
南开大学

Nankai University

# 南　　开　　大　　学

## 计　算　机　学　院

**并行程序设计实验报告**

---

# 体系结构相关及性能测试

---

刘沿辰

年级：2020 级

专业：计算机科学与技术

指导教师：王刚

2023 年 3 月 12 日

# 摘要

本实验针对矩阵乘法和 n 数累加两个问题作出并行算法的优化，优化思路分别是 Cache 优化和超标量优化，并对两种算法优化前后的时间性能作出比较并分析。实验平台为 x86 个人电脑，对于 Cache 优化算法，额外使用了 Vtune 进行了硬件占用率的监测，验证了算法对于 Cache 命中率的优化；对于超标量算法，额外进行了循环耗时的分析以精确影响时间性能的因素。最后对两种优化算法做出总结。


**关键字：并行，Cache 优化，超标量优化**


# 目录

# 一、 实验要求

## (一) 问题描述

计算给定 1*n 向量与 n*n 矩阵的内积，考虑两种算法的设计思路：

1. 逐列访问元素的平凡算法；
2. Cache 优化算法；

计算 n 个数的和，考虑两种算法设计思路：

1. 逐个累加的平凡算法；
2. 超标量优化算法；

# 二、 实验平台配置

- x86 平台：搭载 Windows11 的个人 PC
- CPU：Intel Core i7-10510U
- CPU 主频：2.300GHz
- L1 cache：256K
- L2 cache：1M
- L3 cache：8M



# 三、 实验设计

## (一) 性能测试方式

- 程序运行时间：本次性能测试引入了 Windows 平台下的 QueryPerformance() 高精度计时来更精确地记录程序运行时长。为了避免实验结果的偶然性，报告中的实验结果均为 5 次测试的平均值。
- Cache 命中与干扰：Cache 中存储了程序需要的数据称为 Cache 命中，否则不命中且进入下一层次 Cache 寻找。本机的 L3 Cache 大小为 8M，大约能存储 $2 * 10^6$ 个 int。因此为了避免 Cache 对实验结果的影响，每次测试之前都会加载一个 1500 阶以上的无关矩阵来清空 Cache.

## (二) 计算给定 1*n 向量与 n*n 矩阵的内积

### 1. 算法设计

平凡算法的设计比较简单，按照计算习惯，循环计算矩阵 n 次，第 i 次循环得到第 i 列的结果，代码如下：

逐列访问平凡算法

```
void nor_nul(int* a, int** b)
{
    int sum[SIZE];
    for (int i = 0; i < SIZE; i++)
    {
        sum[i] = 0;
        for (int j = 0; j < SIZE; j++)
            sum[i] += b[j][i] * a[j];
    }
}
```

C/C++ 中二维数组按行存储，平凡算法的程序中一次需要多次访问每一列的数据，空间局部性较差。当缓存中无法一次性加载时，按列访问会导致 Cache 频繁换入换出。于是考虑将程序读取的数据更改为一整行，循环矩阵 n 次，每一次循环完成矩阵一行与向量的乘积并加入 sum 结果中。

Cache 优化算法

```
void cache_nul(int* a, int** b)
{
    int sum[SIZE];
    for (int i = 0; i < SIZE; i++)
        sum[i] = 0;
    for (int j = 0; j < SIZE; j++)
    {
        for (int i = 0; i < SIZE; i++)
            sum[i] += b[j][i] * a[j];
    }
}
```



### 2. 实验结果

要探究平凡算法和 Cache 优化算法的结果，我选择观察矩阵维度（n）与两算法耗时比值的关系来探究算法运行效率。实验中，在不同的 n 下两者耗时的比值变化关系不同，此处分为 n 较小（1000 以下）和 n 较大（1000 以上）来探讨，部分实验比值如下表所示。

| n | 耗时比值 |
|---|---|
| 10 | 1.0113 |
| 30 | 1.0200 |
| 50 | 1.0377 |
| 70 | 1.0478 |
| 100 | 1.0889 |
| 200 | 1.1021 |
| 300 | 1.2022 |
| 400 | 1.3320 |
| 500 | 1.4299 |
| 600 | 1.6720 |
| 700 | 1.8911 |
| 800 | 2.0201 |
| 1000 | 2.401 |

