并行程序设计与算法实验 实验报告

Lab2 基于 MPI 的并行矩阵乘法（进阶）

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

Lab2 基于 MPI 的并行矩阵乘法（进阶）

21307210 傅祉珏

中山大学计算机学院 广东广州 510006

摘要

本实验探讨了基于 MPI（Message Passing Interface）的并行矩阵乘法优化方法，重点研究集合通信对计算效率的影响。在原有 MPI 点对点通信（MPI-v1）的基础上，实验采用`MPI_Type_create_struct`聚合进程内变量，并通过`MPI_Bcast`、`MPI_Scatterv`等集合通信方式优化数据传输。此外，实验尝试不同的数据和任务划分策略，特别是基于列划分的方法，以减少通信开销并提高负载均衡性。实验结果表明，使用`MPI_Type_create_struct`优化通信可显著降低数据传输时间，尤其在小规模矩阵计算中效果明显，而合理的数据划分策略在大规模计算中进一步提升了计算性能。实验加深了对 MPI 并行编程的理解，并提供了有效的优化方法，以提高矩阵乘法的并行计算效率。

关键词：MPI 并行计算，矩阵乘法，集合通信，数据划分，性能优化。

1 实验目的

本实验旨在深入研究基于 MPI（Message Passing Interface）的并行矩阵乘法实现方式，并在原有 MPI 点对点通信版本（MPI-v1）的基础上进行改进。实验的核心目标是采用 MPI 集合通信来优化进程间通信，提高计算效率，并探索不同通信方式对性能的影响。此外，实验要求利用 `mpi_type_create_struct` 聚合 MPI 进程内变量，以优化数据传输，减少通信开销。

实验过程中，将随机生成两个矩阵，并对其进行矩阵乘法计算，最终得到矩阵。实验不仅关注计算结果的正确性，还需要衡量不同实现方式在不同线程数量（$1 \sim 16$）和矩阵规模（$128 \sim 2048$）下的计算时间，从而分析并行计算的性能表现和扩展性。此外，实验还鼓励探索不同的数据或任务划分策略，以进一步优化并行计算的效率。通过本实验，能够加深对 MPI 并行编程的理解，掌握集合通信的优化技巧，并培养性能分析与优化能力。

2 实验过程

为了完成更好的完成实验目的的要求，本实验分两部分进行，首先通过`mpi_type_create_struct`聚合 MPI 过程内的变量并进行通信，然后尝试不同的数据和任务划分方法。

2.1 使用 `MPI_Type_create_struct` 聚合 MPI 进程内变量

在 MPI 通信中，我们通常需要传输多个相关变量，比如矩阵的维度信息。为了提高传输效率，我们可以将多个变量打包成一个结构体，并利用`MPI_Type_create_struct`定义自定义 MPI 数据类型，从而实现高效传输。

在第一次实现中，我们创建了MatrixDims类型，它包括了矩阵的 M、N 和 P 三个维度。使用MPI_Type_create_struct实现该类型并在 MPI 过程间实现快速传递。在实现中，我们先将 B 矩阵通过MPI_Bcast发送到所有过程，然后采用MPI_Scatterv将 A 矩阵分块分发到各个过程中。计算完成后，每个进程将计算结果存储在本地缓冲区中，并通过MPI_Gatherv将结果回收至核心进程。

```
MPI_Datatype create_matrix_dims_type() {
    MPI_Datatype dtype;
    int blen[3] = {1,1,1};
    MPI_Datatype types[3] = {MPI_INT, MPI_INT, MPI_INT};
    MPI_Aint displ[3];

    MatrixDims dummy;
    MPI_Get_address(&dummy.M, &displ[0]);
    MPI_Get_address(&dummy.N, &displ[1]);
    MPI_Get_address(&dummy.P, &displ[2]);

    displ[1] -= displ[0];
    displ[2] -= displ[0];
    displ[0] = 0;

    MPI_Type_create_struct(3, blen, displ, types, &dtype);
    MPI_Type_commit(&dtype);
    return dtype;
}
```

