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 点对点通信并行程序，以单边通信实现一个 2D5P 模板计算的 MPI 并行程序，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.ys = 1;
loc_prob.ye = loc_prob.lny + 1;
if ( coords[0]==0 ) loc_prob.ys = 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。

```
void ExcangeByWin(MPI_Comm &comm, Prob &prob) {
    MPI_Status status;
    MPI_Win win;
    MPI_Win_create((void*)prob.val,
                sizeof(float)*(prob.lnx+2)*(prob.lny+2),
                sizeof(float),
                MPI_INFO_NULL,
                MPI_COMM_WORLD,
                &win);
    for(int i=0;i<parms.nts;i++)
    {
        MPI_Win_fence(0,win);

        MPI_Get(prob.val,                         1,vecd0,nld0,prob.lnx*(prob.lny+2),1,vecd0,win);
        MPI_Get(prob.val+(prob.lny+2)*(prob.lnx+1),1,vecd0,nrd0,prob.lny+2,          1,vecd0,win);
        MPI_Get(prob.val,                         1,vecd1,nld1,prob.lny,             1,vecd1,win);
        MPI_Get(prob.val+prob.lny+1,              1,vecd1,nrd1,1,                    1,vecd1,win);

        update_data(prob);
        MPI_Win_fence(0,win);
    }
    MPI_Win_free(&win);

}
```

图 2 窗口访问的代码

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

    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

```cpp
//2016年12月12日23:32:00, 1501214648  李师尧
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>
#include <time.h>
#define NXPROB 500
#define NYPROB 500
#define NANO           1000000000
struct Parms {
    float cx;
    float cy;
    int nts;
} parms = {0.1, 0.1, 5000};
struct Prob {
    int lbx, lby;
    int lnx, lny;
    int xs, xe, ys, ye;
    float *val, *temp;
};
MPI_Datatype vecd0, vecd1;
int dims[2], periods[2], coords[2], nld0, nrd0, nld1, nrd1;
//nld0: left neigbor in dim[0]; nrd0: right neigbor in dim[0]
float *bufsendnld0, *bufrecvnld0, *bufsendnrd0, *bufrecvnrd0, *bufsendnld1, *bufrecvnld1,*bufsendnrd1, *bufrecvnrd1;
//bufsendnld0: buf for sending data to left neigbor in dim[0]
//bufsendnrd0: buf for sending data to right neigbor in dim[0]
//bufrecvnld0: buf for recieving data to left neigbor in dim[0]
//bufrecvnrd0: buf for recieving data to right neigbor in dim[0]
void init_data(Prob &prob) {
    for (int x = 1; x <= prob.lnx; x++) {
        for (int y = 1; y <= prob.lny; y++) {
            float ix = prob.lbx + x - 1;
            float iy = prob.lby + y - 1;
            prob.val[x*(prob.lny+2) + y] = (ix * (NXPROB - ix - 1) * iy * (NYPROB - iy - 1));
        }
    }
}

void set_buf(Prob &prob) {
    bufrecvnld0 = prob.val + 1;
    bufsendnld0 = prob.val + prob.lny + 3;
    bufsendnrd0 = prob.val + (prob.lny + 2)*prob.lnx + 1;
    bufrecvnrd0 = prob.val + (prob.lny + 2)*(prob.lnx + 1) + 1;
    bufrecvnld1 = prob.val;
    bufsendnld1 = prob.val + 1;
    bufsendnrd1 = prob.val + prob.lny;
```

```
    bufrecvnrd1 = prob.val + prob.lny + 1;
}
void update_data(Prob &prob) {
    int extenty = prob.lny+2;
    for (int x = prob.xs; x < prob.xe; x++) {
        for (int y = prob.ys; y < prob.ye; y++) {
            prob.temp[x*extenty + y] = prob.val[x*extenty + y] +parms.cx * (prob.val[(x-1)*extenty + y] +
                                                prob.val[(x+1)*extenty + y] - 2.0 * prob.val[x*extenty + y])
                                            +   parms.cx * (prob.val[x*extenty + y-1] + prob.val[x*extenty + y+1]
                                            - 2.0 * prob.val[x*extenty + y]);
        }
    }
    float *temp = prob.val;
    prob.val = prob.temp;
    prob.temp = temp;
}
void standard_blocking(MPI_Comm &comm, Prob &prob) {
    MPI_Status status;
    for(int i=0; i<parms.nts; i++) {
        set_buf(prob);
        MPI_Send(bufsendnld0, 1, vecd0, nld0, 50, comm);
