

## 作业 8 MPI 单边通信和共享文件访问(12 月 12 日)

150\*\*\*\*\* 李师尧

参考 4.5.1 节中 2D5P 模板计算的 MPI 点对点通信并程序，以单边通信实现一个 2D5P 模板计算的 MPI 并程序，并将计算结果输出到一个二进制文件中。矩阵在输出文件中按照行优先存储。

首先更正一下教材上 4.5.1 节中的一个错误，如图 1 所示，虚线框住的地方，应该是 1 而不是 0，表示进程的 y 方向的笛卡尔坐标。

```
loc_prob.xs = 1;
loc_prob.xe = loc_prob.lnx + 1;
if ( coords[0]==0 ) loc_prob.xs = 2;
if ( coords[0]==dims[0]-1 ) loc_prob.xe = loc_prob.lnx;
loc_prob.ye = 1;
loc_prob.ye = loc_prob.lny + 1;
if ( coords[0]==0 ) loc_prob.ye = 2;
if ( coords[0]==dims[0]-1 ) loc_prob.ye = loc_prob.lny;
```

图 1 教材中的一个小错误

本题的两大知识点是窗口访问和并行 IO，关键的代码段如图 2 和图 3 所示。

```
122 void ExchangeByWin(MPI_Comm &comm, Prob &prob) {
123     MPI_Status status;
124     MPI_Win win;
125     MPI_Win_create((void*)prob.val,
126                   sizeof(float)*(prob.lnx+2)*(prob.lny+2),
127                   sizeof(float),
128                   MPI_INFO_NULL,
129                   MPI_COMM_WORLD,
130                   &win);
131     for(int i=0;i<parms.nts;i++)
132     {
133         MPI_Win_fence(0,win);
134
135         MPI_Get(prob.val, 1,vecd0,nld0,prob.lnx*(prob.lny+2),1,vecd0,win);
136         MPI_Get(prob.val+(prob.lny+2)*(prob.lnx+1),1,vecd0,nrd0,prob.lny+2, 1,vecd0,win);
137         MPI_Get(prob.val, 1,vecd1,nld1,prob.lny, 1,vecd1,win);
138         MPI_Get(prob.val+prob.lny+1, 1,vecd1,nrd1,1, 1,vecd1,win);
139
140         update_data(prob);
141         MPI_Win_fence(0,win);
142     }
143     MPI_Win_free(&win);
144 }
145 }
```

图 2 窗口访问的代码

```
218 MPI_File fh;
219 MPI_Offset disp,fsize;
220 MPI_File_open(MPI_COMM_WORLD, "2d5p.dat", MPI_MODE_RDWR|MPI_MODE_CREATE,MPI_INFO_NULL, &fh);
221 disp=(NXPROB/dims[0]+2)*(NYPROB/dims[1]+2)*sizeof(float)*rank;
222 fsize=sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2);
223 MPI_File_set_size(fh, fsize);
224 MPI_File_set_view(fh, disp, MPI_FLOAT, MPI_FLOAT, "native", MPI_INFO_NULL);
225 //printf("%d %ld %ld \n", rank, (NXPROB/dims[0]+2)*(NYPROB/dims[1]+2)*sizeof(float)*rank, size
226 MPI_File_write_all(fh, loc_prob.val, (loc_prob.lnx+2)*(loc_prob.lny+2), MPI_FLOAT, &status);
227
228 MPI_File_close(&fh);
```

图 3 并行 IO 的代码

最后实现的效果，如图 4 所示。并行环境为 24 核小工作站，每个 CPU 主频为 2.5GHz。根据两个不同 CPU 规模的结果，标准的阻塞式通信耗时最长，而非阻塞式、就绪模式和窗口访问模式均能提高通信速度，提高并行效率。从编写代码来看，窗口访问的方式较为便捷，可以在实践中推广。

```
业八\ltcs@ltcs-ThinkStation-C30:~/lsy0511/作业八$ mpirun -np 10 ./2d5p-win
standard blocking: 5.111192
standard nonblocking: 3.938875
ready blocking: 2.593024
window: 2.650772
业八\ltcs@ltcs-ThinkStation-C30:~/lsy0511/作业八$ mpirun -np 5 ./2d5p-win
standard blocking: 5.015412
standard nonblocking: 4.932870
ready blocking: 4.930240
window: 4.974005
业八\ltcs@ltcs-ThinkStation-C30:~/lsy0511/作业八$ _
```