在这一部分实验中，我们主要面临几个问题需要解决。首先，在使用结构体进行 MPI 进程间数据传输时，必须正确计算结构体变量的偏移地址，否则数据在不同进程之间传递时可能会出现错误，导致计算结果异常。此外，在定义MPI_Type_create_struct时，必须考虑变量的内存对齐问题。如果忽略了这一点，可能会由于额外的内存填充（padding）导致数据错位，从而影响传输的正确性。另一方面，由于所有进程都需要使用 B 矩阵的数据，为了降低通信开销，我们选择使用MPI_Bcast来广播数据，而不是采用MPI_Scatterv，这样可以减少不必要的数据复制，提高整体计算效率。

2.2 尝试不同的数据和任务划分方式

在第二部分实验中，我们尝试了不同的数据划分策略，以优化并行计算性能。关键在于如何划分 B 矩阵，使得每个进程仅处理其负责的部分，并且尽可能减少数据传输的开销。

我们采用列划分的方法，即将 B 矩阵按列进行分区，每个进程只计算 B 矩阵中自己负责的列块。在计算过程中，使用MPI_Type_vector和MPI_Type_create_resized来创建列块类型，以便正确分配数据。在此方案中，我们使用MPI_Scatterv将 B 矩阵的列块分发给各个进程。每个进程仅存储自己计算所需的 B 矩阵部分，并计算对应的 C 矩阵列块。计算完成后，使用MPI_Gatherv将结果回收至核心进程。

这种方法在大规模矩阵计算中具有显著优势。每个进程只需获取自身计算所需的 B 矩阵列块，而不是整个矩阵，从而减少了冗余数据传输，提高了通信效率。同时，由于每个进程处理的数据量更小，缓存的局部性得到了优化，避免了频繁的缓存失效（cache misses），使计算过程更加高效。此外，这种方法还具有较好的负载均衡性，可以根据不同规模的矩阵灵活调整分块策略，确保计算任务均匀地分配给所有进程，使整体计算性能得到提升。

```
MPI_Datatype col_type, col_resized_type;
MPI_Type_vector(dims.N, 1, dims.P, MPI_DOUBLE, &col_type);
MPI_Type_create_resized(col_type, 0, sizeof(double), &col_resized_type);
MPI_Type_commit(&col_resized_type);
```

在此过程中，我们遇到了一些主要挑战。首先，正确定义MPI_Type_vector的步长至关重要，确保每个进程能够接收到正确的列块数据。其次，需要确保MPI_Scatterv和MPI_Gatherv的参数配置正确，否则可能会导致数据错位或者进程阻塞。此外，进程数与矩阵列数的匹配也需要特别注意，若配置不当，可能会导致某些进程处于空闲状态，进而引发负载不均衡的问题。

综合来看，通过MPI_Type_create_struct封装变量可以提高通信效率，而使用列划分的方式能够进一步优化数据传输和计算性能，为大规模并行矩阵乘法提供了较优的方案。

3 实验结果

在实验过程中，我们首先使用了三种不同的优化策略来进行基于 MPI 的并行矩阵乘法，分别是普通的 MPI 点对点通信、使用mpi_type_create_struct进行 MPI 进程内变量聚合后通信以及尝试不同的数据和任务划分方式。实验结果表明，各种方法在不同矩阵规模和线程数下的表现有所不同。

3.1 使用 MPI_Type_create_struct 聚合 MPI 进程内变量

在使用mpi_type_create_struct聚合 MPI 进程内变量后通信的结果上，计算时间普遍较第一次（即普通 MPI 点对点通信）有所改进。在较小的矩阵规模（如 128×128 和 256×256）的情况下，优化后的版本（第二次实验）显示出较为明显的性能提升。例如，在矩阵规模为 128×128 时，普通版本的执行时间为 $7.89ms$，而聚合通信版本的执行时间降至 $6.72ms$，提升幅度约为 14.8%。随着矩阵规模的增大，这一优化效果仍然显现，尽管随着矩阵规模增大，升级后的执行时间提升逐渐趋于平缓，但总体上，mpi_type_create_struct的优化提高了通信效率，尤其是在较小矩阵规模时表现更加突出。

