**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： 李文俊 学号2023150001班级： 高性能班**

**实验时间： 2025年 5 月 14 日至 5月 28 日**

**实验报告提交时间： 2025年 5 月 15 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？   **3、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 | 0.003743 | 0.274894 | 2.699657 | 11.870786 | 30.583262 | 70.540749 | 125.633544 | | 优化算法执行时间 | 0.003594 | 0.262902 | 2.120739 | 7.025034 | 16.684690 | 30.274917 | 50.259630 | | 加速比  speedup | 1.041457986 | 1.045613955 | 1.272979372 | 1.689783423 | 1.833013499 | 2.330006355 | 2.499690985 |   **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：    图 1  **根源：存储时a和b都是按行存储，但是访问时a是按行访问，b是按列访问，导致程序的空间局部性差，B的cache命中率低**    图 2  优化方式：  比较特殊的是矩阵的行和列都是size，所以可以直接调换i和j，按列存储矩阵b，使Cache按列存储b的一列元素——**保证结果不变**    图 3    图 4  但是矩阵b访问时还是按列访问，Cache命中率不变，因此还要改变b对应的访问顺序**——提高空间局部性**    图 5  优化效果：  vim B.c 创建优化后代码，gcc -Wall B.c -o B创建可执行文件, ./B arg运行代码，将运行时间填入表1中。    图 6    图 7  **随着矩阵大小增加，两个算法的运行时间差距逐渐增大，加速比越大，优化效果越明显。**  **2、测量分析出Cache 的层次结构、容量以及L1 Cache Line有多少？**  （1）实验原理  **测量Cache层次结构和容量**：现代CPU通常用三个层次的Cache，分别为L1,L2和L3，呈金字塔结构，从L1到L3，容量越大，速度越慢，一般访问L1是4个时钟周期，访问L2是10个时钟周期，访问L3是50个时钟周期，操作系统会优先访问金字塔顶层的存储层次，因此，我们可以通过设置递增的内存大小记录访问需要的时间，**如果时间发生较大的变化，说明前一个层次结构已满**，需要用到较慢但较大的下一层存储结构。  因为DRAM和Cache差距较大，因此当**出现非常大的时间变化时说明Cache容量已满。**  **测量L1 Cache Line大小：**已知L1 Cache容量后，内存数据会按 ​**连续地址** 划分到不同的缓存行,例如内存地址 0x0000 ~ 0x003F（64 字节）属于第一个缓存行，当以不同步长（stride）访问内存时，访问模式会决定缓存行的命中率，当步长小于Cache Line大小时部分数据在同一缓存行内，无需频繁加载新行；当步长超过大小，每次访问都触发缓存未命中（Cache Miss），需从内存或更慢的缓存层级加载数据，会比部分命中需要更大的时间。  在容量范围内逐步增加内存步长顺序，当超过某个步长，时间发生较大变化（相对于之前步长），说明超过Cache Line大小，需要到另一个Cache Line中访问，会增加访问时间。  （2）测量方案及代码  **1.测量Cache层次结构和容量：**  **①设置访问内存大小**  根据主流CPU Cache大小，从128KB开始按内存翻倍增加，总体上测试存储结构的层次  在速率变化较大的区间缩小精度按16KB递增测试。    图 8  **②随机访问内存**  设置访问内存次数为1亿次，增大效率的差异，更好地体现容量大小，随机访问Cache记录所有次数所需要的时间，每一个容量进行10次取平均值    图 9  **③计算访问速率MB/s**  **V=总数据量（B）/ 时间（ns）×1e9 / MB，最后输出结果**    图 10  **2.测量L1 Cache Line大小：**  ①设置内存大小：固定Buffer容量，从1B到128B翻倍递增，逐步增加步长顺序    图 11  ②跳跃内存访问：外层循环控制偏移，内层循环遍历所有偏移位置，确保覆盖缓存行的所有偏移位置，强制每次访问跨越不同缓存行（无论步长是否小于 64），放大性能差异，能更精准捕捉到 stride=64 时的吞吐量骤降点，否则可能会误判缓存行大小（吞吐量下降不明显）。    