**中山大学计算机院本科生实验报告**

**（2024学年春季学期）**

**课程名称：并行程序设计 批改人：**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验** | **Lab0-环境设置与串行矩阵乘法** | **专业（方向）** | **计算机科学与技术** |
| **学号** | **21307174** | **姓名** | **刘俊杰** |
| **Email** | **liujj255@mail2.sysu.edu.cn** | **完成日期** | **2024/3/25** |

# **实验目的**

本实验的主要目的是通过实现串行矩阵乘法，并对比不同版本的实现在性能上的差异，分析不同因素对最终性能的影响。具体来说，实验将探讨以下几个方面：

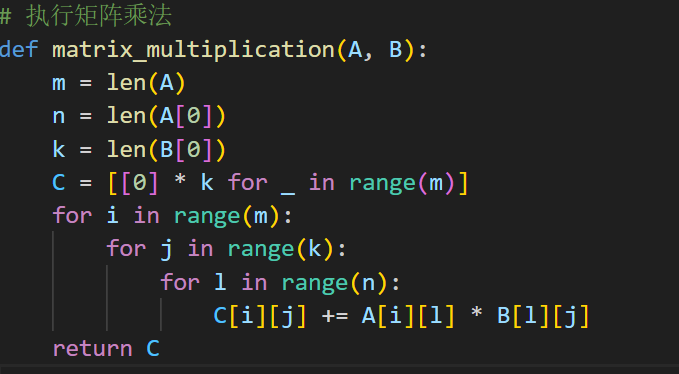
实现多个版本的串行矩阵乘法，包括Python、C/C++等不同语言的实现，以及针对C/C++代码的优化方式，如调整循环顺序、编译优化、循环展开等。

通过对比不同版本的运行时间，分析各种实现方式的性能差异，探讨不同因素对性能的影响。

计算相对加速比、绝对加速比、浮点性能（GFLOPS）以及峰值性能百分比等指标，评估各种实现方式的效果。

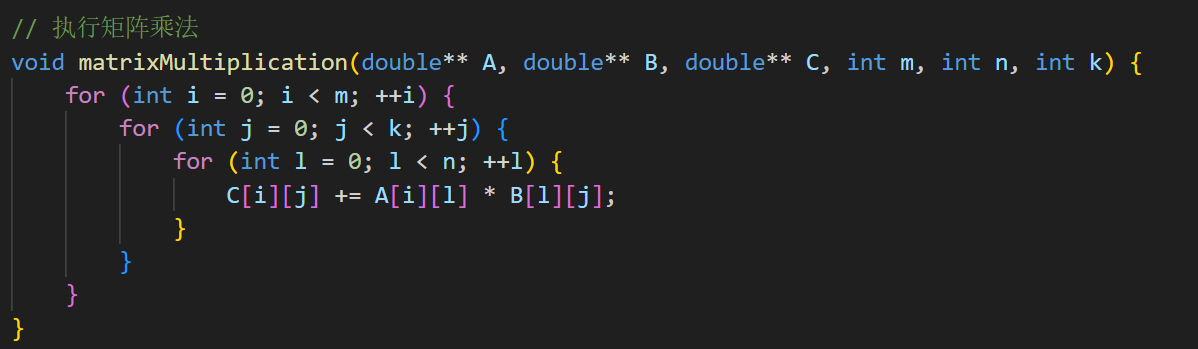
# **实验过程和核心代码**

## 2.1 Python版本实现



(这里初步的三重循环设计的原因在2.2中提及)

## 2.2 C/C++版本实现



为了比较方便后续的循环调整，这里**在矩阵乘法进行前先将矩阵C赋值为0矩阵**。

其次可以先通过**局部性**，确定三重循环的最外层循环为:

