中山大学计算机学院本科生实验报告

课程名称：并行程序设计与算法

实验	环境配置与串行矩阵乘法	专业（方向）	计算机科学与技术
学号	21307035	姓名	邓栩瀛
Email	dengxy66@mail2.sysu.edu.cn	完成日期	2024.3.20

1、实验目的

用C/C++语言实现一个串行矩阵乘法，并通过对比实验分析其性能。

输入：m,n,k三个整数，每个整数的取值范围均为[512,2048]

问题描述：随机生成m×n的矩阵A及n×k的矩阵B，并对这两个矩阵进行矩阵乘法运算，得到矩阵C

输出：A，B，C三个矩阵，以及矩阵计算所消耗的时间t

要求：实现多个版本的串行矩阵乘法（可以考虑多种语言，如Python/编译选项/实现方式/算法/库，IntelMKL），并对比分析不同因素对最终性能的影响。

2、实验过程和核心代码

Intel oneAPI Math Kernel Library (MKL)配置

```
wget https://registrationcenter-download.intel.com/akdlm/IRC_NAS/86d6a4c1-c998-4c6b-9fff-ca004e9f7455/l_onemkl_p_2024.0.0.49673.sh
sudo sh ./l_onemkl_p_2024.0.0.49673.sh
source /opt/intel/oneapi/setvars.sh
# example
gcc -g main.cpp -lstdc++ -lmkl_rt -o main
```

版本1 Python

```
def matrix_multiply_serial(A, B):
    m, n = A.shape
    n, k = B.shape
    C = np.zeros((m, k))

    for i in range(m):
        for j in range(k):
            for l in range(n):
                C[i][j] += A[i][l] * B[l][j]

    return C
```

版本2 C++

```
vector<vector<double>> matrixMultiplication(const vector<vector<double>>&
matrixA, const vector<vector<double>>& matrixB) {
    int rowsA = matrixA.size();
    int colsA = matrixA[0].size();
    int colsB = matrixB[0].size();

    vector<vector<double>> result(rowsA, vector<double>(colsB));

    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsB; j++) {
            for (int k = 0; k < colsA; k++) {
                result[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }

    return result;
}
```

版本3 调整循环顺序

```
vector<vector<double>> matrixMultiplication(const vector<vector<double>>&
matrixA, const vector<vector<double>>& matrixB) {
    int rowsA = matrixA.size();
    int colsA = matrixA[0].size();
    int colsB = matrixB[0].size();

    vector<vector<double>> result(rowsA, vector<double>(colsB));

    for (int i = 0; i < rowsA; i++) {
        for (int k = 0; k < colsA; k++) {
            for (int j = 0; j < colsB; j++) {
                result[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }

    return result;
}
```

版本4 编译优化

```
gcc -g -O3 -fomit-frame-pointer -ffast-math main.cpp -lstdc++ -o main
```

`-ffast-math`: 关闭某些数学函数的严格准确性检查，以换取更高的性能，对浮点运算有帮助

`-fomit-frame-pointer`:用于决定是否在汇编代码中包含帧指针。帧指针是一个在函数调用时保存当前栈帧地址的寄存器。如果不使用帧指针，那么函数调用的开销会更小。

版本5 循环展开

```
gcc -g -O3 -funroll-loops  main.cpp -lstdc++ -o main
```

`-funroll-loops`: 这个选项会展开循环，以减少循环控制的开销，对循环嵌套的程序有性能提升作用。

版本6 Intel MKL

```
void multiply_matrices(int m, int n, int k, double* A, double* B, double* C)
{
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m, k, n, 1.0, A, n, B, k, 0.0, C, k);
}
```

```
gcc -g main.cpp -lstdc++ -lmkl_rt -o main
```

3、实验结果

`m=600`, `n=700`, `k=800`

版本	实验描述	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	231.433815	/	/	2903637.911775	0.007970%
2	C/C++	5.91043	0.9745	0.9745	113697330.59986	5753.42466%
3	调整循环顺序	3.71451	0.3715	0.9840	180912250.02073	5753.42466%
4	编译优化	0.53627	0.8556	0.9977	1253100117.47814	5753.42466%
5	循环展开	0.523683	0.0235	0.9977	1283219046.63699	5753.42466%
6	Intel MKL	0.055911	0.8932	0.9998	12019101786.76826	5753.42466%

注：“相对加速比”为相对于前一版本的加速比；“绝对加速比”为相对于版本1的加速比；“峰值性能百分比”为当前浮点性能相对于计算设备峰值性能的百分比。

使用高级语言（如Python）可能导致较长的运行时间和较低的浮点性能，而使用低级语言（如C/C++）可以显著提升性能。对代码进行优化，如调整循环顺序、编译优化和循环展开，可以进一步提高性能。此外，使用专门的库（如Intel MKL）也可以显著提升性能。因此，选择适当的语言、进行代码优化和利用专门库都是提高性能的关键因素。

4、实验感想

1. 在本次尝试中使用不同的串行矩阵乘法实现方式，使用不同语言（C++和Python）、不同编译选项（如优化级别）、以及Intel MKL。通过比较它们的性能，可以了解不同实现方式的优缺点和适用场景。

2. 矩阵乘法的性能受到多个因素的影响，如矩阵的维度、实现方式、编译选项和算法等都会对性能产生影响。在不同的实验条件下，这些因素会综合地影响最终的性能结果。因此，在选择实现方式和参数配置时，需要综合考虑这些因素，以获得最佳性能。
3. 通过尝试不同的优化技术，如矩阵分块、并行计算和内存访问模式的优化，可以大幅提升矩阵乘法的性能。特别是在处理大规模矩阵时，优化技术的应用可以减少计算时间，提高效率。
4. 通过记录实验过程中使用的参数、编译选项和算法等细节，可以在不同的环境中重复实验并比较结果。
5. 通过性能分析，对不同实现方式和参数配置的性能进行客观评估，有助于理解不同因素对性能的影响，并为进一步的优化提供指导。