B c s e b

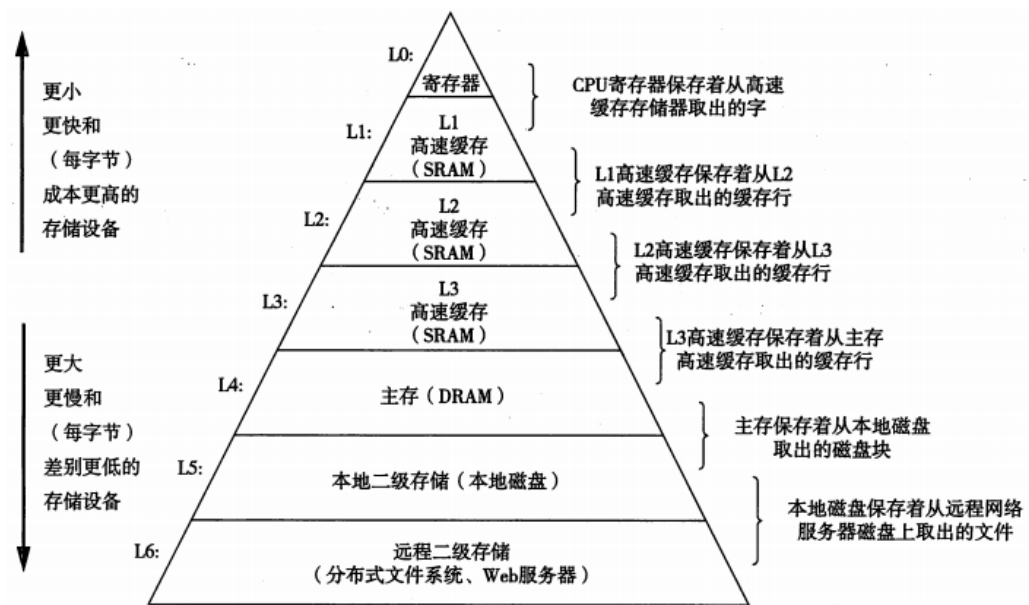
写出 Cache 的基本结构与参数

写出各类 Cache 的读策略与写策略

掌握 Valgrind 与 Gprof 的使用方法

第 2 章 实验预习

2.1 画出存储器层级结构，标识容量价格速度等指标变化 (5 分)




2.2 用 CPUZ 等查看你的计算机 Cache 各参数，写出各级 Cache 的 C S E B s e b (5 分)

CPU-Z

CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About

Processor

Name	Intel Core i5 6200U		
Code Name	Skylake-U/Y	Max TDP	15.0 W
Package	Socket 1168 BGA		
Technology	14 nm	Core VID	0.734 V



Specification: Intel® Core™ i5-6200U CPU @ 2.30GHz

Family	6	Model	E	Stepping	3
Ext. Family	6	Ext. Model	4E	Revision	D0/K0/K1

Instructions: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3

Clocks (Core #0)

Core Speed	797.85 MHz
Multiplier	x 8.0 (4 - 28)
Bus Speed	99.73 MHz
Rated FSB	

Cache

L1 Data	2 x 32 KBytes	8-way
L1 Inst.	2 x 32 KBytes	8-way
Level 2	2 x 256 KBytes	4-way
Level 3	3 MBytes	12-way

Selection: Socket #1 | Cores: 2 | Threads: 4

CPU-Z Ver. 1.86.0.x64 | Tools | Validate | Close

CPU-Z

CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About

L1 D-Cache

Size	32 KBytes	x 2
Descriptor	8-way set associative, 64-byte line size	

L1 I-Cache

Size	32 KBytes	x 2
Descriptor	8-way set associative, 64-byte line size	

L2 Cache

Size	256 KBytes	x 2
Descriptor	4-way set associative, 64-byte line size	

L3 Cache

Size	3 MBytes	
Descriptor	12-way set associative, 64-byte line size	

Size		
Descriptor		
Speed		

CPU-Z Ver. 1.86.0.x64 | Tools | Validate | Close

2.3 写出各类 Cache 的读策略与写策略 (5 分)

