****

**计算机组成原理实验报告**

**题目：**cache和程序访问的局部性（一）

**姓 名： 臧祝利**

**专 业： 计算机科学与技术**

**年 级： 2020级**

**学 号： 202011998088**

**任课教师： 王志春**

**完成日期： 2022年05月25日**

**人工智能学院**

1. 实验要求

在以下程序中，修改或添加必要的语句（如计时函数等），以计算和打印主

体程序段的**执行时间**。

分别以 M=100000,N=10、M=1000,N=1000、M=10,N=100000，执行程序A和程序B，以比较两个程序执行时间的长短。[保证总空间不变，N应为1e5]

1. 实验结果与分析

## 程序A

程序A代码如下所示；其中M和N的值根据情况进行变换

#include <stdio.h>

#include <time.h>

#include <windows.h>

#define M 1000

#define N 1000

**int** a**[**M**][**N**];**

**int** i**,**j**;**

**void** Assign\_array\_rows**(){**

**for** **(**i**=**0**;**i**<**M**;**i**++){**

**for** **(**j**=**0**;**j**<**N**;**j**++){**

a**[**i**][**j**]=**i**+**j**;**

**}**

**}**

**}**

**int** main**(){**

*/\*clock\_t start,end;*

*start=clock();\*/*

**double** run\_time**,**dqFreq**;** *//运行时间和计时器频率*

LARGE\_INTEGER time\_start**,**time\_over**,**f**;** *//开始时间、结束时间、计时器频率*

QueryPerformanceFrequency**(&**f**);**

dqFreq**=(double)**f**.**QuadPart**;**

QueryPerformanceCounter**(&**time\_start**);** *//计时开始*

Assign\_array\_rows**();**

QueryPerformanceCounter**(&**time\_over**);** *//计时结束*

run\_time**=**1000000**\*(**time\_over**.**QuadPart**-**time\_start**.**QuadPart**)/**dqFreq**;** *//精度为微秒*

printf**("\nrun\_time：%fus\n",**run\_time**);**

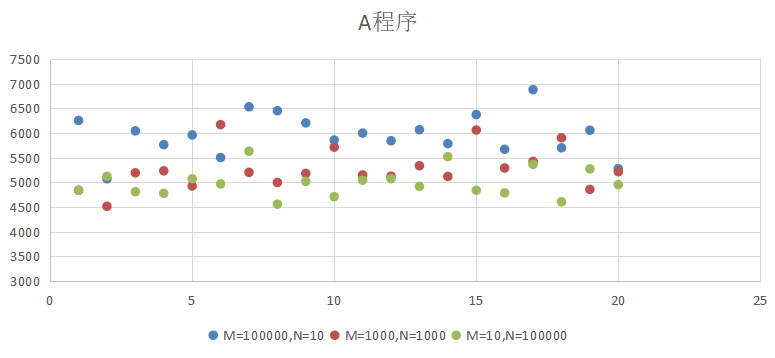
*/\*end=clock();*

*printf("time=%fms\n",(double)(end-start)\*1000/CLK\_TCK);\*/*

**}**

由于运行相同代码时运行结果时间仍存在差异，这里用运行20次的均值来代表最终的运行结果；

运行时间图像为：



平均值为：

|  |  |  |  |
| --- | --- | --- | --- |
| M,N | M=100000,N=10 | M=1000,N=1000 | M=10,N=100000 |
| Ta(μs) | 5966.32 | 5274.56 | 4996.445 |

## 程序B

程序B代码如下所示；其中M和N的值根据情况进行变换

#include <stdio.h>

#include <time.h>

#include <windows.h>

#define M 1000

#define N 1000

**int** a**[**1000**][**1000**];**

**int** i**,**j**;**

**void** Assign\_array\_cols**(){**

**for** **(**j**=**0**;**j**<**N**;**j**++){**

**for** **(**i**=**0**;**i**<**M**;**i**++){**

a**[**i**][**j**]=**i**+**j**;**

**}**

**}**

**}**

**int** main**(){**

**double** run\_time**,**dqFreq**;**

LARGE\_INTEGER time\_start**,**time\_over**,**f**;**

QueryPerformanceFrequency**(&**f**);**

dqFreq**=(double)**f**.**QuadPart**;**

QueryPerformanceCounter**(&**time\_start**);** *//计时开始*

Assign\_array\_cols**();**

QueryPerformanceCounter**(&**time\_over**);** *//计时结束*

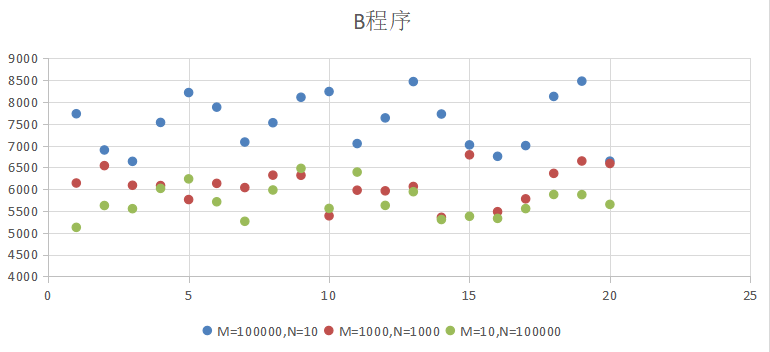
run\_time**=**1000000**\*(**time\_over**.**QuadPart**-**time\_start**.**QuadPart**)/**dqFreq**;** *//精度为微秒*

printf**("\nrun\_time：%fus\n",**run\_time**);**

**}**

由于运行相同代码时运行结果时间仍存在差异，这里用运行20次的均值来代表最终的运行结果；

运行时间图像为：



平均值为：

|  |  |  |  |
| --- | --- | --- | --- |
| M,N | M=100000,N=10 | M=1000,N=1000 | M=10,N=100000 |
| Tb(μs) | 7537.64 | 6091.35 | 5724.365 |

## 两程序运行时间比较

|  |  |  |  |
| --- | --- | --- | --- |
| M,N | M=100000,N=10 | M=1000,N=1000 | M=10,N=100000 |
| Ta(μs) | 5966.32 | 5274.56 | 4996.445 |
| Tb(μs) | 7537.64 | 6091.35 | 5724.365 |

可以简单看出，在M和N对应相等的情况下，相比先行后列(程序A)访问，先列后行（程序B）运行所需要的运行时间更长；对于某一个程序，M越小，N越大，即总空间不变的情况下，一行的长度越短，所需要的运行时间越短。

1. 实验小结

一开始选择<time.h>中的clock()函数进行计时，发现精度比较低，部分程序运行时间为0，于是更改成了<windows.h>库里的QueryPerformanceCounter()函数，使得最终结果可以精确到微秒级别；

根据实验结果，对于A程序的访问，是按行访问的，符合计算机的存储逻辑，具有良好的空间局部性，是一个步长为1的引用模式;因此效率比较高。对于局部数据块而言，主存块比较大可以使得一个比较大的空间调用到cache里，增加命中机会，从而使得运行时间减少，但是也不能太大，否则会造成缺失的损失更大；数组访问顺序而言，由于数组在存储器中是按照行存储的，行优先会使cache的命中率更高，从而提高程序的效率，减少运行时间。

1. 参考资料（如有参考资料可以列出）