**for(int i = 0; i< m; ++i)**

因为将这一层循环放在最外层能够保证矩阵C和矩阵A不会出现频繁的未命中而导致的延时，否则

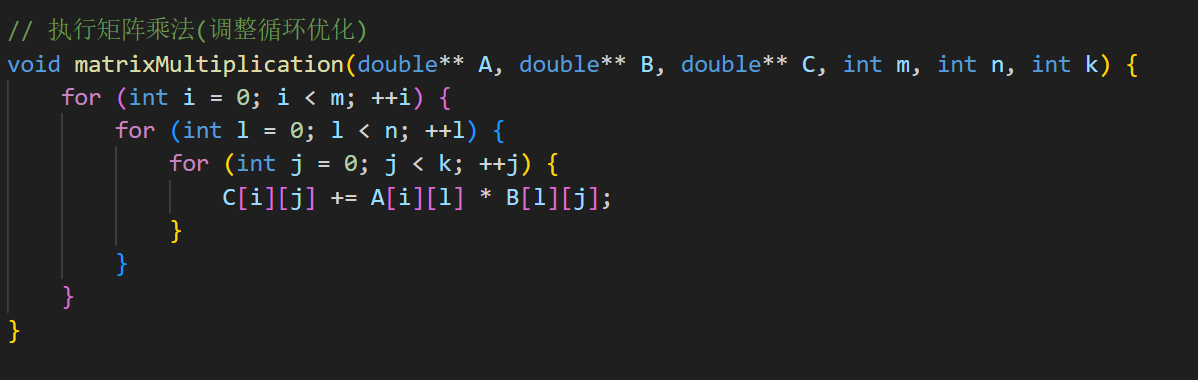
会出现多次类似取C[i]到取C[i+1]而导致的c[i+1]元素的未命中。

为了与后续比较这里先确定 **m -> k -> n**的三重循环顺序:

## 2.3 调整循环顺序

相较于上一步优化**m -> k -> n**的循环顺序，这里调整循环顺序为 **m -> n -> k**。

**因为将n放在第二重循环能相较于上一步更大的利用矩阵B的局部性。**



## 2.4 编译优化

### 编译命令:

#### -O1:

-O1 是一种基本的优化级别，它启用了一些简单的优化，例如删除未使用的变量、内联简单函数、去除无效的代码、简化表达式等。这些优化不会增加代码的大小，但可以提高代码的执行效率。

#### -O2:

-O2 是一个中等优化级别，它在 -O1 的基础上增加了更多的优化手段，例如更进一步的代码内联、函数调用优化、循环展开、消除冗余计算、基于数据流的优化等。这些优化可以显著提高程序的执行速度，但可能会增加编译时间和生成的代码大小。

#### -O3:

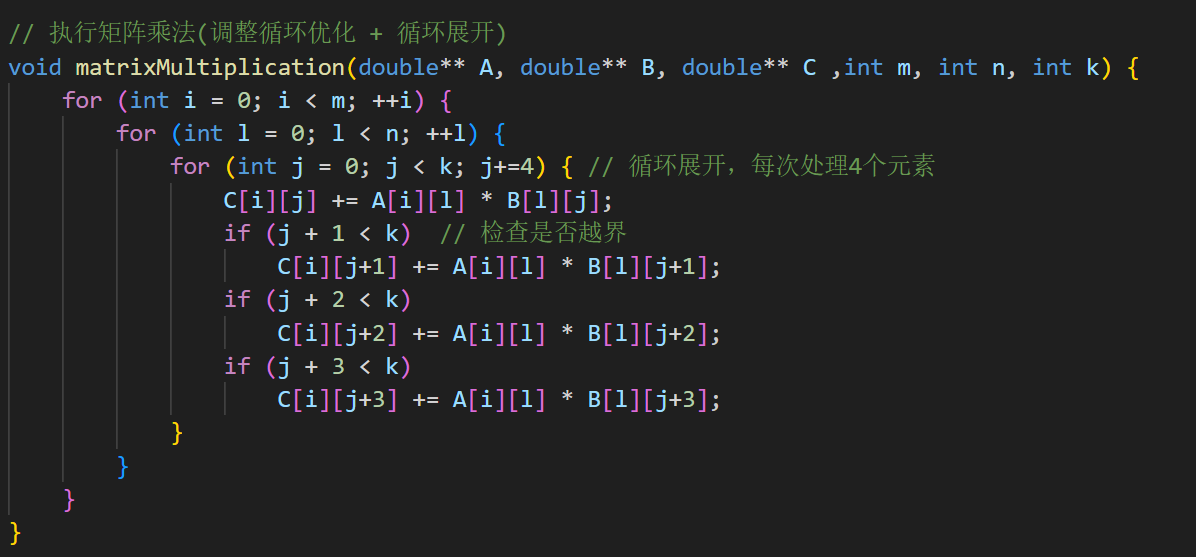
-O3 是最高级别的优化选项，它在 -O2 的基础上进行了更加激进的优化，例如更大规模的循环展开、更多的向量化操作、更深入的函数内联、更多的优化阶段等。这些优化可能会导致编译时间的显著增加，以及生成的代码大小的增加，但通常能够带来最大的性能提升。

