# 并行程序设计报告二

# CPU架构编程

## 2313911 黄尚扬 2025.3.21

### Git链接：https://gitee.com/hsy23333/parallel-jobs

### 实验一：n\*n矩阵与向量内积

#### 算法设计

##### 平凡算法设计思路

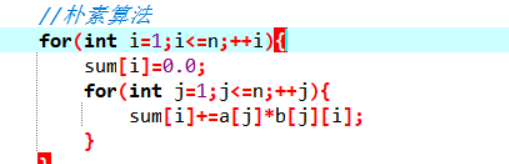
暴力地，我们采用模拟手算的过程，对于结果向量的每一位，遍历矩阵的每一列，分别累加，算完每个结果后跳到下一个要计算的位置。

##### Cache优化算法设计思路

考虑到行优先的cache存取方式，在遍历时优先按照行来读取；每次遍历矩阵一行，叠加一部分到结果向量的对应位置上；最后遍历完整个矩阵得到结果。

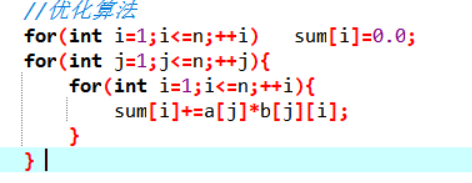
#### 编程实现

##### 平凡算法



实现如图，非常暴力，不做解释。

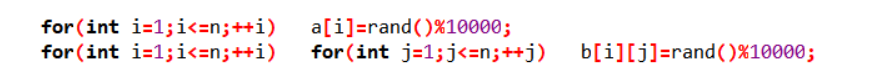
##### Cache优化算法



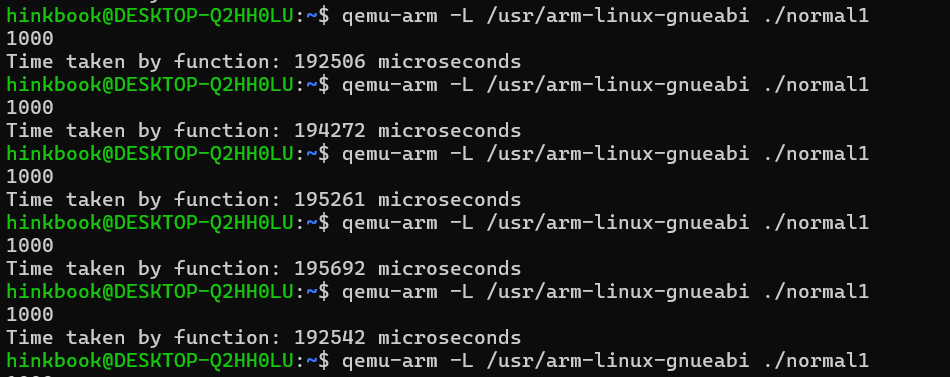
如图，改为了行优先读取，结果很显然不变；而同时利用了cache的存储特点。

#### 性能测试

在输入n后，采取随机生成的方式模拟相应规模的执行，如下：



笔者测试的n规模有：50，100，200，300，500，1000。每一算法每一规模重复测试5次，例子如下：



如图所示，这是优化算法n=1000的测试结果，随后取平均值即可得到最终结果。**注意，由于是使用rand()函数简单取随机数，所以可以保证相同规模每次实验数据相同。**

##### 平凡算法

表1：测试结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 时间1（ms） | 时间2（ms） | 时间3（ms） | 时间4（ms） | 时间5（ms） | 平均（ms） |
| 50 | 1.292 | 1.323 | 1.156 | 1.140 | 1.186 | 1.2194 |
| 100 | 2.613 | 2.539 | 2.772 | 2.521 | 2.483 | 2.5856 |
| 200 | 8.320 | 8.290 | 8.134 | 9.886 | 8.113 | 8.5486 |
| 300 | 17.908 | 23.007 | 16.950 | 17.762 | 17.222 | 18.5698 |
| 500 | 48.817 | 47.713 | 48.064 | 48.010 | 49.563 | 48.4334 |
| 1000 | 197.932 | 200.063 | 195.827 | 197.168 | 187.095 | 195.617 |