表 1: n 较小时性能测试结果

趋势变化如图1所示，可见此时比值增长趋近于线性。

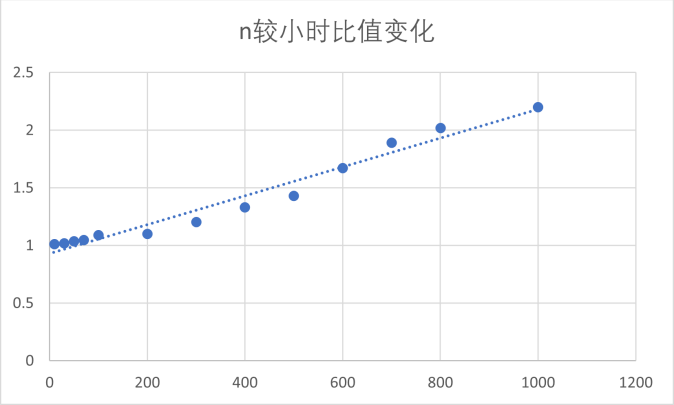


图 1: n 较小时比值变化关系

当 n 较大时（1000-10000），程序运行耗时比值如下表所示。

| n | 耗时比值 |
|---|---|
| 1000 | 2.4010 |
| 2000 | 2.4778 |
| 3000 | 4.9024 |
| 4000 | 6.6444 |
| 5000 | 9.0331 |
| 6000 | 9.2990 |
| 7000 | 10.0074 |
| 8000 | 12.0897 |
| 9000 | 17.3312 |
| 10000 | 21.7370 |

表 2: n 较大时性能测试结果

耗时比值如图2所示。

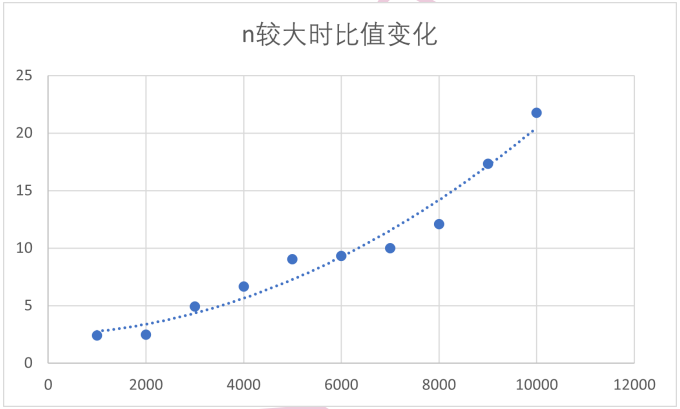


图 2: n 较大时比值变化关系

### 3. 结果分析

对于平凡算法和 Cache 优化算法，在矩阵规模较小（n < 1000）时，由于 Cache 还能存储所有数据，时间消耗趋近于理论推测的 $y = a * n^2$，此时两者的速度差异并不大。但平凡算法在问题规模变大后，缓存的命中率开始下降，其缺页带来的时间消耗逐渐变长，程序需要频繁访存导致性能大幅下降，使得 Cache 优化算法在问题规模变大后明显优于平凡算法，且两者速度之差逐渐趋近 $y = a * n^2$。使用 Vtune 工具跟踪 Cache 命中率结果如下表。

| n | L1 Cache | L2 Cache | L3 Cache |
|---|---|---|---|
| 100 | 99.81% | 99.89% | 99.91% |
| 500 | 94.19% | 99.65% | 99.90% |
| 1000 | 90.33% | 99.13% | 99.90% |
| 10000 | 90.98% | 91.32% | 99.46% |