## 2.5 循环展开

循环展开，英文中称Loop unwinding或loop unrolling，是一种牺牲程序的尺寸来加快程序的执行速度的优化方法。可以由程序员完成，也可由编译器自动优化完成。循环展开最常用来降低循环开销，为具有多个功能单元的处理器提供指令级并行。也有利于指令流水线的调度。

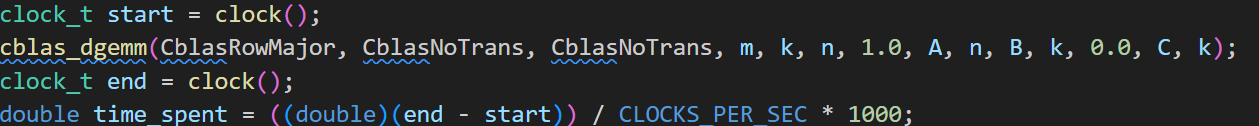
循环展开对程序性能有着很重要的影响，**可以减少分支预测错误次数，增加取消数据相关**进一步利用并行执行提高速度的机会。

这里利用循环展开进行优化(要注意越界问题)



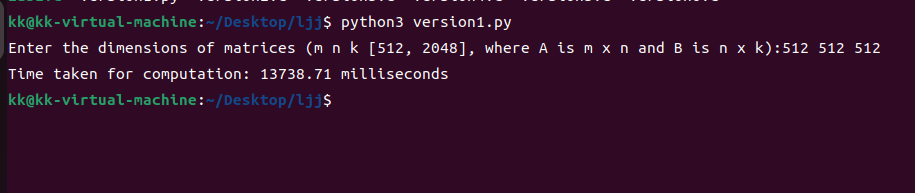
## 2.6 Intel MKL

Intel Math Kernel Library（MKL）是英特尔提供的数学核心库，旨在提高在英特尔处理器上执行数学和科学计算的性能。MKL 提供了一系列高度优化的数学函数和算法，包括线性代数、傅立叶变换、随机数生成等，可以显著加速科学计算、工程计算和数据分析等应用程序。

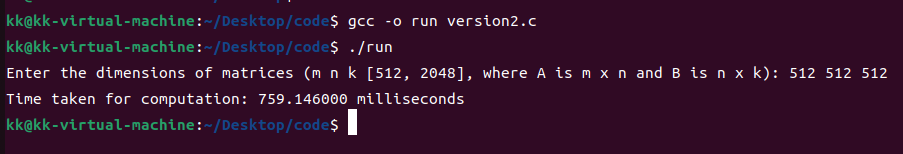


# **实验结果**

## 3.1 Python版本



## 3.2 C/C++版本

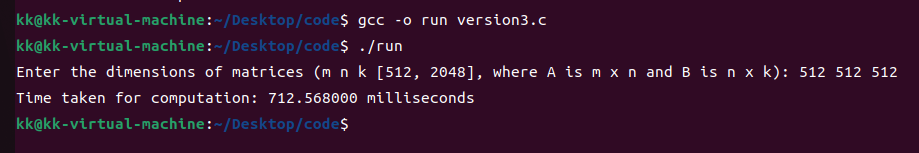


Python执行比C/C++慢的原因:

