中山大學
SUN YAT-SEN UNIVERSITY

并行程序设计与算法实验

Lab2-基于 MPI 的并行矩阵乘法 (进阶)

姓 名 李源卿

学 号 22336128

学 院 计算机学院

专 业 计算机科学与技术

2025 年 4 月 2 日

1 实验目的

- 掌握 MPI 集合通信在并行矩阵乘法中的应用
- 学习使用 MPI_Type_create_struct 创建派生数据类型
- 分析不同通信方式和任务划分对并行性能的影响
- 研究并行程序的扩展性和性能优化方法

2 实验内容

- 使用 MPI 集合通信实现并行矩阵乘法
- 使用 MPI_Type_create_struct 聚合进程内变量后通信
- 选做：尝试不同数据/任务划分方式
  - 请描述你的数据/任务划分方式。
  - 回答：.....

3 代码说明

3.1 SPMD

与之前使用 Recv 和 Send 不同的是，Bcast，Scatterv 和 Gatherv 需要每个进程都调用。所以没有使用如下格式:

```
    if(rank==0){
        ...
    }
    else{
        ...
    }
```

而是采取：

```
    MPI_Bcast(...);
    if(rank==0){
        ...
    }
    MPI_Scatterv(...);
```

3.2 Scatterv 和 Gatherv

Scatterv 和 Gatherv 的好处在于可以将不同长度的数据散射到各个进程，在无法均匀分配实验数据时，我们可以用这两个函数来实现集合通讯。

```
rows_per_process = (int *)malloc(size * sizeof(int));
displs = (int *)malloc(size * sizeof(int));
recvcounts = (int *)malloc(size * sizeof(int));
// 规划数据分组
int base_rows = m / size;
int remainder = m % size;
int cur_row = 0;
if (rank == 0) {
    for (int i = 0; i < size; i++) {
        // 前 remainder 个进程多处理一行
        rows_per_process[i] = base_rows + (i < remainder ? 1 : 0);
        displs[i] = cur_row * k; //偏移量
        cur_row += rows_per_process[i];
        recvcounts[i] = rows_per_process[i] * k; // 每个进程接收的元
            素数
    }
}
...
MPI_Scatterv(A, recvcounts, displs, MPI_DOUBLE,
        A_local, recvcounts[rank], MPI_DOUBLE,
        0, MPI_COMM_WORLD);
//A-发送缓冲区 recvcounts[i]表示第i个进程接收的数组大小displs[i]表示
   每个进程接收的数据在A中的起始位置。
//0表示0是根进程，像MPI_DOUBLE则是数据类型，MPI_COMM_WORLD是通讯集合
...
MPI_Gatherv(C_local, rows_per_process[rank] * n, MPI_DOUBLE,
        C, recvcounts, displs, MPI_DOUBLE,
        0, MPI_COMM_WORLD);
//C_local 待聚合的本地缓冲区（发送缓冲区），rows_per_process[rank] *
   n为发送数据量，c为接收缓冲区，recvcounts是一个存储了各个进程发送量
   的数组，displs决定了每个进程发送来的数据将在C的什么位置开始记录。
```

3.3 MPI_Type_create_struct

MPI_Type_create_struct 的优势在于可以把不同类型的数据打包，但是在本次实验中不涉及打包不同类型的数据，于是我自定义了三个类型的数据：

3.3.1 mkn_type

```
typedef struct {
    int m;
    int k;
    int n;
} mkn;
MPI_Datatype mkn_type;
{
    int blocklengths[3] = {1, 1, 1};
    MPI_Datatype types[3] = {MPI_INT, MPI_INT, MPI_INT};
    MPI_Aint displacements[3];
    displacements[0] = 0;
    displacements[1] = 4;
    displacements[2] = 8;
    MPI_Type_create_struct(3, blocklengths, displacements, types, &
        mkn_type);
    MPI_Type_commit(&mkn_type);
}
```

3.3.2 size_len_block

以上是比较标准的流程，对于不同类型的数据很有必要，因为每个数据的偏移量可能不同。但是对于相同的数据，像下面这样创建更加方便：

```
MPI_Datatype size_len_block;
{
    int blocklengths = size;
    MPI_Datatype types = MPI_INT;
    MPI_Aint displacements = 0;
    //这里都需要取地址
    MPI_Type_create_struct(1, &blocklengths, &displacements, &types,
        &size_len_block);
    MPI_Type_commit(&size_len_block);
}
```

