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、Fedora 13 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) {
```

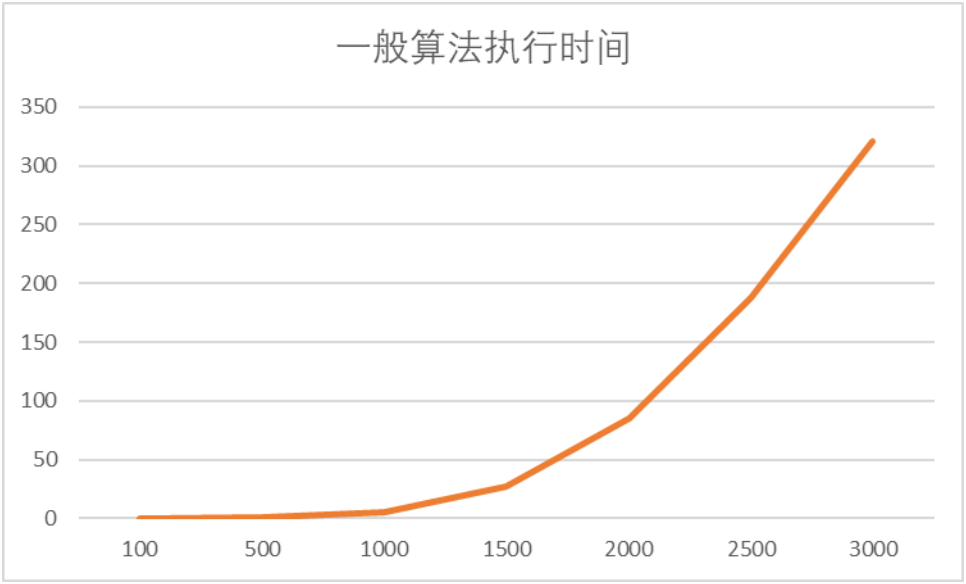
```
        time2.tv_usec+=1000000L;
        time2.tv_sec-=1;
    }

    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	0.003511	0.511947	5.785495	26.953632	85.128116	187.514837	320.786085

分析：




```
dongyunhao2019284073@ubuntu:~/SZU$ ./1 100
Executiontime=0.003353 seconds
dongyunhao2019284073@ubuntu:~/SZU$ ./1 150
Executiontime=0.011780 seconds
dongyunhao2019284073@ubuntu:~/SZU$ ./1 1000
Executiontime=5.455171 seconds
```

通过对代码进行分析，可以发现，代码中对数组中的每个元素进行的是以列为顺序的访问（行优先）

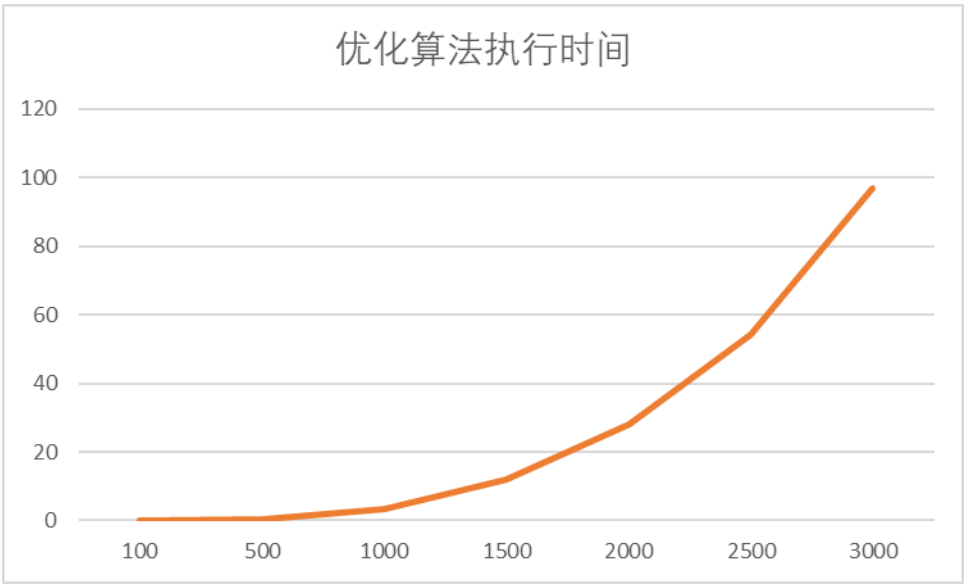
```
for(i=0;i<size;i++) {
    for(j=0;j<size;j++)  {
        c[i*size+j] = 0;
        for (k=0;k<size;k++)
            c[i*size+j] += a[i*size+k]*b[k*size+j];
    }
}
```

