**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(2)**

**实验项目名称： Cache实验**

**学 院： 计算机与软件学院**

**专 业： 计算机科学与技术**

**指 导 教 师： 王 毅**

**报告人： 沈晨玙 学号： 2019092121 班级： 19计科国际**

**实 验 时 间： 2021年 6 月 10 日**

**实验报告提交时间： 2021年 7 月 2 日**

**教务处制**

**一、实验目标：**

了解Cache对系统性能的影响

**二、实验环境：**

1、个人电脑（Intel CPU）

2、Ubuntu Linux 操作系统

**三、实验内容与步骤**

1. 编译并运行程序A，记录相关数据。
2. 不改变矩阵大小时，编译并运行程序B，记录相关数据。
3. 改变矩阵大小，重复1和2两步。
4. 通过以上的实验现象，分析出现这种现象的原因。

**程序A：**

#include <sys/time.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

int main(int argc,char \*argv[])

{

float \*a,\*b,\*c, temp;

long int i, j, k, size, m;

struct timeval time1,time2;

if(argc<2) {

printf("\n\tUsage:%s <Row of square matrix>\n",argv[0]);

exit(-1);

} //if

size = atoi(argv[1]);

m = size\*size;

a = (float\*)malloc(sizeof(float)\*m);

b = (float\*)malloc(sizeof(float)\*m);

c = (float\*)malloc(sizeof(float)\*m);

for(i=0;i<size;i++) {

for(j=0;j<size;j++) {

a[i\*size+j] = (float)(rand()%1000/100.0);

b[i\*size+j] = (float)(rand()%1000/100.0);

}

}

gettimeofday(&time1,NULL);

for(i=0;i<size;i++) {

for(j=0;j<size;j++) {

c[i\*size+j] = 0;

for (k=0;k<size;k++)

c[i\*size+j] += a[i\*size+k]\*b[k\*size+j];

}

}

gettimeofday(&time2,NULL);

time2.tv\_sec-=time1.tv\_sec;

time2.tv\_usec-=time1.tv\_usec;

if (time2.tv\_usec<0L) {

time2.tv\_usec+=1000000L;

time2.tv\_sec-=1;

}

printf("Executiontime=%ld.%06ld seconds\n",time2.tv\_sec,time2.tv\_usec);

return(0);

}//main

**程序B：**

#include <sys/time.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

int main(int argc,char \*argv[])

{

float \*a,\*b,\*c, temp;

long int i, j, k, size, m;

struct timeval time1,time2;

if(argc<2) {

printf("\n\tUsage:%s <Row of square matrix>\n",argv[0]);

exit(-1);

} //if

size = atoi(argv[1]);

m = size\*size;

a = (float\*)malloc(sizeof(float)\*m);

b = (float\*)malloc(sizeof(float)\*m);

c = (float\*)malloc(sizeof(float)\*m);

for(i=0;i<size;i++) {

for(j=0;j<size;j++) {

a[i\*size+j] = (float)(rand()%1000/100.0);

c[i\*size+j] = (float)(rand()%1000/100.0);

}

}

gettimeofday(&time1,NULL);

for(i=0;i<size;i++) {

for(j=0;j<size;j++) {

b[i\*size+j] = c[j\*size+i];

}

}

for(i=0;i<size;i++) {

for(j=0;j<size;j++) {

c[i\*size+j] = 0;

for (k=0;k<size;k++)

c[i\*size+j] += a[i\*size+k]\*b[j\*size+k];

}

}

gettimeofday(&time2,NULL);

time2.tv\_sec-=time1.tv\_sec;

time2.tv\_usec-=time1.tv\_usec;

if (time2.tv\_usec<0L) {

time2.tv\_usec+=1000000L;

time2.tv\_sec-=1;

}

printf("Executiontime=%ld.%06ld seconds\n",time2.tv\_sec,time2.tv\_usec);

return(0);

