Introduction to Parallel Programing Techniques

Deferred Assessment

Professor: Stylianos Sygletos

Nestor Edgar Sandoval Alaguna
200243856

Assignment 3

Exercise – 1 [4 points]

a) 2-sided communication based on MPI_Send and MPI_Recv;
b) 2-sided communication based on MPI_Isend and MPI_Recv;
c) 2-sided communication based on MPI_Send and MPI_Irecv;

Answer:

3 forms of P2P communication are implemented. In ex1_a.c the implementation depends on MPI_Recv and MPI_Send calls. The primary matrices ATL, ATR, ABL, and ABR were initialized with 1, 2, 3, 4 respectively in their all $n2*n2$ indexes. The computed result was correctly found to be 10, -2, -4, and 0 respectively in all indexes of the corresponding matrix. Refer to the screenshots below that proves correct operation:

Code:

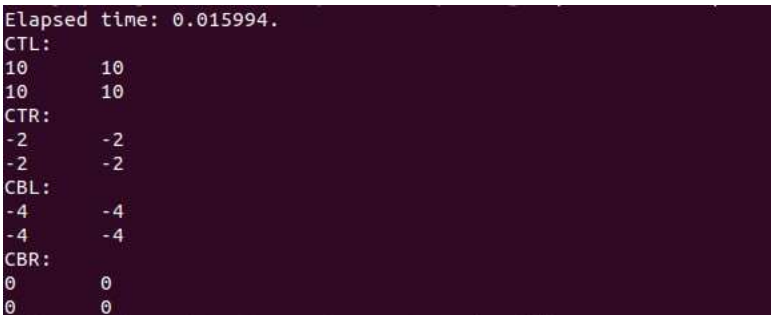
```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

void generate_numbers(int argc, char *argv[], int *n_p, int current_rank,
MPI_Comm common) {
if (current_rank == 0) {
if (argc != 2)
printf("Missing arguments, please care about inputs.\n");
else {
if (atoi(argv[1]) % 2 == 0)
*n_p = atoi(argv[1]);
else
printf("Specify even number.\n");
}
}
}

void fill_array(int *a_p, int n, int current_rank) {
int i;
for (i = 0; i < (n / 2 * n / 2); i++)
a_p[i] = current_rank + 1;
}
int main(int argc, char *argv[]) {
int common_sz, current_rank, n, i, j;
MPI_Comm common = MPI_COMM_WORLD;
int *A_TL, *A_TR, *A_BL, *A_BR, *B_TL, *B_TR, *B_BL, *B_BR, *C_TL, *C_TR,
*C_BL, *C_BR, *a, *A;
double start, finish, elapsed, Elapsed;
MPI_Init(&argc, &argv);
MPI_Comm_size(common, &common_sz);
MPI_Comm_rank(common, &current_rank);
generate_numbers(argc, argv, &n, current_rank, common);
MPI_Barrier(common);
start = MPI_Wtime();
MPI_Bcast(&n, 1, MPI_INT, 0, common);
switch (current_rank) {
```

```
case 0:
A_TL = malloc(sizeof(int) * n / 2 * n / 2);
A_TR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_TL, n, current_rank);
MPI_Send(A_TL, n / 2 * n / 2, MPI_INT, 1, 0, common);
MPI_Recv(A_TR, n / 2 * n / 2, MPI_INT, 1, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < (n / 2 * n / 2); i++)
A_TL[i] += A_TR[i];
free(A_TR);
B_TL = A_TL;
B_BL = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Send(B_TL, n / 2 * n / 2, MPI_INT, 2, 0, common);
MPI_Recv(B_BL, n / 2 * n / 2, MPI_INT, 2, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
B_TL[i] += B_BL[i];
free(B_BL);
C_TL = B_TL;
a = C_TL;
break;
case 1:
A_TL = malloc(sizeof(int) * n / 2 * n / 2);
A_TR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_TR, n, current_rank);
MPI_Recv(A_TL, n / 2 * n / 2, MPI_INT, 0, 0, common, MPI_STATUS_IGNORE);
MPI_Send(A_TR, n / 2 * n / 2, MPI_INT, 0, 0, common);
for (i = 0; i < n / 2 * n / 2; i++)
A_TL[i] -= A_TR[i];
free(A_TR);
B_TR = A_TL;