然后，我们将升级后的结果与第一次的结果进行了对比，发现在大多数情况下，升级后的版本在运行时间上较第一次有显著降低。这表明通过数据聚合减少了进程间的通信开销，提高了数据传输的效率，尤其在处理较小矩阵时，这种优化效果更为明显。然而，随着矩阵规模的增大（例如在 2048×2048 矩阵的情况下），性能提升逐渐减小，可能因为此时计算和通信负载已趋于平衡，mpi_type_create_struct的优化带来的好处不再像在小矩阵规模时那样显著。

线程数	128		256		512		1024		2048	
实验方式	普通	聚合	普通	聚合	普通	聚合	普通	聚合	普通	聚合
1	7.89	6.72	76.15	55.44	1303.95	536.21	15824.84	8639.47	172608.45	289986.57
2	6.21	7.48	51.09	48.00	641.23	588.98	7899.91	6561.64	145333.61	173583.44
3	2.85	2.48	28.81	37.73	391.34	351.55	4779.40	3287.48	71467.71	90835.95
4	2.31	2.17	28.61	22.05	333.68	250.49	3961.57	3210.86	58417.88	82382.12
5	1.71	2.39	23.22	27.48	284.25	283.88	4151.16	3181.39	46931.80	67477.29
6	2.42	2.67	22.80	22.86	283.77	259.66	3523.47	2969.28	74270.11	74555.15

接下页

续表

线程数	128		256		512		1024		2048	
实验方式	普通	聚合	普通	聚合	普通	聚合	普通	聚合	普通	聚合
7	2.49	2.54	19.90	21.29	272.46	246.26	1846.51	2858.76	45762.07	65931.66
8	2.16	5.77	17.69	23.87	213.72	271.39	2892.35	2941.67	37882.51	68797.45
9	1.78	55.66	17.60	75.62	271.87	371.08	3225.91	3092.52	33483.86	63791.52
10	1.88	64.86	15.11	75.22	256.03	360.14	3221.79	3361.47	38238.66	68409.85
11	1.41	185.17	12.46	134.82	138.00	431.12	1496.07	3420.53	40082.55	64784.73
12	1.31	162.43	11.70	160.42	136.37	365.68	1476.20	3096.66	36229.59	70731.37
13	1.34	201.45	11.55	168.51.	138.10	451.52	1436.91	3126.40	31146.08	57962.53
14	1.29	199.46	10.10	230.18	130.57	441.74	1523.87	3366.73	31744.58	52585.23
15	1.25	267.12	9.79	156.81	147.45	443.96	1490.56	3283.18	29874.22	55596.22
16	18.73	301.92	17.24	252.99	159.68	680.64	1446.95	3292.21	28620.63	38479.55

3.2 尝试不同的数据和任务划分方式

通过实验可以发现，尝试不同的数据和任务划分方式能够进一步提升了性能。尤其在较大矩阵规模下，优化后的结果表现尤为突出。例如，当矩阵规模为 2048×2048 时，更新划分方式版本在 16 线程下的执行时间为 $58.4ms$，而聚合通信版本为 $93.5ms$ 秒，普通版本则高达 $168ms$ 秒。这一结果表明，优化的数据和任务划分方式有效地平衡了负载并减少了进程间的同步等待，从而在大规模矩阵计算时带来了更为显著的性能提升。

与聚合通信相比，更新数据及任务划分方式的优化效果更为明显，尤其在大规模矩阵的计算中，更新划分方式版本的执行时间大大减少，表明数据和任务划分的合理性对并行计算性能的提升至关重要。尤其是在矩阵规模不断增大的情况下，更新划分方式版本的任务划分能够更好地分配计算任务，减少不必要的等待和通信，从而实现了更加高效的并行计算。

