哈爾濱工業大學

实验报告

实 验（三）

题　　目　优化

专　　业　计算机类

学　　号　120L022115

班　　级　2003007

学　　生　王炳轩

指 导 教 师　吴锐

实 验 地 点　G709

实 验 日 期　2022-04-15

计算学部

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 ..... - 4 -
2.1 程序优化的十大方法（5 分） ..... - 4 -
2.2 性能优化的方法概述（5 分） ..... - 4 -
2.3 Linux 下性能测试的方法（5 分） ..... - 5 -
2.4 Windows 下性能测试的方法（5 分） ..... - 5 -
第 3 章 性能优化的方法 ..... - 6 -
第 4 章 性能优化实践 ..... - 7 -
第 5 章 总结 ..... - 13 -
5.1 请总结本次实验的收获 ..... - 13 -
5.2 请给出对本次实验内容的建议 ..... - 13 -
参考文献 ..... - 14 -

第 1 章 实验基本信息

1.1 实验目的

理解程序优化的 10 个维度。

熟练利用工具进行程序的性能评价、瓶颈定位。

掌握多种程序性能优化的方法。

熟练应用软件、硬件等底层技术优化程序性能。

1.2 实验环境与工具

1.2.1 硬件环境

Surface Go 3：x64、Pentium G6500Y @ 1.1GHz、16GB RAM、128GB SSD。

1.2.2 软件环境

Windows 11、Windows Subsystem for Linux、Ubuntu 20.04

1.2.3 开发工具

DevCpp、gcc、vim、edb、gdb

1.3 实验预习

上实验课前，必须认真预习实验指导书。

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。

程序优化的十个维度。

如何编写面向编译器、CPU、存储器友好的程序。

性能测试方法：time、RDTSC、clock

性能测试准确性的文献查找：流水线、超线程、超标量、向量、多核、GPU、多级 CACHE、编译优化 Ox、多进程、多线程等多种因素对程序性能的综合影响。

第 2 章 实验预习

总分 20 分

2.1 程序优化的十大方法（5 分）

代码移动、复杂指令简化、公共子表达式、面向超标量 CPU 的优化、分离的循环展开、MMX/SSE/AVR、CMOVxx 等指令代替 test/cmp+jxx、嵌入式汇编、面向编译器的优化、面向存储器的优化：重新排列提高空间局部性分块提高时间局部性、多进程优化、文件访问优化、并行计算、网络计算优化、GPU 编程、算法优化。