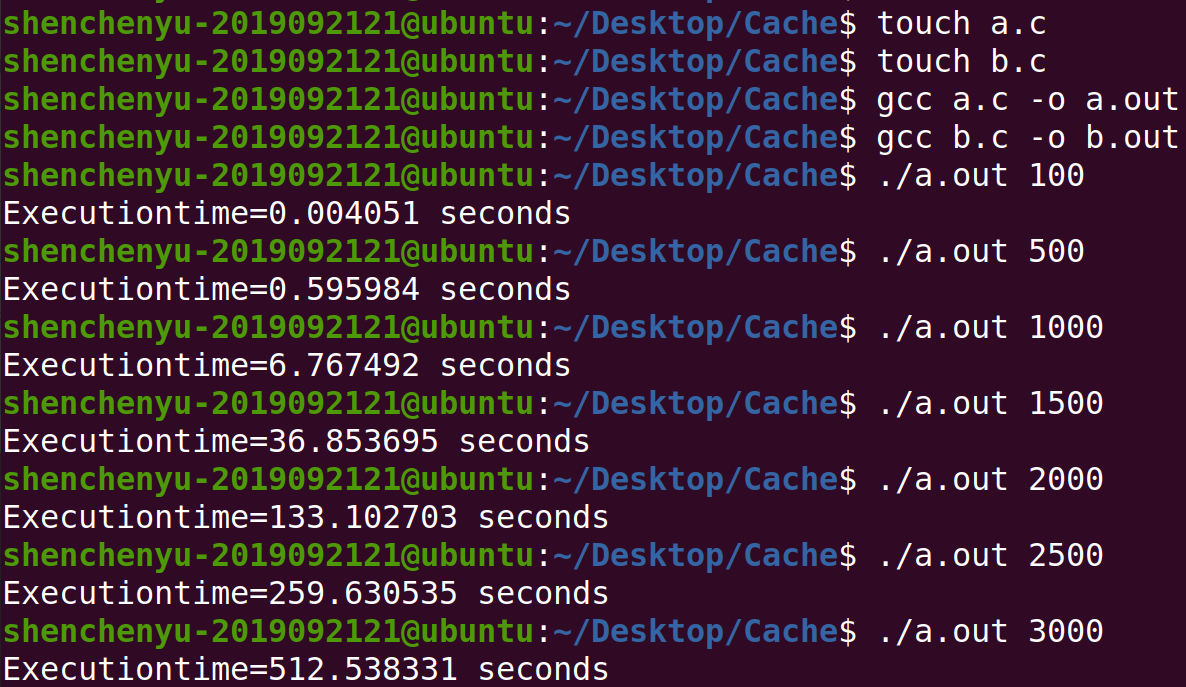
}//main

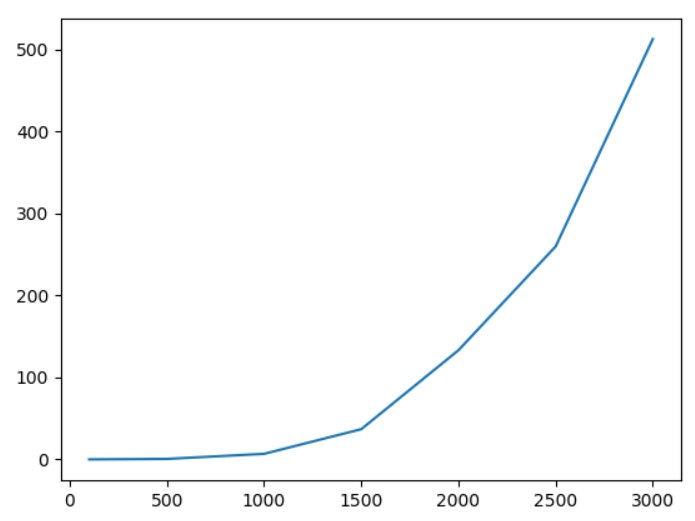
**四、实验结果及分析**

1. 用C语言实现矩阵（方阵）乘积一般算法（程序A），填写下表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 矩阵大小 | 100 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 |
| 一般算法执行时间 | 0.004051 | 0.595984 | 6.767492 | 36.853695 | 133.10270 | 259.6305 | 512.538331 |

