哈爾濱工業大學

实验报告

实 验（三）

题　　目 优化

专　　业 计算学部

学　　号 120L022109

班　　级 2003007

学　　生 李世轩

指 导 教 师 吴锐

实 验 地 点 G709

实 验 日 期 2022 年 4 月 15 日

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 - 5 -

2.1 程序优化的十大方法（5 分） - 5 -
2.2 性能优化的方法概述（5 分） - 5 -
2.3 LINUX 下性能测试的方法（5 分） - 6 -
2.4 WINDOWS 下性能测试的方法（5 分） - 6 -

第 3 章 性能优化的方法 - 7 -

第 4 章 性能优化实践 - 8 -

第 5 章 总结 - 19 -

5.1 请总结本次实验的收获 - 19 -

参考文献 - 20 -

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

X64 CPU；2.3GHz；16G RAM；512GHD Disk

1.2.2 软件环境

Windows11 64 位；VirtualBox;Ubuntu 20.04 LTS 64 位；

1.2.3 开发工具

CLion 64 位以上；CodeBlocks 64 位；vi/vim/gedit+gcc

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。

请写出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）

更快（本课程重点!）
更省（存储空间、运行空间）
更美（UI 交互）
更正确（本课程重点！各种条件下）
更可靠
可移植
更强大（功能）
更方便（使用）
更范（格式符合编程规范、接口规范 ）
更易懂（能读明白、有注释、模块化）

2.2 性能优化的方法概述（5 分）

1.一般有用的优化
2.面向编译器的优化：障碍
3.面向超标量 CPU 的优化
4.面向向量 CPU 的优化：MMX/SSE/AVR
5. CMOVxx 等指令
6. 嵌入式汇编
7.面向编译器的优化
8.面向存储器的优化：Cache 无处不在
9.内存作为逻辑磁盘：内存够用的前提下。
10.多进程优化
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）

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

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。

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#define Data long long
#define N 1920
#define M 1080
//typedef long long Data;

int getMiTime() {
    SYSTEMTIME currentTime;
    GetSystemTime(&currentTime);

    return (3600 * currentTime.wHour + 60 * currentTime.wMinute + currentTime.wSecond) * 1000 + currentTime.wMilliseconds;
}
int main()
{
    if(sizeof(Data)!=8){
        printf("DataClass Error\n");
        exit(-1);
    }
    //生成随机数组
    Data *img = (Data *)malloc(M * N * sizeof(Data));
    srand((unsigned)time(NULL));
    for(int i = 0;i<N;i++){
        for(int j = 0;j<M;j++){
            img[i * M +j]=rand();

        }
    }

    Data *line1 =(Data*) malloc(M*sizeof(Data));
    Data *line2 =(Data*) malloc(M*sizeof(Data));

    //记录开始时间
    int t_start = getMiTime();
```

```
        for(int i = 0;i<10;i++) {
            //算出 line1
            for(int j = 1;j<M-1;j++){
                line1[j] = (img[(1-1)*M+j] + img[(1+1)*M+j] + img[1*M+j-1] +
img[1*M+j+1])/4;
            }
            int k;
            for(k = 2;k<N-1;k++){
                //算出 line2
                for(int j = 1;j<M-1;j++){
                    line2[j] = (img[(k-1)*M+j] + img[(k+1)*M+j] + img[k*M+j-1]
+ img[k*M+j+1])/4;
                }
                //改变原值
                for(int j = 1;j<M-1;j++){
                    img[(k-1)*M+j] = line1[j];
                }
                //交换 line1，line2
                Data *temp = line1;
                line1 = line2;
                line2 = temp;
            }
            //改变原值,最后一行在双重循环中未改变
            for(int j = 1;j<M-1;j++){
                img[(k-1)*M+j] = line1[j];
            }
        }

