



中山大學
SUN YAT-SEN UNIVERSITY

并行程序设计 with 算法

实验 2-基于 MPI 的并行矩阵乘法（进阶）

姓 名 王志杰

学 号 22331095

学 院 计算机学院

专 业 计算机科学与技术

2025 年 4 月 15 日

目录

1	实验目的	1
2	实验方法	1
2.1	并行策略	1
2.2	通信优化	1
2.3	关键实现	1
3	实验结果	4
3.1	示例输出	4
3.2	结果总结	4
4	讨论	4
4.1	内存受限优化策略	4
4.1.1	分块计算	4
4.1.2	外存交换	6
4.1.3	精度压缩	6
4.2	稀疏矩阵优化策略	6
4.2.1	存储格式优化	6
4.2.2	计算路径优化	6
4.2.3	负载重平衡	6

1 实验目的

改进上次实验中的 MPI 并行矩阵乘法 (MPI-v1)，并讨论不同通信方式对性能的影响。采用 MPI 集合通信实现并行矩阵乘法中的进程间通信；使用 `mpi_type_create_struct` 聚合 MPI 进程内变量后通信；尝试不同数据/任务划分方式，实现基于 MPI 的并行矩阵乘法算法 $C = A \times B$ ，其中：

- A 为 $m \times n$ 矩阵
- B 为 $n \times k$ 矩阵
- C 为 $m \times k$ 结果矩阵

通过任务划分和进程间通信，验证并行计算的加速效果。

2 实验方法

2.1 并行策略

1. **数据划分**：矩阵 A 按行分块，每个进程处理 m/p 行
2. **通信模式**：
 - `MPI_Bcast` 广播矩阵 B （全连接通信）
 - `MPI_Scatter` 分发矩阵 A 的行块（一对多通信）
 - `MPI_Gather` 收集计算结果（多对一通信）
3. **负载均衡**：要求 m 必须能被进程数整除

2.2 通信优化

- **结构体打包**：使用 `MPI_Type_create_struct` 聚合性能数据
- **单次通信**：通过 `MPI_Gather` 一次性收集所有进程的性能数据

2.3 关键实现

核心代码结构如下（完整代码见附件）：

```
1 // 每个进程分配B矩阵
2 srand(SEED);
3 B = (float*)malloc(n * k * sizeof(float));
4 if (B == NULL) {
```

```

5         fprintf(stderr, "Failed to allocate memory for B(rank %d)\n"
6             , rank);
7         MPI_Abort(MPI_COMM_WORLD, 1);
8     }
9
10    // 根进程初始化A和C
11    if (rank == 0) {
12        A = (float*)malloc(m * n * sizeof(float));
13        C = (float*)malloc(m * k * sizeof(float));
14        if (A == NULL || C == NULL) {
15            fprintf(stderr, "Failed to allocate memory for A or C\n");
16            ;
17            MPI_Abort(MPI_COMM_WORLD, 1);
18        }
19        initialize_matrix(A, m, n);
20        initialize_matrix(B, n, k);
21        global_start = MPI_Wtime();
22    }
23
24    // 广播矩阵B到所有进程
25    MPI_Bcast(B, n * k, MPI_FLOAT, 0, MPI_COMM_WORLD);
26
27    // 分发A的行到各进程
28    float *local_A = (float*)malloc(rows_per_proc * n * sizeof(float)
29        );
30    if (local_A == NULL) {
31        fprintf(stderr, "Failed to allocate local_A(rank %d)\n",
32            rank);
33        MPI_Abort(MPI_COMM_WORLD, 1);
34    }
35    MPI_Scatter(A, rows_per_proc * n, MPI_FLOAT,
36        local_A, rows_per_proc * n, MPI_FLOAT,
37        0, MPI_COMM_WORLD);
38
39    // 各进程计算局部结果
40    float *local_C = (float*)malloc(rows_per_proc * k * sizeof(float)
41        );
42    if (local_C == NULL) {
43        fprintf(stderr, "Failed to allocate local_C(rank %d)\n",
44            rank);

```

```

39     MPI_Abort(MPI_COMM_WORLD, 1);
40 }
41
42 double local_start = MPI_Wtime();
43 for (int i = 0; i < rows_per_proc; i++) {
44     for (int j = 0; j < k; j++) {
45         float sum = 0.0;
46         for (int l = 0; l < n; l++) {
47             sum += local_A[i * n + l] * B[l * k + j];
48         }
49         local_C[i * k + j] = sum;
50     }
51 }
52 double local_end = MPI_Wtime();
53 double local_comp_time = local_end - local_start;
54
55 // 收集各局部结果到根进程
56 MPI_Gather(local_C, rows_per_proc * k, MPI_FLOAT,
57           C, rows_per_proc * k, MPI_FLOAT,
58           0, MPI_COMM_WORLD);
59
60 // 创建自定义MPI数据类型收集性能数据
61 PerfData my_perf;
62 my_perf.rank = rank;
63 my_perf.comp_time = local_comp_time;
64
65 MPI_Datatype mpi_perf_type;
66 int block_lengths[2] = {1, 1};
67 MPI_Aint displacements[2];
68 MPI_Datatype types[2] = {MPI_INT, MPI_DOUBLE};
69
70 displacements[0] = offsetof(PerfData, rank);
71 displacements[1] = offsetof(PerfData, comp_time);
72
73 MPI_Type_create_struct(2, block_lengths, displacements, types, &
74                       mpi_perf_type);
75 MPI_Type_commit(&mpi_perf_type);
76
77 PerfData *all_perf = NULL;
78 if (rank == 0) {

