天津大学

计算机系统课程实验

选择题目：第五章—性能优化

学生姓名 王雨朦

学生学号 2016229082

学院名称 国际工程师学院

专业 计算机

时间 2017/12/5

目录

一、 实验条件

机器型号	**Acer Aspire E1-471G**	
内存大小	10GB	
系统类型	64 位操作系统	
SSD	Sumsung 256GB	
CPU	Intel® Core™ i5-3210M CPU @2.50GHz 2.50GHz	
高速缓存	L1	2*32KB，8 路
	L2	2*256KB，8 路
	L3	3.0MB，12 路
操作系统	Windows 7 旗舰版 64 位	

二、 优化程序的方法

在第五章的学习中，我们总结了优化程序性能的几种方法：

1. 消除循环的低效率

例如：在 for 循环中，将每次函数调用移到函数外面。这样减少每次在循环中进行函数调用所耗费的时间，只用去内存中找变量的值就可以。

源代码	修改后

`for(int i=0; i<length(v); i++){`
`  …`
`}`
	`Int len=length(v);`
	`for(int i=0; i<length; i++){`
	`  …`
	`}`

2. 减少过程调用

即消除循环中的函数调用和数组访问，这种优化虽然可能代码没有显示性能提升，但有其他方面的优化，最终会产生显著的性能提升。

3. 消除不必要的内存引用

可将累积的结果写在临时变量中，消除每次循环迭代中从内存中读出并将更新值写回的必要；同理可以使用局部变量消除别名引用；例如我们常用的 for 循环中的 i++和++i 也是如此，i++需要先在内存中定义一个变量保存 i 的值，再加一，而++i 是直接加一。

源代码	修改后
`b[j] = a[i*n +j];`	`tmp = a[i*n +j];` `b[j] = tmp;`

4. 循环展开

例如将左边代码 2×1 循环展开如右表格，这种变换能减小循环开销的影响。

源代码	修改后

for(int i=0;i<length;i++)
{
sum *= data[i];
}
	int i;
	for(i=0;i<limit;i+=2)
	{
	sum=sum*data[i]*data[i+1];
	}
	for(;i<length;i++)
	{
	sum *= data[i];
	}

5. 提高并行性

如图所示，将乘法进行并行操作，不用一个等待另一个，可以大大提升程序的执行效率。

源代码	修改后

for(i=0;i<limit;i+=2)
{
sum=sum*data[i]*data[i+1];
}
	for(i=0;i<limit;i+=2)
	{
	// 合并下标为偶数的值， 0 按偶数算
	sum1 *= data[i];
	// 合并下标为奇数的值
	sum2 *= data[i+1];
	}
	double sum = sum1*sum2;

三、 改进第六章程序

1. 实验一：存储器山实验

A. 程序优化

由于该存储器山的代码中，没有很多循环，且循环中没有多余的函数调用，程序中也没有变量直接赋值引起别名引用的危险。因此，对照以上程序性能优化的方法，没有找到能够进行优化的地方。

存储器山的绘制是 python 写的，绘制过程中也没有对应可以优化的地方，因此没有对这个实验进行优化。

2. 实验二：图像增强中用均方误差测试空间局部性实验

A. 程序优化

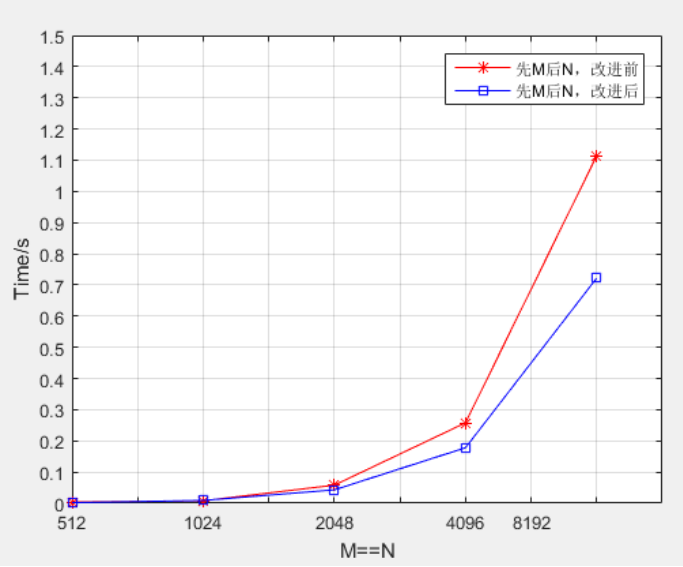
原代码有一段循环中如下，我们做两点改进。
一是利用方法 3 消除不必要的内存引用，将 i++改为++i;
二也是可以用方法 3 消除不必要的内存引用，即使用中间变量 tmp 进行改进。

```
for(j=0; j<N; j++){
    for(i=0; i<M; i++){
            sum += (a[i][j]-b[i][j])*(a[i][j]-b[i][j]);
    }
}
```

