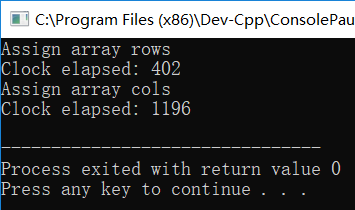
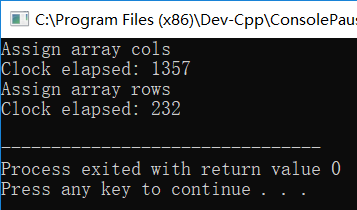
HW3

马逸君 17300180070

第一题 程序访问局部性对cache性能的影响

程序代码见附件exam1.cpp。

一开始我尝试了一次运行两个函数，但发现函数的耗时会随着它们的先后顺序而变化，函数先运行比后运行耗时更长。因此，我们得到的计时是有误差的，不能直接作为量化的性能评价指标，只能用于定性分析。（此结论已经过重复试验验证）

然后我改为每次只运行一个函数，且为了减小系统误差的影响，把要求的数组规模都扩大了10倍。实验结果如下：（为节约版面，截图略去）

实验1

实验1-1：1e6行\*1e2列（计时单位：系统clock数）

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值(取整) |
| assign\_array\_cols | 1416 | 1361 | 1344 | 1374 |
| assign\_array\_rows | 322 | 337 | 337 | 332 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 2162241 | 2172909 | 2267089 | 2200746 |
| assign\_array\_rows | 668285 | 648657 | 663336 | 660093 |

实验1-2：1e4行\*1e4列

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 726 | 746 | 704 | 725 |
| assign\_array\_rows | 345 | 342 | 333 | 340 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 3365847 | 3350482 | 3444198 | 3386842 |
| assign\_array\_rows | 673714 | 663279 | 669684 | 668892 |

实验1-3：1e2行\*1e6列

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 360 | 394 | 334 | 363 |
| assign\_array\_rows | 339 | 349 | 336 | 341 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 616767 | 608941 | 605638 | 610449 |
| assign\_array\_rows | 671610 | 669783 | 669073 | 670155 |

实验2 修改程序使a作为非静态局部变量并分配在栈区

实验2-1：1e6行\*1e2列（计时单位：系统clock数）

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 1286 | 1275 | 1328 | 1296 |
| assign\_array\_rows | 258 | 252 | 248 | 253 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 2160546 | 2127151 | 2169703 | 2152467 |
| assign\_array\_rows | 676038 | 674572 | 690573 | 680394 |

实验2-2：1e4行\*1e4列

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 678 | 682 | 685 | 682 |
| assign\_array\_rows | 278 | 276 | 264 | 273 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 3312301 | 3290034 | 3531194 | 3377843 |
| assign\_array\_rows | 725055 | 708952 | 716147 | 716718 |

实验2-3：1e2行\*1e6列

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 375 | 370 | 336 | 360 |
| assign\_array\_rows | 283 | 285 | 274 | 281 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 714697 | 697570 | 700892 | 704386 |
| assign\_array\_rows | 764028 | 710827 | 713896 | 729584 |

实验3 修改程序使a作为非静态局部变量并分配在堆区

实验3-1：1e6行\*1e2列（计时单位：系统clock数）

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 1707 | 1688 | 1688 | 1694 |
| assign\_array\_rows | 253 | 210 | 207 | 223 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 2153332 | 2161801 | 2459928 | 2258354 |
| assign\_array\_rows | 434945 | 432678 | 431671 | 433098 |

实验3-2：1e4行\*1e4列

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 806 | 957 | 773 | 845 |
| assign\_array\_rows | 270 | 252 | 335 | 286 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 3485876 | 3395830 | 3689086 | 3523597 |
| assign\_array\_rows | 517573 | 513590 | 509993 | 513719 |

实验3-3：1e2行\*1e6列

Windows 10 x64：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 1620 | 1631 | 1607 | 1619 |
| assign\_array\_rows | 298 | 300 | 295 | 298 |

Ubuntu 18.04：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数 | 第一次运行 | 第二次运行 | 第三次运行 | 平均值 |
| assign\_array\_cols | 3358540 | 3338824 | 3382206 | 3359857 |
| assign\_array\_rows | 554897 | 543596 | 538395 | 545629 |

从以上的实验数据，可以得出以下结论：

1. 无论数组a被分配在内存的哪一区，按行访问的性能都不差于按列访问的性能。
2. 按行访问时局部数据块(行)的大小对效率影响不大；按列访问时，在Windows下的全局变量和栈区两组实验中，都出现随局部数据块(行)变大效率显著提升的情况，但其他几组实验中，则是执行时间随局部数据块(行)大小变化呈现无规则的明显波动。
3. 总体效率：栈>堆，栈>全局变量

