Homework 4 for Chapter 8-9

120033910005

May 2, 2021

Problem 8.16

the overall process is similar to the program shown in Figure 8.14. except that We use the 'read_replicated_vector' function to replicate the vector to all process. And the corresponding product operation is changed to

```
for (i = 0; i < m; i++) {
    c_part_out[i] = 0.0;
    for (j = 0; j < BLOCK_SIZE(id,p,n); j++)
        c_part_out[i] += a[i][j] * b[BLOCK_LOW(id,p,n) + j];
}
```

and the result is as follows

```
>>mpirun -np 4 ./8-6/exer08_06 ./8-6/matrix.txt ./8-6/vector.txt
m = 4, n = 5
 2.000  1.000  3.000  4.000  0.000
 5.000 -1.000  2.000 -2.000  4.000
 0.000  3.000  4.000  1.000  2.000
 2.000  3.000  1.000 -3.000  0.000

n = 5
 3.000  1.000  4.000  0.000  3.000

19.000 34.000 25.000 13.000
```

Problem 8.12

Problem 9.7

In manager process, we create a, b, c. a is the matrix. b is the vector. c is the result vector.

$$a * b = c$$

We broadcast the b vector to all process. For each row of a, we assign it to one worker process and get the result(which is a single number).

```
void manager (int argc, char *argv[], int p) {

    double *a;          /* Store matrix here */
    double *b;           /* Store vector here */
    double *c;
    double res;
    int n;               /* column of matrix */
    int m;               /* row of matrix */
    int i, j;
    int assign_cnt;
```

```
// MPI_Request pending;/* Handle for recv request */
int src;              /* Message source process */
MPI_Status status;    /* Message status */
int tag;              /* Message tag */
int terminated;       /* Count of terminated procs */
FILE * infileptr;
/* read matrix and vectors */

infileptr = fopen(argv[INPUT_ARG], "r");
if (infileptr == NULL){
    m = 0; n = 0;
    printf("failed to open file: %s\n", argv[INPUT_ARG]);
    MPI_Abort(MPI_COMM_WORLD, OPEN_FILE_ERROR);
}else{
    fscanf(infileptr, "%d %d", &m, &n);
    printf("m = %d, n = %d\n", m, n);
}
a = (double*) malloc(m * n * sizeof(double));
b = (double*) malloc(n * sizeof(double));
c = (double*) malloc(m *sizeof(double));
if (a == NULL || b == NULL || c == NULL){
    printf("error in manager process: can allocate enough \
            memory for a, b, c\n");
    MPI_Abort(MPI_COMM_WORLD, MALLOC_ERROR);
}
for (i = 0; i < m; ++i){
    for (j = 0; j < n; ++j)
        fscanf(infileptr, "%lf", a + i * n + j);
}
for (j = 0; j < n; ++j)
    fscanf(infileptr, "%lf", b + j);
fclose(infileptr);

/* distribute copy of vector to all workers */
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(b, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);

/* Respond to requests by workers. */
terminated = 0;
assign_cnt = 0;
do {
    MPI_Recv (&res, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                MPI_COMM_WORLD, &status);
    src = status.MPI_SOURCE;
    tag = status.MPI_TAG;
    if (tag == VECTOR_MSG){
        c[assign_cnt] = res;
        assign_cnt++;
    }
    /* Assign more work or tell worker to stop. */
    if (assign_cnt < m) {
        MPI_Send (a + assign_cnt * n, n, MPI_DOUBLE, src,
                    FILE_NAME_MSG, MPI_COMM_WORLD);
    } else {
        MPI_Send (NULL, 0, MPI_CHAR, src, FILE_NAME_MSG,
                MPI_COMM_WORLD);
```

```
            terminated++;
        }
    } while (terminated < (p-1));

    printf("result: ");
    for (int i = 0; i < m; ++i){
        printf("%lf ", c[i]);
    }
    printf("\n");
    free(a);free(b);free(c);
}
```

In worker process, as soon as it recive a row of vector, denoted as a_i. It did a production on a_i and b.

