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：0	专业（方向）：计算机科学与技术
学号：22336226	姓名：王泓沣
Email：wanghf59@mail2.sysu.edu.cn	完成日期：2025/3/23

实验目的

熟悉 Linux 编程及编译环境（shell, vim, gcc, gdb）；掌握基本矩阵乘法实现方式。

实验过程和核心代码

使用 Python 实现三层嵌套矩阵乘法

计算 256×256 的矩阵相乘，矩阵元素为随机数值。

```
start = time.time()
    for i in range(N):
        for j in range(N):
            for k in range(N):
                C[i][j] += A[i][k] * B[k][j]
end = time.time()
```

Listing 1: Python 实现三层嵌套矩阵乘法

使用 C 实现三层嵌套矩阵乘法

Python 为解释性语言，在编译时需要先转换为 C 语言，因而直接使用 C 语言实现能够减少运行时间。

```
clock_t start = clock();
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        for (int k = 0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j];
clock_t end = clock();
```

Listing 2: C 实现三层嵌套矩阵乘法

调整循环顺序

循环顺序从外到内遍历 i,j,k 时，没有利用局部性原理，因而调整循环顺序，在访问 C[i][j] 后，访问 C[i][j+1] 可以减少运行时间。

```
//仅核心循环变化
clock_t start = clock();
for (int i = 0; i < N; i++)
    for (int k = 0; k < N; k++)
        for (int j = 0; j < N; j++)
            C[i][j] += A[i][k] * B[k][j];
clock_t end = clock();
```

Listing 3: 调整嵌套循环顺序

编译优化

编译优化包括对编译器和指令集的优化，此处使用/O2，加快编译速度。

图 1: 编译器优化

循环展开

对 for 循环进行 4 次循环展开，每个循环执行 4 次浮点数乘法，提高了指令的并行度，同时注意避免数据依赖，能够提升运行速度。

```
clock_t start = clock();
int unroll_factor = 4; //展开因子为4
for (int i = 0; i < N; i++)
    for (int k = 0; k < N; k++)
        for (int j = 0; j < N; j += unroll_factor) {
            C[i][j] += A[i][k] * B[k][j];
            C[i][j + 1] += A[i][k] * B[k][j + 1];
            C[i][j + 2] += A[i][k] * B[k][j + 2];
            C[i][j + 3] += A[i][k] * B[k][j + 3];
        }
clock_t end = clock();
```

Listing 4: 循环 4 次展开

Intel MKL

MKL 实现了 BLAS（Basic Linear Algebra Subprograms）接口，并对其中的矩阵乘法例程（如 cblas_dgemm, cblas_sgemm）做了深入优化，同时使用 Blocking（或称 Tiling）技术将大矩阵分解成小块（block），提高 cache 局部性，减少内存带宽瓶颈，在指令集方面利用了现代 CPU 的 SIMD（Single Instruction, Multiple Data）指令集，利用 AVX/AVX2/AVX-512 指令，在一个 CPU 周期内同时计算多个浮点数。MKL 内部通过矢量化算法，将多个矩阵元素的乘加操作并行化。此外，MKL 支持多线程并行计算，可自动利用多核 CPU，内部使用 OpenMP 或 Intel TBB 管理线程，自动划分矩阵子任务到不同线程，提升利用率。

```
    double alpha = 1.0, beta = 0.0;
    MKL_INT m = N, k = N, n = N;
    clock_t start = clock();

    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
        m, n, k, alpha, A, k, B, n, beta, C, n);

    clock_t end = clock();
```

Listing 5: 使用 Intel MKL 计算矩阵乘法

实验结果

表格所示实验中未包含 Strassen 等矩阵乘法算法来改变运算次数，因此在本次 256×256 矩阵的乘法实验中，浮点运算次数始终为 $O(n^3)$ 级，即 2×256^3 次。

CPU 配置：AMD Ryzen 7 6800H with Radeon Graphics 3.20 GHz 在不使用高级指令集的情况下，使用 SSE2 指令集，双精度峰值为 8 核 ×3.2GHz×2 浮点/周期 =51.2 GLFOPS; Intel MKL 自动调用高级指令集 AVX2，双精度峰值为 8 核 ×3.2GHz×8 浮点/周期 =204.8 GLFOPS.

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
Python	10.864194	-	-	0.00	0.01
C/C++	0.055000	197.53	197.53	0.61	1.19
调整循环顺序	0.033000	1.67	329.22	1.02	1.99
编译优化	0.003000	11.00	3621.40	11.18	21.84
循环展开	0.005000	0.60	2172.84	6.71	13.10
Intel MKL	0.002000	2.50	5432.10	16.78	8.19

表 1: 实验结果汇总

实验感想

在优化编译时，我对各个种类的优化 (O1,O2,O3) 不清楚底层原理，后通过 IDE 内部描述与查询资料解决；在计算浮点性能和峰值性能时，我一度忽略了不同指令集产生的差异，后查询 SSE2 和 AVX2 解决；同时，此次实验加深了我对指令集与 CPU 双向提升的理解，并对 Intel MKL 库有了一定程度的认识。