分析如下：

1. 数组访问顺序与执行时间的关系

我们知道数组在内存中是按行存放的，所以按行访问的空间局部性好；由于局部性，cache命中率较高，大多数情况下CPU能直接从cache中取得指令和数据，不必访问慢速的主存，执行时间较短。

按列访问则需要看具体情况（包括局部数据块大小）。

大部分情况下，按行访问的效率是要优于按列访问的，因而执行时间较短。

1. 局部数据块大小与执行时间

首先显而易见的是，按行访问时，因为无论行的长度如何，访问的地址在内存上都是连续的，所以cache使用情况与行大小无关，效率变化不大。

接下来讨论按列访问，设L1数据缓存大小为4\*32K，块大小64byte，8路组相联（则一共64组，每组8行，一行一块）；L2缓存大小4\*256K，块大小64byte，4路组相联（则一共1024组，每组4行），不考虑L3缓存。假设a[0][0]地址为0。

当M=1e2，N=1e6的时候，

按行访问：第一次miss，后31次命中，如此循环。每32次miss一次，总共miss 3125000次。

按列访问：前100次都miss，后面31\*100次命中，如此循环。同样是每32次miss一次，总共miss 3125000次，所以与按行访问相差不大。

同理，当M=1e4，N=1e4的时候，按列访问前1e4次miss，后310000次命中（L1和L2 cache总大小大于64\*10000），所以同样是每32次miss一次，总共3125000次miss。但L1只有32K，小于64\*10000，所以很多都是在L2 cache中命中的，比L1慢。综合下来比按行慢。

当M=1e7，N=10的时候，按列访问第一次miss，后2次命中（因为每块对应32个short），如此循环，平均每3.2次miss一次，总共31250000次miss。比按行慢得多。

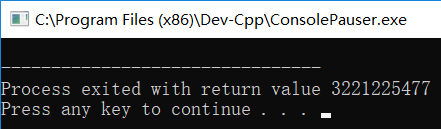
（另：如果要讨论的是cache与主存交换数据的“块”的大小，则应是：随着块增大，命中率是先上升后下降的。当块很小的时候(例如只有一个存储单元)，显然不能很好地利用局部性，访问中会发生较多次的替换，命中率较低；随着块大小增加，命中率也是增加的；超过某个最佳值以后，虽然对空间局部性好的程序来说命中率更高，但块大小更大导致需要花费更多时间存取，miss penalty变大，且块的数量也在减少导致对时间局部性的利用下降，所以对空间局部性不是很好的程序来说，会显著地变慢。）

1. 堆区、栈区、全局变量区存取速度问题

堆区比栈区慢的原因是，栈区是系统底层直接支持的，具体来说，有专门的寄存器指向栈所在的地址，有专门的机器指令完成数据入栈出栈的操作；堆区则是由函数库提供的，由malloc/realloc/free函数完成维护，而且存放不连续。显而易见栈会快一些。但堆区也有其存在的意义：它更灵活。[1]

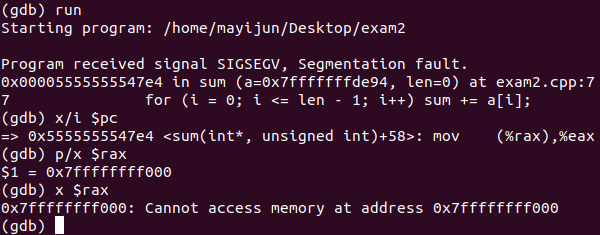
栈区局部变量比全局变量快的原因之一是，访问栈中的数据往往只需要一条指令，这是因为有专门的栈指针存在；访问全局变量可能需要2-3条指令，因为全局变量的地址计算较复杂、涉及到相对定位的问题，而且因为x86-64指令是32位而地址是64位，64位的地址不能被编码到32位的指令中，所以访问全局变量一般需要至少2条指令的。另外也有人认为，局部变量往往是按顺序分配空间的，而且过程调用时访问返回地址也会导致邻近的局部变量被读到cache中，提高了命中率，全局变量则无此优势。[2]

第二题 存储保护

Windows下执行结果：

Linux下执行结果：

在Linux下，gdb调试结果如下：



由此可见，发生异常的指令为：

mov (%rax), %eax

发生的异常的内容为段错误(Segmentation Fault)，结合指令的内容，只可能是访问了不可访问的内存地址。该结论和gdb中测试的结果相符。

发生访问违例的存储单元地址为0x7ffffffff000。

参考资料：

[1] <https://blog.csdn.net/baidu_37964071/article/details/81428139>

C/C++的内存分配？栈和堆的区别？为什么栈快？ honeyRJ 2018.8.5

[2] <https://bbs.csdn.net/topics/350109504?list=2557603>

为什么局部变量速度要比全局变量速度快？ 2010.10.19