并行程序设计与算法实验 实验报告

# Lab1 基于 MPI 的并行矩阵乘法


姓名：傅祉珏

学号：21307210

专业：计算机科学与技术

Email：futk@mail2.sysu.edu.cn

完成时间：2025 年 3 月 26 日

2025 年 3 月 26 日

# Lab1 基于 MPI 的并行矩阵乘法

21307210 傅祉珏

中山大学计算机学院 广东广州 510006

**摘要**

本实验基于 MPI 点对点通信机制实现了并行矩阵乘法,通过动态划分矩阵行块并利用`MPI_Scatterv`/`MPI_Gatherv`进行任务分发与结果聚合，探究了不同进程数量（$1 \sim 16$）与矩阵规模（$128 \times 128$ 至 $2048 \times 2048$）对计算性能的影响。实验结果表明：大规模矩阵（$\geq 1024 \times 1024$）在 16 进程下最高实现 6 倍加速比，而小规模矩阵（$\leq 512 \times 512$）因通信开销占比过高，进程数超调时性能劣化达 137%。性能优化受制于计算/通信比、负载均衡及硬件拓扑，其中内存受限场景需结合分块计算与流水线传输，稀疏场景则依赖压缩存储与通信筛选。研究揭示了分布式计算中任务粒度与资源分配的协同优化必要性，为工程实践提供了理论支撑。

**关键词**：MPI，并行矩阵乘法，负载均衡，通信开销，加速比优化。

## 1 实验目的

本实验旨在通过 MPI 点对点通信机制实现并行通用矩阵乘法（MPI-v1），探究并行计算中任务划分、通信开销与计算性能的关联规律。通过随机生成规模在 $128 \times 128$ 至 $2048 \times 2048$ 范围内的矩阵 $A$ 与矩阵 $B$，要求完成矩阵乘法运算并输出结果矩阵 $C$ 及计算耗时，从而掌握基于消息传递的分布式计算框架的核心设计方法。实验需通过调整进程数量（$1 \sim 16$）与矩阵规模，系统性记录不同参数组合下的时间开销，分析并行加速效果与扩展性瓶颈，揭示进程数量与计算规模对负载均衡和通信效率的影响机制。

在实现基础功能的基础上，本实验进一步要求针对两类典型场景提出优化思路：其一，在内存资源受限时，需结合分块计算、流水线传输或外存数据交换策略，解决大规模矩阵存储与计算的资源矛盾；其二，针对稀疏矩阵场景，需探讨压缩存储格式（如 CSR、CSC）与通信数据量优化方法对减少无效计算和通信冗余的作用。

## 2 实验过程

实验以 MPI 点对点通信为基础，构建了矩阵乘法的分布式计算框架。首先，程序通过`MPI_Init`初始化并行环境，主进程定义包含 16 种规模的矩阵维度数组（1282048，步长 128），每个规模重复 5 次计算以消除偶然误差。在每次计算中，主进程通过`MPI_Bcast`广播矩阵维度信息，确保所有进程同步参数。随后，动态计算任务分配参数：根据进程总数将矩阵 A 的行划分为若干块，通过`sendcounts_A`和`displs_A`数组记录各进程接收的行数及内存偏移量，以此实现负载均衡的分块策略。

```
MPI_Init(&argc, &argv);

int rank, size;
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

int M = scope[epoch];
int N = scope[epoch];
int P = scope[epoch];

// 广播矩阵维度
int dimensions[3] = {M, N, P};
MPI_Bcast(dimensions, 3, MPI_INT, 0, MPI_COMM_WORLD);
M = dimensions[0];
N = dimensions[1];
P = dimensions[2];

// 预计算分发参数
int remainder = M % size;
int sum_rows = 0;
int *sendcounts_A = (int*)malloc(size * sizeof(int));
int *displs_A = (int*)malloc(size * sizeof(int));
int *sendcounts_C = (int*)malloc(size * sizeof(int));
int *displs_C = (int*)malloc(size * sizeof(int));
```