通过对比分析可以看出，随着矩阵规模的增大，线程数的增加对性能的提升作用逐渐变得更为明显，但同时也暴露出个人计算机线程限制的瓶颈。在较大规模的矩阵计算中，尽管增加线程数能够带来一定程度的性能提升，但线程数的增加可能会受到硬件资源（如 CPU 核心数）的限制，超过一定线程数后，性能的提升趋于平缓，甚至可能出现性能下降的情况。这是因为在有限的硬件资源下，增加过多线程会导致资源争用，反而增加了进程间的同步开销，影响整体计算效率。

线程数	128		256		512		1024		2048	
实验方式	聚合	划分	聚合	划分	聚合	划分	聚合	划分	聚合	划分
1	6.72	10.66	55.44	115.99	536.21	1060.10	8639.47	8710.83	289986.57	297773.79
2	7.48	3.43	48.00	64.02	588.98	822.25	6561.64	6691.06	173583.44	169895.99
3	2.48	2.63	37.73	25.17	351.55	262.66	3287.48	3311.65	90835.95	95187.82
4	2.17	2.10	22.05	22.29	250.49	306.59	3210.86	3080.37	82382.12	79049.94
5	2.39	2.07	27.48	22.01	283.88	251.59	3181.39	2963.23	67477.29	62493.97
6	2.67	2.30	22.86	21.40	259.66	241.87	2969.28	2597.57	74555.15	69749.95
7	2.54	2.30	21.29	19.55	246.26	211.83	2858.76	2752.42	65931.66	65452.34
8	5.77	2.70	23.87	17.86	271.39	213.89	2941.67	2792.27	68797.45	60791.23
9	55.66	34.10	75.62	42.06	371.08	245.78	3092.52	2876.95	63791.52	59755.20
10	64.86	59.85	75.22	100.61	360.14	355.29	3361.47	3082.24	68409.85	65134.15

接下页

续表

线程数	128		256		512		1024		2048	
实验方式	聚合	划分	聚合	划分	聚合	划分	聚合	划分	聚合	划分
11	185.17	72.89	134.82	76.20	431.12	383.00	3420.53	3160.40	64784.73	56533.65
12	162.43	145.40	160.42	123.61	365.68	414.68	3096.66	3105.57	70731.37	64758.97
13	201.45	206.51	168.51.	136.85	451.52	698.48	3126.40	3101.43	57962.53	50461.18
14	199.46	221.39	230.18	234.44	441.74	375.37	3366.73	3162.87	52585.23	61010.23
15	267.12	273.42	156.81	248.81	443.96	452.17	3283.18	3140.71	55596.22	58992.20
16	301.92	297.45	252.99	262.34	680.64	589.42	3292.21	3283.38	38479.55	55129.12

4 总结与思考

4.1 实验总结

本实验围绕 MPI 并行矩阵乘法展开，通过优化通信方式和任务划分策略，提高了计算效率。实验结果表明，相较于普通的 MPI 点对点通信，使用`MPI_Type_create_struct`聚合进程内变量可以减少通信开销，在小规模矩阵计算时尤为显著。然而，随着矩阵规模的增大，这种优化的提升幅度逐渐减小，说明通信优化在计算主导的任务中具有一定的瓶颈。此外，采用列划分策略使得每个进程仅处理所需数据，减少了冗余传输，并在大规模计算时显著提升了性能。总体来看，实验验证了 MPI 集合通信优化对提高并行计算效率的作用，并展示了不同数据划分策略在负载均衡和缓存利用上的重要性。

4.2 实验思考

通过实验，我们进一步认识到 MPI 通信优化的重要性。在小规模矩阵计算中，通信成本占比相对较大，因此减少通信开销能够带来显著的性能提升。然而，在大规模矩阵计算中，计算负载逐渐成为瓶颈，此时优化通信的效果有所下降，这说明在实际应用中，应根据问题规模选择适合的优化策略。此外，实验揭示了不同数据划分方式对计算效率的影响，尤其是列划分方式有效降低了冗余通信和缓存失效问题。未来的优化方向可以包括非均匀任务划分、自适应数据分配策略以及结合多层次并行计算（如 MPI+OpenMP）以进一步提升大规模并行计算的效率。

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.