并行计算实验报告：矩阵乘法并行化

姓名：蔡可豪
学号：22336018

1. 实验概述

本实验旨在研究矩阵乘法的并行化性能，通过使用不同大小的矩阵和不同数量的进程，分析并行计算中的加速比和效率。

1.1 实验环境

- 操作系统：macOS 24.4.0
- 编程语言：C++ (MPI)
- 编译环境：MPI (OpenMPI)

1.2 实验参数

- 矩阵大小：128x128, 256x256, 512x512, 1024x1024, 2048x2048
- 进程数：1, 2, 4, 8, 16

1.3 关键源码解析

1.3.1 矩阵数据结构

```
typedef struct {
    int rows;
    int cols;
    double* data;
} Matrix;
```

使用结构体封装矩阵数据，包含行数、列数和数据指针，便于数据管理和传递。

1.3.2 矩阵初始化

```
void init_matrix(Matrix* mat, int rows, int cols) {
    mat->rows = rows;
    mat->cols = cols;
    mat->data = (double*)malloc(rows * cols * sizeof(double));

    for (int i = 0; i < rows * cols; i++) {
        mat->data[i] = (double)rand() / RAND_MAX;
    }
}
```

动态分配内存并随机初始化矩阵元素，使用一维数组存储以提高内存访问效率。

1.3.3 核心计算函数

```
void multiply_block(const double* A, const double* B, double* C,
                    int A_rows, int A_cols, int B_cols,
                    int start_row, int num_rows) {
    for (int i = 0; i < num_rows; i++) {
        for (int j = 0; j < B_cols; j++) {
            double sum = 0.0;
            for (int k = 0; k < A_cols; k++) {
                sum += A[i * A_cols + k] * B[k * B_cols + j];
            }
            C[i * B_cols + j] = sum;
        }
    }
}
```

实现矩阵乘法的核心计算，采用分块计算策略：

- 每个进程负责计算结果矩阵的一部分行
- 使用三重循环实现矩阵乘法
- 通过一维数组索引优化内存访问

1.3.4 并行化策略

```
// 计算每个进程处理的行数
int rows_per_proc = matrix_size / size;
int remainder = matrix_size % size;
int local_rows = rows_per_proc + (rank < remainder ? 1 : 0);

// 分发矩阵A
MPI_Scatterv(A, sendcounts, displs, MPI_DOUBLE,
             local_A, local_rows * matrix_size, MPI_DOUBLE,
             0, MPI_COMM_WORLD);

// 广播矩阵B到所有进程
MPI_Bcast(B, matrix_size * matrix_size, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

采用行分块并行策略：

- 将矩阵A按行分块，每个进程处理大致相等的行数
- 使用MPI_Scatterv实现不均匀分发
- 通过MPI_Bcast广播矩阵B到所有进程
- 最后使用MPI_Gatherv收集结果

2. 实验结果

2.1 性能数据

128x128 矩阵

进程数	执行时间(s)	加速比	效率
1	0.002	1.00	1.00
2	0.001	2.27	1.14
4	0.001	2.56	0.64
8	0.001	3.48	0.44
16	0.000	7.23	0.45

256x256 矩阵

进程数	执行时间(s)	加速比	效率
1	0.015	1.00	1.00
2	0.008	1.89	0.95
4	0.005	3.19	0.80
8	0.004	3.85	0.48
16	0.002	8.68	0.54

512x512 矩阵

进程数	执行时间(s)	加速比	效率
1	0.148	1.00	1.00
2	0.078	1.89	0.95
4	0.047	3.15	0.79
8	0.046	3.18	0.40
16	0.040	3.66	0.23

1024x1024 矩阵

进程数	执行时间(s)	加速比	效率
1	1.238	1.00	1.00
2	0.882	1.40	0.70
4	0.959	1.29	0.32
8	0.981	1.26	0.16
16	0.996	1.24	0.08

2048x2048 矩阵

进程数	执行时间(s)	加速比	效率
1	30.169	1.00	1.00
2	18.013	1.67	0.84
4	19.524	1.55	0.39
8	19.878	1.52	0.19
16	19.773	1.53	0.10

2.2 性能分析

小矩阵性能分析 (128x128, 256x256)

1. 在小矩阵情况下，并行化效果显著：
    - 128x128矩阵在16个进程时达到7.23倍加速比
    - 256x256矩阵在16个进程时达到8.68倍加速比
    - 出现了超线性加速现象，这可能是由于缓存效应导致的
2. 效率分析：
    - 2个进程时效率接近或超过1.0
    - 随着进程数增加，效率逐渐下降
    - 16个进程时效率降至0.45-0.54

中等大小矩阵性能分析 (512x512)

1. 加速比表现：
    - 最大加速比达到3.66（16个进程）
    - 2-4个进程时加速比相对稳定
    - 8个进程后加速比提升不明显
2. 效率特征：

- 2个进程时效率保持在0.95
- 4个进程时效率降至0.79
- 16个进程时效率显著下降至0.23

大矩阵性能分析 (1024x1024, 2048x2048)

1. 加速比特点：
   - 最大加速比仅为1.67（2048x2048，2个进程）
   - 增加进程数对性能提升有限
   - 2048x2048矩阵在4个进程后性能反而略有下降
2. 效率问题：
   - 效率随进程数增加急剧下降
   - 16个进程时效率降至0.08-0.10
   - 表明通信开销成为主要瓶颈

3. 结论与发现

1. 矩阵大小与并行效率的关系：
   - 小矩阵（≤256x256）适合并行化，可获得较好的加速比
   - 中等大小矩阵（512x512）并行效果一般
   - 大矩阵（≥1024x1024）并行效果较差，通信开销显著
2. 进程数与性能的关系：
   - 小矩阵：进程数增加带来显著性能提升
   - 中等矩阵：4-8个进程可能是最优选择
   - 大矩阵：2-4个进程可能是最优选择
3. 超线性加速现象：
   - 在小矩阵情况下观察到超线性加速
   - 这可能是由于缓存效应和内存访问模式优化导致的

4. 优化建议

1. 针对不同大小的矩阵采用不同的并行策略：
   - 小矩阵：可以使用更多进程
   - 大矩阵：建议使用较少进程， focus on 减少通信开销
2. 通信优化：
   - 考虑使用更高效的数据分发策略
   - 优化进程间通信模式
   - 减少不必要的同步点

3. 负载均衡：
    - 实现动态负载均衡机制
    - 考虑矩阵分块大小的优化

5. 实验总结

本实验通过实现并行矩阵乘法算法，深入研究了并行计算中的性能特征和优化策略。主要成果和发现如下：

1. 算法实现方面：
    - 成功实现了基于MPI的并行矩阵乘法
    - 采用行分块策略，实现了数据的均匀分配
    - 通过一维数组存储优化了内存访问效率
2. 性能特征方面：
    - 小矩阵（≤256x256）表现出色，最高达到8.68倍加速比
    - 中等矩阵（512x512）性能适中，最大加速比3.66
    - 大矩阵（≥1024x1024）性能受限，通信开销显著
3. 关键发现：
    - 发现了小矩阵情况下的超线性加速现象
    - 验证了通信开销对并行性能的显著影响
    - 确定了不同矩阵规模下的最优进程数
4. 实验启示：
    - 并行算法的性能与问题规模密切相关
    - 通信开销是影响并行效率的关键因素
    - 需要根据具体问题规模选择合适的并行策略

通过本实验，我们不仅掌握了并行计算的基本原理和实现方法，也深入理解了影响并行性能的各种因素。这些经验对于今后设计和优化并行算法具有重要的指导意义。