并行程序设计与算法实验2

实验	MPI集合通信矩阵乘法	专业	计算机科学与技术
学号	21311525	姓名	马梓培
Email	mazp@mail2.sysu.edu.cn	完成日期	2024/04/02

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。
>
> 1. 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（选做）。
> 2. 对于不同实现方式，调整并记录不同线程数量（1-16）及矩阵规模（128-2048）下的时间开销，填写表格，并分析其性能及扩展性。

2. MPI集合通信并行矩阵乘法实现

并行矩阵乘法的大体思路是用A的部分行与整体B相乘，得到结果C的部分行。因此，在矩阵乘法中，我们可以用MPI_Scatter散射矩阵A到通信子中的各个进程；用MPI_Bcast广播矩阵B到通信子中的各个进程。在各个进程完成自己部分的矩阵乘法后，再通过MPI_Gather将各个进程的localC收集到进程0.

2.1 MPI_Scatter

核0(master core)分发矩阵至[0 to comm_sz - 1]核：

```
int avg_rows = N / comm_sz;
...
// scatter A
MPI_Scatter(A, N * avg_rows, MPI_FLOAT, localA, N * avg_rows, MPI_FLOAT, 0,
MPI_COMM_WORLD);
```

根据MPI_Scatter函数，进行参数的解释：

```
int MPI_Scatter(
    void*         send_buf_p      /* in  */,
    int           send_count      /* in  */,
    MPI_Datatype  send_type       /* in  */,
    void*         recv_buf_p      /* out */,
    int           recv_count      /* in  */,
    MPI_Datatype  recv_type       /* in  */,
    int           src_proc        /* in  */,
    MPI_Comm      comm            /* in  */);
```

将矩阵A（一维数组实现）分成comm_sz份，第一份给0号进程，第二份给1号进程，以此类推。每份的大小为N * avg_rows，即为send_count；数据类型为MPI_FLOAT。接收buffer为localA，同样的，接收大小为N * avg_rows，接收数据类型为MPI_FLOAT，0号进程为源程序，通信子为MPI_COMM_WORLD。

2.2 MPI_Bcast

整个矩阵B广播给所有进程：

```
// broadcast B
MPI_Bcast(B, N * N, MPI_FLOAT, 0, MPI_COMM_WORLD);
```

根据MPI_Scatter函数，进行参数的解释：

```
int MPI_Bcast(
    void*         data_p        /* in/out */,
    int           count         /* in     */,
    MPI_Datatype  datatype      /* in     */,
    int           source_proc   /* in     */,
    MPI_Comm      comm          /* in     */);
```

广播的数据即为矩阵B，数据个数为N * N，数据类型为MPI_FLOAT，源进程为0，处于MPI_COMM_WORLD通信子下。

2.3 MPI_Gather

当各个进程计算结束后，MPI_Gather收集localC，汇聚到矩阵C。

```
// gather c
MPI_Gather(localC, avg_rows * N, MPI_FLOAT, C, avg_rows * N, MPI_FLOAT, 0,
MPI_COMM_WORLD);
```

同样，根据MPI_Gather函数，进行参数的解释：

```
int MPI_Gather(
    void*         send_buf_p      /* in  */,
    int           send_count      /* in  */,
    MPI_Datatype  send_type       /* in  */,
    void*         recv_buf_p      /* out */,
    int           recv_count      /* in  */,
    MPI_Datatype  recv_type       /* in  */,
    int           dest_proc       /* in  */,
    MPI_Comm      comm            /* in  */);
```

数据源为localC，发送数据大小为N * avg_rows，发送数据类型为MPI_FLOAT。矩阵C接收，接收数据大小为N * avg_rows，接收数据类型为MPI_FLOAT，目标进程即为0号进程。处于MPI_COMM_WORLD通信子内。

3. 实验结果

N = 128

```
PP2 ) mpiexec -n 1 mpicol.out
Cost time: 0.005921
PP2 ) mpiexec -n 2 ./mpicol.out
Cost time: 0.002949
PP2 ) mpiexec -n 4 ./mpicol.out
Cost time: 0.001551
PP2 ) mpiexec -n 8 ./mpicol.out
Cost time: 0.000946
PP2 ) mpiexec -n 16 ./mpicol.out
Cost time: 0.000482
```

N = 256

```
PP2 ) mpiexec -n 1 mpicol.out
Cost time: 0.048979
PP2 ) mpiexec -n 2 ./mpicol.out
Cost time: 0.025063
PP2 ) mpiexec -n 4 ./mpicol.out
Cost time: 0.015384
PP2 ) mpiexec -n 8 ./mpicol.out
Cost time: 0.006214
PP2 ) mpiexec -n 16 ./mpicol.out
Cost time: 0.003992
```

N = 512

```
PP2 ) mpiexec -n 1 mpicol.out
Cost time: 0.391363
PP2 ) mpiexec -n 2 ./mpicol.out
Cost time: 0.197304
PP2 ) mpiexec -n 4 ./mpicol.out
Cost time: 0.101385
PP2 ) mpiexec -n 8 ./mpicol.out
Cost time: 0.062479
PP2 ) mpiexec -n 16 ./mpicol.out
Cost time: 0.085388
```

N = 1024

```
PP2 ) mpiexec -n 1 mpicol.out
Cost time: 3.582811
PP2 ) mpiexec -n 2 ./mpicol.out
Cost time: 1.892759
PP2 ) mpiexec -n 4 ./mpicol.out
Cost time: 1.070382
PP2 ) mpiexec -n 8 ./mpicol.out
Cost time: 0.643595
PP2 ) mpiexec -n 16 ./mpicol.out
Cost time: 0.661593
```

N = 2048

```
PP2 ) mpiexec -n 1 mpicol.out
Cost time: 45.888383
PP2 ) mpiexec -n 2 mpicol.out
Cost time: 21.289783
PP2 ) mpiexec -n 4 mpicol.out
Cost time: 11.637980
PP2 ) mpiexec -n 8 mpicol.out
Cost time: 6.418614
PP2 ) mpiexec -n 16 mpicol.out
Cost time: 6.849250
```

根据如上实验结果，得到表格：

P\N	128	256	512	1024	2048
1	0.005921	0.048979	0.391363	3.582811	45.888383
2	0.002949	0.025063	0.197304	1.892759	21.289783
4	0.001551	0.015384	0.101385	1.070382	11.637980
8	0.000946	0.006214	0.062479	0.643595	6.418614
16	0.000482	0.003992	0.085388	0.661593	6.849250

- 横向来看，对于给定的线程数，随着矩阵规模的增加，执行时间增加。这是因为工作量随着矩阵规模的增长呈指数增加。
- 纵向来看，对于给定的矩阵规模，增加线程数通常会减少执行时间，这体现了并行计算的优势。
- 而由于我的虚拟机只分配了8核，因此，使用超过8个线程可能不会给性能带来太多改善，甚至可能会因为线程上下文切换的开销而降低性能。
- 对于较小的矩阵（例如128x128），线程数的增加对执行时间的改善不大。但对于较大的矩阵（2048x2048），增加线程数可以显著降低执行时间，直到达到物理核心的数量。