分析：





核心代码:

for(i=0;i<size;i++) {

for(j=0;j<size;j++) {

c[i\*size+j] = 0;

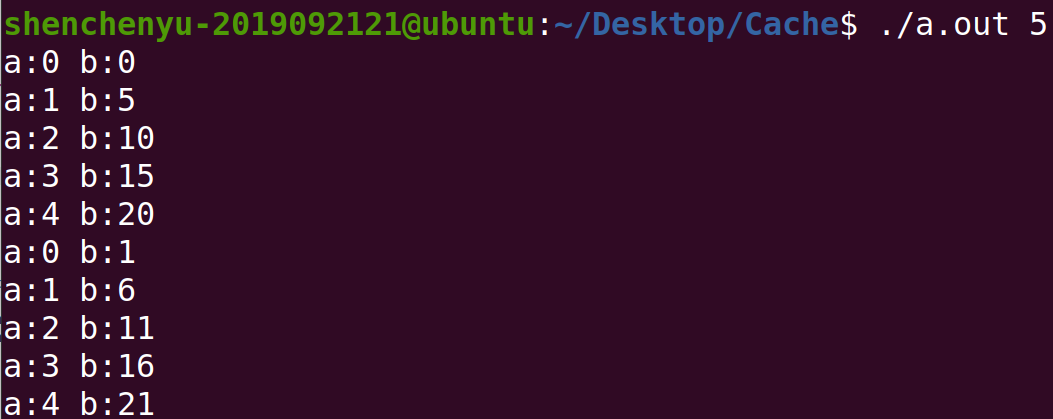
for (k=0;k<size;k++)

c[i\*size+j] += a[i\*size+k]\*b[k\*size+j];

