# 并行程序设计方法实验总结

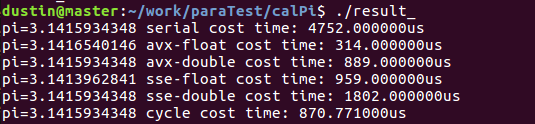
## **实验环境**

|  |  |
| --- | --- |
| 工具清单 | |
| 虚拟机 | VMware Workstation 14.1.1 build-7528167 |
| 系统镜像 | ubuntu-16.04.3-desktop-amd64.iso |
| 内核版本 | Linux ubuntu 4.13.0-41-generic |
| gcc版本 | version 5.4.0 20160609 |
| 处理器内核总数 | 4 |
| 运行内存 | 4GB |

## **专题：积分计算圆周率**

### 2.1向量优化

根据求pi公式，尝试进行编程实践，分别给出串行、sse-float、sse-double、avx-float、avx-double、循环展开六个版本的代码（result.c），观察实验结果：



由于计算pi时计算结果的精度和耗费时间与dt大小的选取密切相关，在该专题实验中统一取dt=1280000，最终发现符合实验指导书中所给规律：

**AVX-float>AVX-double≈SSE-float>SSE-double>serial**

在规定的五个版本基础之上，做了一次简单的循环展开测试，优化结果也比较明显。由于通过手工进行循环展开，进而为每一次循环体中更多指令的流水并行提供了可能，但由于指令系统和编译器本身的局限性，不可能完全依靠手工循环展开进行优化，最佳的方案需要取一个折中。

* 问题一：编译选项问题

最开始代码未整合，各个版本分别放在一个c文件中，这样在编译时加的优化编译选项有所区别，如对串行版本，仅添加-O3，对sse版本添加-O3和-msse2,对avx版本添加-O3和-mavx2，分别编译运行，发现结果非常不好。另外，在对串行版本代码的编译过程中发现仅添加-O3和添加-O3 -msse2 -mavx2的结果也大不相同，明显后者较于前者做了更多“潜”优化，这样的话我们得到的结果从试验方法上来讲就不科学。

考虑到机器每次运行时的环境状态差异，以及编译选项不统一引来的“潜优化”问题，最终选择统一到一个c文件中，并对整个文件添加统一的优化选项，并进行多次试验求其平均值，发现试验结果良好，比较有说服力，验证了向量优化的实用性。

**编译命令：gcc -mavx2 -msse2 -fopenmp -O3 result.c -o result**

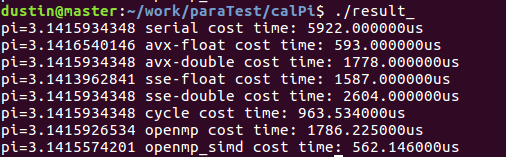
* 问题二：精度问题

double和float型数据在机器中占的字节数是不同的，复习计组过程中重新学习了浮点数float和double型数据在机器中的存储方式。根据计算公式理论上来讲，dt选择越大，试验结果越逼近pi的真实值；然而当dt较大时会发现float版本的试验结果与真实pi的值相差甚远，可以认为已经从“误差”级别转为“错误”级别。因此我们需要找到一个dt的折中值，最终选择dt=1280000，试验结果比较好。精度的问题在大型科学计算中显得异常重要，启示我们在未来追求速度和精度的计算之道中也要更加理性的选择，妥善考虑。

### 2.2 OpenMP优化

OpenMP使用FORK-JOIN并行执行模型。所有的OpenMP程序开始于一个单独的主线程。主线程会一直串行地执行，一旦遇到并行域会根据机器情况和编译配置属性实现多线程并行，特别适和对与该专题中类似计算密集型任务的优化。

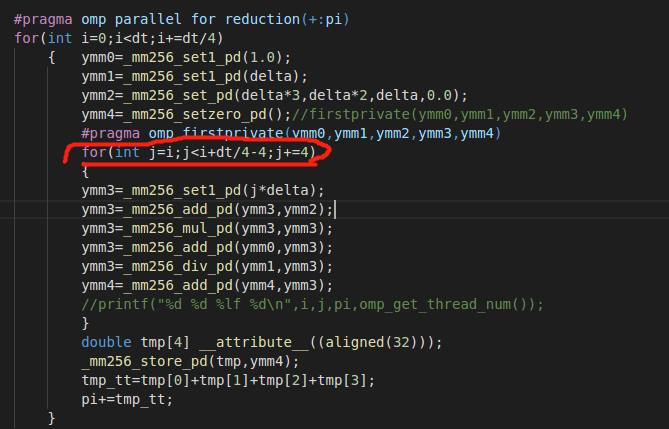
给出openmp和openmp+simd两种版本的代码，观察实验结果，符合预期：



实验中dt依然取1280000；

* 问题一：openmp任务两级并行（手工向量化）

对计算任务进行openmp优化时，首先考虑任务的划分问题，由于实验环境中处理器核心数为4，对dt级别的计算任务分解为4个dt/4级别的任务，然后针对dt/4大小的任务考虑进行simd的进一步优化；



图中二层循环处比较关键，由于已经给每个线程划分一定的计算任务量，所以需要设计一个统一的计算任务体针对于每一个线程能如期执行。最开始由于考虑不周，将j的初始值定为0，这就导致每个线程分的的计算任务量是不同的，最终pi的累加值远大于真实值；

在寻找问题原因时采用打印变量值进行一步步的摸索，最终将问题定位于二层循环处；最终，成功实现openmp+simd的两级并行优化，进一步增加优化程度。