南開大學
Nankai University

计算机学院

并行程序设计实验报告

cache 优化与超标量优化

姓名：张传畅

学号：2110805

专业：计算机科学与技术

2023 年 3 月 10 日

目录

1 实验一：n*n 矩阵与向量内积

1.1 实验环境

本次实验是 windows11 平台上，用 intel 的 CPU i7-12700H，配有 14 核 20 线程。使用 code::blocks，搭载 TDM-GCC 编译器。CPU 每个核心 48KB 的指令一级缓存以及 48KB 的数据一级缓存，有 512KB 的私有二级缓存，有 24MB 的三级缓存。

Gitee 链接：张传畅的 Gitee

1.2 实验摘要

计算机为解决冯诺依曼瓶颈，将一些数据和代码存放在靠近 cpu 寄存器的特殊容器中，也叫缓存（cache）。在这段区域，相对于主存，cpu 能更快的访问其中的数据。

为了探究 cache 对程序运行速度的影响，本实验选择了 n*n 矩阵与向量内积作为研究对象，设计了平凡算法与 cache 优化算法，在程序运行时间上对二者做了对比，并使用性能测量工具 vtune 对两个程序的性能作具体探究。

1.3 算法设计

1.3.1 平凡算法

n*n 矩阵与向量的内积利用二重循环，逐列对矩阵元素进行访问，将矩阵的每一列与向量作相应的运算，一次外部循环计算出内积的一个值，运算结果储存在结果向量的相应位置。

1.3.2 cache 优化算法

同样是利用两重循环，对矩阵的元素作逐行访问，一部外部循环计算不出内积的任何一个值，只是其中一部分，当整个外部循环结束时，整个内积运算结束。

1.4 编程实现

1.4.1 平凡算法

逐列访问平凡算法

```
for(int i=0;i<MAXN;i++)//MAXN表示数组规模
{
    sum[i]=0;
    for(int j=0;j<MAXN;j++)
    {
        sum[i]+=b[j][i]*a[j];
    }
}
```

1.4.2 cache 优化算法

逐行访问 cache 算法

```
for(int i=0;i<MAXN;i++)//MAXN表示数组规模
{
    sum[i]=0;
}
for(int j=0;j<MAXN;j++)
{
    for(int i=0;i<MAXN;i++)
    {
        sum[i]+=b[j][i]*a[j];
    }
}
```

1.5 性能测试

用 MAXN 表示数组长度。

1.5.1 MAXN 为 1000

次数	1	2	3	4	5	6	7	8	9	10	平均
平凡算法/s	0.017	0.012	0.017	0.018	0.017	0.011	0.016	0.014	0.010	0.013	0.0145
cache 优化算法/s	0.012	0.011	0.011	0.017	0.013	0.010	0.013	0.014	0.013	0.013	0.0127

1.5.2 MAXN 为 3000

次数	1	2	3	4	5	6	7	8	9	10	平均
平凡算法/s	0.061	0.054	0.059	0.053	0.062	0.056	0.057	0.057	0.058	0.054	0.0571
cache 优化算法/s	0.026	0.033	0.032	0.031	0.031	0.032	0.030	0.027	0.044	0.032	0.0318

1.5.3 MAXN 为 5000

次数	1	2	3	4	5	6	7	8	9	10	平均
平凡算法/s	0.142	0.149	0.140	0.147	0.160	0.155	0.148	0.152	0.149	0.157	0.1519
cache 优化算法/s	0.062	0.063	0.057	0.070	0.062	0.054	0.047	0.061	0.057	0.050	0.0583

1.5.4 MAXN 为 10000

次数	1	2	3	4	5	6	7	8	9	10	平均
平凡算法	0.556	0.567	0.561	0.552	0.570	0.564	0.565	0.567	0.575	0.568	0.5635
cache 优化算法	0.188	0.182	0.188	0.181	0.183	0.180	0.178	0.176	0.174	0.178	0.1802