        //记录结束时间
        int t_end = getMiTime();
        printf("Using time: %d ms",(t_end-t_start)/10);
        free(img);
        free(line1);
        free(line2);
    }
```

可能瓶颈

每次都计算(k-1)*M+j 等下标，多次进行乘法运算，占用时间

计算时采用除法“/”运算，占用大量时间

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。

面向 CPU 优化

将下标计算改为使用变量 n = k * M +j;然后 n-M，减少乘法运算数量

除法运算改为移位运算，大幅度减少运算时间
循环展开，减少了不直接有助于有助于程序程序结果的操作的数量

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#define Data long long
#define N 1920
#define M 1080
//typedef long long Data;

int getMiTime() {
    SYSTEMTIME currentTime;
    GetSystemTime(&currentTime);

    return (3600 * currentTime.wHour + 60 * currentTime.wMinute + currentTime.wSecond) * 1000 + currentTime.wMilliseconds;
}
int main()
{
    if(sizeof(Data)!=8){
        printf("DataClass Error\n");
        exit(-1);
    }
    //生成随机数组
    Data *img = (Data *)malloc(M * N * sizeof(Data));
    srand((unsigned)time(NULL));
    for(int i = 0;i<N;i++){
        for(int j = 0;j<M;j++){
            img[i * M +j]=rand();

        }
    }

    Data *line1 =(Data*) malloc(M*sizeof(Data));
    Data *line2 =(Data*) malloc(M*sizeof(Data));

    //记录开始时间
    int t_start = getMiTime();
    int n;
    for(int i = 0;i<10;i++) {
        //算出 line1
        for(int j = 1;j<M-1;j++){
            n=1*M+j;
            line1[j] = ((img[n-M] + img[n+M]) + (img[n-1] + img[n+1]))>>2;
        }
```

```
            int k;
            for(k = 2;k<N-1;k++){
                //算出 line2
                for(int j = 1;j<M-1;j+=4){
                    n = k * M +j;
                    line2[j] = ((img[n-M] + img[n+M]) + (img[n-1] + img[n+1]))>>2;
                    line2[j+1] = ((img[n-M+1] + img[n+M+1]) + (img[n] + img[n+2]))>>2;
                    line2[j+2] = ((img[n-M+2] + img[n+M+2]) + (img[n+1] + img[n+3]))>>2;
                    line2[j+3] = ((img[n-M+3] + img[n+M+3]) + (img[n+2] + img[n+4]))>>2;
                }
                //改变原值
                for(int j = 1;j<M-1;j++){
                    img[(k-1)*M+j] = line1[j];
                }
                //交换 line1，line2
                Data *temp = line1;
                line1 = line2;
                line2 = temp;
            }
            //改变原值,最后一行在双重循环中未改变
            for(int j = 1;j<M-1;j++){
                img[(k-1)*M+j] = line1[j];
            }
        }

        //记录结束时间
        int t_end = getMiTime();
        printf("Using time: %d ms",(t_end-t_start)/10);
        free(img);
        free(line1);
        free(line2);
    }
```

面向 Cache 优化

对 img 进行分块处理，使最内层循环的工作集变小，使其放进缓存中，降低不命中率，减少程序运行时间

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <time.h>
    #include <windows.h>
    #define Data long long
    #define N 1920
```

```
#define M 1080
//typedef long long Data;

int getMiTime() {
    SYSTEMTIME currentTime;
    GetSystemTime(&currentTime);

    return (3600 * currentTime.wHour + 60 * currentTime.wMinute + currentTime.wSecond) * 1000 + currentTime.wMilliseconds;
}
int main()
{
    if(sizeof(Data)!=8){
        printf("DataClass Error\n");
        exit(-1);
    }
    //生成随机数组
    Data *img = (Data *)malloc(M * N * sizeof(Data));
    srand((unsigned)time(NULL));
    for(int i = 0;i<N;i++){
        for(int j = 0;j<M;j++){
            img[i * M +j]=rand();

        }
    }

    Data *line1 =(Data*) malloc(M*sizeof(Data));
    Data *line2 =(Data*) malloc(M*sizeof(Data));

    //记录开始时间
    int t_start = getMiTime();
    int n;
    //分块
    int q=M/4;
    for(int i = 0;i<10;i++) {
        for(int p =1;p<=M;p+=q){
            //算出 line1
            for(int j = p;j<M-1&&j<(p+q);j++){
                n=1*M+j;
                line1[j] = ((img[n-M] + img[n+M]) + (img[n-1] + img[n+1]))>>2;
            }
            int k;
            for(k = 2;k<N-1;k++){
                //算出 line2
                for(int j = p;j<M-1&&j<(p+q);j++){
```

```
                    n = k * M +j;
                    line2[j] = ((img[n-M] + img[n+M]) + (img[n-1] + img[n+1]))>>2;
                }
                //改变原值
                for(int j = p;j<M-1&&j<(p+q);j++){
                    img[(k-1)*M+j] = line1[j];
                }
                //交换 line1，line2
                Data *temp = line1;
                line1 = line2;
                line2 = temp;
            }
            //改变原值,最后一行在双重循环中未改变
            for(int j = p;j<M-1&&j<(p+M/3);j++){
                img[(k-1)*M+j] = line1[j];
            }