1. 当 CPU 发出读操作命令时, 根据它产生的主存地址分为两种情形:

一种是需要的数据已在 Cache 中, 那么只需直接访问 Cache, 从对应单元中读取信息到数据总线;

另一种是需要的数据尚未装入 Cache, CPU 需从主存中读取信息的同时, Cache 替换部件把该地址所在的那块存储内容从主存拷贝到 Cache 中; 若 Cache 中相应位置已被字块占满, 就必须去掉旧的字块。常见的替换策略有两种:

1. 先进先出策略 (FIFO)

FIFO (First In First Out) 策略总是把最先调入的 Cache 字块替换出去, 它不需要随时记录各个字块的使用情况, 较容易实现; 缺点是经常使用的块, 如一个包含循环程序的块也可能由于它是最早的块而被替换掉。

2. 最近最少使用策略 (LRU)

LRU (Least Recently Used) 策略是把当前近期 Cache 中使用次数最少的那块信息块替换出去, 这种替换算法需要随时记录 Cache 中字块的使用情况。LRU 的平均命中率比 FIFO 高, 在组相联映像方式中, 当分组容量加大时, LRU 的命中率也会提高。

2. 当 CPU 发出写操作命令时, 也要根据它产生的主存地址分为两种情形: 一种是不命中时, 只向主存写入信息, 不必同时把这个地址单元所在的整块内容调入 Cache 中; 另一种是命中时, 这时会遇到如何保持 Cache 与主存的一致性问题, 通常有三种处理方式:

1. 直写式 (write through)

即 CPU 在向 Cache 写入数据的同时, 也把数据写入主存以保证 Cache 和主存中相应单元数据的一致性, 其特点是简单可靠, 但由于 CPU 每次更新时都要对主存写入, 速度必然受影响。

2. 缓写式 (post write)

即 CPU 在更新 Cache 时不直接更新主存中的数据, 而是把更新的数据送入一个缓存器暂存, 在适当的时候再把缓存器中的内容写入主存。在这种方式下, CPU 不必等待主存写入而造成的时延, 在一定程度上提高了速度, 但由于缓存器

只有有限的容量，只能锁存一次写入的数据，如果是连续写入，CPU 仍需要等待。

3. 回写式 (write back)

即 CPU 只向 Cache 写入，并用标记加以注明，直到 Cache 中被写过的块要被进入的信息块取代时，才一次写入主存。这种方式考虑到写入的往往是中间结果，每次写入主存速度慢而且不必要。其特点是速度快，避免了不必要的冗余写操作，但结构上较复杂。

此外，还有一种设置不可 Cache 区 (Non-cacheable Block) 的方式，即在主存中开辟一块区域，该区域中的数据不受 Cache 控制器的管理，不能调入 Cache，CPU 只能直接读写该区域的内容。由于该区域不与 Cache 发生关系，也就不存在数据不一致性问题。目前微机系统的 BIOS 设置程序大多允许用户设置不可 Cache 区的首地址和大小。

2.4 写出用 gprof 进行性能分析的方法 (5 分)

基本用法：

1. 使用 -pg 选项编译和链接你的应用程序。
2. 运行你的应用程序，使之运行完成后生成供 gprof 分析的数据文件（默认是 gmon.out）。
3. 使用 gprof 程序分析你的应用程序生成的数据。
4. 可用 python 生成 png 图来直观看。

举例

```
gcc -pg -o test test.c //程序文件名称 test.c 编译时使用
```

-pg

现在我们可以再次运行生成的结果文件 test，并使用我们前面使用的测试数据。这次我们运行的时候，test 运行的分析数据会被搜集并保存在 'gmon.out' 文件中，我们可以通过运行 'gprof test gmon.out>result' 到一个 result 文件来查看结果。

```
a.exe
```

```
gprof a.exe gmon.out>result
```

或

```
gprof a.exe(直接在命令行下查看)
```


2.5 写出用 Valgrind 进行性能分析的方法 (5 分)

Valgrind 的使用非常简单，valgrind 命令的格式如下：

```
valgrind [valgrind-options]your-prog [your-prog options]
```