1.5.5 归纳

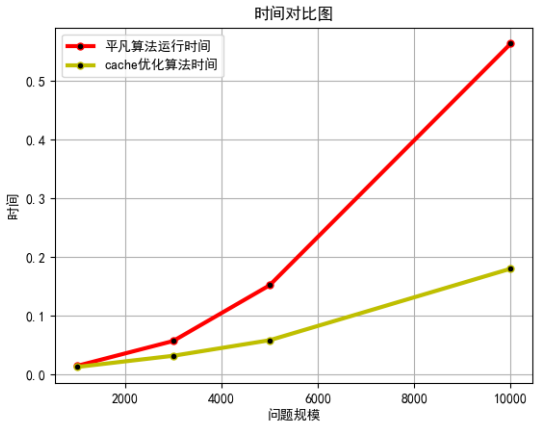

图 1.1: 优化前后时间对比

如上图所示：第一、相同算法的运行速度随着数组规模的增大而增大。第二、在问题规模较小的情况下，cache 优化算法相对平凡算法并没有展现出太多的优势。随着问题规模增大，cache 优化算法的运行时间上的优势越来越大。

1.6 Profiling

问题规模	平凡 L1cacheHIT	平凡 L1cacheMISS	优化 L1cacheHIT	优化 L1cacheMISS
1000	4	3	3	1
3000	12	25	9	1
5000	29	71	39	0
10000	105	264	96	37

问题规模	平凡 L2cacheHIT	平凡 L2cacheMISS	优化 L2cacheHIT	优化 L2cacheMISS
1000	4	0	0	0
3000	23	3	0	0
5000	68	10	0	1
10000	151	231	32	4

问题规模	平凡 L3cacheHIT	平凡 L3cacheMISS	优化 L3cacheHIT	优化 L3cacheMISS
1000	0	1	0	0
3000	1	5	0	0
5000	0	16	0	0
10000	185	74	0	8

1.7 结果分析

理论上说，二维矩阵中的数组是按行存储的，而计算机回一次将多个数据同时拿到 cache 中存储，如果 cache 中没有需要的元素时，就需要访问内存，将需要的数据替换进去。在平凡算法中，二维数组的数据读取是按照列进行的，读取的前后两个元素的存储位置有很大差距，cache 缺失次数较多，需要经常访问内存。

而 cahce 优化算法中，二维数组的数据读取是按照行进行的，访问的元素会有很大一部分在 cache 中能找到，cache 缺失次数较少，从而节省了运行时间。

从 vtune 程序测量的结果当中也确实能够反映出来，总的来说 cache 优化算法三级 cache 的命中率较平凡算法来说有着提升，也充分支持了理论分析结果。

2 超标量优化

2.1 实验环境

本次实验是 windows11 平台上，用 intel 的 CPU i7-12700H，配有 14 核 20 线程。使用 code::blocks，搭载 TDM-GCC 编译器。CPU 每个核心 48KB 的指令一级缓存以及 48KB 的数据一级缓存，有 512KB 的私有二级缓存，有 24MB 的三级缓存。

Gitee 链接：张传畅的 Gitee

2.2 实验摘要

对比普通的链式算法，循环展开算法能更好地利用 CPU 超标量架构，在一次循环步中能完成多步操作，大大降低循环周期。

本实验选取 n 个数求和作为研究对象，同样设计了平凡算法以及优化算法，在运行时间上对二者进行对比，并使用性能测量工具对二者进行具体研究。

2.3 算法设计

2.3.1 平凡算法

从头开始对整个数组遍历，依次累加到指定变量。

2.3.2 循环展开算法

执行次数为数组长度的一半，在每个循环步内内进行两次加法运算，两个相邻的数组元素依次累加到两个不同的变量上，最后在把两个变量相加。

2.4 编程实现

利用 MAXN 表示累加数据数量大小。

2.4.1 平凡算法

平凡算法

```
1 for(int i=0;i<MAXN;i++)//用MAXN表示有MAXN个数相加
```

```
{
    sum+=a[i];
}
```