B_BR = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Send(B_TR, n / 2 * n / 2, MPI_INT, 3, 0, common);
MPI_Recv(B_BR, n / 2 * n / 2, MPI_INT, 3, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
B_TR[i] += B_BR[i];
free(B_BR);
C_TR = B_TR;
a = C_TR;
break;
case 2:
A_BR = malloc(sizeof(int) * n / 2 * n / 2);
A_BL = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_BL, n, current_rank);
MPI_Send(A_BL, n / 2 * n / 2, MPI_INT, 3, 0, common);
MPI_Recv(A_BR, n / 2 * n / 2, MPI_INT, 3, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
A_BL[i] += A_BR[i];
free(A_BR);
B_BL = A_BL;
B_TL = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Recv(B_TL, n / 2 * n / 2, MPI_INT, 0, 0, common, MPI_STATUS_IGNORE);
MPI_Send(B_BL, n / 2 * n / 2, MPI_INT, 0, 0, common);
for (i = 0; i < n / 2 * n / 2; i++)
B_TL[i] -= B_BL[i];
free(B_BL);
C_BL = B_TL;
a = C_BL;
break;
case 3:
A_BL = malloc(sizeof(int) * n / 2 * n / 2);
A_BR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_BR, n, current_rank);
MPI_Recv(A_BL, n / 2 * n / 2, MPI_INT, 2, 0, common, MPI_STATUS_IGNORE);
MPI_Send(A_BR, n / 2 * n / 2, MPI_INT, 2, 0, common);
```

```
for (i = 0; i < n / 2 * n / 2; i++)
A_BL[i] -= A_BR[i];
free(A_BR);
B_BR = A_BL;
B_TR = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Recv(B_TR, n / 2 * n / 2, MPI_INT, 1, 0, common, MPI_STATUS_IGNORE);
MPI_Send(B_BR, n / 2 * n / 2, MPI_INT, 1, 0, common);
for (i = 0; i < n / 2 * n / 2; i++)
B_TR[i] -= B_BR[i];
free(B_BR);
C_BR = B_TR;
a = C_BR;
break;
}
finish = MPI_Wtime();
elapsed = finish - start;
MPI_Reduce(&elapsed, &Elapsed, 1, MPI_DOUBLE, MPI_MAX, 0, common);
A = malloc(sizeof(int) * n * n);
MPI_Gather(a, n / 2 * n / 2, MPI_INT, A, n / 2 * n / 2, MPI_INT, 0, common);
if (current_rank == 0) {
printf("Elapsed time: %lf.\n", Elapsed);
printf("CTL: \n");
for (i = 0; i < n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + i * n / 2]);
printf("\n");
}
printf("CTR: \n");
for (i = 0; i < n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + n / 4 * n + i * n / 2]);
printf("\n");
}
printf("CBL: \n");
for (i = n; i < 3 * n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + i * n / 2]);
printf("\n");
}
printf("CBR: \n");
for (i = n; i < 3 * n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + n / 4 * n + i * n / 2]);
printf("\n");
}
}
free(A);
free(a);
MPI_Finalize();
return 0;
}
```



This program was running with 4 threads

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

void generate_numbers(int argc, char *argv[], int *n_p, int current_rank,
MPI_Comm common) {
if (current_rank == 0) {
if (argc != 2)
printf("Missing arguments, please care about inputs.\n");
else {
if (atoi(argv[1]) % 2 == 0)
*n_p = atoi(argv[1]);
else
printf("Specify even number.\n");
}
}
}

void fill_array(int *a_p, int n, int current_rank) {
int i;
for (i = 0; i < (n / 2 * n / 2); i++)
a_p[i] = current_rank + 1;
}