```

```

78     all_perf = (PerfData*)malloc(size * sizeof(PerfData));
79     if (all_perf == NULL) {
80         fprintf(stderr, "Failed to allocate all_perf\n");
81         MPI_Abort(MPI_COMM_WORLD, 1);
82     }
83 }
84
85 MPI_Gather(&my_perf, 1, mpi_perf_type,
86           all_perf, 1, mpi_perf_type,
87           0, MPI_COMM_WORLD);

```

如何运行程序：

```

1  mpicc -o mpi_matrix  mpi_matrix.c
2  mpirun -np 4 ./mpi_matrix 2048 2048 2048 // 只作为示例，参数可改

```

3 实验结果

3.1 示例输出

如图 1 所示，改变了进程数，计算结果可以保持准确（控制了相同的随机种子去生成矩阵），同时输出了每个进程的 local 计算时间

3.2 结果总结：性能和扩展性分析

表 1 展示了所有的进程数和矩阵规模对应实验得到的结果。其中每项单位为秒。可以注意到，矩阵规模扩大运行时间显著提高，而进程数增加则运行时间减少，这都是十分显然的，说明并行计算能够有效缩短任务执行时间。但是可以注意到，进程增加过多的时候，运行时间的缩减并不完全和进程数呈线性关系，这是因为进程间的数据交换成本随进程数增长，另外频繁的上下文切换导致 L3 缓存命中率下降可能也有影响，这种进程间切换的成本开销已经足够大，以至于甚至如果使用更多进程，已经不能再为计算提速了，反而会减速。另外，当扩大了矩阵的规模，这种通信成本会相较于计算时间来说显得更少，因此矩阵规模更大的时候并行加速更有效

4 优化方向

可以后续加入 openmp 线程并行和 MPI 进程并行共同提高并行程度，另外可以使用 SIMD 向量化，循环分块，流水线等技术优化并行，进一步提高计算速度

```
● (openr1) (base) root@0a2d09754970:~/nfs/parallel computing/lab2# mpirun -np 4 ./mpi_matrix 128 128 128
Matrix A (partial):
  0.33   3.30
  3.31   7.52

Matrix B (partial):
  5.97   4.94
  4.59   1.05

Matrix C (partial):
 3389.84 3598.29
 3214.34 3587.97

Global time elapsed: 0.004795 seconds

Per-process performance:
Rank 0: Local computation time: 0.003613 seconds
Rank 1: Local computation time: 0.003116 seconds
Rank 2: Local computation time: 0.003963 seconds
Rank 3: Local computation time: 0.002186 seconds
● (openr1) (base) root@0a2d09754970:~/nfs/parallel computing/lab2# mpirun -np 8 ./mpi_matrix 128 128 128
Matrix A (partial):
  0.33   3.30
  3.31   7.52

Matrix B (partial):
  5.97   4.94
  4.59   1.05

Matrix C (partial):
 3389.84 3598.29
 3214.34 3587.97

Global time elapsed: 0.002964 seconds

Per-process performance:
Rank 0: Local computation time: 0.001078 seconds
Rank 1: Local computation time: 0.001059 seconds
Rank 2: Local computation time: 0.001065 seconds
Rank 3: Local computation time: 0.001167 seconds
Rank 4: Local computation time: 0.002030 seconds
Rank 5: Local computation time: 0.001932 seconds
Rank 6: Local computation time: 0.001802 seconds
Rank 7: Local computation time: 0.002031 seconds
○ (openr1) (base) root@0a2d09754970:~/nfs/parallel computing/lab2#
```

图 1: 示例输出, 控制了相同的随机种子, 可以看到改变进程数计算结果保持准确

进程数	矩阵规模 (n×n)				
	128	256	512	1024	2048
1	0.009226	0.092216	0.810985	6.819849	73.611963
2	0.004733	0.072287	0.452271	3.453686	41.474989
4	0.004620	0.037701	0.253157	1.693229	23.334249
8	0.002919	0.024012	0.129899	1.059637	11.781277
16	0.002315	0.013495	0.106179	0.917778	7.681015

表 1: 不同进程数与矩阵规模的执行时间 (单位: 秒)