**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： 计算机与软件学院所有专业**

**指导教师： 罗胜**

**报告人： 赵美玲 学号： 2023155025**

**实验时间： 2025年 6 月 1日至 6月 20 日**

**实验报告提交时间： 2025年 6 月 20 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？   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 | 0.002704 | 0.367576 | 3.374580 | 14.545891 | 27.353415 | 70.266762 | 113.959578 | | 优化算法执行时间 | 0.002276 | 0.279886 | 2.391224 | 8.081079 | 19.112433 | 37.338461 | 64.096240 | | 加速比  speedup | 1.188049 | 1.313306 | 1.411235 | 1.799994 | 1.431184 | 1.881887 | 1.777945 |   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。  **优化代码：**    分析原因：  分块原理：将大矩阵划分为BLOCK\_SIZE×BLOCK\_SIZE的小块，每次只处理小块数据，确保数据能完全放入CPU缓存，减少缓存未命中（cache miss）次数  BLOCK\_SIZE选择：典型值为32-128，取决于CPU缓存大小，可通过实验找到最佳值（本代码设为32），现代CPU L1缓存通常为32KB，适合32×32的float块（32×32×4B=4KB）  三重循环优化：外层循环遍历所有块（ii, jj, kk），内层循环处理当前块内的元素（i, j, k），边界处理确保不越界  与其他优化的结合：仍然保持了内存对齐、循环重排序等优化，可以与多线程并行计算结合（每个线程处理不同块）  **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；**  基本原理  通过测量不同大小内存块的访问时间差异来检测Cache层次结构。当测试数据大小超过某一级Cache容量时，访问时间会显著增加。  **（2）测量方案及代码；**  分配一系列不同大小的内存块  顺序访问这些内存块并测量访问时间  分析访问时间随内存大小变化的拐点 代码如下：    **（3）测试结果；**  **（4）分析过程；** 由数据可以看出，384KB-512KB首次出现较大时间增加，所以估计第一级cache大小在384KB-512KB；3072KB-4096KB再次出现较大时间增加，所以估计第二级cache大小在3072KB-4096KB；在16384KB的时间变化较大，就应该是超过第三级cache的区域，所以估计第三级cache在16384KB左右。  **（5）验证实验结果。**  **dc46883861ae577aa6f225c36f9e238** |
| **五、实验结论与心得体会**  通过本次实验，我借助两份代码，借助矩阵乘法对C语言程序运行中Cache对程序运行的影响进行了探究。可以看到，Cache对大数据下的程序运行有比较明显的优化效果。这说明具有良好空间局部性的代码运行起来往往较快，在实际编程中，我们也要尽量写出具有空间局部性的代码，以缩短程序运行时间。 |

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  2025年 月 日 |
| 备注： |