int main(int argc, char *argv[]) {
int common_sz, current_rank, n, i, j;
MPI_Comm common = MPI_COMM_WORLD;
int *A_TL, *A_TR, *A_BL, *A_BR, *B_TL, *B_TR, *B_BL, *B_BR, *C_TL, *C_TR,
*C_BL, *C_BR, *a, *A;
double start, finish, elapsed, Elapsed;
MPI_Init(&argc, &argv);
MPI_Comm_size(common, &common_sz);
MPI_Comm_rank(common, &current_rank);
MPI_Request request, req1, req2, req3;
MPI_Status status;
generate_numbers(argc, argv, &n, current_rank, common);
MPI_Barrier(common);
start = MPI_Wtime();
MPI_Bcast(&n, 1, MPI_INT, 0, common);
switch (current_rank) {
case 0:
A_TL = malloc(sizeof(int) * n / 2 * n / 2);
A_TR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_TL, n, current_rank);
MPI_Isend(A_TL, n / 2 * n / 2, MPI_INT, 1, 0, common, &request);
MPI_Recv(A_TR, n / 2 * n / 2, MPI_INT, 1, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < (n / 2 * n / 2); i++)
```

```
A_TL[i] += A_TR[i];
free(A_TR);
B_TL = A_TL;
B_BL = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Isend(B_TL, n / 2 * n / 2, MPI_INT, 2, 0, common, &request);
MPI_Recv(B_BL, n / 2 * n / 2, MPI_INT, 2, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
B_TL[i] += B_BL[i];
free(B_BL);
C_TL = B_TL;
a = C_TL;
break;
case 1:
A_TL = malloc(sizeof(int) * n / 2 * n / 2);
A_TR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_TR, n, current_rank);
MPI_Recv(A_TL, n / 2 * n / 2, MPI_INT, 0, 0, common, MPI_STATUS_IGNORE);
MPI_Isend(A_TR, n / 2 * n / 2, MPI_INT, 0, 0, common, &req1);
MPI_Wait(&req1, &status);
for (i = 0; i < n / 2 * n / 2; i++)
A_TL[i] -= A_TR[i];
free(A_TR);
B_TR = A_TL;
B_BR = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Isend(B_TR, n / 2 * n / 2, MPI_INT, 3, 0, common, &request);
MPI_Recv(B_BR, n / 2 * n / 2, MPI_INT, 3, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
B_TR[i] += B_BR[i];
free(B_BR);
C_TR = B_TR;
a = C_TR;
break;
case 2:
A_BR = malloc(sizeof(int) * n / 2 * n / 2);
A_BL = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_BL, n, current_rank);
MPI_Isend(A_BL, n / 2 * n / 2, MPI_INT, 3, 0, common, &request);
MPI_Recv(A_BR, n / 2 * n / 2, MPI_INT, 3, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
A_BL[i] += A_BR[i];
free(A_BR);
B_BL = A_BL;
B_TL = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Recv(B_TL, n / 2 * n / 2, MPI_INT, 0, 0, common, MPI_STATUS_IGNORE);
MPI_Isend(B_BL, n / 2 * n / 2, MPI_INT, 0, 0, common, &req2);
MPI_Wait(&req2, &status);
for (i = 0; i < n / 2 * n / 2; i++)
B_TL[i] -= B_BL[i];
free(B_BL);
C_BL = B_TL;
a = C_BL;
break;
case 3:
A_BL = malloc(sizeof(int) * n / 2 * n / 2);
A_BR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_BR, n, current_rank);
MPI_Recv(A_BL, n / 2 * n / 2, MPI_INT, 2, 0, common, MPI_STATUS_IGNORE);
MPI_Isend(A_BR, n / 2 * n / 2, MPI_INT, 2, 0, common, &req3);
MPI_Wait(&req3, &status);
for (i = 0; i < n / 2 * n / 2; i++)
A_BL[i] -= A_BR[i];
free(A_BR);
B_BR = A_BL;
```