##### Cache优化算法

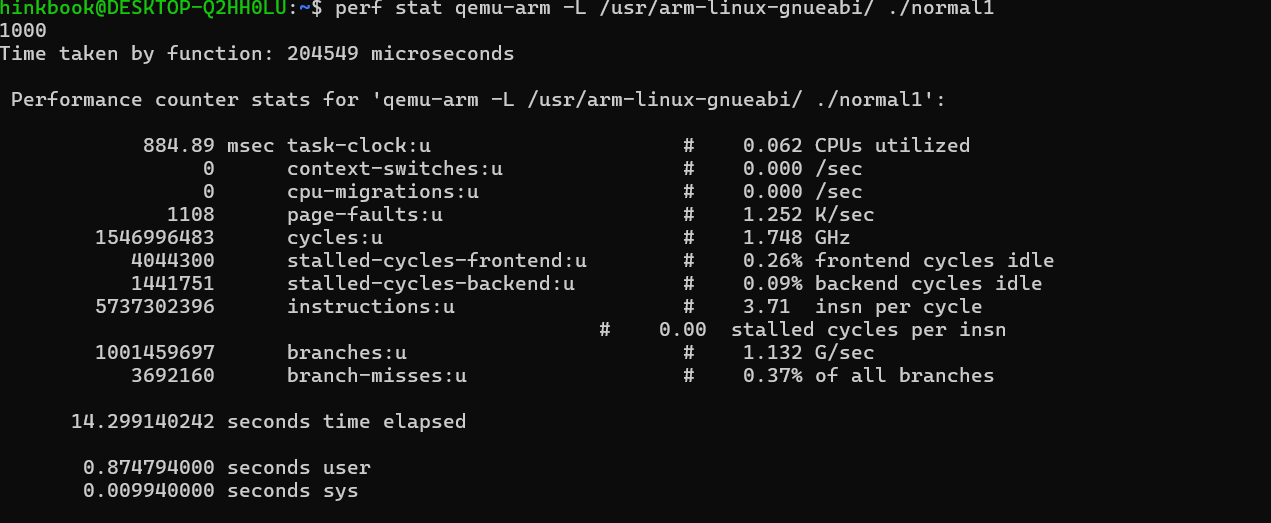
表2：测试结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 时间1（ms） | 时间2（ms） | 时间3（ms） | 时间4（ms） | 时间5（ms） | 平均（ms） |
| 50 | 1.147 | 1.332 | 1.227 | 1.150 | 1.789 | 1.329 |
| 100 | 2.524 | 2.526 | 2.501 | 2.670 | 2.484 | 2.5410 |
| 200 | 8.840 | 7.946 | 8.457 | 7.917 | 7.912 | 8.2144 |
| 300 | 17.216 | 17.853 | 17.024 | 17.137 | 17.028 | 17.2516 |
| 500 | 46.693 | 46.733 | 47.008 | 46.469 | 47.786 | 46.7978 |
| 1000 | 192.506 | 194.272 | 195.261 | 195.682 | 192.542 | 194.0526 |