改进后：

```
for(i=0; i<M; ++i){
    for(j=0; j<N; ++j){
            tmp=a[i][j]-b[i][j];
           sum += tmp*tmp;
    }
}
```

B. 优化结果



由图中看出，改进后大大提升了运行速度，而且随着矩阵的增大，性能提升越明显。因为原来代码中每次循环时减少了一次数据读取和一次数据计算的过程，且每次循环过程中减少了 i 在内存中的新建。

3. 实验三：分块矩阵相乘

A. 程序优化

实验中有一段代码如下，我们做三点改进。

一是根据方法 3 消除不必要的内存引用，将 i++改为++i；

二是用方法 2 减少过程调用，使用中间变量 in 进行改进，in=i1*n 保存中间值，减少了内层循环中的数组下标的计算；

三是用方法 4 循环展开最内层循环，这里进行 2×1 展开。

```
for (i = 0; i < n; i+=B)
    for (j = 0; j < n; j+=B)
        for (k = 0; k < n; k+=B)
     /* B x B mini matrix multiplications */
            for (i1 = i; i1 < i+B; i1++)
                for (j1 = j; j1 < j+B; j1++)
                    for (k1 = k; k1 < k+B; k1++)
                          c[i1*n+j1] += a[i1*n + k1]*b[k1*n + j1];
```

改进后：

```
for (i = 0; i < n; i+=B)
    for (j = 0; j < n; j+=B)
        for (k = 0; k < n; k+=B)
     /* B x B mini matrix multiplications */
            for (i1 = i; i1 < i+B; ++i1){
                  in = i1 * n;
                  for (j1 = j; j1 < j+B; ++j1){
                      for (k1 = k; k1 < k+B; k1+=2){
                          c[in+j1] += a[in + k1] * b[k1*n + j1] + a[in + k1+1] * b
                      }
                }
            }
```

另外有一段代码如下：

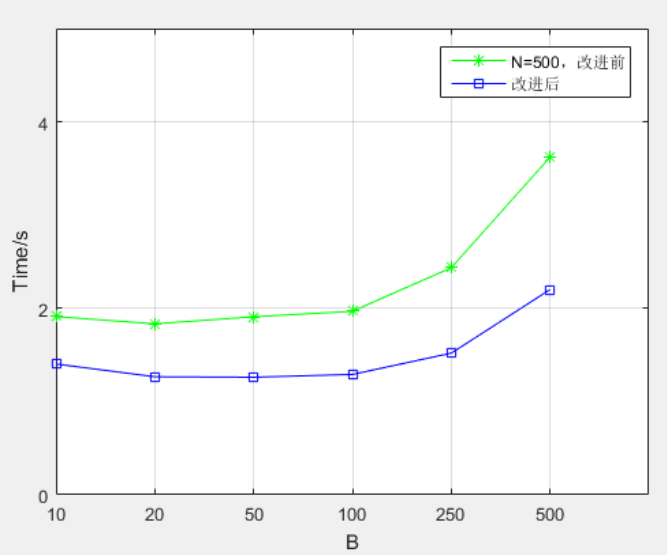
根据方法 1 消除循环的低效率，改写变量 tmp 将 N*N 的值进行保存。

```
for (i = 0; i < N*N; ++i)
   c[i]=0.0;
```