```
B_TR = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Recv(B_TR, n / 2 * n / 2, MPI_INT, 1, 0, common, MPI_STATUS_IGNORE);
MPI_Isend(B_BR, n / 2 * n / 2, MPI_INT, 1, 0, common, &request);
for (i = 0; i < n / 2 * n / 2; i++)
B_TR[i] -= B_BR[i];
free(B_BR);
C_BR = B_TR;
a = C_BR;
break;
}
finish = MPI_Wtime();
elapsed = finish - start;
MPI_Reduce(&elapsed, &Elapsed, 1, MPI_DOUBLE, MPI_MAX, 0, common);
A = malloc(sizeof(int) * n * n);
MPI_Gather(a, n / 2 * n / 2, MPI_INT, A, n / 2 * n / 2, MPI_INT, 0, common);
if (current_rank == 0) {
printf("Elapsed time: %lf.\n", Elapsed);
printf("CTL: \n");
for (i = 0; i < n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + i * n / 2]);
printf("\n");
}
printf("CTR: \n");
for (i = 0; i < n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + n / 4 * n + i * n / 2]);
printf("\n");
}
printf("CBL: \n");
for (i = n; i < 3 * n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + i * n / 2]);
printf("\n");
}
printf("CBR: \n");
for (i = n; i < 3 * n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + n / 4 * n + i * n / 2]);
printf("\n");
}
}
free(A);
free(a);
MPI_Finalize();
return 0;
}
```