        }

    }

    //记录结束时间
    int t_end = getMiTime();
    printf("Using time: %d ms",(t_end-t_start)/10);
    free(img);
    free(line1);
    free(line2);
}
```

4.3 优化前后的性能测试（10 分）

测试方法、测试结果

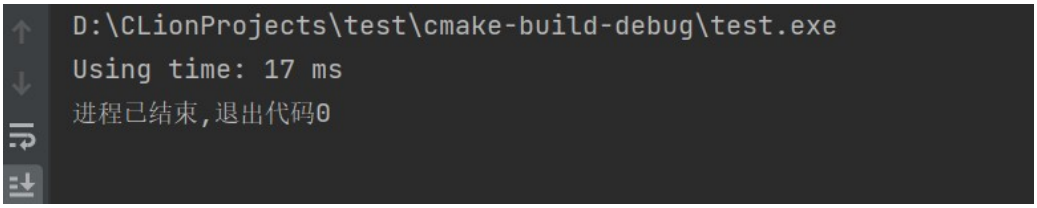
测试方法：

将<windows.h>中的函数 GetSystemTime()封装

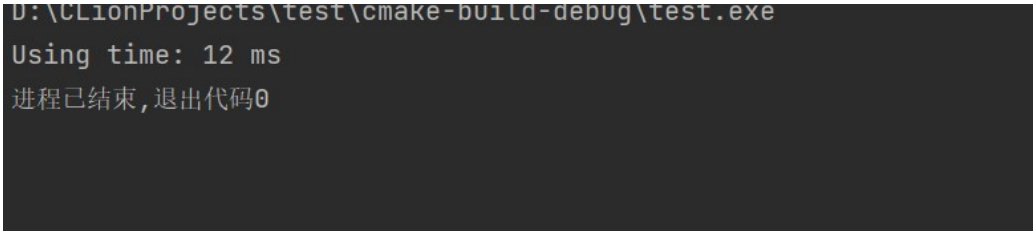
```
int getMiTime() {
    SYSTEMTIME currentTime;
    GetSystemTime(&currentTime);

    return (3600 * currentTime.wHour + 60 * currentTime.wMinute + currentTime.wSecond) * 1000 + currentTime.wMilliseconds;
}
```

然后程序块运行前后分别获取时间，做差后输出。

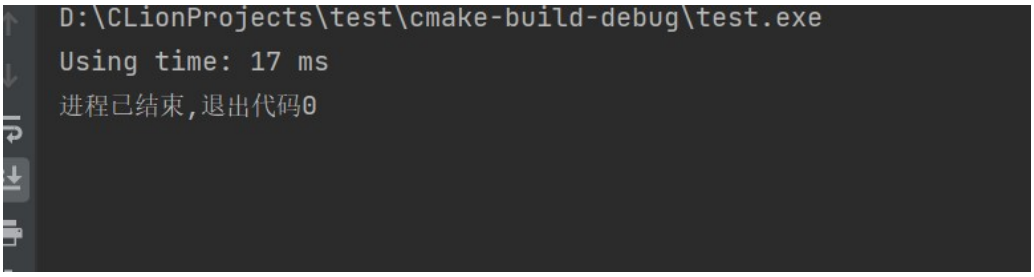
优化前，运行时间为



面向 CPU 的优化



面向 Cache 优化



发现针对 Cache 优化后，程序性能变化不大，分析发现，计算机一级缓存大小足以将内层循环的工作集完全覆盖，再进行分块，对其意义不大

4.4 面向泰山服务器优化后的程序与测试结果（15 分）

面向泰山服务器的程序

将性能测试方法改为使用<sys/timb.h>中的 ftime 函数；并进行封装

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/timeb.h>
#include <unistd.h>
#define Data long long
#define N 1920
#define M 1080
long long getSystemTime(){
    struct timeb t;
    ftime(&t);
    return 1000 *t.time +t.millitm;
}

int main()
{
    if(sizeof(Data)!=8){
        printf("DataClass Error\n");
```

```
            exit(-1);
        }
        //生成随机数组
        Data *img = (Data *)malloc(M * N * sizeof(Data));
        //srand((unsigned)time(NULL));
        for(int i = 0;i<N;i++){
            for(int j = 0;j<M;j++){
                img[i * M +j]=rand();

            }
        }

        Data *line1 =(Data*) malloc(M*sizeof(Data));
        Data *line2 =(Data*) malloc(M*sizeof(Data));

        //记录开始时间
        //int t_start = getMiTime();
        long long start=getSystemTime();
        int n;
        for(int i = 0;i<10;i++) {
            //算出 line1
            for(int j = 1;j<M-1;j++){
                line1[j] = (img[(1-1)*M+j] + img[(1+1)*M+j] + img[1*M+j-1] +
img[1*M+j+1])/4;
            }
            int k;
            for(k = 2;k<N-1;k++){
                //算出 line2
                for(int j = 1;j<M-1;j++){
                    line2[j] = (img[(k-1)*M+j] + img[(k+1)*M+j] + img[k*M+j-1]
+ img[k*M+j+1])/4;
                }
                //改变原值
                for(int j = 1;j<M-1;j++){
                    img[(k-1)*M+j] = line1[j];
                }
                //交换 line1，line2
                Data *temp = line1;
                line1 = line2;
                line2 = temp;
            }
            //改变原值,最后一行在双重循环中未改变
            for(int j = 1;j<M-1;j++){
                img[(k-1)*M+j] = line1[j];
            }
        }
```

```
    //记录结束时间
    //int t_end = getMiTime();
    long long end=getSystemTime();
    printf("Using time: %lld ms\n",(end-start)/10);
    free(img);
    free(line1);
    free(line2);
}
```