改进后：

```
int tmp=N*N, i;
for (i = 0; i < tmp; ++i)
   c[i]=0.0;
```

B. 优化结果



如图中结果所示，程序运行速度提升的比较明显，提升了将近原速度的三分之一。没步改进对程序的性能提升都比较明显。

4. 实验四：矩阵转置

A. 程序优化

实验中有一段代码如下，我们做三点改进。

一是根据方法 3 消除不必要的内存引用，将 i++改为++i；

二是根据方法 1 消除循环的低效率，将 int X,Y 提出了 for 循环外；

三是根据 2 减少过程调用，将循环中与循环无关的 LinePS[0]在循环外存进 tmp，在循环中只取 tmp 的值即可，减少循环中的数组访问。

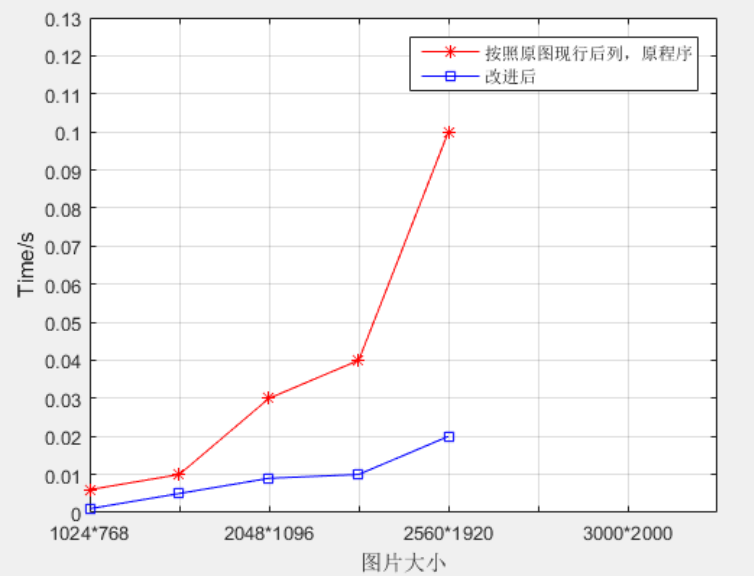
四是根据 4 循环展开最内层循环，这里进行 2×1 展开；

```
for (int X = 0; X < DstW; X++)
{
    LinePD[X] = LinePS[0];
    LinePS += StrideS;
}
```

改进后：

```
unsigned char tmp= LinePS[0];
for (X = 0; X < DstW; X+=2)
{
    LinePD[X] = tmp;
    LinePD[X+1]=tmp;
    LinePS += StrideS + StrideS;
}
```

B. 优化结果



如结果所示，改进效果非常明显。这四点改进都是在对循环进行优化修改，可见循环对程序性能的影响之大。

四、 改进第九章程序

1. 实验二：设计实验分析 TLB 对顺序读的影响

A. 程序优化

程序中这段循环，改进方法如下：

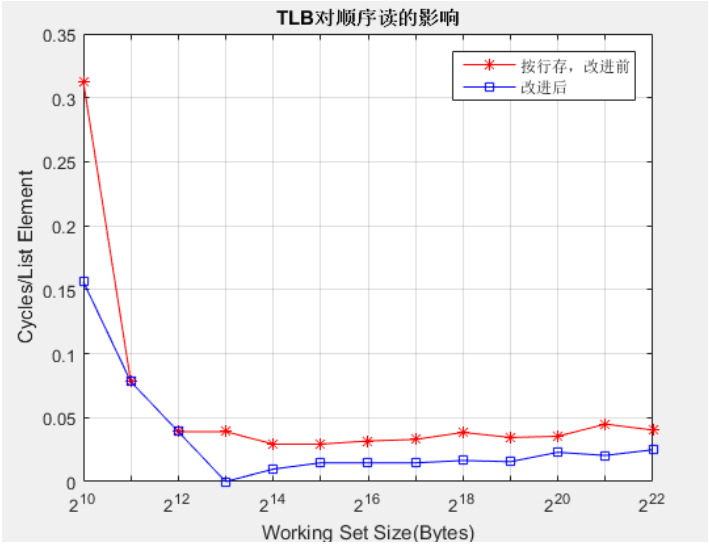
可用方法 1 消除循环的低效率，改写用变量 len 将 set/64 的值进行保存。避免每次循环调用时的计算。

```
for (i = 0; i < set /64; i++)
```

改进后，

```
len=set/64;
for (i = 0; i < len; i++)
```

B. 优化结果



如图中所示，仅优化了一点，程序的性能就有如此提升，可见优化程序是十分重要的。

五、 感想

通过第五章的学习，以前很多不知道如何处理程序的地方，既学会了如何处理又知道了为什么这样处理更好。学了程序优化之后，看到不规范的代码会有不舒服的感觉，离一名合格的工程师又近了一步。