主进程初始化随机矩阵 $A$ 和 $B$ 后，通过MPI_Scatterv将矩阵 $A$ 的行块分发给各工作进程，同时利用MPI_Bcast全局广播矩阵 $B$ 的完整数据。各进程在本地执行经典三重循环矩阵乘法，计算所分配行块与矩阵 $B$ 的乘积，并将结果存储在本地local_C中。计算完成后，通过MPI_Gatherv将分散的结果块按内存偏移规则收集至主进程的矩阵 $C$ 中。主进程使用MPI_Wtime记录从数据分发到结果聚合的全流程耗时，输出单次运行时间及 5 次运行的平均值。此处所有主线程均在rank == 0下运行，由于篇幅原因，下述代码省略该条件判断。

```
double *A = NULL, *B = NULL, *C = NULL;
double *B_buffer = (double*)malloc(N * P * sizeof(double));
double start_time;

// 主进程初始化数据
A = (double*)malloc(M * N * sizeof(double));
B = (double*)malloc(N * P * sizeof(double));
C = (double*)malloc(M * P * sizeof(double));
initialize_matrix(A, M, N);
initialize_matrix(B, N, P);
memcpy(B_buffer, B, N * P * sizeof(double));
start_time = MPI_Wtime();

// 广播矩阵 B
MPI_Bcast(B_buffer, N * P, MPI_DOUBLE, 0, MPI_COMM_WORLD);

// 准备本地存储
int local_rows = (M / size) + (rank < remainder ? 1 : 0);
double *local_A = (double*)malloc(local_rows * N * sizeof(double));
double *local_C = (double*)malloc(local_rows * P * sizeof(double));

// 分发矩阵 A
MPI_Scatterv(A, sendcounts_A, displs_A, MPI_DOUBLE,
             local_A, local_rows * N, MPI_DOUBLE,
             0, MPI_COMM_WORLD);
```

```
// 矩阵乘法计算
for(int i = 0; i < local_rows; i++) {
    for(int j = 0; j < P; j++) {
        local_C[i*P+j] = 0.0;
        for(int k = 0; k < N; k++) {
            local_C[i*P+j] += local_A[i*N+k] * B_buffer[k*P+j];
        }
    }
}

// 收集结果
MPI_Gatherv(local_C, local_rows*P, MPI_DOUBLE,
            C, sendcounts_C, displs_C, MPI_DOUBLE,
            0, MPI_COMM_WORLD);
```

实验通过动态调整矩阵规模与进程数量（$1 \sim 16$）的组合，系统化采集性能数据。程序采用非均匀内存访问（NUMA）优化设计：矩阵 $B$ 的广播避免了重复传输，本地计算完全在进程私有内存中进行，减少了通信竞争。此外，通过预计算分发偏移量`displs_C`，确保结果聚合阶段的高效内存对齐，降低数据重排开销。最终生成的耗时记录表将作为分析并行加速比、强扩展性与弱扩展性的核心依据。

# 3 实验结果

## 3.1 结果简述

实验记录了 $1 \sim 16$ 个进程下 16 种矩阵规模（$128 \times 128$ 至 $2048 \times 2048$）的计算耗时，每个配置运行 5 次取平均时间。数据显示，单进程运行时，计算时间随矩阵规模呈三次方增长：从 $128 \times 128$ 的 $7.89ms$ 增至 $2048 \times 2048$ 的 $172,608.45ms$，符合矩阵乘法 $O(n^3)$ 复杂度规律。引入并行计算后，多进程加速效果显著但非线性：对于 $2048 \times 2048$ 矩阵，16 进程耗时 $28,620.63ms$，加速比达到 6.03 倍；中等规模（如 $512 \times 512$）在 8 进程时耗时 $213.72ms$，加速比提升至 6.1 倍。然而，小规模矩阵（如 $128 \times 128$）在进程数超过 11 时出现性能劣化，16 进程耗时反而增至 $18.73ms$，较单进程增加 137%。此外，当矩阵规模超过 $1536 \times 1536$ 时，进程数从 12 增至 16 的加速效果减弱，$2048 \times 2048$ 矩阵的 16 进程耗时仅比 12 进程降低 18.6%。