一些常用的选项如下：

`-h --help`

显示帮助信息。

`--version`

显示 valgrind 内核的版本，每个工具都有各自的版本。

`-q --quiet`

安静地运行，只打印错误信息。

`-v --verbose`

打印更详细的信息。

`--tool= [default: memcheck]`

最常用的选项。运行 valgrind 中名为 toolname 的工具。如果省略工具名，默认运行 memcheck。

`--db-attach= [default: no]`

绑定到调试器上，便于调试错误。

第 3 章 Cache 模拟与测试

3.1 Cache 模拟器设计

提交 csim.c

程序设计思想：

1.getopt ()

如果函数声明丢失，则在 Unix 命令行上自动解析元素，通常在循环中调用以检索参数，它的返回值存储在局部变量中，

当 getopt () 返回 -1 时，没有更多的选项。（百度翻译的）一句话就是它是解析你的那个命令行的。

2.fscanf ()

读入测试文件的，自带有调用就好了。具体使用方式如下：（写的时候粘贴复制一下就好了。）

```
FILE * pFile; //pointer to FILE object

pFile = fopen ("tracefile.txt","r"); //open file for reading

char identifier;

unsigned address;

int size;

// Reading lines like " M 20,1" or "L 19,3"

while(fscanf(pFile," %c %x,%d", &identifier, &address, &size)>0){

// Do stuff

}

fclose(pFile); //remember to close file when done
```

3.Malloc/free

分配和释放内存空间的函数。

具体用法如下：

```
Some_pointer_you_malloced = malloc(sizeof(int));
```

```
Free(some_pointer_you_malloced);
```

分配了内存用完的时候记得释放，还有不要释放没有分配的内存。

三、根据（一）可以写出下面结构体：（基于 c 语言）

定义行的属性：

```
typedef struct{  
    int valid;      //有效位  
  
    int tag;        //标识位  
  
    int LruNumber;  //牺牲行的时候要用的，具体上面说了。  
} Line;
```

定义组的属性：

```
typedef struct{  
    Line* lines;    //用于存储一组中包含的行  
} Set;
```

定义 cache 的属性：

```
typedef struct {  
    int set_num;    //组数  
  
    int line_num;  //行数  
  
    Set* sets;     //cache 的空间，模拟 cache
```

```
} Sim_Cache;
```

测试用例 1 的输出截图 (5 分):

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim -s 1 -E 1
-b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim-ref -s 1
-E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
```

测试用例 2 的输出截图 (5 分):

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim -s 4 -E 2
-b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim-ref -s 4
-E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
```

测试用例 3 的输出截图 (5 分):

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim -s 2 -E 1
-b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim-ref -s 2
-E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
```

测试用例 4 的输出截图 (5 分):

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim -s 2 -E 1
-b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim-ref -s 2
-E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
```

测试用例 5 的输出截图 (5 分):

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim -s 2 -E 2
-b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim-ref -s 2
-E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
```

测试用例 6 的输出截图 (5 分):

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim -s 2 -E 4
-b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim-ref -s 2
-E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
```

测试用例 7 的输出截图 (5 分):

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim -s 5 -E 1
-b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim-ref -s 5
-E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
```

测试用例 8 的输出截图 (10 分):

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim -s 5 -E 1
-b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./csim-ref -s 5
-E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
```

注: 每个用例的每一指标 5 分 (最后一个用例 10) ——与参考 csim-ref 模拟器输出指标相同则判为正确

3.2 矩阵转置设计

提交 trans.c

程序设计思想:

程序的目的是优化矩阵转置算法, 即使用分块的方法, 让 Cache 的 miss 次数更少。题目的要求, 缓存采用的是直接映射高速缓存, $s = 5$, $b = 5$, $E = 1$ 。即 32 个组, 每个组共 32 个字节, 可以装入 8 个 int 变量。

.32x32

首先对矩阵进行分块处理。矩阵分块的目的在于将大的、不能完全加载进入缓存的大矩阵分块成小的、可以完全加载进入缓存的小矩阵块来处理。小矩阵块具有良好的局部性, 性能显著增加。