        MPI_Recv(bufrecvnrd0, 1, vecd0, nrd0, 50, comm, &status);
        MPI_Send(bufsendnrd0, 1, vecd0, nrd0, 50, comm);
        MPI_Recv(bufrecvnld0, 1, vecd0, nld0, 50, comm, &status);
        MPI_Send(bufsendnld1, 1, vecd1, nld1, 50, comm);
        MPI_Recv(bufrecvnrd1, 1, vecd1, nrd1, 50, comm, &status);
        MPI_Send(bufsendnrd1, 1, vecd1, nrd1, 50, comm);
        MPI_Recv(bufrecvnld1, 1, vecd1, nld1, 50, comm, &status);
        update_data(prob);
    }
}
void standard_nonblocking(MPI_Comm &comm, Prob &prob) {
    MPI_Status status[8];
    MPI_Request request[8];
    for(int i=0; i<parms.nts; i++) {
        set_buf(prob);
        MPI_Isend(bufsendnrd0, 1, vecd0, nrd0, 50, comm, &request[0]);
        MPI_Isend(bufsendnld0, 1, vecd0, nld0, 50, comm, &request[1]);
        MPI_Isend(bufsendnld1, 1, vecd1, nld1, 50, comm, &request[2]);
```

```
        MPI_Isend(bufsendnrd1, 1, vecd1, nrd1, 50, comm, &request[3]);
        MPI_Irecv(bufrecvnld0, 1, vecd0, nld0, 50, comm, &request[4]);
        MPI_Irecv(bufrecvnrd0, 1, vecd0, nrd0, 50, comm, &request[5]);
        MPI_Irecv(bufrecvnrd1, 1, vecd1, nrd1, 50, comm, &request[6]);
        MPI_Irecv(bufrecvnld1, 1, vecd1, nld1, 50, comm, &request[7]);
        MPI_Waitall(8, request, status);
        update_data(prob);
    }
}
void ready_blocking(MPI_Comm &comm, Prob &prob) {
    MPI_Status status[4];
    MPI_Request request[4];
    char msg_send[]="ready", msg_recv[10];
    for(int i=0; i<parms.nts; i++) {
        set_buf(prob);
        MPI_Irecv(bufrecvnld0, 1, vecd0, nld0, 50, comm, &request[0]);
        MPI_Irecv(bufrecvnrd0, 1, vecd0, nrd0, 50, comm, &request[1]);
        MPI_Irecv(bufrecvnrd1, 1, vecd1, nrd1, 50, comm, &request[2]);
        MPI_Irecv(bufrecvnld1, 1, vecd1, nld1, 50, comm, &request[3]);
        MPI_Send(msg_send, strlen(msg_send), MPI_CHAR, nld0, 60, comm);
        MPI_Send(msg_send, strlen(msg_send), MPI_CHAR, nrd0, 60, comm);
        MPI_Send(msg_send, strlen(msg_send), MPI_CHAR, nld1, 60, comm);
        MPI_Send(msg_send, strlen(msg_send), MPI_CHAR, nrd1, 60, comm);
        MPI_Recv(msg_recv, 10, MPI_CHAR, nrd0, 60, comm, &status[0]);
        MPI_Recv(msg_recv, 10, MPI_CHAR, nld0, 60, comm, &status[0]);
        MPI_Recv(msg_recv, 10, MPI_CHAR, nrd1, 60, comm, &status[0]);
        MPI_Recv(msg_recv, 10, MPI_CHAR, nld1, 60, comm, &status[0]);
        MPI_Rsend(bufsendnld0, 1, vecd0, nld0, 50, comm);
        MPI_Rsend(bufsendnrd0, 1, vecd0, nrd0, 50, comm);
        MPI_Rsend(bufsendnld1, 1, vecd1, nld1, 50, comm);
        MPI_Rsend(bufsendnrd1, 1, vecd1, nrd1, 50, comm);
```

```
            MPI_Waitall(4, request, status);
            update_data(prob);
        }
    }

void ExcangeByWin(MPI_Comm &comm, Prob &prob) {
        MPI_Status status;
        MPI_Win win;
        MPI_Win_create((void*)prob.val,
                        sizeof(float)*(prob.lnx+2)*(prob.lny+2),
                        sizeof(float),
                        MPI_INFO_NULL,
                        MPI_COMM_WORLD,
                        &win);
        for(int i=0;i<parms.nts;i++)
        {
            MPI_Win_fence(0,win);

            MPI_Get(prob.val, 1,vecd0,nld0,prob.lnx*(prob.lny+2),1,vecd0,win);

            MPI_Get(prob.val+(prob.lny+2)*(prob.lnx+1),1,vecd0,nrd0,prob.lny+2,           1,vecd0,win);
            MPI_Get(prob.val, 1,vecd1,nld1,prob.lny,                  1,vecd1,win);
            MPI_Get(prob.val+prob.lny+1, 1,vecd1,nrd1,1,                  1,vecd1,win);

            update_data(prob);
            MPI_Win_fence(0,win);
        }
        MPI_Win_free(&win);

    }
    int main(int argc, char**argv) {
        int rank, size;
        int source, dest, tag=50;
        double time_standard_blocking, time_standard_nonblocking, time_ready_blocking,time_win;
