**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： 软件工程**

**指导教师： 马晨琳**

**报告人： 学号： 班级：**

**实验时间： 2025年 5 月 14 日至 5 月 27 日**

**实验报告提交时间： 2024年 5 月 24 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 | 0.002999 | 0.355014 | 3.317456 | 14.285250 | 31.363163 | 101.123598 | 214.852792 | | 优化算法执行时间 | 0.003001 | 0.332496 | 2.732219 | 9.332219 | 22.266843 | 43.785236 | 76.675017 | | 加速比  speedup | 0.999 | 1.068 | 1.214 | 1.531 | 1.409 | 2.310 | 2.802 |   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：  优化算法为将其中一个矩阵倒置，这样，在进行乘法操作时，数据的访问会更加连续，从而减少Cache未命中的情况，其他内容基本不变      数据局部性提升：通过转置矩阵B，优化后的算法改进了数据访问模式，使得CPU访问内存时更可能从Cache中命中数据。这是因为转置后，数据访问在内存中是连续的，符合Cache设计的局部性原理。  减少Cache未命中的开销：原始算法中，对矩阵B的访问是按列进行的，这导致较高的Cache未命中率和内存访问开销。优化后，对矩阵B的访问变为按行访问，减少了Cache未命中情况。  执行时间与矩阵大小的关系：随着矩阵大小的增加，优化效果更加明显，尤其在较大的矩阵上。这表明当工作集大小超过Cache容量时，优化算法更能有效地利用Cache。  加速比的变化趋势：对于较小的矩阵（如500x500），加速比较低，因为整个矩阵可能完全适合于Cache中，使得优化效果不明显。但随着矩阵大小的增加，原始算法因多次Cache未命中而性能下降更严重，因此优化后的加速比提高。  算法的效率提升：总体来看，优化算法改善了计算的时间复杂度，尤其在高维数据处理时，有效地减少了操作数和改善了Cache的命中率。  **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；   实验的基本原理是通过特定的内存访问模式来触发和测量Cache的行为。通过逐渐增加访问数据的步长和大小，我们可以观察到不同Cache层次的容量和换出策略。主要原理包括：  缓存行大小测量：通过访问跳跃特定字节大小的数据，可以观察到程序性能的突变点，该点即为缓存行的大小。  缓存容量测量：通过逐渐增加数据的大小并观察Cache失效率的增加，可以推断出Cache的容量。  多级Cache识别：通过改变访问的数据大小，可以识别出不同级别的Cache（如L1, L2, L3），每个级别的Cache在特定数据大小阈值时表现出性能下降。   1. 测量方案及代码；   方案使用（1）中的操作思路  1. 测量L1 Cache行大小    2.测量各级Cache的容量     1. 测试结果；   1. 测量L1 Cache行大小      2.测量各级Cache的容量     1. 分析过程；   **Cache行大小分析**  我们可以观察到随着stride增加，时间总体趋势是逐渐减少，这是因为增大stride减少了每次迭代访问的数据总量，进而减少了内存访问次数。特别注意，Stride 16之后，时间的下降趋于平稳。这种趋势暗示了Cache行可能是64字节（因为当Stride为16时，整型（int）大小为4字节，16\*4 = 64字节）。当stride超过Cache行大小时，每次数组访问都会触发一个新的Cache行加载，因此性能改进边际逐渐减少。  **Cache 的层次结构、容量**  第二个数据图显示了随着数组大小的增加，时间如何变化。关键点在于注意时间增加的显著变化，这可能表明超过了某级Cache的容量。我们可以观察到数据访问时间随着数组大小的增加而增加，特别是在接近2MB、20MB和24MB时，时间增加更为显著。即处理器有**三级缓存**，L1 Cache约为1MB至2MB之间，L2 Cache约为20MB，L3 Cache约为24MB。   1. 验证实验结果。   查询英特尔官方文档，得证：i5-13600K的Cache行是64字节，L1 Cache缓存为1.2MB，L2 Cache为20MB，L3 Cache为24MB，与实验结果相符  **3、尝试测量你的x86机器TLB有多大？**  我打算通过修改访问的内存大小和访问模式来测量TLB大小。以下是测试方法：  创建一个足够大的数据集：这个数据集需要大到足以覆盖多个页表项。  逐渐增加步长访问数据：从访问数组的每个元素开始（步长为1），逐步增加步长，每次跳过几个元素，最终达到一个较大的步长。  测量访问时间：对于每种步长，测量完成所有访问的时间。当步长增加到超过TLB容量时，预期会看到访问时间的显著增加，因为这会导致频繁的TLB缺失。    运行程序，得到如图结果    初始步长（1-8页）：在步长为1到8页时，时间逐步减少。这表明TLB在这些步长范围内基本能够覆盖访问的地址，命中率较高。  中间步长（16-128页）：随着步长的增加，访问时间进一步减少。这可能是因为每次访问的数据局部性降低，预取机制和缓存优化效果更加显著，TLB的作用开始减弱。  大步长（256页及以上）：当步长增加到256页及以上时，时间趋于稳定，甚至接近于零。此时访问的每个页面间隔较大，内存访问的局部性非常低，基本每次访问都是TLB miss，但由于步长过大，实际执行的内存访问次数减少，导致时间反而减少。  基于上述分析，可以推测CPU的L1 TLB大小大约在32到64个页面之间。 |
| **五、实验结论与心得体会**  通过对矩阵乘法代码的优化，可以显著提高程序性能。优化后的算法通过将矩阵B进行转置，使得矩阵乘法中的数据访问更加连续，从而减少了Cache未命中的情况，显著提升了执行效率。  从实验结果可以看出，优化算法在较大的矩阵上表现出更显著的性能提升，尤其当矩阵大小超过Cache容量时，优化效果更加明显。  实验数据表明，i5-13600K处理器具有三级缓存结构。通过逐步增加数据大小并测量访问时间，可以推断出L1 Cache的容量约为1MB，L2 Cache的容量约为20MB，L3 Cache的容量约为24MB。进一步查阅官方文档验证了这一推测。  本次实验通过实际编程和性能测试，深刻体会到Cache优化对程序性能的显著影响。特别是在大数据处理和高性能计算中，合理的内存访问模式和数据布局可以极大地提升计算效率。  实验过程中使用了时间测量工具和多次重复测试，确保了数据的稳定性和准确性。这种方法不仅适用于本次实验，在其他性能测试和优化中同样具有重要参考价值。  深入理解计算机体系结构：  通过本次实验，对计算机体系结构中的Cache层次结构、TLB工作原理有了更深入的理解。这些知识不仅有助于优化程序性能，也为未来进一步研究和开发高效算法奠定了基础。 |

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