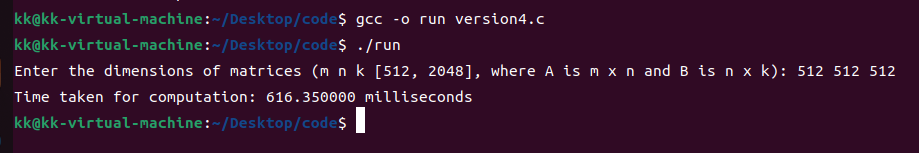
1. 动态类型和解释执行： Python 是一种动态类型语言，而 C 是一种静态类型语言。Python 在运行时需要进行类型检查和解释执行，这会导致额外的性能开销。相比之下，C 是一种静态类型语言，在编译时进行类型检查并生成机器码，因此执行速度更快。

1. 解释器和虚拟机开销：\*Python 是一种解释执行的语言，它的代码由解释器逐行解释并执行。另外，Python 代码通常在虚拟机中运行，这也会增加一些额外的开销。相比之下，C 代码是直接编译成机器码执行的，没有解释器和虚拟机的开销。
2. 优化和编译器技术： C 是一种编译型语言，它的代码在编译时会经过优化和静态分析，生成高效的机器码。与此相比，Python 的代码是在运行时动态解释执行的，编译器无法进行静态优化。

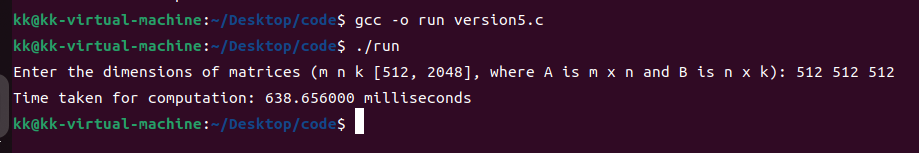
## 3.3调整循环顺序



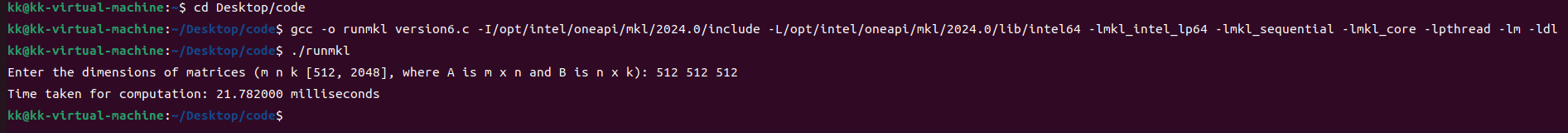
## 3.4 编译优化



## 3.5 循环展开



## 3.6 Intel MKL



## 3.7综合对比

矩阵规模:

A: 512 X 512

B: 512 X 512

C: 512 X 512

虚拟机配置:

1个处理器，共2个核

浮点计算单元为8

时钟频率为 2495.311 MHZ

峰值性能估计为 39.925 GFLOPS

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 版本 | 实现描述 | 运行时间  （sec.） | 相对  加速比 | | 绝对  加速比 | | 浮点性能  （GFLOPS） | | 峰值性能  百分比 |
| 1 | Python | 13.73871 | | 1 | | 1 | | 0.019 | 0.048% |
| 2 | C/C++ | 0.759146 | | 18.097586 | | 18.097586 | | 0.353 | 0.885% |
| 3 | 调整循环顺序 | 0.712568 | | 1.065366 | | 19.280560 | | 0.376 | 0.943% |
| 4 | 编译优化 | 0.616350 | | 1.156109 | | 22.290436 | | 0.435 | 1.090% |
| 5 | 循环展开 | 0.638656 | | 0.965074 | | 21.511909 | | 0.420 | 1.052% |
| 6 | Intel MKL | 0.021782 | | 29.320356 | | 630.736847 | | 12.323 | 30.867% |

# **实验感想**

在本次实验中，我对串行矩阵乘法的实现和优化进行了深入探索，在实验中收获了一些感想：

在本次实验中，我掌握了串行矩阵乘法的实现和优化技巧，也提升了自己在编程和性能优化方面的能力。通过本次实验，对比了多种串行矩阵乘法的优化方法，包括循环调整、编译优化、循环展开以及使用Intel MKL库等，我了解了这些优化方法是如何优化矩阵乘法的。并且通过对实验结果的比较，我加深了算法优化对性能提升的重要性的认识。