2.4.2 循环展开算法

循环展开算法

```
for(int i=0;i<MAXN;i+=2)//用MAXN表示有MAXN个数相加
{
    sum1+=a[i];
    sum2+=a[i+1];
}
sum+=sum1+sum2;
```

2.5 性能测量

我们用 MAXN 表示累加数据长度，对数据的核心算法作循环一万次处理，以使得差距更加明显。

2.5.1 MAXN=10000

次数	1	2	3	4	5	6	7	8	9	10	平均
平凡算法/s	0.039	0.041	0.040	0.040	0.040	0.041	0.039	0.040	0.040	0.041	0.0401
循环展开算法/s	0.029	0.029	0.029	0.028	0.027	0.028	0.029	0.027	0.028	0.028	0.0282

2.5.2 MAXN=30000

次数	1	2	3	4	5	6	7	8	9	10	平均
平凡算法/s	0.085	0.085	0.088	0.087	0.089	0.088	0.086	0.086	0.087	0.088	0.0869
循环展开算法/s	0.054	0.054	0.055	0.054	0.055	0.054	0.054	0.055	0.053	0.054	0.0542

2.5.3 MAXN=50000

次数	1	2	3	4	5	6	7	8	9	10	平均
平凡算法/s	0.136	0.135	0.139	0.134	0.140	0.136	0.135	0.139	0.139	0.132	0.1365
循环展开算法/s	0.080	0.080	0.079	0.080	0.081	0.079	0.081	0.083	0.079	0.082	0.0804

2.5.4 MAXN=100000

次数	1	2	3	4	5	6	7	8	9	10	平均
平凡算法/s	0.258	0.259	0.257	0.254	0.253	0.261	0.258	0.261	0.262	0.262	0.2585
循环展开算法/s	0.143	0.142	0.138	0.142	0.140	0.142	0.145	0.143	0.142	0.141	0.1418

2.5.5 归纳

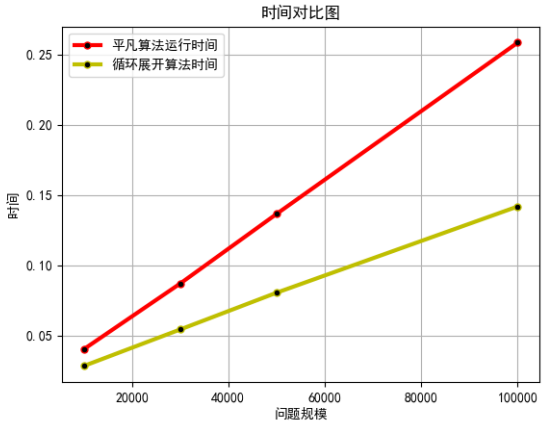

图 2.2: 优化前后时间对比

如下图所示：第一、运算时间基本与数据累加数目成线性关系。第二、在数据规模小的时候，平凡算法和优化算法的差距很小，甚至平凡算法更快。第三，随着数据规模增大，优化算法的优势更能显示出来。

2.6 Profiling

2.6.1 平凡算法

规模	Clockticks	Instructions Retired	CPI
10000	124200000	510300000	0.225
30000	321300000	1512000000	0.212
50000	534600000	2513700000	0.213
100000	1058400000	5019300000	0.211

2.6.2 循环展开算法

规模	Clockticks	Instructions Retired	CPI
10000	70200000	359100000	0.195
30000	194400000	1061100000	0.183
50000	313200000	1763100000	0.178
100000	577800000	3512700000	0.164

2.7 结果分析

理论上说，使用循环展开算法，能够在一个循环步内执行多条指令，减小程序的 CPI，并且能够减少循环周期，从而加快程序运行。

而根据 vtune 的测量结果来看，优化算法与平凡算法的指令数目大幅度减少，并且优化算法的时钟周期数大概为平凡算法的一半，CPI 的测量结果也有所降低，也充分支持了理论分析结果。