表 3: 平凡算法 Cache 命中率

平凡算法在矩阵阶数较小时 Cache 由于能存储下矩阵，命中率很高。但是随着问题规模的增大，L1 和 L2 Cache 的命中率下降至 91% 左右，L3 Cache 下降至 99.5% 左右。作为对比的 Cache 优化算法命中率如下表所示。

| n | L1 Cache | L2 Cache | L3 Cache |
|---|---|---|---|
| 100 | 99.74% | 99.87% | 99.96% |
| 500 | 99.83% | 99.87% | 99.97% |
| 1000 | 99.81% | 99.90% | 99.97% |
| 10000 | 99.88% | 99.93% | 99.97% |

表 4: Cache 优化算法 Cache 命中率

可见 Cache 优化算法的命中率始终维持在 99.7% 以上，并没有因为矩阵规模增大而出现明显的性能下滑问题，实验结果与上文的分析结果相符合。有意思的是，在采用不同优化级别对代码进行优化时，程序实际表现出的性能差距并不大，但是编译优化后的 Cache 程序在 n 较大时和平凡算法之间出现了更大的差距。

## (三) 计算 n 个数的和

### 1. 算法设计

类似于矩阵乘法，在此给出平凡算法与超标量优化算法的代码。

逐个相加平凡算法

```
void nor_plu(int* a)
{
    int sum = 0;
    for (int i = 0; i < SIZE; i++)
    {
        sum += a[i];
    }
}
```

对于超标量优化代码，本次实验采取四路超标量进行累加，即用四个变量线性累加所有数，四路之间没有依赖关系，可以并行执行，最后对四个数求和。理论分析可知，该算法大约可以将性能提升四倍（计算机线程数量大于 4）。为了算法的正确执行，数组长度会被预先扩充为 4 的倍数，多余的值赋 0。

四路超标量优化算法

```
void ove_plu(int* a)
{
    int sum = 0, sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;
    for (int i = 0; i < SIZE; i += 4)
    {
        sum1 += a[i];
        sum2 += a[i + 1];
        sum3 += a[i + 2];
        sum4 += a[i + 3];
```

```
    }
    sum = sum1 + sum2 + sum3 + sum4;
}
```

### 2. 实验结果

本实验依然以平凡算法耗时和优化算法耗时之比作为测试指标，结果如下表。

| n | 耗时比值 |
|---|---|
| 10 | 1.0000 |
| 50 | 1.0000 |
| 100 | 1.0000 |
| 1000 | 2.0001 |
| 10000 | 3.4500 |
| 100000 | 2.9430 |
| 1000000 | 3.0848 |
| 10000000 | 2.9817 |
| 100000000 | 3.1166 |

表 5: 平凡算法与优化算法耗时之比 (4 线程)

### 3. 结果分析

在数组规模太小时，由于两种算法时间之差超出了测量精度范围，因此表格中耗时比例显示均为 1.0，此时的时间差别可以忽略不计。平凡算法所需循环次数为每线程 n 次，超标量优化后理论上将循环次数降低到每个线程 n/4 次，可能由于并行带来的额外时间代价以及缺页等必要时间代价的原因，实际程序运行效率在 3 倍左右。

## 四、 总结

本实验主要就 Cache 优化以及超标量优化两种优化方式进行了测试。Cache 优化依据于程序设计的空间局部性原理，将原本的空间局部性很差的代码优化为空间局部性较优的代码，实现了运行效率的大幅提升。而超标量优化算法源于处理器的多核架构，通过分多个线程充分利用处理器的多核心优势提高程序性能，由于代码本身不是按照并行样式书写，程序实现的是更底层的并行，由于对变量 i 的依赖导致每个线程每次循环都需要等待，可以考虑直接把每个线程解耦合单独运行来优化这个算法。