图 12  ③计算访问速率MB/s，分析容量和速率关系，得到结果    图 13  （3）测试结果    图 14    图 15    图 16    图 17    图 18    图 19  L1 Cache Line:    图 20  （4）分析过程  根据总体Cache容量与速率差距：**L1 缓存容量在512KB~1024KB，L2缓存容量在8192KB~16384KB，L3缓存容量在16384KB~32768KB**  测试容量从800到1024，发现864KB之后速率突然较大幅度减少**，L1 缓存可能在864KB附近。**  测试容量从9088到10240，发现9728KB之后速率突然较大幅度减少，**L2 缓存可能在9728KB**附近。  测试容量从20000到20960，发现20480KB之后速率突然较大幅度减少**，L3 缓存可能在20480KB**附近。  **L1 Cache Line 大小判定：**当 stride 达到 64 字节时，吞吐量显著下降，说明此时每次访问跨越不同缓存行，命中率降低。  （5）验证实验结果    图 21  从任务管理器-性能中查看CPU各级缓存大小，可以看出L1 Cache的大小为864KB,L2 Cache的大小为9.5MB，L3 Cache的大小为20.0MB，**与分析结果一致，分析正确。**    图 22    图 23  从英特尔处理器参数表可以得到L1 Cache Line大小是64B，**与分析结果一致，分析正确。**  **3、尝试测量你的x86机器TLB有多大？**  （1）实验原理  **TLB（Translation Lookaside Buffer）**：CPU 用于缓存虚拟地址到物理地址转换结果（页表条目）的硬件缓存，本质就是一个高速缓存，数据cache缓存地址(虚拟地址或者物理地址)和数据，而TLB缓存虚拟地址和其映射的物理地址。虚拟地址首先发往TLB确认是否命中cache，如果cache hit直接可以得到物理地址。否则，一级一级查找页表获取物理地址。并将虚拟地址和物理地址的映射关系缓存到TLB中。  ​**页大小：通常为 4KB**（x86默认），大页可能为 2MB 或 1GB。  因此，测试思路可以是分配连续内存并按页大小步长访问，强制触发 TLB 未命中。  逐渐增加访问的内存范围，当范围超过 TLB 容量时，访问时间会显著上升。  （2）测量方案及代码  **①分配大块内存**  使用 mmap 分配一块按页对齐的大内存（例如 1GB），确保访问模式跨越不同页    图 24  **②​遍历不同页数，测量访问时间**    图 25  （3）测试结果和分析    图 26  在4096时TLB单页周期数发生较大增加，说明超出TLB容量大小，TLB容量可能在1024\*4KB~2048\*4KB，即4MB~8MB附近。  （4）验证实验结果    图 27    图 28  Linux的页大小为4KB，整体的TLB容量是2M/4M, **与分析结果一致，分析正确。** |
| **五、实验结论与心得体会**  本次实验围绕Cache与TLB的工作原理展开，通过代码优化和结构探测，深入理解了计算机体系结构中存储子系统的关键机制及其对程序性能的影响。在矩阵乘法优化实验中，观察到访存模式对性能的显著作用：当按列访问矩阵时，相邻元素因跨越不同缓存行（64B）导致频繁Cache Miss，计算速度下降；而改为按行访问后，空间局部性使得连续元素被集中加载至同一缓存行，性能提升1-2倍。这一对比验证了局部性原理在程序优化中的核心地位——合理利用缓存行预取机制可大幅减少内存访问延迟。  进一步通过步进式内存访问测试（Stride Access）探测多级Cache结构，发现当访问步长超过64B时，L1 Cache（864KB）的延迟（约1ns）开始突增；步长增至9.5MB后，访问时间跳变至L2 Cache级别（约3ns）；而步长接近20MB时，访问延迟进一步上升至L3 Cache（约10ns）。这一结果与lscpu命令输出的硬件参数一致，表明通过编程实测能够有效推断缓存层级结构。值得注意的是，L1 Cache行大小的验证实验中，以64B为步长的访问时间最短，进一步佐证了缓存行对齐对性能优化的必要性。  在TLB测量实验中，通过分配4KB页对齐的连续内存区域并生成随机页访问序列，结合计时发现：当访问页数超过1024（即4MB内存范围）时，单页访问时间因TLB Miss出现明显跃升。结合coreinfo工具输出的硬件信息（如L1 DTLB条目数为64），推断实验机器的TLB可覆盖约4MB的虚拟地址空间（即1024页×4KB）。这一结果揭示了TLB容量对程序内存布局的约束——若程序频繁访问超出TLB条目限制的离散内存页，会因页表遍历引入额外开销，进一步说明内存紧凑化与页对齐的重要性。  实验的深层启示在于，存储子系统的性能优化需综合考量Cache与TLB的协作机制。例如，即使程序已充分利用缓存行，若内存页访问模式过于随机，仍可能因TLB Thrashing导致性能瓶颈。因此，高性能代码设计需在数据布局（如结构体紧凑排列）、访存模式（如顺序访问）和内存分配策略（如页对齐）间达成平衡。此次实践不仅深化了对Cache-TLB层级协同的理解，更凸显了“从内存视角思考性能”的工程哲学——在摩尔定律趋缓的当下，存储墙的突破愈发依赖于软硬件协同的精细化优化。 |

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