2.2 性能优化的方法概述（5 分）

1.一般有用的优化

代码移动

复杂指令简化

公共子表达式

2.面向编译器的优化：障碍

函数副作用

内存别名

3.面向超标量 CPU 的优化

流水线、超线程、多功能部件、分支预测投机执行、乱序执行、多核：分离的循环展开！

只有保持能够执行该操作的所有功能单元的流水线都是满的，程序才能达到这个操作的吞吐量界限

4.面向向量 CPU 的优化：MMX/SSE/AVR

5. CMOVxx 等指令

代替 test/cmp+jxx

6.嵌入式汇编

7.面向编译器的优化

Ox:0 1 2 3 g

8.面向存储器的优化：Cache 无处不在

重新排列提高空间局部性

分块提高时间局部性

9.内存作为逻辑磁盘：内存够用的前提下。

10.多进程优化

fork，每个进程负责各自的工作任务，通过mmap共享内存或磁盘等进行交互。

11.文件访问优化：带 Cache 的文件访问

12.并行计算：多线程优化：第 12 章

13.网络计算优化：第 11 章、分布式计算、云计算

14.GPU 编程、算法优化

15.超级计算

2.3 Linux 下性能测试的方法（5 分）

Linux 下 Oprofile 等工具（gprof、google-perftools）

https://blog.csdn.net/Blaider/article/details/7730792 用 OProfile 彻底了解性能

https://www.cnblogs.com/jkkkk/p/6520381.html《Linux 调优工具 oprofile 的演示分析》

https://www.cnblogs.com/MYSQLZOUQI/p/5426689.html

Linux 下的 valgrind： callgrind/Cachegrind

https://www.jianshu.com/p/1e423e3f5ed5 将 Cachegrind 和 Callgrind 用于性能调优

https://blog.csdn.net/u010168781/article/details/84303954

2.4 Windows 下性能测试的方法（5 分）

Windows 下 VS，本身就有性能评测的组件

调试：性能探测器：CPU、RAM、GPU

第 3 章 性能优化的方法

总分 20 分

逐条论述性能优化方法名称、原理、实现方案（至少 10 条）

3.1 代码移动：把循环体内部多次执行与单次执行的效果相同的无效代码挪出，减少执行次数达到同样的效果。
3.2 复杂指令简化：如除 4 变为右移 2 位、乘 8 变为左移 3 位等，有些 CPU 会计算的更快——使用低开销的指令代替高开销的指令。
3.3 公共子表达式：对于多次使用的值表达式，可以先行计算，然后存储在变量中，下次使用直接调用，而不必重新计算。
3.4 n*n 循环展开：减少循环次数，增加每次循环所做的事情，达到同样的效果，节省时间开销。保持能够执行该操作的所有功能单元的流水线都是满的，达到这个操作的吞吐量界限。
3.5 重新排列提高空间局部性：对多维数组进行步长为 1 的访问/存储，利用局部性原理，提高访存（cache）效率，有利于提高执行效率。
3.6 分块提高时间局部性：对于很大的内存区域计算时有重复的访问某一小块区域，则可以将大区域分块计算，优先计算使用小区域的数据，使其一直在高速缓存中，提高访问的效率。
3.7 多进程优化：fork，每个进程负责各自的工作任务，通过 mmap 共享内存或磁盘等进行交互，提高执行效率。
3.8 网络计算优化：分布式计算、云计算：将需要进行大量计算的项目数据分割成小块，由多台计算机分别计算，再上传运算结果后统一合并得出数据结论。
3.9 文件访问优化：带 Cache 的文件访问。对文件读写并不是每次都进行磁盘 IO，而是将对应的磁盘文件缓存到内存上，之后对该文件的操作实际上也是对内存的读写。
3.10 面向向量 CPU 的优化：MMX/SSE/AVR。使用新一代的专用指令与硬件资源来达到同样效果，新指令使用更新的专用硬件资源，具有更高的计算速度。

第 4 章 性能优化实践

总分 60 分

4.1 原始程序及说明（10 分）

说明程序的功能、流程，分析程序可能瓶颈

功能：一个图像处理程序。实现图像的平滑，其图像分辨率为 1920*1080，每一点颜色值为 64bits，用 long img[1920][1080]存储屏幕上的所有点颜色值，颜色值可从 0 依行列递增，或真实图像。

平滑算法为：任一点的颜色值为其上下左右 4 个点颜色的平均值，即：

img[i][j] = (img[i-1][j] + img[i+1][j] +img[i][j-1] + img[i][j+1]) /4。

流程：从文件读入图片、执行平滑算法、写出到文件。

读入文件并记录时间。

```
FILE* f = fopen(".\\IMAGE.64BMP","rb");
time1 = clock();
for(int i=1;i<HEIGHT+1;i++){
    fread(&img[i][1],sizeof(IMGTYPE),WIDTH,f);
}
fclose(f);
```

处理边缘

```
for(int i=0;i<WIDTH+2;i++){
    line[last][i] = img[0][i] = img[1][i];
    img[HEIGHT+1][i]= img[HEIGHT][i];
}

for(int i=0;i<HEIGHT+2;i++){
    img[i][0] = img[i][1];
    img[i][WIDTH+1]= img[i][WIDTH];
}
```

执行算法（使用 2 行的缓存）

```
for(int i=1;i<HEIGHT+1;i++){
    for(int j=1;j<WIDTH+1;j++){
        line[curr][j] = (img[i][j-1]+img[i][j+1]+img[i-1][j]+img[i+1][j])/4;
    }
    for(int j=1;j<WIDTH+1;j++){
        img[i-1][j] = line[last][j];
        line[last][j] = line[curr][j];
    }

}
```