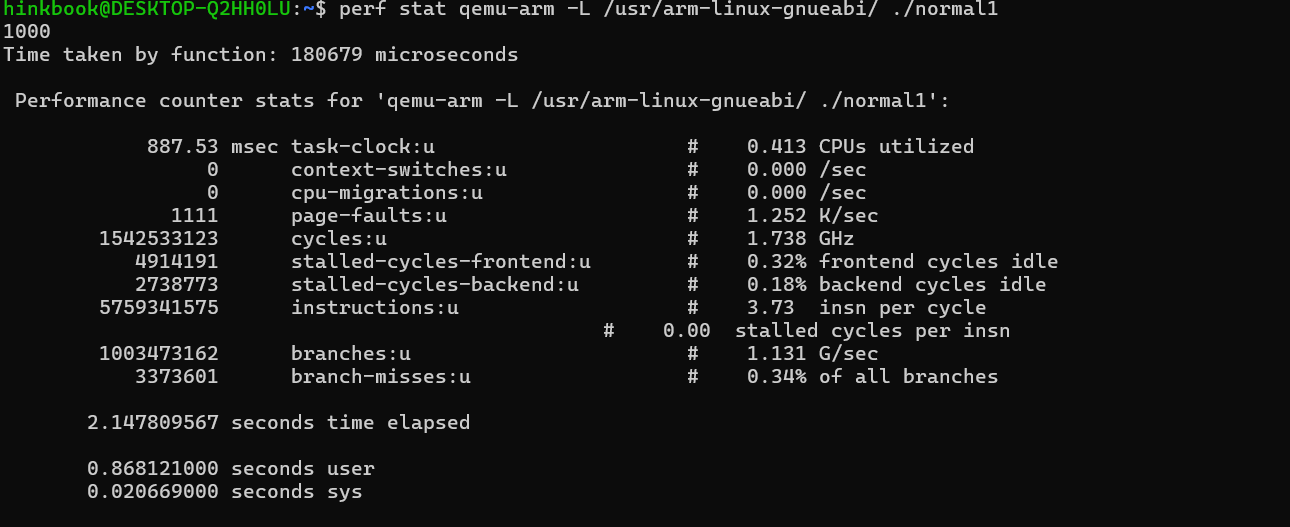
#### Profiling

这里我们皆以n=1000的情况为例进行性能分析。本模块基于perf，仅放出实验结果截图；而具体的分析在下一模块。**注：所有编译出来的文件名皆为normal1，此处是进行了重新编译后的实验结果，并不代表二者是同一个程序。**

##### 平凡算法



##### Cache优化算法



#### 结果分析

总体而言，cache优化确实有帮助，但效果有限，优化效率基本在1%量级，如下图。推测原因是编译器采用了较为先进的优化，使得朴素算法能够获得比预期更高的效率。

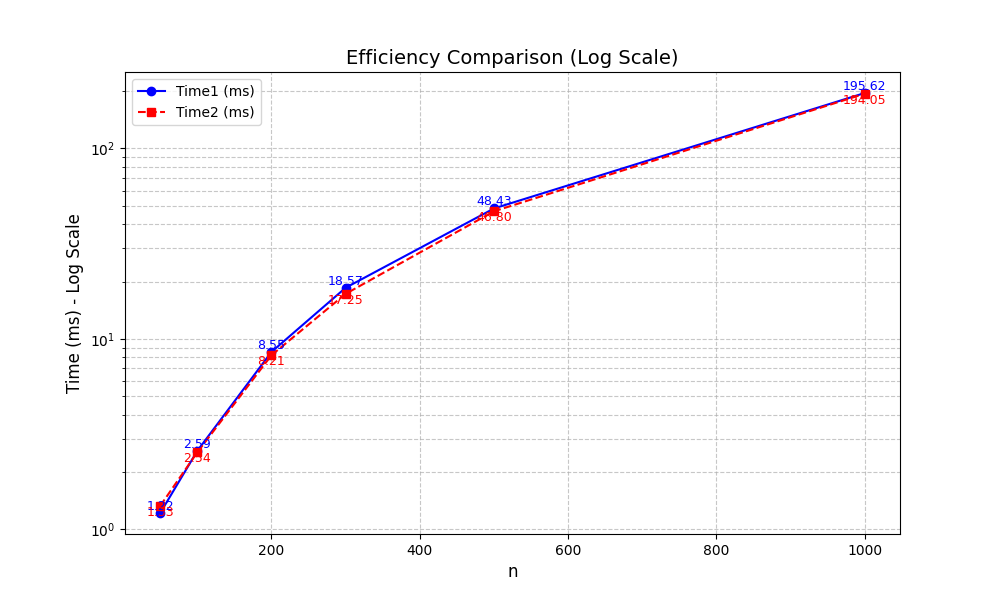


表 3 性能对比

##### 平凡算法

基于perf，我们可以看到，在这次测试中耗时约204微秒，与表内数据接近。分支预测3292160，占比0.37%。

##### Cache优化算法

基于perf，本次测试耗时约181微秒，稍稍快于先前测试结果。推测原因是在进行大量计算之后，cache命中率和分支预测带来的性能提升显著变强。分支3373601，占比0.34%，可以发现确实效率更高。

### 实验二：n个数求和

#### 算法设计

##### 平凡算法设计思路

根据下标顺序遍历数组，逐个累加到sum中，十分暴力，不再赘述。

##### Cache优化算法设计思路

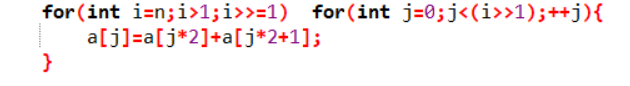
基于类似归并或分治的思维，每次执行当前规模/2大小的累加；再利用递归逐层加和。共执行logn级别层数。累加后仍然是O（n）的算法。

