MPI 编程实习: 用点对点通信实现 MPI_Allgather

1. 内容

用 MPI 点对点通信函数 MPI_Sendrecv() 实现 MPI_Allgather() 函数，同时设计一个测试程序测试所实现的函数的正确性和性能。

2. 目的

了解复杂通信的组织。

3. 算法

采用循环算法。将所有进程按进程号首尾相连排成一个有向环 (最后一个进程与首个进程相连)。各进程先将自己的发送缓冲区拷贝到接收缓冲区的相应位置，然后将数据块发送给下一个进程同时接收前一个进程发来的数据块到接收缓冲区的相应位置，然后再将新收到的数据块发送给下一个进程同时接收前一个进程发来的数据块，如此重复 $p-1$ 次 (p 为进程数)，即完成了 MPI_Allgather() 的操作。图 1 为 4 个进程时的算法示例。

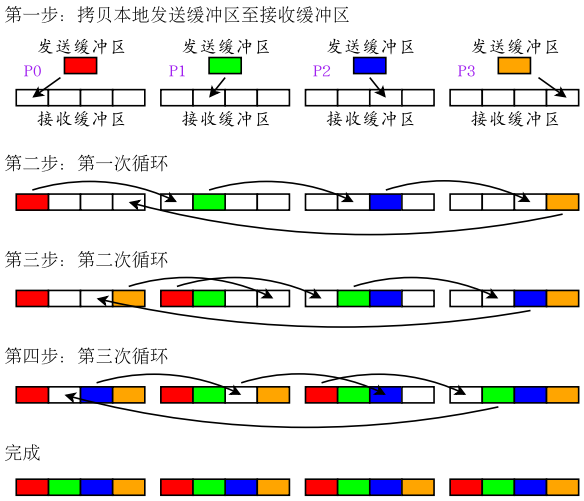

图 1 循环算法实现 MPI_Allgather 函数图示

4. 程序

我们准备了一个用 C 语言编写的程序框架，它负责完成数据准备及结果检查。这个程序先调用一次 MPI_Allgather 并统计其运行时间。要求学员在程序第 76 行处实现 3.的算法, 完成 MPI_Allgather 相同的通信操作。程序源文件可以从下述地址下载:

ftp://ftp.cc.ac.cn/pub/home/zlb/bxjs/allgather-template.c

```
1  /* MPI 程序实例: 用循环算法实现 Allgather
2   *
3   * 整个过程需要 p-1 步完成，每个进程发送 (p-1)*size 数据，接收 (p-1)*size 数据
4   *
5   * 用法: mpirun -np 4 allgather 4m        (数据块大小取 4MB)
6   *
7   * */
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <ctype.h>
12 #include <string.h>
13 #include <mpi.h>
14
```

```
int
My_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype,
             void *recvbuf, int recvcount, MPI_Datatype recvtype,
             MPI_Comm comm)
{
    /* 该函数由学员完成 */
}

/*--------------------------------------------------------------------------*/
#if 1
typedef unsigned char byte;

static void
check(int nprocs, int myrank, size_t size, byte *buffer)
/* 结果检查: 数据块 j 的值应该等于 j + 1 */
{
    size_t i, j;

    for (j = 0; j < nprocs; j++)
        for (i = 0; i < size; i++)
            if (buffer[j * size + i] != ((j + 1) & 255)) {
                fprintf(stderr, "Process %d: incorrect value at block %d, "
                                "position %d\n", myrank, j, i);
                MPI_Abort(MPI_COMM_WORLD, 1);
            }
}

int
main(int argc, char **argv)
{
    int nprocs, myrank;
    byte *send_buffer, *recv_buffer;
    size_t size = 0 /* this makes gcc happy */ ;
    double time0, time1;

    /* 初始化 MPI 环境 */
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    /* 获取命令行参数 */
```

```
112     if (argc != 2) {
113         if (myrank == 0)
114             fprintf(stderr, "Usage:   %s buffersize[K|M|G]\n", argv[0]);
115         /*MPI_Abort(MPI_COMM_WORLD, 1);*/
116         MPI_Finalize();
117         exit(1);
118     }
119     else {
120         char *p;
121         size = strtol(argv[1], &p, 10);
122         switch (toupper(*p)) {
123             case 'G':
124                 size *= 1024;
125             case 'M':
126                 size *= 1024;
127             case 'K':
128                 size *= 1024;
129                 break;
130         }
131     }
132     if (size <= 0) {
133         fprintf(stderr, "Process %d: invalid size %d\n", myrank, size);
134         MPI_Abort(MPI_COMM_WORLD, 1);
135     }
136 
137     if (myrank == 0) {
138         fprintf(stderr, "Allgather with %d processes, buffer size: %d\n",
139                         nprocs, size);
140     }
141 
142     /* 申请发送和接收缓冲区 */
143     send_buffer = malloc(size);
144     recv_buffer = malloc(nprocs * size);
145     if (send_buffer == NULL || recv_buffer == NULL) {
146         fprintf(stderr, "Process %d: memory allocation error!\n", myrank);
147         MPI_Abort(MPI_COMM_WORLD, 1);
148     }
149 
150     /* 发送缓冲区赋值 */
151     memset(send_buffer, myrank + 1, size);
152 
```

```
153     /* 测试 My_Allgather() 函数 */
154
155     /* 清除接收缓冲区 */
156     memset(recv_buffer, 0, nprocs * size);
157     MPI_Barrier(MPI_COMM_WORLD);
158     time0 = MPI_Wtime();
159     My_Allgather(send_buffer, size, MPI_BYTE, recv_buffer, size, MPI_BYTE,
160                  MPI_COMM_WORLD);
161     MPI_Barrier(MPI_COMM_WORLD);
162     time1 = MPI_Wtime();
163     if (myrank == 0)
164         fprintf(stderr, "The circular algorithm: wall time = %lf\n",
165                         time1 - time0);
166     check(nprocs, myrank, size, recv_buffer);
167
168     /* 测试 MPI_Allgather() 函数 */
169
170     /* 清除接收缓冲区 */
171     memset(recv_buffer, 0, nprocs * size);
172     MPI_Barrier(MPI_COMM_WORLD);
173     time0 = MPI_Wtime();
174     MPI_Allgather(send_buffer, size, MPI_BYTE, recv_buffer, size, MPI_BYTE,
175                   MPI_COMM_WORLD);
176     MPI_Barrier(MPI_COMM_WORLD);
177     time1 = MPI_Wtime();
178     if (myrank == 0)
179         fprintf(stderr, "MPI_Allgather: wall time = %lf\n", time1 - time0);
180     check(nprocs, myrank, size, recv_buffer);
181
182     MPI_Finalize();
183     return 0;
184 }
185 #endif
```

该程序运行方式如下 (假设可执行文件名为 `allgather`):

```
mpirun -np 进程数 allgather 数据块大小
```

其中 *数据块大小* 给出数据块的大小，缺省为字节数，如果后面跟随 `K`, `M` 或 `G` 则表示 KB, MB 或 GB。如:

```
mpirun -np 4 allgather 4M
```

表示数据块大小为 4MB。

5. 习题与思考

1. 完成 `allgather.c` 第 76 行处的代码。
2. 比较不同机器上循环算法和 `MPI_Allgather` 函数的通信时间。
3. 考虑、设计其它实现方法，如树型算法 (相当于一次 `MPI_Gather` 调用和一次 `MPI_Bcast` 调用)。论述不同算法的优缺点，并通过实际测试加以验证。