        Prob loc_prob;
        MPI_Comm comm_cart;
        MPI_Status status;
        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Dims_create(size, 2, dims);
        //将size个进程分成二维结构，每个维度的进程数目存储在dims数组里
```

```cpp
    periods[0]=false; periods[1]=false; //为下面的创建做准备，表示没有周期性，不连接成环
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, false, &comm_cart);   //创建一个新的通信器
    MPI_Comm_rank(comm_cart, &rank);   //获取在新通信器中的编号
    MPI_Cart_get(comm_cart, 2, dims, periods, coords); //获取维数、周期性、以及进程的笛卡尔坐标
    MPI_Cart_shift(comm_cart, 0, 1, &nld0, &nrd0); //计算数据发送接收的源地址和目的地址
    MPI_Cart_shift(comm_cart, 1, 1, &nld1, &nrd1);
    if (comm_cart==MPI_COMM_NULL) {
        MPI_Finalize();
        return 0;
    }

    loc_prob.lnx = NXPROB/dims[0]; //进程负责的局部数据块的x方向长度
    if ( coords[0] < NXPROB%dims[0] ) {
        loc_prob.lbx = loc_prob.lnx * coords[0] + coords[0];
        loc_prob.lnx++;
    } else
        loc_prob.lbx = loc_prob.lnx * coords[0] + NXPROB%dims[0];
    loc_prob.lny = NYPROB/dims[1]; ////进程负责的局部数据块的y方向长度
    if ( coords[1] < NXPROB%dims[1] ) {
        loc_prob.lby = loc_prob.lny * coords[1] + coords[1];
        loc_prob.lny++;
    } else
        loc_prob.lby = loc_prob.lny * coords[1] + NYPROB%dims[1];

    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.ys = 1;
    loc_prob.ye = loc_prob.lny + 1;
    if ( coords[1]==0 ) loc_prob.ys = 2;
    if ( coords[1]==dims[0]-1 ) loc_prob.ye = loc_prob.lny;

    loc_prob.val = (float*)malloc(sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2));
    loc_prob.temp = (float*)malloc(sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2));
    memset(loc_prob.val, 0, sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2));
    memset(loc_prob.temp, 0,
```

```
        sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2));

        init_data(loc_prob);

        MPI_Type_contiguous(loc_prob.lny, MPI_FLOAT, &vecd0); //创建连续存放的数据类型
        MPI_Type_vector(loc_prob.lnx+2, 1, loc_prob.lny+2, MPI_FLOAT, &vecd1);  //创建矢量性质的数据类型
        MPI_Type_commit(&vecd0);   //提交数据类型
        MPI_Type_commit(&vecd1);   //提交数据类型

        time_standard_blocking=MPI_Wtime();
        standard_blocking(comm_cart, loc_prob);
        if (rank==0) printf("standard blocking: %f \n", MPI_Wtime()-time_standard_blocking);

        time_standard_nonblocking=MPI_Wtime();
        standard_nonblocking(comm_cart, loc_prob);
        if (rank==0) printf("standard nonblocking: %f \n", MPI_Wtime()-time_standard_nonblocking);

        time_ready_blocking=MPI_Wtime();
        ready_blocking(comm_cart, loc_prob);
        if (rank==0) printf("ready blocking: %f \n", MPI_Wtime()-time_ready_blocking);

        time_win=MPI_Wtime();
        ExcangeByWin(comm_cart, loc_prob);
        if (rank==0) printf("window: %f \n", MPI_Wtime()-time_win);

        MPI_File fh;
        MPI_Offset disp,fsize;
        MPI_File_open(MPI_COMM_WORLD, "2d5p.dat", MPI_MODE_RDWR|MPI_MODE_CREATE,MPI_INFO_NULL, &fh);

        disp=(NXPROB/dims[0]+2)*(NYPROB/dims[1]+2)*sizeof(float)*rank;
        fsize=sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2);
        MPI_File_set_size(fh, fsize);
        MPI_File_set_view(fh, disp, MPI_FLOAT, MPI_FLOAT, "native", MPI_INFO_NULL);
        //printf("%d %ld %ld \n", rank, (NXPROB/dims[0]+2)*(NYPROB/dims[1]+2)*sizeof(float)*rank, sizeof(float)*(loc_prob.lnx+2)*(loc_prob.lny+2));
        MPI_File_write_all(fh, loc_prob.val, (loc_prob.lnx+2)*(loc_prob.lny+2), MPI_FLOAT, &status);

        MPI_File_close(&fh);
```

```

    MPI_Type_free(&vecd0);
    MPI_Type_free(&vecd1);
    free(loc_prob.val);
    free(loc_prob.temp);
    MPI_Comm_free(&comm_cart);
    MPI_Finalize();
}
```