```
Elapsed time: 0.018106.
CTL:
10      10
10      10
CTR:
-2      -2
-2      -2
CBL:
-4      -4
-4      -4
CBR:
0       0
0       0
```

This program was running with 4 threads

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

void generate_numbers(int argc, char *argv[], int *n_p, int current_rank,
MPI_Comm common) {
if (current_rank == 0) {
if (argc != 2)
printf("Missing arguments, please care about inputs.\n");
else {
if (atoi(argv[1]) % 2 == 0)
*n_p = atoi(argv[1]);
else
printf("Specify even number.\n");
}
}
}

void fill_array(int *a_p, int n, int current_rank) {
int i;
for (i = 0; i < (n / 2 * n / 2); i++)
a_p[i] = current_rank + 1;
}

 int main(int argc, char *argv[]) {
int common_sz, current_rank, n, i, j;
MPI_Comm common = MPI_COMM_WORLD;
int *A_TL, *A_TR, *A_BL, *A_BR, *B_TL, *B_TR, *B_BL, *B_BR, *C_TL, *C_TR,
*C_BL, *C_BR, *a, *A;
MPI_Request req1, req2, req3, req4;
double start, finish, elapsed, Elapsed;
MPI_Init(&argc, &argv);
MPI_Comm_size(common, &common_sz);
MPI_Comm_rank(common, &current_rank);
generate_numbers(argc, argv, &n, current_rank, common);
MPI_Barrier(common);
start = MPI_Wtime();
MPI_Bcast(&n, 1, MPI_INT, 0, common);
switch (current_rank) {
case 0:
A_TL = malloc(sizeof(int) * n / 2 * n / 2);
A_TR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_TL, n, current_rank);
MPI_Send(A_TL, n / 2 * n / 2, MPI_INT, 1, 0, common);
MPI_Irecv(A_TR, n / 2 * n / 2, MPI_INT, 1, 0, common, &req1);
MPI_Wait(&req1, MPI_STATUS_IGNORE);
for (i = 0; i < (n / 2 * n / 2); i++)
A_TL[i] += A_TR[i];
free(A_TR);
B_TL = A_TL;
B_BL = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Send(B_TL, n / 2 * n / 2, MPI_INT, 2, 0, common);
MPI_Irecv(B_BL, n / 2 * n / 2, MPI_INT, 2, 0, common, &req1);
MPI_Wait(&req1, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
B_TL[i] += B_BL[i];
free(B_BL);
C_TL = B_TL;
a = C_TL;
break;
case 1:
A_TL = malloc(sizeof(int) * n / 2 * n / 2);
A_TR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_TR, n, current_rank);
MPI_Irecv(A_TL, n / 2 * n / 2, MPI_INT, 0, 0, common, &req2);
MPI_Send(A_TR, n / 2 * n / 2, MPI_INT, 0, 0, common);
MPI_Wait(&req2, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
A_TL[i] -= A_TR[i];
free(A_TR);
```

```
B_TR = A_TL;
B_BR = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Send(B_TR, n / 2 * n / 2, MPI_INT, 3, 0, common);
MPI_Irecv(B_BR, n / 2 * n / 2, MPI_INT, 3, 0, common, &req2);
MPI_Wait(&req2, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
B_TR[i] += B_BR[i];
free(B_BR);
C_TR = B_TR;
a = C_TR;
break;
case 2:
A_BR = malloc(sizeof(int) * n / 2 * n / 2);
A_BL = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_BL, n, current_rank);
MPI_Send(A_BL, n / 2 * n / 2, MPI_INT, 3, 0, common);
MPI_Irecv(A_BR, n / 2 * n / 2, MPI_INT, 3, 0, common, &req3);
MPI_Wait(&req3, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
A_BL[i] += A_BR[i];
free(A_BR);
B_BL = A_BL;
B_TL = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Irecv(B_TL, n / 2 * n / 2, MPI_INT, 0, 0, common, &req3);
MPI_Send(B_BL, n / 2 * n / 2, MPI_INT, 0, 0, common);
MPI_Wait(&req3, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
B_TL[i] -= B_BL[i];
free(B_BL);
C_BL = B_TL;
a = C_BL;
break;
case 3:
A_BL = malloc(sizeof(int) * n / 2 * n / 2);
A_BR = malloc(sizeof(int) * n / 2 * n / 2);
fill_array(A_BR, n, current_rank);
MPI_Irecv(A_BL, n / 2 * n / 2, MPI_INT, 2, 0, common, &req4);
MPI_Send(A_BR, n / 2 * n / 2, MPI_INT, 2, 0, common);
MPI_Wait(&req4, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
A_BL[i] -= A_BR[i];
free(A_BR);
B_BR = A_BL;
B_TR = malloc(sizeof(int) * n / 2 * n / 2);
MPI_Irecv(B_TR, n / 2 * n / 2, MPI_INT, 1, 0, common, &req4);
MPI_Send(B_BR, n / 2 * n / 2, MPI_INT, 1, 0, common);
MPI_Wait(&req4, MPI_STATUS_IGNORE);
for (i = 0; i < n / 2 * n / 2; i++)
B_TR[i] -= B_BR[i];
free(B_BR);
C_BR = B_TR;
a = C_BR;
break;
}
finish = MPI_Wtime();
elapsed = finish - start;
MPI_Reduce(&elapsed, &Elapsed, 1, MPI_DOUBLE, MPI_MAX, 0, common);
A = malloc(sizeof(int) * n * n);
MPI_Gather(a, n / 2 * n / 2, MPI_INT, A, n / 2 * n / 2, MPI_INT, 0, common);
if (current_rank == 0) {
printf("Elapsed time: %lf.\n", Elapsed);
printf("CTL: \n");
for (i = 0; i < n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + i * n / 2]);
printf("\n");
}
printf("CTR: \n");
for (i = 0; i < n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + n / 4 * n + i * n / 2]);
```

```
printf("\n");
}
printf("CBL: \n");
for (i = n; i < 3 * n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + i * n / 2]);
printf("\n");
}
printf("CBR: \n");
for (i = n; i < 3 * n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", A[j + n / 4 * n + i * n / 2]);
printf("\n");
}
}
free(A);
free(a);
MPI_Finalize();
return 0;
}
```

```
Elapsed time: 0.030129.
CTL:
10      10
10      10
CTR:
-2      -2
-2      -2
CBL:
-4      -4
-4      -4
CBR:
0       0
0       0
```

This program was running with 4 threads

Exercise – 2 [3 points]

The goal of this task is to optimize for memory usage. We ask you to solve the same problem as in the above task, this time with restriction in the amount of memory that is available per process. Now, each process can only store one submatrix of size n2×n2 plus one vector of size n2. Implement one single version using the communication mode of your choice.