$$res = a_i * b$$

and res is sent to manager process using MPI_Send;

```
void worker (int argc, char *argv[], MPI_Comm worker_comm) {
    double *b;
    double *row_a;
    double res;
    int n; /* Profile vector size */
    int i;
    int name_len; /* Chars in file name */
    MPI_Request pending; /* Handle for MPI_Isend */
    MPI_Status status; /* Info about message */
    int worker_id; /* Rank in worker_comm */

    MPI_Comm_rank(worker_comm, &worker_id);

    b = (double*) malloc(n * sizeof(double));
    row_a = (double*) malloc(n *sizeof(double));
    if(b == NULL || row_a == NULL)
    {
        if(!worker_id)
            printf("error in worker process: can't allocate \
            enough memory for b and row_a\n");
        MPI_Abort(MPI_COMM_WORLD, MALLOC_ERROR);
    }

    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(b, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);


    MPI_Isend(NULL, 0, MPI_DOUBLE, 0, EMPTY_MSG, MPI_COMM_WORLD,
                &pending);
    for(;;){
        MPI_Probe(0, FILE_NAME_MSG, MPI_COMM_WORLD, &status);
        MPI_Get_count (&status, MPI_DOUBLE, &name_len);
        if (name_len != n) break;

        MPI_Recv(row_a, n, MPI_DOUBLE, 0, FILE_NAME_MSG,
                    MPI_COMM_WORLD, &status);

        res = 0;
        for (i = 0; i < n; ++i) res += row_a[i] * b[i];
```

```
        MPI_Send(&res, 1, MPI_DOUBLE, 0, VECTOR_MSG, MPI_COMM_WORLD);
    }
    free(b); free(row_a);
}
```

Problem 9.10

We adopt the Euclid method, for each integer n, we check whether $2^n - 1$ is a prime. If it is a prime, we adopt $(2^n - 1) * 2^{n-1}$ as a newly found perfect number.

The manager process is responsible for distribute all integer to worker process until eight perfect number is found.

```
void manager (int argc, char *argv[], int p) {

    int *e;          /* Store matrix here */
    int n;
    int i;
    int assign_cnt;
    int src;             /* Message source process */
    MPI_Status status;   /* Message status */
    int tag;             /* Message tag */
    int terminated;      /* Count of terminated procs */
    double perfect_number;

    e = (int *) malloc (NUM_PERFECT * sizeof(int));

    if (e == NULL){
        printf("error in manager process: can allocate enough \
                memory for e\n");
        MPI_Abort(MPI_COMM_WORLD, MALLOC_ERROR);
    }

    /* Respond to requests by workers. */
    terminated = 0;
    assign_cnt = 0;
    i = 1;
    do {
        MPI_Recv (&n, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
                    MPI_COMM_WORLD, &status);
        src = status.MPI_SOURCE;
        tag = status.MPI_TAG;

        if (tag == SUCCESS_MSG){
            e[assign_cnt] = n;
            assign_cnt++;
        }
        /* Assign more work or tell worker to stop. */
        if (assign_cnt < NUM_PERFECT) {
            MPI_Send (&i, 1, MPI_INT, src, SUCCESS_MSG, MPI_COMM_WORLD);
        } else {
            MPI_Send (NULL, 0, MPI_INT, src, SUCCESS_MSG, MPI_COMM_WORLD);
            terminated++;
        }
        ++i;
    } while (terminated < (p-1));
```

```
    printf("the first %d perfect numbers : \n", NUM_PERFECT);
    for (int i = 0; i < NUM_PERFECT; ++i){
        perfect_number = (pow(2, e[i])-1) * pow(2, e[i]-1);
        printf("%d: %.0lf\n", i, perfect_number);
    }
    free(e);
}
```

The worker process is responsible for determine whether $2^n - 1$ is a prime.

```
void worker (int argc, char *argv[], MPI_Comm worker_comm) {
    int n; /* Profile vector size */
    int name_len; /* Chars in file name */
    MPI_Request pending; /* Handle for MPI_Isend */
    MPI_Status status; /* Info about message */
    int worker_id; /* Rank in worker_comm */

    int is_prime(int x);

    MPI_Comm_rank(worker_comm, &worker_id);

    MPI_Isend(NULL, 0, MPI_INT, 0, EMPTY_MSG, MPI_COMM_WORLD, &pending);
    for(;;){
        MPI_Probe(0, SUCCESS_MSG, MPI_COMM_WORLD, &status);
        MPI_Get_count (&status, MPI_INT, &name_len);
        if (!name_len) break;

        MPI_Recv(&n, 1, MPI_INT, 0, SUCCESS_MSG, MPI_COMM_WORLD, &status);

        if (is_prime(n)){
            MPI_Send(&n, 1, MPI_INT, 0, SUCCESS_MSG, MPI_COMM_WORLD);
        }else{
            MPI_Send(&n, 1, MPI_INT, 0, FAIL_MSG, MPI_COMM_WORLD);
        }
    }
}
```