这有很差的空间局部性，也不能够高效借助 Cache 完成数据传递，效率很低。

2、 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	0.003768	0.430997	3.475389	11.838706	28.083379	54.43081	97.124569

分析：




```
dongyunhao2019284073@ubuntu:~/SZU$ ./2 100
Executiontime=0.003424 seconds
dongyunhao2019284073@ubuntu:~/SZU$ ./2 150
Executiontime=0.011362 seconds
```

通过对代码进行分析，可以发现，代码中对数组中的每个元素进行的是以行为顺序的访问（列优先）

```
for(i=0;i<size;i++) {
    for(j=0;j<size;j++)  {
        c[i*size+j] = 0;
        for (k=0;k<size;k++)
```

```
5.                 c[i*size+j] += a[i*size+k]*b[j*size+k];
6.         }
7.     }
```

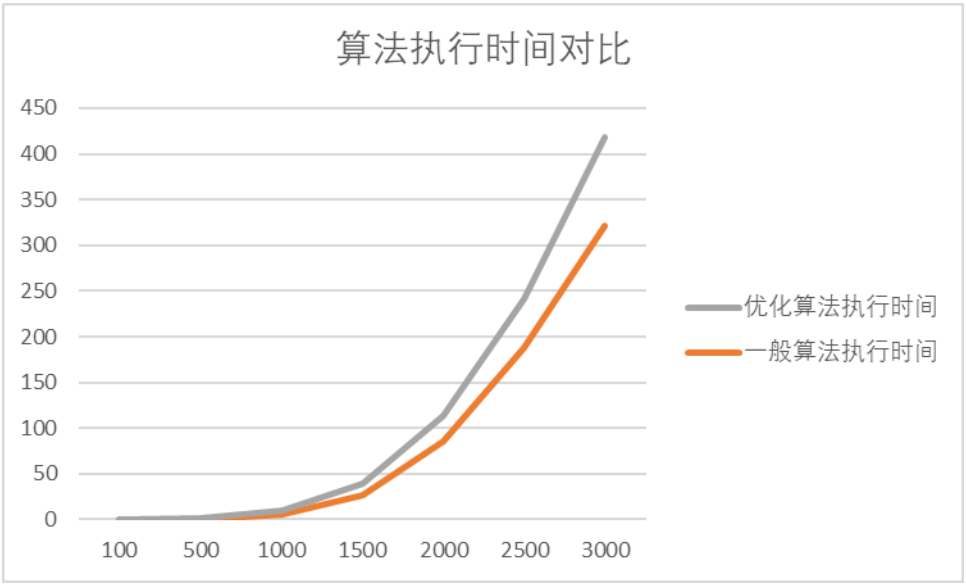
即按顺序对数组元素进行访问，有比较好的空间局部性，比较高效的借助了Cache进行数据传递。因此效率比较高。

3、 优化后的加速比（speedup）

矩阵大小	100	500	1000	1500	2000	2500	3000
加速比	0.931794055	1.187820333	1.664704296	2.276738015	3.031263296	3.445012797	3.302831491

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：



可以看到，加速比随着数据量的增大而增大，即数据越多，利用 Cache 进行优化得越明显。这说明，Cache 对大数据量下的优化效果比小数据量下好。

五、实验总结与体会

通过本次实验，我借助两份代码，借助矩阵乘法对 C 语言程序运行中 Cache 对程序运行的影响进行了探究。可以看到，Cache 对大数据下的程序运行有比较明显的优化效果。这说明具有良好空间局部性的代码运行起来往往较快，在实际编程中，我们也要尽量写出具有空间局部性的代码，以缩短程序运行时间。