图 4 实现效果截图

```

1 //2016年12月12日23:32:00, 1501214648 李师尧
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <mpi.h>
6 #include <time.h>
7 #define NXPROB 500
8 #define NYPROB 500
9 #define NANO 1000000000
10 struct Params {
11     float cx;
12     float cy;
13     int nts;
14 } parms = {0.1, 0.1, 5000};
15 struct Prob {
16     int lbx, lby;
17     int lnx, lny;
18     int xs, xe, ys, ye;
19     float *val, *temp;
20 };
21 MPI_Datatype vecd0, vecd1;
22 int dims[2], periods[2], coords[2], nld0, nrd0, nld1, nrd1;
23 //nld0: left neighbor in dim[0]; nrd0: right neighbor in dim[0]
24 float *bufsendnld0, *bufrecvnld0, *bufsendnrd0, *bufrecvnrd0,
25 *bufsendnld1, *bufrecvnld1, *bufsendnrd1, *bufrecvnrd1;
26 //bufsendnld0: buf for sending data to left neighbor in dim[0]
27 //bufsendnrd0: buf for sending data to right neighbor in dim[0]
28 //bufrecvnld0: buf for receiving data to left neighbor in dim[0]
29 //bufrecvnrd0: buf for receiving data to right neighbor in dim[0]
30 void init_data(Prob &prob) {
31     for (int x = 1; x <= prob.lnx; x++) {
32         for (int y = 1; y <= prob.lny; y++) {
33             float ix = prob.lbx + x - 1;
34             float iy = prob.lby + y - 1;
35             prob.val[x*(prob.lny+2) + y] = (ix * (NXPROB - ix - 1) * iy * (NYPROB - iy - 1));
36         }
37     }
38 }
39 void set_buf(Prob &prob) {
40     bufrecvnld0 = prob.val + 1;
41     bufsendnld0 = prob.val + prob.lny + 3;
42     bufsendnrd0 = prob.val + (prob.lny + 2)*prob.lnx + 1;
43     bufrecvnrd0 = prob.val + (prob.lny + 2)*(prob.lnx + 1) + 1;
44     bufrecvnld1 = prob.val;
45     bufsendnld1 = prob.val + 1;
46     bufsendnrd1 = prob.val + prob.lny;

```

```

47     bufrecvnrd1 = prob.val + prob.lny + 1;
48 }
49 void update_data(Prob &prob) {
50     int extenty = prob.lny+2;
51     for (int x = prob.xs; x < prob.xe; x++) {
52         for (int y = prob.ys; y < prob.ye; y++) {
53             prob.temp[x*extenty + y] = prob.val[x*extenty + y]
54                 + parms.cx * (prob.val[(x-1)*extenty + y] +
55                     prob.val[(x+1)*extenty +
56                         y] - 2.0 *
57                         prob.val[x*extenty + y])
58                 + parms.cx *
59                 (prob.val[x*extenty + y-1]
60                 + prob.val[x*extenty + y+1]
61                 - 2.0 * prob.val[x*extenty
62                     + y]);
63         }
64     }
65     float *temp = prob.val;
66     prob.val = prob.temp;
67     prob.temp = temp;
68 }
69 void standard_blocking(MPI_Comm &comm, Prob &prob) {
70     MPI_Status status;
71     for(int i=0; i<parms.nts; i++) {
72         set_buf(prob);
73         MPI_Send(bufsendnld0, 1, vecd0, nld0, 50, comm);
74         MPI_Recv(bufrecvnrd0, 1, vecd0, nrd0, 50, comm, &status);
75         MPI_Send(bufsendnrd0, 1, vecd0, nrd0, 50, comm);
76         MPI_Recv(bufrecvnld0, 1, vecd0, nld0, 50, comm, &status);
77         MPI_Send(bufsendnld1, 1, vecd1, nld1, 50, comm);
78         MPI_Recv(bufrecvnrd1, 1, vecd1, nrd1, 50, comm, &status);
79         MPI_Send(bufsendnrd1, 1, vecd1, nrd1, 50, comm);
80         MPI_Recv(bufrecvnld1, 1, vecd1, nld1, 50, comm, &status);
81         update_data(prob);
82     }
83 }
84 void standard_nonblocking(MPI_Comm &comm, Prob &prob) {
85     MPI_Status status[8];
86     MPI_Request request[8];
87     for(int i=0; i<parms.nts; i++) {
88         set_buf(prob);
89         MPI_Isend(bufsendnrd0, 1, vecd0, nrd0, 50, comm,
90             &request[0]);
91         MPI_Isend(bufsendnld0, 1, vecd0, nld0, 50, comm,
92             &request[1]);
93         MPI_Isend(bufsendnld1, 1, vecd1, nld1, 50, comm,
94             &request[2]);