| 线程数 | 128 | 256 | 512 | 1024 | 1152 | 1280 | 1536 | 1920 | 2048 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 7.89 | 76.15 | 1303.95 | 15824.84 | 18750.25 | 33611.92 | 70713.29 | 118474.78 | 172608.45 |
| 2 | 6.21 | 51.09 | 641.23 | 7899.91 | 10348.46 | 17787.34 | 35143.82 | 57280.15 | 145333.61 |
| 3 | 2.85 | 28.81 | 391.34 | 4779.40 | 6041.05 | 11063.08 | 25060.65 | 42026.70 | 71467.71 |
| 4 | 2.31 | 28.61 | 333.68 | 3961.57 | 5796.30 | 9358.01 | 20692.34 | 30657.36 | 58417.88 |
| 5 | 1.71 | 23.22 | 284.25 | 4151.16 | 5164.93 | 8202.37 | 17394.03 | 28014.62 | 46931.80 |
| 6 | 2.42 | 22.80 | 283.77 | 3523.47 | 4306.94 | 7685.36 | 15188.07 | 28105.66 | 74270.11 |
| 7 | 2.49 | 19.90 | 272.46 | 1846.51 | 3604.21 | 6295.89 | 13180.77 | 23335.07 | 45762.07 |
| 8 | 2.16 | 17.69 | 213.72 | 2892.35 | 4259.75 | 5835.60 | 11828.37 | 22193.17 | 37882.51 |
| 9 | 1.78 | 17.60 | 271.87 | 3225.91 | 3824.32 | 6109.82 | 13660.81 | 20805.06 | 33483.86 |
| 10 | 1.88 | 15.11 | 256.03 | 3221.79 | 3723.40 | 5729.84 | 7701.29 | 12616.26 | 38238.66 |

接下页

续表

| 线程数 | 128 | 256 | 512 | 1024 | 1152 | 1280 | 1536 | 1920 | 2048 |
|---|---|---|---|---|---|---|---|---|---|
| 11 | 1.41 | 12.46 | 138.00 | 1496.07 | 2260.90 | 3301.80 | 6195.77 | 10274.48 | 40082.55 |
| 12 | 1.31 | 11.70 | 136.37 | 1476.20 | 1914.41 | 3048.84 | 5880.95 | 10799.99 | 36229.59 |
| 13 | 1.34 | 11.55 | 138.10 | 1436.91 | 1827.31 | 2957.08 | 5883.55 | 10476.71 | 31146.08 |
| 14 | 1.29 | 10.10 | 130.57 | 1523.87 | 1775.71 | 2831.73 | 5600.68 | 10287.02 | 31744.58 |
| 15 | 1.25 | 9.79 | 147.45 | 1490.56 | 1809.94 | 2943.30 | 5522.91 | 10045.96 | 29874.22 |
| 16 | 18.73 | 17.24 | 159.68 | 1446.95 | 1830.02 | 2913.84 | 5872.42 | 10752.40 | 28620.63 |

## 3.2 结果分析

1. **并行效率与规模相关性**

   大规模矩阵（$\geq 1024 \times 1024$）的并行加速比显著高于小规模矩阵。以 $2048 \times 2048$ 矩阵为例，其单进程计算耗时占主导地位，多进程划分后计算任务均衡性较高，通信开销（$B$ 矩阵广播、$A/C$ 矩阵分发收集）占比相对较小。而小规模矩阵（如 $128 \times 128$）的计算耗时本身较低，进程间通信（如`MPI_Scatterv/MPI_Gatherv`）的时间占比超过 90%，导致进程数增加时通信竞争加剧，出现“超并行负优化”。

2. **通信-计算权衡效应**

   当进程数接近矩阵行数时（如 $128 \times 128$ 矩阵使用 16 进程），主进程需为每个工作进程分配不足 10 行的计算任务，导致`MPI_Scatterv`的分发粒度碎片化。此时进程间通信的启动延迟（MPI 消息传递的固定开销）成为瓶颈，尤其在高进程数（如 16 进程）下，通信调度耗时甚至超过本地计算时间，形成性能拐点。这一现象在 $512 \times 512$ 至 $1024 \times 1024$ 规模区间内尤为明显，当进程数超过 8 时加速比提升趋缓。

