**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： 马晨琳**

**报告人： 彭逸远 学号： 2023152031 班级： 数计班**

**实验时间： 2025年 5 月 15 日至 5 月 24 日**

**实验报告提交时间： 2024年 5 月 23 日**

**教务处制**

|  |
| --- |
| **一、实验目的：**   1. 加强对Cache工作原理的理解； 2. 体验程序中访存模式变化是如何影响cahce效率进而影响程序性能的过程； 3. 学习在X86真实机器上通过调整程序访存模式来探测多级cache结构以及TLB的大小。 |
| **二、实验环境**  X86真实机器 |
| **三、实验内容和步骤**  **1、分析Cache访存模式对系统性能的影响**   * 1. 给出一个矩阵乘法的普通代码A，设法优化该代码，从而提高性能。   2. 改变矩阵大小，记录相关数据，并分析原因。   **2、编写代码来测量x86机器上（非虚拟机）的Cache 层次结构和容量**   1. 设计一个方案，用于测量x86机器上的Cache层次结构，并设计出相应的代码； 2. 运行你的代码获得相应的测试数据； 3. 根据测试数据来详细分析你所用的x86机器有**几级Cache**，**各自容量**是多大？ 4. 根据测试数据来详细分析**L1 Cache行**有多少？   **4、尝试测量你的x86机器TLB有多大？**  代码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 |
| **四、实验结果及分析**  **1、分析Cache访存模式对系统性能的影响**  表1、普通矩阵乘法与及优化后矩阵乘法之间的性能对比   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 矩阵大小 | 100 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 | | 一般算法执行时间/s | 0.002004 | 0.277290 | 2.756737 | 12.963930 | 29.267048 | 81.288047 | 118.634738 | | 优化算法执行时间/s | 0.001999 | 0.223383 | 1.769253 | 5.954850 | 14.154809 | 28.596098 | 48.076917 | | 加速比  speedup | 1.002501 | 1.241321 | 1.558136 | 2.177037 | 2.067640 | 2.842627 | 2.467603 |   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。  分析原因：  对于计算矩阵相乘的代码进行优化：    图1 优化代码  原始代码的访存模式存在缓存不友好的问题。在代码A中，b[k\*size+j]的访问是跳跃式的，每次均访问b的列元素，导致缓存行频繁失效 ，增加缓存未命中率。因此，我们将将矩阵划分为小块，使每个块能完全放入缓存，则数据在缓存中的命中率增大，减少从外部内存读取。  **2、测量分析出Cache 的层次结构、容量以及L1 Cache行有多少？**   1. 实验原理；   Cache 是 CPU 和主存之间的重要缓冲结构，通常包括 L1、L2、L3，不同层容量和速度差异较大。CPU 一次从内存读取的不只是一个字节，而是一整个缓存行（如 64 字节）。每一层 Cache 都有一定容量，超出后会替换已有数据。通过测量访问不同大小的数组或跳跃不同步长的数据，观察访问延迟，从而反推出各级 Cache 的容量和行大小。   1. 测量方案及代码；   **1.测试Cache行大小**  当跨越不同 Cache 行时，访问延迟明显变高。因此，通过逐步增加访问步长（stride），测量单位访问时间变化，从而推测Cache行大小。  代码如下：    图2 测试缓存行大小代码  **2.测试L1，L2，L3大小**  通过逐步增加数组的总大小，观察访问延迟何时明显上升，来判断缓存容量的上限。固定 stride 为 64 字节（1 个 cache line），避免影响测量；每次扩大数组大小，从几十 KB 到几十 MB；观察哪个大小下，访问延迟发生明显上升。  测试代码如下：    图3 三种缓存测试代码  **3.L1 Cache行**  L1 Cache行的数量可以通过L1缓存容量大小除以每行的大小来计算   1. 测试结果；   **1.对Cache的行测试结果如下**  **：**  表1 测试Cache行大小   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 组数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | 步长(Byte) | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | | 平均访问时间(ns) | 1.53 | 1.57 | 1.73 | 2.29 | 3.68 | 6.36 | 6.82 | 6.64 |   图5 不同Cache行大小访问时间  **2. L1，L2，L3缓存大小测试结果**  图6 数组大小与组别关系  图7 平均运行时间与组别关系   1. 分析过程；   **1.对Cache行大小的分析：**  观察到在第六组测试，缓存行大小为64 字节时延迟从 2.29 上升到 3.68ns，有明显跳跃，说明：缓存行大小 ≈ 64 Bytes 在此之前，访问都在一个 Cache line 内，硬件预取器效果好，从此以后，访问每次都跨越新行，需要从缓存或主存加载。  **2.L1，L2，L3缓存大小测试分析**  观察图像，发现在第12组、第27组和第37组分别出现了耗时的突变，观察组别对应的数组大小，第12组时数组大小为708KB故L1大小大约为708KB左右，第27组的数组大小为11268KB，则L2的大小为11MB左右。第37组的数组大小为21508KB，大小约为21MB。  **3.L1 Cache Line数量计算**  根据求出的Cache行大小64B和L1缓存大小708KB，我们可以计算得到行数量大约为11328行。   1. 验证实验结果。   查询Intel官网的信息，得知i5 12600KF处理器的Cache Line大小为64KB，测试无误。  进入任务管理器查看i5 12600KF处理器的缓存信息：    图8 CPU缓存信息  比对测试得到的缓存信息，缓存大小十分接近。由于测试L1缓存过程中，数组大小步长设置为64MB，L2，L3步长设置为1MB，因此会有一定的大小差异，测试结果较为精准。  **3、尝试测量你的x86机器TLB有多大？**  每次访问一个新页时，CPU 会查找 TLB，如果命中，访问很快。如果页不在 TLB 中（TLB miss），则会发生页表查找（慢）。因此，我们可以逐步增加访问的页数，观察访问延迟何时突然增加，这就是 TLB miss 开始发生的点，也就是 TLB 容量边界。  表2 不同页面大小访问时间表   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 组数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | 页面大小 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | | 访问时间 | 1 | 1.4 | 1.27 | 1.05 | 2.24 | 4.28 | 5.58 | 6.16 | 7.24 | 7.19 |   图9 不同页面大小的访问时间  观察上图，在第四组中，当页数小于8，访问时间基本稳定在 1 ns左右；在第六组中，当页数增加到 32 时，访问时间明显增加至 4.28 ns；说明 TLB 大小大约为 32。现代 CPU中的 TLB 数量在32–64左右，与测试的结果几乎相同。 |
| **五、实验结论与心得体会**  通过本次实验，我成功测量并分析了处理器中各级缓存和 TLB 的容量与访问特性。实验结果显示，随着数据访问跨度或数据规模的变化，访问延迟会呈明显分段上升趋势，反映了缓存与 TLB 命中率的变化。特别是通过对 stride 访问与页数遍历的分析，我估算出 L1 缓存行数及 TLB 的条目数与理论值基本一致，验证了实验的合理性。本实验不仅加深了我对现代处理器内存层次结构的理解，也锻炼了我设计测试方法和分析实验数据的能力。 |

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  2024年 月 日 |
| 备注： |