**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：谢弘烨 学号：2020151036 班级： 软工02**

**实验时间： 2022年 6 月 2日至 6月 20日**

**实验报告提交时间： 2022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？   **4、选做：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 | **0.0033** | **0.5286** | **4.7912** | **26.9899** | **70.4438** | **153.4221** | **288.1525** | | 优化算法执行时间 | **0.0034** | **0.4160** | **3.3160** | **11.4218** | **27.2721** | **52.3509** | **90.6125** | | 加速比  speedup | **0.98** | **1.27** | **1.44** | **2.36** | **2.58** | **2.93** | **3.18** |   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：  优化前的算法中矩阵采用了默认的行优先存储，导致了在进行矩阵乘法的过程中，取出第二个矩阵的某一列中的元素时需要取出该元素所在行的所有元素。  如：  易得结果中第一行第一列元素的值为：  在进行上述计算的过程中，优化前算法在取出第二个矩阵中第一列第一个元素5时，会取出该元素所在行的所有元素5 3 8存入Cache中。第二、三个元素同理。  由于优化前的算法中存在大量上述在Cache中读写数据的操作，消耗了大量计算机资源，导致执行时间较长。  在优化算法中，利用列优先的方式存储第二个矩阵，如此在乘法执行过程时仅需一次就可以将所需列中的元素存入Cache中，减少了Cache和内存之间的数据传输次数，大大提高了算法性能，减少了执行时间。  同时，在实验结果中可以看出随着矩阵大小增大，优化效果也更加明显。这可能是因为：矩阵越大，在读取列中元素时减少的数据传输次数就越多，优化效果也就越明显。  **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；   在计算机中普遍存在Cache结构用于解决CPU和主存之间速度不匹配的问题。同时不同Cache的结构、容量和性能之间也有差异。  今天的CPU中普遍存在L1、L2、L3三级缓存，三者容量和性能各不相同，需要设计实验测得三级缓存各自的容量大小。  又因为缓存在存取数据时以cache line为基本单位。在测得L1缓存大小后，可以进一步试探L1缓存中行的大小。当前处理的数据如果比行小，那么处理时间几乎相同，否则时间将会迅速增大。   1. 测量方案及代码；   缓存之间速度差异明显，因此当数据大小介于两级缓存之间时，程序运行时间会增大。  故设计数据量不同的数组进行求和，测得求和过程需要的时间。当时间发生大幅改变时，此时求和过程处理的数据量就是上一级缓存的大小。         1. 测试结果；   Cache：   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 尺寸/KB | **8** | **16** | **32** | **64** | **128** | **256** | | 时间 | **0.498371** | **0.494326** | **0.506066** | **0.503809** | **0.508738** | **0.53974** |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **384** | **512** | **768** | **1024** | **1536** | **2048** | | **0.635152** | **0.667122** | **0.691619** | **0.715149** | **0.736387** | **0.725322** |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **3072** | **4096** | **5120** | **6144** | **7168** | **8192** | | **0.746017** | **0.789466** | **0.824335** | **1.00931** | **1.30309** | **1.67611** |  |  |  |  |  | | --- | --- | --- | --- | | **10240** | **12288** | **16384** | **20480** | | **2.20185** | **2.3704** | **2.60386** | **2.76934** |   Cacheline：   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 尺寸/B | **1** | **2** | **4** | **8** | **16** | **32** | **64** | | 时间 | **0.170908** | **0.172894** | **0.171863** | **0.184543** | **0.214426** | **0.272834** | **0.439061** |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **96** | **128** | **192** | **256** | **512** | **1024** | **1536** | **2048** | | **0.593186** | **0.735834** | **0.903543** | **0.827641** | **0.905476** | **0.986259** | **1.01052** | **0.970536** |  1. 分析过程；   Cache：  可以发现红色数据增长不明显，基本可以确定此时数据没有超出当前缓存。因此可以猜测L1缓存大小在256KB到512KB之间，L2缓存大小在1.5MB到3MB之间，L3缓存大小在7MB到12MB之间。  Cache line：  可以发现在8B到32B之间时间有了明显增加，说明cache line大小应该介于8B和32B。   1. 验证实验结果。   观察系统信息可见CPU三级缓存大小分别为384KB，1.5MB，12MB。猜测基本正确。    观察CPU-Z软件中现实的CPU信息，可以看到一级缓存的行大小为32B，猜测基本正确。    **3、选做：尝试测量你的x86机器TLB有多大？** |
| **五、实验结论与心得体会**  在本次实验中，我意识到在代码编写时要尽量考虑cache的优化，按行优先存储可以相应的提高性能。  除此之外，我还尝试运用cache的知识进行代码优化，同时尝试编程实现真实x86机器cache的测量，加深了我对cache运行原理的理解。 |

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  2022年 月 日 |
| 备注： |