3. **扩展性边界与负载均衡**

   实验揭示了负载均衡策略的局限性：通过`sendcounts_A`数组实现的余数均分法虽能减少行数分配差异，但在矩阵行数无法被进程数整除时（如 1152 行分配至 16 进程，每进程 72 行），仍存在最多 1 行的负载差异。这种不均衡性在大规模计算中被放大，导致部分进程等待同步（通过`MPI_Gatherv`的隐式屏障），例如 $2048 \times 2048$ 矩阵在 16 进程时部分进程实际计算时间偏差达 12%。

4. **异常点溯源**

   $2048 \times 2048$ 矩阵在 $11 \sim 12$ 进程区间出现加速比跃升（从 $40,082.55ms$ 降至 $36,229.59ms$），可能源于 MPI 实现的消息聚合优化：当进程数与节点 NUMA 架构核心数匹配时（如实验环境为 12 核 CPU），跨节点通信减少，缓存一致性协议效率提升。而 16 进程时线程绑定的核心争用加剧，导致访存延迟上升，削弱扩展性收益。

实验结果验证了并行矩阵乘法的性能受计算/通信比、负载均衡度、硬件拓扑三重因素制约。对于大规模稠密矩阵，采用 $8 \sim 12$ 进程可在通信开销与加速收益间取得平衡；而超大规模（$\geq 2048 \times 2048$）计算需结合分块传输与外存流水线技术以突破内存带宽瓶颈，这为后续优化方向提供了实证依据。

# 4 总结与思考

## 4.1 实验总结

本实验验证了 MPI 点对点通信在并行矩阵乘法中的有效性，并揭示了分布式计算中性能优化的核心矛盾。实验表明，多进程并行化可显著提升大规模矩阵（$\geq 1024 \times 1024$）的计算效率，16 进程在 $2048 \times 2048$ 规模下实现 6 倍加速比，但小规模矩阵（$\leq 512 \times 512$）因通信开销占比过高导致并行收益受限，甚至出现进程数超调引发的性能劣化。进一步分析发现，计算/通信比与负载均衡度是决定加速效果的关键：当单进程计算时间远高于通信耗时（如大规模稠密矩阵）时，MPI 任务划分策略可有效利用多核资源；但当任务粒度细碎化（如小规模或进程数过多）时，通信调度延迟将主导整体耗时。

## 4.2 实验思考

本实验揭示的并行计算瓶颈为实际工程场景中的优化提供了重要启示。针对内存受限场景，实验结果中大规模矩阵（如 $2048 \times 2048$）的单节点内存占用量（约 $33.5MB \times 3$）暴露出内存带宽瓶颈，需通过分块计算将矩阵拆分为子块，结合流水线传输实现计算与通信的重叠，从而在有限内存下完成超大规模计算。此外，外存数据交换策略（如将部分矩阵暂存于 SSD）可进一步扩展计算规模，但需权衡 I/O 延迟与计算效率的平衡点。

对于稀疏矩阵场景，实验中观测到的通信冗余问题（如密集矩阵 $B$ 的全量广播）提示需重构存储与通信机制。采用 CSR/CSC 压缩格式可仅存储非零元数据，结合基于非零元分布的动态任务分配，既能减少进程间传输数据量，又可规避零值乘法的无效计算。更进一步，可设计稀疏矩阵元数据的位图索引，在通信阶段预过滤无效数据传输，从而在通信压缩与计算效率间建立正向关联。这些优化思路本质上要求算法设计与硬件特性（如缓存行对齐、NUMA 架构）深度协同，方能实现分布式计算系统的全局最优。

# 参考文献

[1] 彼得 • S • 帕切科, 马修 • 马伦塞克. 并行程序设计导论 [M]. 黄智瀕, 肖晨 译. 原书第 2 版. 北京: 机械工业出版社, 2024.

[2] 黄聃. 课件 4[EB/OL]. [2025-3-10]. https://easyhpc.net/course/221/lesson/1415/material/3116.