#### 编程实现

##### 平凡算法



##### Cache优化算法



#### 性能测试

和上一个实验相似，我们采取多次测量取平均值的方式测试。进一步地，由于上一个是n^2复杂度，本次为n复杂度，于是为凸显差别，本次n的规模扩大为：10000，100000，1000000，2000000，5000000，10000000。

##### 平凡算法

表4：测试结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 时间1（ms） | 时间2（ms） | 时间3（ms） | 时间4（ms） | 时间5（ms） | 平均（ms） |
| 10000 | 2.874 | 2.913 | 2.777 | 2.900 | 2.790 | 2.8508 |
| 100000 | 3.268 | 3.206 | 3.152 | 3.290 | 3.966 | 3.3764 |
| 1000000 | 7.879 | 7.590 | 7.449 | 7.685 | 7.475 | 7.6156 |
| 2000000 | 10.586 | 10.270 | 11.123 | 11.526 | 11.267 | 10.9544 |
| 5000000 | 22.523 | 21.353 | 21.869 | 22.998 | 21.349 | 22.0184 |
| 10000000 | 39.948 | 41.863 | 41.982 | 41.024 | 39.741 | 40.9116 |

##### Cache优化算法

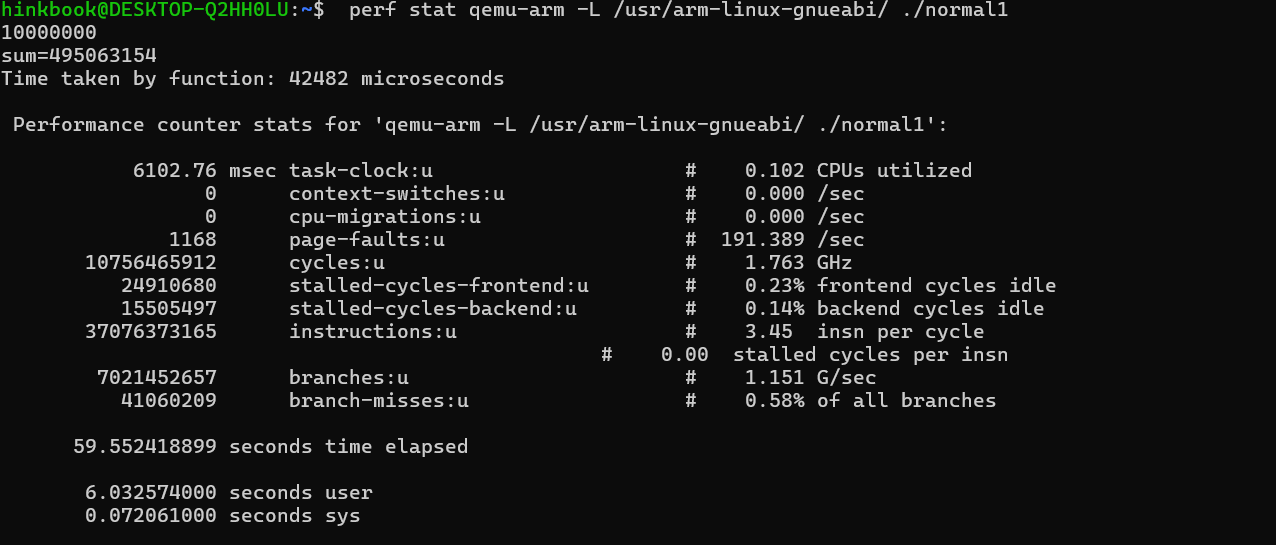
表5：测试结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 时间1（ms） | 时间2（ms） | 时间3（ms） | 时间4（ms） | 时间5（ms） | 平均（ms） |
| 10000 | 2.888 | 2.963 | 2.874 | 2.910 | 2.835 | 2.894 |
| 100000 | 3.627 | 3.542 | 3.534 | 3.474 | 3.643 | 3.564 |
| 1000000 | 9.131 | 8.660 | 8.339 | 8.938 | 8.903 | 8.7942 |
| 2000000 | 15.052 | 15.026 | 14.527 | 14.387 | 14.808 | 14.76 |
| 5000000 | 33.511 | 32.269 | 32.909 | 32.445 | 32.791 | 32.785 |
| 10000000 | 63.373 | 61.134 | 61.602 | 61.294 | 62.476 | 61.9758 |