我还打包了矩阵 B，不过和上面的 size_len_block 一致，就不放到报告里了。

4 实验结果

4.1 正确性

之前的实验其实我都验证过，但是没有在报告里指出。此处用 9×9 的矩阵做验证。

```
test.py > ...
1   import numpy as np
2
3   # 定义两个矩阵
4   A = np.array([[i+j*9+1 for i in range(0,9)] for j in range(0,9)])  # 2x2
5   B = np.array([[i+j*9+1 for i in range(0,9)] for j in range(0,9)])  # 2x2
6
7   # 矩阵乘法
8   C = A @ B
9
10  print(C)
```

```
问题 9   输出   调试控制台   终端   端口

 [17190 17640 18090 18540 18990 19440 19890 20340 20790]
 [20187 20718 21249 21780 22311 22842 23373 23904 24435]
 [23184 23796 24408 25020 25632 26244 26856 27468 28080]
 [26181 26874 27567 28260 28953 29646 30339 31032 31725]]
PS C:\Users\31169\Desktop\parallel> & C:/Users/31169/anaconda3/python.exe c:/Users/31169/Desktop/parallel/
[[ 2205  2250  2295  2340  2385  2430  2475  2520  2565]
 [ 5202  5328  5454  5580  5706  5832  5958  6084  6210]
 [ 8199  8406  8613  8820  9027  9234  9441  9648  9855]
 [11196 11484 11772 12060 12348 12636 12924 13212 13500]
 [14193 14562 14931 15300 15669 16038 16407 16776 17145]
 [17190 17640 18090 18540 18990 19440 19890 20340 20790]
 [20187 20718 21249 21780 22311 22842 23373 23904 24435]
 [23184 23796 24408 25020 25632 26244 26856 27468 28080]
 [26181 26874 27567 28260 28953 29646 30339 31032 31725]]
```

图 1: numpy 结果

```
coder@ccfa80596251:~/project$ mpirun -n 2 ./main

Matrix C (9x9) = A x B:
2205.000000 2250.000000 2295.000000 2340.000000 2385.000000 2430.000000 2475.000000 2520.000000 2565.000000
5202.000000 5328.000000 5454.000000 5580.000000 5706.000000 5832.000000 5958.000000 6084.000000 6210.000000
8199.000000 8406.000000 8613.000000 8820.000000 9027.000000 9234.000000 9441.000000 9648.000000 9855.000000
11196.000000 11484.000000 11772.000000 12060.000000 12348.000000 12636.000000 12924.000000 13212.000000 13500.000000
14193.000000 14562.000000 14931.000000 15300.000000 15669.000000 16038.000000 16407.000000 16776.000000 17145.000000
17190.000000 17640.000000 18090.000000 18540.000000 18990.000000 19440.000000 19890.000000 20340.000000 20790.000000
20187.000000 20718.000000 21249.000000 21780.000000 22311.000000 22842.000000 23373.000000 23904.000000 24435.000000
23184.000000 23796.000000 24408.000000 25020.000000 25632.000000 26244.000000 26856.000000 27468.000000 28080.000000
26181.000000 26874.000000 27567.000000 28260.000000 28953.000000 29646.000000 30339.000000 31032.000000 31725.000000
the max time: 0.000385s
```

图 2: 程序结果

4.2 性能分析

根据运行结果，填入下表以记录不同线程数量和矩阵规模下的运行时间：

表 1: 用 MPI 集合通信实现

进程数	矩阵规模				
	128	256	512	1024	2048
1	0.012799s	0.098125s	0.778544s	6.149306s	44.429201s
2	0.007039s	0.051697s	0.392966s	2.658302s	20.433961s
4	0.004355s	0.027613s	0.220402s	1.378771s	10.776583s
8	0.005023s	0.016130s	0.372274s	1.538535s	10.776583s
16	0.004399s	0.104787s	0.493304s	2.714182s	20.777025s

表 2: 用 MPI_Type_create_struct 聚合进程内变量

进程数	矩阵规模				
	128	256	512	1024	2048
1	0.012969s	0.099214s	0.776359s	6.149796s	40.722858s
2	0.007144s	0.051306s	0.397161s	2.688867s	20.630480s
4	0.004386s	0.023349s	0.202343s	1.347274s	10.715022s
8	0.005222s	0.016376s	0.197371s	1.472587s	11.182086s
16	0.005011s	0.092633s	0.397388s	2.615108s	19.298225s

表 3: 选做题实验结果请填写在此表

进程数	矩阵规模				
	128	256	512	1024	2048
1					
2					
4					
8					
16					

5 实验分析

根据运行时间，分析程序并行性能及扩展性

- 使用 MPI 集合通信实现并行矩阵乘法：是强可拓展的。问题规模为 128 情况下，当进程数扩大为原来两倍的时候，加速比都约为 1.7，效率几乎没变，其余情况加速比都约为 2。所以是强可拓展的。
- 使用 MPI_Type_create_struct 聚合进程内变量后通信：是强可拓展的，当核数加倍时，时间缩短为原来的 1/2 左右（规模为 128 时，问题规模增加为原来 2 倍，加速比约为 1.7，效率保持稳定）。另外在性能方面我们可以看进程数为 16 时，使用普通集合通信和使用 MPI_Type_create_struct 的对比，我们会发现使用 MPI_Type_create_struct 的时间会稍微短一些，这说明虽然创建自定义类型需要耗费一些时间，但是也能带来一些收益。
- 你的方法 (选做)：