写出到文件并记录时间、写出 LOG

```
f = fopen(".\\IMAGE_Blur.64BMP","wb");
for(int i=1;i<HEIGHT+1;i++){
    //for(int j=1;j<HEIGHT+1;j++){
        fwrite(&img[i][1],sizeof(IMGTYPE),WIDTH,f);
    //}
}
time2 = clock();
fclose(f);
FILE* flog = fopen(".\\log.txt","a+");
fprintf(flog,"BLUR \t START:%lld\t END:%lld\n",time1,time2)
fclose(flog);
flog = fopen(".\\log_BL.txt","a+");
fprintf(flog,"%lld\n",time2-time1);
fclose(flog);
```

瓶颈：文件读写效率、算法执行计算效率。

4.2 优化后的程序及说明（20 分）

至少包含面向 CPU、Cache 的两种优化策略（20 分），额外每增加 1 种优化方法加 5 分至第 4 章满分。

优化使用了一下七种方法：代码移动、复杂指令简化、公共子表达式、4*4 循环展开、重新排列、文件访问优化。

代码移动：将部分循环中申请的变量移动到外部，将一些循环合并。

复杂指令简化：将除 4 变为右移 2 位。

公共子表达式：有一些循环体内的重复计算，提前存储于变量，直接调用。

面向 CPU 的 4*4 循环展开：将步长为 1 的循环变为步长为 4 的循环，减少循环次数。

```
for(;i<max;i++){
    for(int j=1;j<WIDTH;j+=4){
        i1 = i-1;i2 = i+1;j1 = j+1;j2 = j+2;j3=j+3;
        line[id][curr][j] = (img[i][j-1]+img[i][j1]+img[i1][j]+img[i2][j])>>2;
        line[id][curr][j1] = (img[i][j]+img[i][j2]+img[i1][j1]+img[i2][j1])>>2;
        line[id][curr][j2] = (img[i][j1]+img[i][j3]+img[i1][j2]+img[i2][j2])>>2;
        line[id][curr][j3] = (img[i][j2]+img[i][j+4]+img[i1][j3]+img[i2][j3])>>2
        img[i1][j] = line[id][last][j];
        img[i1][j1] = line[id][last][j1];
        img[i1][j2] = line[id][last][j2];
        img[i1][j3] = line[id][last][j3];
    }
    curr = !curr;
    last = !last;
}
```

重新排列：部分代码进行了顺序的调整。

面向 Cache 的文件访问优化：将一行一行读写改为整体输入输出。

多线程：程序使用了 pthread.h 启用双线程进行计算。

```
pthread_t th1,th2;

if(pthread_create(&th1,NULL,calculate,(void*)&a1)!=0) exit(1);
if(pthread_create(&th1,NULL,calculate,(void*)&a2)!=0) exit(2);
pthread_join(th1,NULL);
pthread_join(th2,NULL);
```