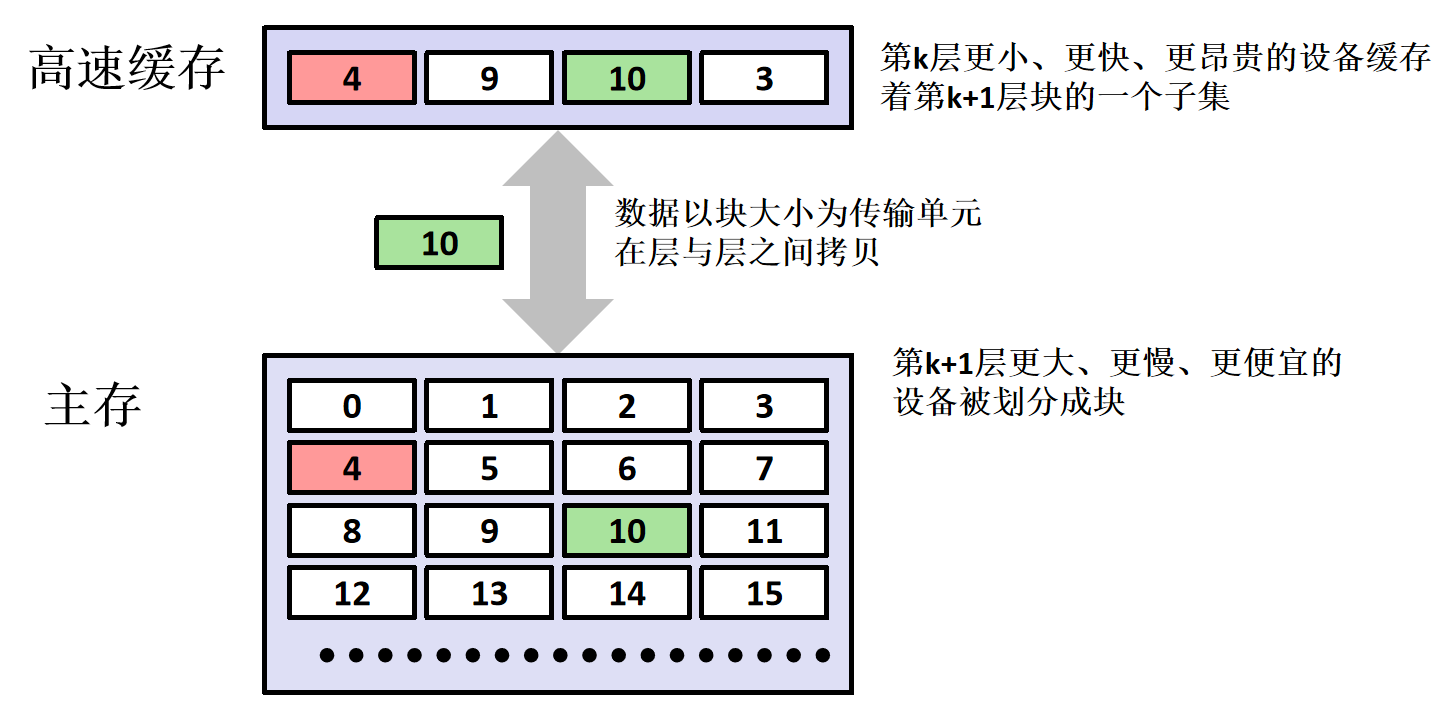
}

}

对于a、b数组的访问部分顺序为：



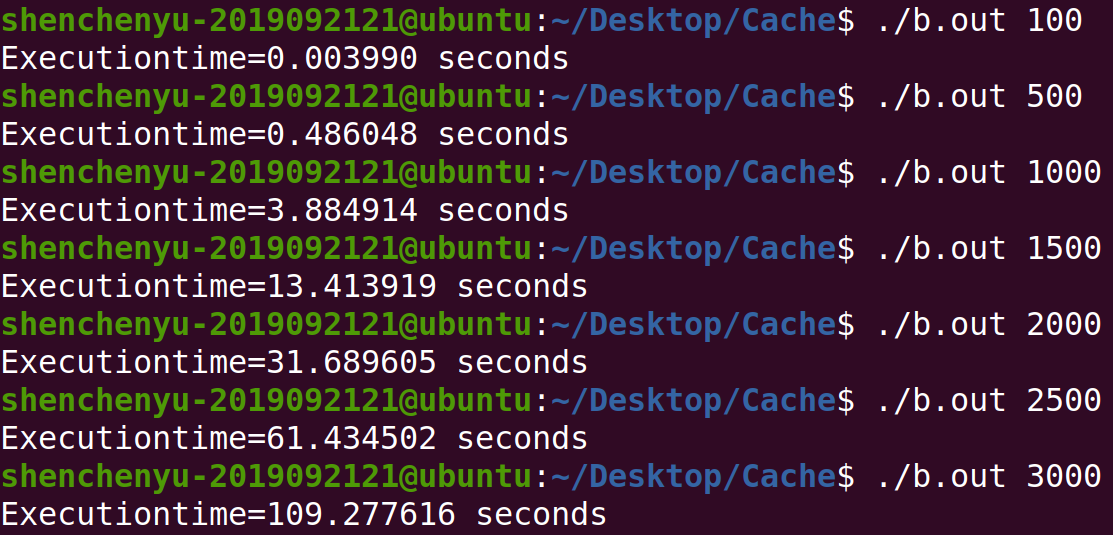
可以看到b数组访问顺序不符合空间局部性。

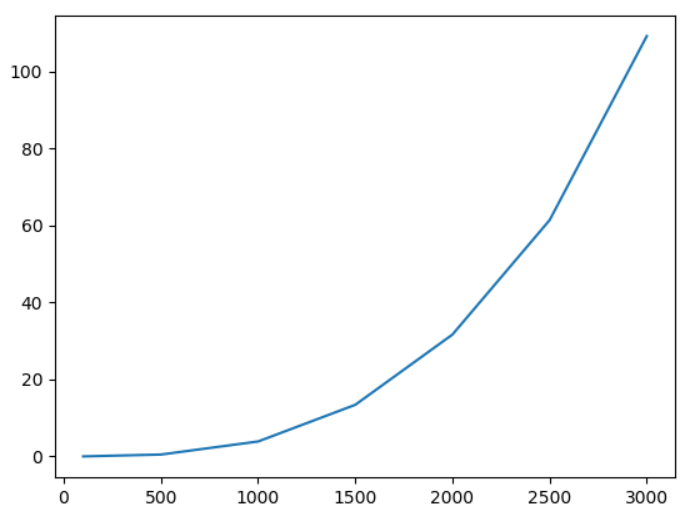


1. 程序B是基于Cache的矩阵（方阵）乘积优化算法，填写下表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 矩阵大小 | 100 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 |