面向 CPU 优化后的程序

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/timeb.h>
#include <unistd.h>
#define Data long long
#define N 1920
#define M 1080
long long getSystemTime(){
    struct timeb t;
    ftime(&t);
    return 1000 *t.time +t.millitm;
}

int main()
{
    if(sizeof(Data)!=8){
        printf("DataClass Error\n");
        exit(-1);
    }
    //生成随机数组
    Data *img = (Data *)malloc(M * N * sizeof(Data));
    //srand((unsigned)time(NULL));
    for(int i = 0;i<N;i++){
        for(int j = 0;j<M;j++){
            img[i * M +j]=rand();

        }
    }

    Data *line1 =(Data*) malloc(M*sizeof(Data));
    Data *line2 =(Data*) malloc(M*sizeof(Data));
```

```
        //记录开始时间
        //int t_start = getMiTime();
        long long start=getSystemTime();
        int n;
        for(int i = 0;i<10;i++) {
            //算出 line1
            for(int j = 1;j<M-1;j++){
                n=1*M+j;
                line1[j] = ((img[n-M] + img[n+M]) + (img[n-1] + img[n+1]))>>2;
            }
            int k;
            for(k = 2;k<N-1;k++){
                //算出 line2
                for(int j = 1;j<M-1;j+=4){
                    n = k * M +j;
                    line2[j] = ((img[n-M] + img[n+M]) + (img[n-1] +
img[n+1]))>>2;
                    line2[j+1] = ((img[n-M+1] + img[n+M+1]) + (img[n] +
img[n+2]))>>2;
                    line2[j+2] = ((img[n-M+2] + img[n+M+2]) + (img[n+1] +
img[n+3]))>>2;
                    line2[j+3] = ((img[n-M+3] + img[n+M+3]) + (img[n+2] +
img[n+4]))>>2;
                }
                //改变原值
                for(int j = 1;j<M-1;j++){
                    img[(k-1)*M+j] = line1[j];
                }
                //交换 line1，line2
                Data *temp = line1;
                line1 = line2;
                line2 = temp;
            }
            //改变原值,最后一行在双重循环中未改变
            for(int j = 1;j<M-1;j++){
                img[(k-1)*M+j] = line1[j];
            }
        }


        //记录结束时间
        //int t_end = getMiTime();
        long long end=getSystemTime();
        printf("Using time: %lld ms\n",(end-start)/10);
        free(img);
```

```
        free(line1);
        free(line2);
    }
```

优化结果

优化前

```
stu_120L022109@node210:~$ ./temp
Using time: 25 ms
stu_120L022109@node210:~$ gcc ./temp.c -o temp
```

优化后

```
stu_120L022109@node210:~$ ./temp
Using time: 18 ms
stu_120L022109@node210:~$ ./temp
Using time: 18 ms
```

4.5 还可以采取的进一步的优化方案（5 分）

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！

.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。

第 5 章 总结

5.1 请总结本次实验的收获

-了解程序优化的十大法则
-了解程序优化的一般方法
-对面向 CPU 和面向 Cache 的优化方法初步掌握
-学会性能测试的一些方法

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.