为了完全利用每一个缓存块 (32 个字节) 采用 8×8 分块。我们知道, 当进行对角线的引用时, 可能会发生缓存的冲突不命中问题, 所以优先处理对角线。同时对于 A 矩阵按列优先 (不连续读)。通过优先处理对角线上的元素, 保证了 B 矩阵的第 a 行被载入缓存中, 接下来对于 A 矩阵的列优先处理保证了 B 矩阵的第 a 行缓存被充分利用。

64x64

对于 64×64 的矩阵，其同一列相邻行的元素之间的地址间隔为 $0x100$ 。对于 8×8 的矩阵分块而言，其第 1、2、3、4 行的元素会和第 5、6、7、8 行的元素占用相同的高速缓存组，出现冲突不命中现象。但是用 4×4 的矩阵分块又无法充分利用每一个高速缓存行，所以依然用 8×8 分组。

采用如下策略：

- 1.按行加载矩阵 A，并且将其存入矩阵 B。依次执行 4 次，直到整个分块的上半部分处理完毕。
- 2.对于分块的下半部分的第一行，先将矩阵 B 的右上分块的 4 个元素载入临时变量，然后从矩阵 A 中的左下分块读取第一列并转置进入矩阵右上分块的第一行，然后将读出的 4 个元素存入矩阵 B 右下分块的第一行，最后再将矩阵 A 右下分块第一列转置送入矩阵 B 右下分块的第一行。依次处理完下半部分的所有行。

61×67

由于 61×67 的矩阵不是方阵，所以也不用考虑处理对角线问题，经过尝试换用不同的边长分块即可。发现 16×16 的分块就是满足 $\text{miss} < 2000$ 的。

32×32 (10 分)：运行结果截图

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./test-trans -M
32 -N 32

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1766, misses:287, evictions:255

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151

Summary for official submission (func 0): correctness=1 misses=287
TEST_TRANS_RESULTS=1:287
```

64×64 (10 分): 运行结果截图

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./test-trans -M 61  
-N 67  
  
Function 0 (2 total)  
Step 1: Validating and generating memory traces  
Step 2: Evaluating performance (s=5, E=1, b=5)  
func 0 (Transpose submission): hits:6332, misses:1847, evictions:1815  
  
Function 1 (2 total)  
Step 1: Validating and generating memory traces  
Error: Program timed out.  
TEST_TRANS_RESULTS=0:0
```

61×67 (20 分): 运行结果截图

```
l170300901@l170300901-VirtualBox:~/share/lab6/cachelab-handout$ ./test-trans -M 61  
-N 67  
  
Function 0 (2 total)  
Step 1: Validating and generating memory traces  
Step 2: Evaluating performance (s=5, E=1, b=5)  
func 0 (Transpose submission): hits:6332, misses:1847, evictions:1815  
  
Function 1 (2 total)  
Step 1: Validating and generating memory traces  
Error: Program timed out.  
TEST_TRANS_RESULTS=0:0
```

第 4 章 总结

4.1 请总结本次实验的收获

我是韩国留学生. 因此用中文学习这个课程是非常困难的. 但是很多中国朋友告诉我和帮助我了解了课堂内容 所以我提高了很多汉语水平,也了解了很多 linux. 对我来说,这似乎是一次很好的经验,真的很幸福

4.2 请给出对本次实验内容的建议

我是留学生 。请多多关照。 谢谢老师

注：本章为酌情加分项。

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

- [1] 林来兴. 空间控制技术[M]. 北京: 中国宇航出版社, 1992: 25-42.
- [2] 辛希孟. 信息技术与信息服务国际研讨会论文集: A 集[C]. 北京: 中国科学出版社, 1999.
- [3] 赵耀东. 新时代的工业工程师[M/OL]. 台北: 天下文化出版社, 1998 [1998-09-26]. <http://www.ie.nthu.edu.tw/info/ie.newie.htm> (Big5) .
- [4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨: 哈尔滨工业大学, 1992: 8-13.
- [5] KANAMORI H. Shaking Without Quaking[J]. Science, 1998, 279 (5359): 2063-2064.
- [6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science , 1998 , 281 : 331-332[1998-09-23]. <http://www.sciencemag.org/cgi/collection/anatmorp>.