```

```

86     MPI_Isend(bufsendnrd1, 1, vecd1, nrd1, 50, comm,
87             &request[3]);
88     MPI_Irecv(bufrecvnld0, 1, vecd0, nld0, 50, comm,
89             &request[4]);
90     MPI_Irecv(bufrecvnrd0, 1, vecd0, nrd0, 50, comm,
91             &request[5]);
92     MPI_Irecv(bufrecvnrd1, 1, vecd1, nrd1, 50, comm,
93             &request[6]);
94     MPI_Irecv(bufrecvnld1, 1, vecd1, nld1, 50, comm,
95             &request[7]);
96     MPI_Waitall(8, request, status);
97     update_data(prob);
98 }
99 }
100 void ready_blocking(MPI_Comm &comm, Prob &prob) {
101     MPI_Status status[4];
102     MPI_Request request[4];
103     char msg_send[]="ready", msg_recv[10];
104     for(int i=0; i<parms.nts; i++) {
105         set_buf(prob);
106         MPI_Irecv(bufrecvnld0, 1, vecd0, nld0, 50, comm,
107                 &request[0]);
108         MPI_Irecv(bufrecvnrd0, 1, vecd0, nrd0, 50, comm,
109                 &request[1]);
110         MPI_Irecv(bufrecvnrd1, 1, vecd1, nrd1, 50, comm,
111                 &request[2]);
112         MPI_Irecv(bufrecvnld1, 1, vecd1, nld1, 50, comm,
113                 &request[3]);
114         MPI_Send(msg_send, strlen(msg_send), MPI_CHAR, nld0,
115                 60, comm);
116         MPI_Send(msg_send, strlen(msg_send), MPI_CHAR, nrd0,
117                 60, comm);
118         MPI_Send(msg_send, strlen(msg_send), MPI_CHAR, nld1,
119                 60, comm);
120         MPI_Send(msg_send, strlen(msg_send), MPI_CHAR, nrd1,
121                 60, comm);
122         MPI_Recv(msg_recv, 10, MPI_CHAR, nrd0, 60, comm,
123                 &status[0]);
124         MPI_Recv(msg_recv, 10, MPI_CHAR, nld0, 60, comm,
125                 &status[0]);
126         MPI_Recv(msg_recv, 10, MPI_CHAR, nrd1, 60, comm,
127                 &status[0]);
128         MPI_Recv(msg_recv, 10, MPI_CHAR, nld1, 60, comm,
129                 &status[0]);
130         MPI_Rsend(bufsendnld0, 1, vecd0, nld0, 50, comm);
131         MPI_Rsend(bufsendnrd0, 1, vecd0, nrd0, 50, comm);
132         MPI_Rsend(bufsendnld1, 1, vecd1, nld1, 50, comm);
133         MPI_Rsend(bufsendnrd1, 1, vecd1, nrd1, 50, comm);

```

```

117         MPI_Waitall(4, request, status);
118         update_data(prob);
119     }
120 }
121
122 void ExchangeByWin(MPI_Comm &comm, Prob &prob) {
123     MPI_Status status;
124     MPI_Win win;
125     MPI_Win_create((void*)prob.val,
126                   sizeof(float)*(prob.lnx+2)*(prob.lny+2),
127                   sizeof(float),
128                   MPI_INFO_NULL,
129                   MPI_COMM_WORLD,
130                   &win);
131     for(int i=0;i<parms.nts;i++)
132     {
133         MPI_Win_fence(0,win);
134
135         MPI_Get(prob.val,
136               1,vecd0,nld0,prob.lnx*(prob.lny+2),1,vecd0,win);
137
138         MPI_Get(prob.val+(prob.lny+2)*(prob.lnx+1),1,vecd0,nrd0,p
139               rob.lny+2,1,vecd0,win);
140         MPI_Get(prob.val,
141               1,vecd1,nld1,prob.lny,1,vecd1,win);
142         MPI_Get(prob.val+prob.lny+1,
143               1,vecd1,nrd1,1,1,vecd1,win);
144
145         update_data(prob);
146         MPI_Win_fence(0,win);
147     }
148     MPI_Win_free(&win);
149 }
150
151 int main(int argc, char**argv) {
152     int rank, size;
153     int source, dest, tag=50;
154     double time_standard_blocking, time_standard_nonblocking,
155           time_ready_blocking,time_win;
156     Prob loc_prob;
157     MPI_Comm comm_cart;
158     MPI_Status status;
159     MPI_Init(&argc, &argv);
160     MPI_Comm_size(MPI_COMM_WORLD, &size);
161     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
162     MPI_Dims_create(size, 2, dims);
163     //将size个进程分成二维结构，每个维度的进程数目存储在dims数组
164     里