writeImage_opt.cpp

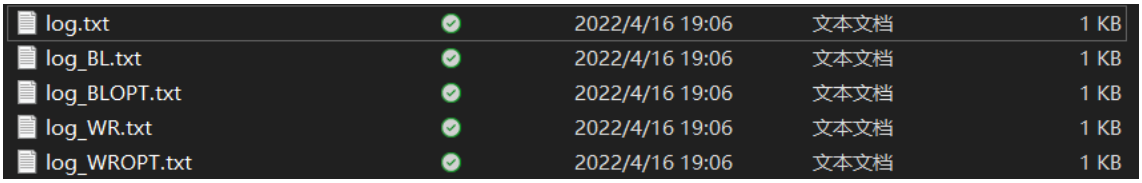

writeImage.cpp


imageBlur_opt.cpp


imageBlur.cpp

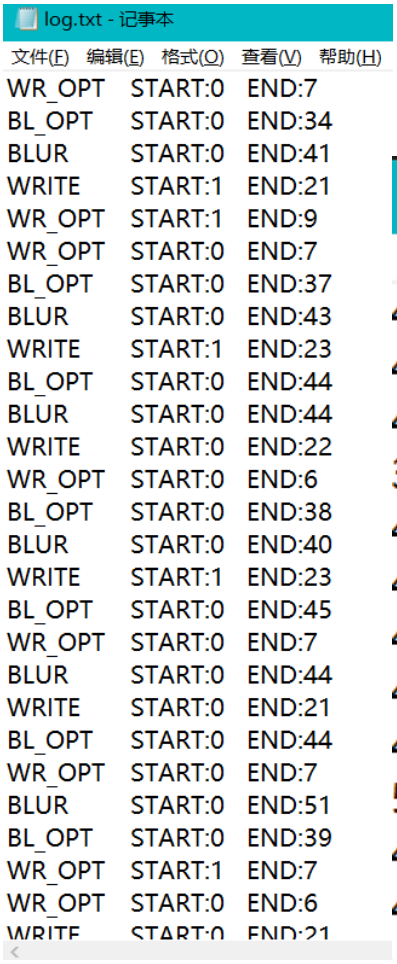
4.3 优化前后的性能测试（10 分）

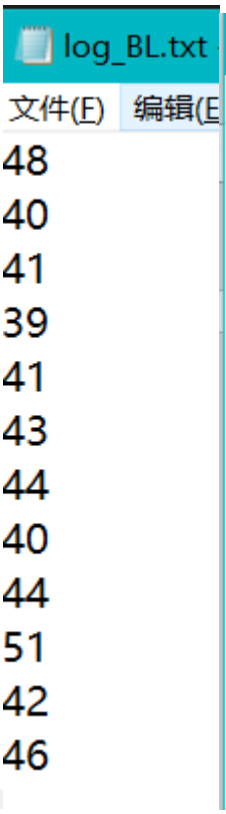
测试方法：使用自建的 clock 函数计算。在读入文件之前记录 start，关闭文件之后记录 end，计算差值（单位：ms）。分为两个情况分别计算：WRITE、BLUR 分别表示创建随机数图像文件和读取图像并执行算法并写出的执行时间。

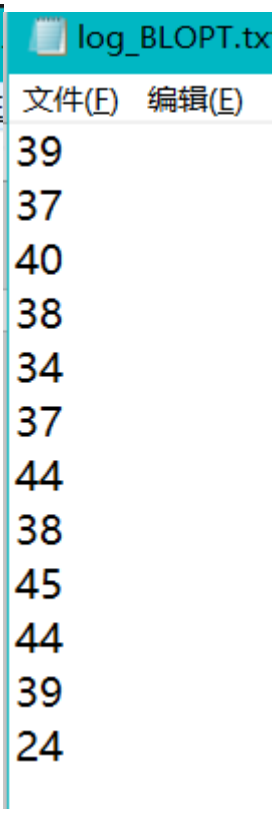


log.txt	2022/4/16 19:06	文本文档	1 KB
log_BL.txt	2022/4/16 19:06	文本文档	1 KB
log_BLOPT.txt	2022/4/16 19:06	文本文档	1 KB
log_WR.txt	2022/4/16 19:06	文本文档	1 KB
log_WROPT.txt	2022/4/16 19:06	文本文档	1 KB

经过查阅 log.txt 并整理得到了以下结果：







测试结果：

WR_OPT	START:0	END:7
BL_OPT	START:0	END:34
BLUR	START:0	END:41
WRITE	START:1	END:21
WR_OPT	START:1	END:9
WR_OPT	START:0	END:7
BL_OPT	START:0	END:37
BLUR	START:0	END:43
WRITE	START:1	END:23
BL_OPT	START:0	END:44
BLUR	START:0	END:44
WRITE	START:0	END:22
WR_OPT	START:0	END:6
BL_OPT	START:0	END:38
BLUR	START:0	END:40
WRITE	START:1	END:23
BL_OPT	START:0	END:45
WR_OPT	START:0	END:7
BLUR	START:0	END:44