| 优化算法执行时间 | 0.00399 | 0.48604 | 3.8849 | 13.4139 | 31.6896 | 61.4345 | 109.2776 |

分析：





核心代码：

for(i=0;i<size;i++) {

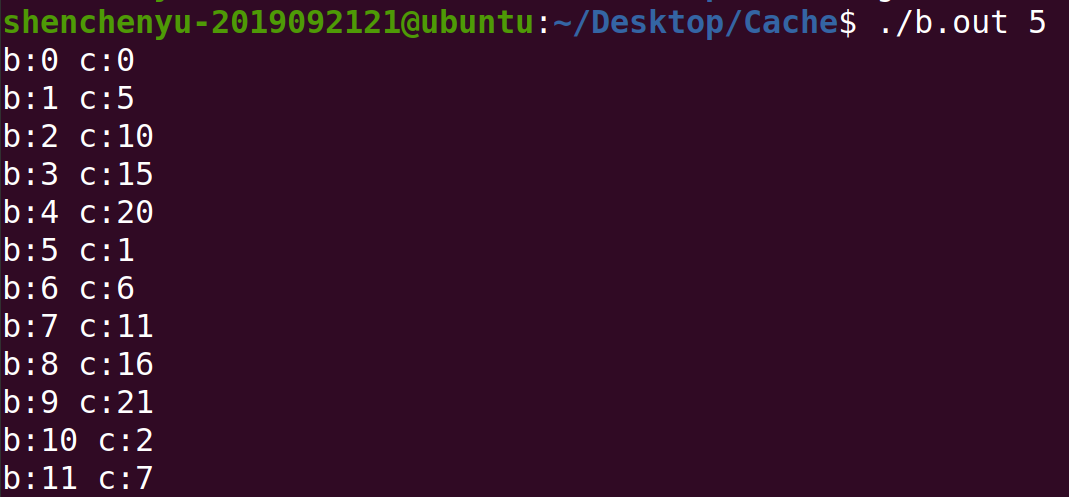
for(j=0;j<size;j++) {

b[i\*size+j] = c[j\*size+i];

}

}

对于b、c数组的访问部分顺序为：



核心代码:

for(i=0;i<size;i++) {

for(j=0;j<size;j++) {

c[i\*size+j] = 0;

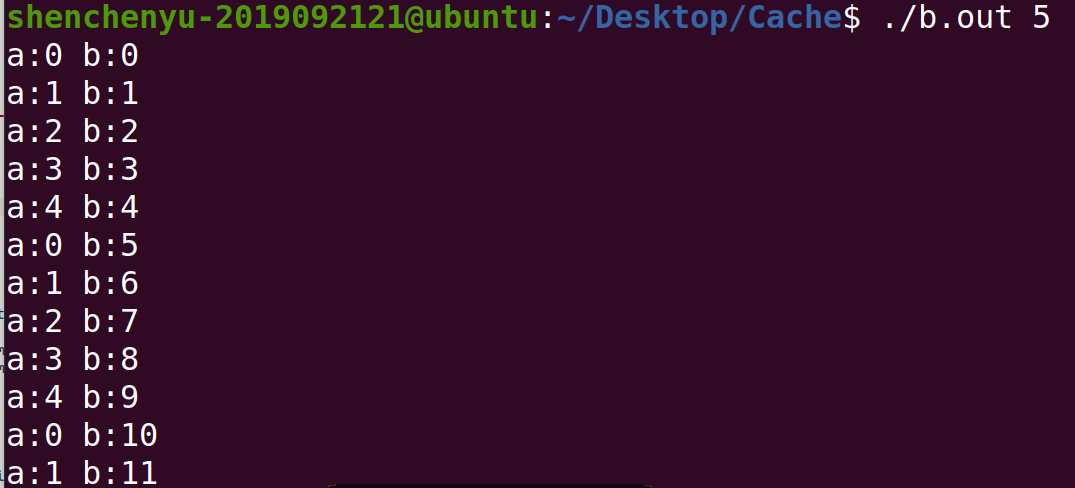
for (k=0;k<size;k++)