```

```

157     periods[0]=false; periods[1]=false;
158     //为下面的创建做准备，表示没有周期性，不连接成环
159     MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, false,
160     &comm_cart);    //创建一个新的通信器
161     MPI_Comm_rank(comm_cart, &rank);    //获取在新通信器中的编号
162     MPI_Cart_get(comm_cart, 2, dims, periods, coords);
163     //获取维数、周期性、以及进程的笛卡尔坐标
164     MPI_Cart_shift(comm_cart, 0, 1, &nld0, &nrd0);
165     //计算数据发送接收的源地址和目的地址
166     MPI_Cart_shift(comm_cart, 1, 1, &nld1, &nrd1);
167     if (comm_cart==MPI_COMM_NULL) {
168         MPI_Finalize();
169         return 0;
170     }
171
172     loc_prob.lnx = NXPROB/dims[0];
173     //进程负责的局部数据块的x方向长度
174     if ( coords[0] < NXPROB%dims[0] ) {
175         loc_prob.lbx = loc_prob.lnx * coords[0] + coords[0];
176         loc_prob.lnx++;
177     } else
178         loc_prob.lbx = loc_prob.lnx * coords[0] + NXPROB%dims[0];
179     loc_prob.lny = NYPROB/dims[1];
180     //进程负责的局部数据块的y方向长度
181     if ( coords[1] < NXPROB%dims[1] ) {
182         loc_prob.lby = loc_prob.lny * coords[1] + coords[1];
183         loc_prob.lny++;
184     } else
185         loc_prob.lby = loc_prob.lny * coords[1] + NYPROB%dims[1];
186
187     loc_prob.xs = 1;
188     loc_prob.xe = loc_prob.lnx + 1;
189     if ( coords[0]==0 ) loc_prob.xs = 2;
190     if ( coords[0]==dims[0]-1 ) loc_prob.xe = loc_prob.lnx;
191     loc_prob.ys = 1;
192     loc_prob.ye = loc_prob.lny + 1;
193     if ( coords[1]==0 ) loc_prob.ys = 2;
194     if ( coords[1]==dims[0]-1 ) loc_prob.ye = loc_prob.lny;
195
196     loc_prob.val =
197     (float*)malloc(sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2
198     ));
199     loc_prob.temp =
200     (float*)malloc(sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2
201     ));
202     memset(loc_prob.val, 0,
203     sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2));
204     memset(loc_prob.temp, 0,

```



```

194     sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2));
195     init_data(loc_prob);
196
197     MPI_Type_contiguous(loc_prob.lny, MPI_FLOAT, &vecd0);
198     //创建连续存放的数据类型
199     MPI_Type_vector(loc_prob.lnx+2, 1, loc_prob.lny+2,
200     MPI_FLOAT, &vecd1); //创建矢量性质的数据类型
201     MPI_Type_commit(&vecd0); //提交数据类型
202     MPI_Type_commit(&vecd1); //提交数据类型
203
204     time_standard_blocking=MPI_Wtime();
205     standard_blocking(comm_cart, loc_prob);
206     if (rank==0) printf("standard blocking: %f \n",
207     MPI_Wtime()-time_standard_blocking);
208
209     time_standard_nonblocking=MPI_Wtime();
210     standard_nonblocking(comm_cart, loc_prob);
211     if (rank==0) printf("standard nonblocking: %f \n",
212     MPI_Wtime()-time_standard_nonblocking);
213
214     time_ready_blocking=MPI_Wtime();
215     ready_blocking(comm_cart, loc_prob);
216     if (rank==0) printf("ready blocking: %f \n",
217     MPI_Wtime()-time_ready_blocking);
218
219     time_win=MPI_Wtime();
220     ExchangeByWin(comm_cart, loc_prob);
221     if (rank==0) printf("window: %f \n", MPI_Wtime()-time_win);
222
223     MPI_File fh;
224     MPI_Offset disp, fsize;
225     MPI_File_open(MPI_COMM_WORLD, "2d5p.dat",
226     MPI_MODE_RDWR | MPI_MODE_CREATE, MPI_INFO_NULL, &fh);
227
228     disp=(NXPROB/dims[0]+2)*(NYPROB/dims[1]+2)*sizeof(float)*rank
229     ;
230     fsize=sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2);
231     MPI_File_set_size(fh, fsize);
232     MPI_File_set_view(fh, disp, MPI_FLOAT, MPI_FLOAT, "native",
233     MPI_INFO_NULL);
234     //printf("%d %ld %ld \n", rank,
235     (NXPROB/dims[0]+2)*(NYPROB/dims[1]+2)*sizeof(float)*rank,
236     sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2));
237     MPI_File_write_all(fh, loc_prob.val,
238     (loc_prob.lnx+2)*(loc_prob.lny+2), MPI_FLOAT, &status);
239
240     MPI_File_close(&fh);

```



```
229
230     MPI_Type_free(&vecd0);
231     MPI_Type_free(&vecd1);
232     free(loc_prob.val);
233     free(loc_prob.temp);
234     MPI_Comm_free(&comm_cart);
235     MPI_Finalize();
236 }
```