WRITE	START:0	END:21
BL_OPT	START:0	END:44
WR_OPT	START:0	END:7
BLUR	START:0	END:51
BL_OPT	START:0	END:39
WR_OPT	START:1	END:7
WR_OPT	START:0	END:6
WRITE	START:0	END:21
WRITE	START:0	END:21
BLUR	START:0	END:42
BLUR	START:0	END:46
BL_OPT	START:0	END:24

WRITE	**WR_OPT**	**BLUR**	**BL_OPT**
22	**7**	**48**	**26**
22	**6**	**40**	**27**
22	**6**	**41**	**24**
18	**6**	**39**	**26**
20	**7**	**41**	**26**
22	**8**	**43**	**24**
22	**7**	**44**	**24**
22	**6**	**40**	**14**
21	**7**	**44**	**25**
21	**7**	**51**	**27**
21	**6**	**42**	**25**
22	**6**	**46**	**24**

经过优化，创建图像的时间明显减少了 14~16ms，而执行算法+读写的执行时间同样也减少了 20~22ms。

4.4 面向泰山服务器优化后的程序与测试结果（15 分）

经过进一步优化：更多的循环展开等

进行相同的测试，得到如下的结果。


writeImage_opt.cpp


writeImage.cpp


imageBlur_opt.cpp


imageBlur.cpp

单位： μs

WRITE	START:2013	END:28990
WR_OPT	START:9820	END:36718
BLUR	START:1983	END:70210

BL_OPT	START:2090	END:67536
BL_OPT	START:1819	END:62950
BLUR	START:1944	END:74467
BL_OPT	START:1873	END:65757
WR_OPT	START:8167	END:34371
WRITE	START:9004	END:32560
WR_OPT	START:7422	END:35839
WRITE	START:8966	END:32189
WRITE	START:9126	END:33035
BL_OPT	START:1739	END:64355
BLUR	START:3098	END:74902
WRITE	START:9224	END:34019
BLUR	START:1897	END:77056
BL_OPT	START:2066	END:63668
WRITE	START:7745	END:32261
BLUR	START:1939	END:80798
WR_OPT	START:9174	END:38910
BL_OPT	START:1996	END:60827
WR_OPT	START:8926	END:34781
WRITE	START:9112	END:36194
WR_OPT	START:8287	END:35476
BL_OPT	START:1582	END:63049
BLUR	START:1890	END:74699
WRITE	START:4234	END:27014
BL_OPT	START:1913	END:61963
WR_OPT	START:9035	END:35034

WRITE	**WR_OPT**	**BLUR**	**BL_OPT**
26977	**26898**	**68227**	**65446**
23556	**25855**	**72523**	**61131**
23223	**27189**	**71804**	**63884**
23909	**25999**	**75159**	**62616**
24795	**28758**	**78859**	**61602**
24516	**25690**	**72809**	**58831**
27082	**26176**		**61467**
22780	**25110**		**60050**

经过优化，创建图像文件的时间并没有明显减少，而执行算法+读写的执行时间大幅减少了 5~10ms。

4.5 还可以采取的进一步的优化方案（5 分）

多进程计算，分块提高空间局部性，面向向量 CPU 的计算优化。

第 5 章 总结

5.1 请总结本次实验的收获

学习了 C 语言的二进制文件读写。
学会了多种程序优化的方法，提高程序的执行效率。
学会了优化效果、性能检测的查看方法。

5.2 请给出对本次实验内容的建议

建议直接改成 24 位真彩色位图 BMP。

注：本章为酌情加分项。

参考文献

[1] SSE_百度百科 (baidu.com)

[2] (13 条消息) Linux 的文件读写缓存 cache 浅析，以及 read 优化_Tecinno4 的博客-CSDN 博客_cache read

[3] (13 条消息) 分布式计算概念_HeavenlyDragon 的博客-CSDN 博客_分布式计算