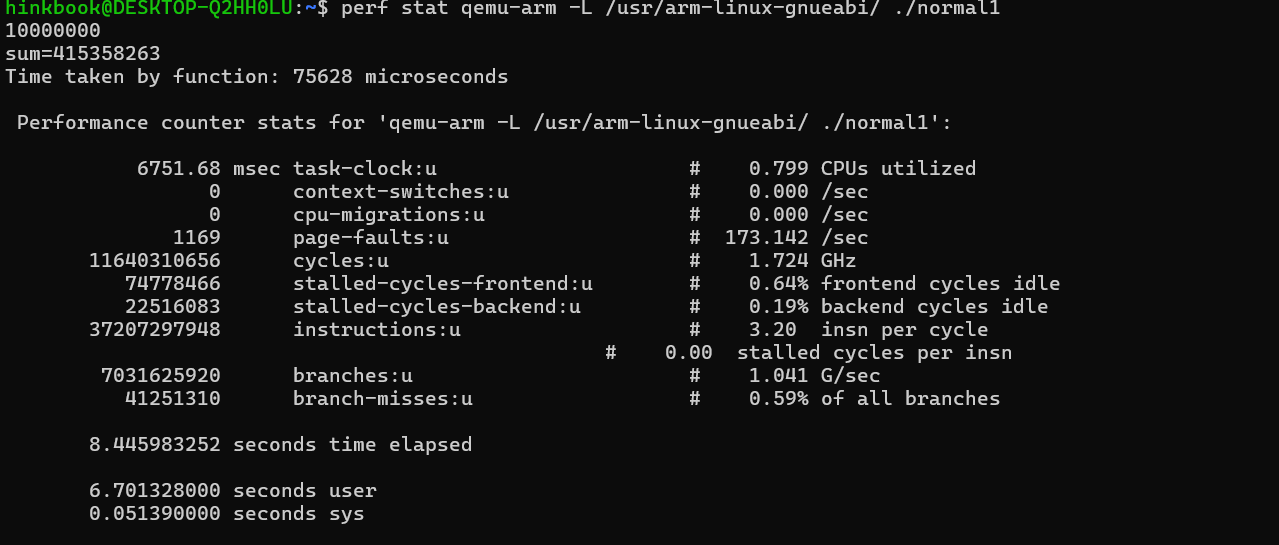
#### Profiling

和上一个实验相同，这里仅采用1e7规模进行性能测试，以凸显差距。

##### 平凡算法



##### Cache优化算法



#### 结果分析

在这里，我们惊奇地发现：Cache优化居然比原先慢出了快一倍！笔者尝试了使用递归和循环两种方式，结果都差不多。于是，我们可以认为：要么耗时的计算方式有歧义，要么这种优化在一定程度上是成问题的。笔者将在下方试图解释。

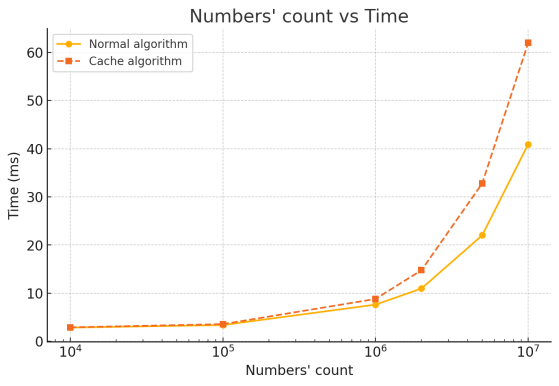


表 6 耗时分析

##### 平凡算法

基于perf，本次主函数耗时42微秒，而总执行耗时59.55秒，出现显著差异。这可能是由于CPU使用效率过低导致。而实际的用户时间为6.03秒，问题不大。

##### Cache优化算法

主函数耗时76微秒，总执行8.44秒，用户时间6.70秒，和前文结果基本一致：略慢于朴素算法。值得一提的是，该算法分支miss率为0.59%，略高于前者的0.58%。可能是这个原因。也有可能是随机数据恰巧使得该算法较劣。

### 三．实验总结与思考

Cache优化往往可以利用存取特性小规模地加速程序运行；但有时并非进行Cache优化就会提升效率，还需要考虑到实际数据的特性。