c[i\*size+j] += a[i\*size+k]\*b[j\*size+k];

}

}

对于a、b数组的访问部分顺序为：



可以看到数组访问顺序较好的符合空间局部性，属于对cache友好的c语言代码。

1. 优化后的加速比（speedup）

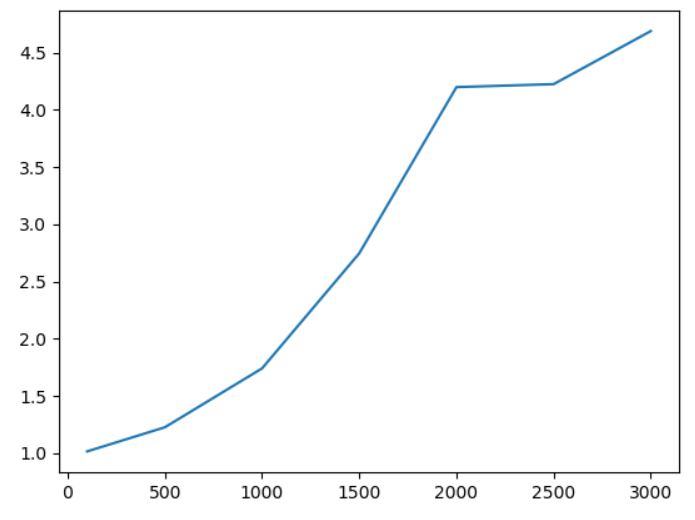
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 矩阵大小 | 100 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 |
| 加速比 | 1.015 | 1.226 | 1.741 | 2.747 | 4.200 | 4.226 | 4.690 |

加速比定义：加速比=优化前系统耗时/优化后系统耗时；

所谓加速比，就是优化前的耗时与优化后耗时的比值。加速比越高，表明优化效果越明显。

分析：

由下图知，总体上来说，当size越大时，对程序B的优化效果越好，即命中率更高，局部空间性更好



**五、实验总结与体会**

1、通过该实验，对cache的原理有了感性上的一点认识，即因为cpu和主存的速度上的差距，故而在CPU和主存之间设计了一级或多级的cache，cache为SRAM（静态随机存储器），因为cache时钟周期明显高于主存，故而加快了计算机的速度；

2、Cache和主存或cache和cache之间的数据传输是以块为最小单位，块里面保存着多个相邻地址的数据。例如：程序中存在一个数组int型b[4][4],块的大小为16字节，当第一次使用该数组时，例如使用了b[0][0]，会将主存中b[0][0]所在块放到cache中，因为块保存着主存相邻的数据，故而它里面很可能包含着b[0][1]、b[0][2]、b[0][3]，如果按b[0][1]、b[0][2]、b[0][3]顺序调用数组，这样就提高了cache的命中率，减少了时钟周期。如果按照b[1][0]、b[2][0]、b[3][0]的顺序访问，则因为cache的空间有限，当引入b[1][0]、b[2][0]、b[3][0]所在块时，可能将b[0][0]所在的块替换了，此时CPU再使用b[0][1]时，则需要重新从主存中拷贝b[0][1]所在块，这使cache的命中率下降了，增加了时间周期。

3、因此我们编写对cache更友好、符合局部性原理的代码，以提高代码的执行效率。

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字： 王毅    2021年 7月5日 |
| 备注： |