Answer:
The prior activity is carried out with limited memory resources, necessitating more frequent communication... The P2P communication overhead caused the elapsed time to increase to 0.174266s as the figure below shows, for n=8 implementation with MPI_Send and Recv pairs. While the same specifications achieved an elapsed time of 0.044031s with the previous implementation. Refer to ex2.c for the code.

Code:

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

void input_n(int argc, char *argv[], int *n_p, int current_rank,
MPI_Comm common) {
if (current_rank == 0) {
if (argc != 2)
```

```
printf("Missing n!\n");
else {
if (atoi(argv[1]) % 2 == 0)
*n_p = atoi(argv[1]);
else
printf("Specify even number.\n");
}
}
}

void generate_array(int *a_p, int n, int current_rank) {
int i;
for (i = 0; i < (n / 2 * n / 2); i++)
a_p[i] = current_rank + 1;
}

int main(int argc, char *argv[]) {
int common_sz, current_rank, n, i, j;
MPI_Comm common = MPI_COMM_WORLD;
int *M_TL, *M_TR, *M_BL, *M_BR, *B_TL, *B_TR, *B_BL, *B_BR, *C_TL, *C_TR,
*C_BL, *C_BR, *a, *M, *v;
double start, finish, elapsed, Elapsed;
MPI_Init(&argc, &argv);
MPI_Comm_size(common, &common_sz);
MPI_Comm_rank(common, &current_rank);
input_n(argc, argv, &n, current_rank, common);
MPI_Barrier(common);
start = MPI_Wtime();
MPI_Bcast(&n, 1, MPI_INT, 0, common); // broadcast n
v = malloc(sizeof(int) * n / 2);
switch (current_rank) {
case 0:
M_TL = malloc(sizeof(int) * n / 2 * n / 2);
generate_array(M_TL, n, current_rank);
for (i = 0; i < n / 2; i++) {
MPI_Send(M_TL + (n / 2 * i), n / 2, MPI_INT, 1, 0, common);
MPI_Recv(v, n / 2, MPI_INT, 1, 0, common, MPI_STATUS_IGNORE);
for (j = 0; j < n / 2; j++)
M_TL[i * n / 2 + j] += v[j];
}
B_TL = M_TL;
for (i = 0; i < n / 2; i++) {
MPI_Send(B_TL + n / 2 * i, n / 2, MPI_INT, 2, 0, common);
MPI_Recv(v, n / 2, MPI_INT, 2, 0, common, MPI_STATUS_IGNORE);
for (j = 0; j < n / 2; j++)
B_TL[i * n / 2 + j] += v[j];
}
C_TL = B_TL;
a = C_TL; // for verification later
break;
case 1:
M_TR = malloc(sizeof(int) * n / 2 * n / 2);
generate_array(M_TR, n, current_rank);
for (i = 0; i < n / 2; i++) {
MPI_Recv(v, n / 2, MPI_INT, 0, 0, common, MPI_STATUS_IGNORE);
MPI_Send(M_TR + n / 2 * i, n / 2, MPI_INT, 0, 0, common);
for (j = 0; j < n / 2; j++)
M_TR[i * n / 2 + j] = v[j] - M_TR[i * n / 2 + j];
}
B_TR = M_TR;
for (i = 0; i < n / 2; i++) {
MPI_Send(B_TR + n / 2 * i, n / 2, MPI_INT, 3, 0, common);
MPI_Recv(v, n / 2, MPI_INT, 3, 0, common, MPI_STATUS_IGNORE);
```

```
for (j = 0; j < n / 2; j++)
B_TR[i * n / 2 + j] += v[j];
}
C_TR = B_TR;
a = C_TR; // for verification later
break;
case 2:
M_BL = malloc(sizeof(int) * n / 2 * n / 2);
generate_array(M_BL, n, current_rank);
for (i = 0; i < n / 2; i++) {
MPI_Send(M_BL + n / 2 * i, n / 2, MPI_INT, 3, 0, common);
MPI_Recv(v, n / 2, MPI_INT, 3, 0, common, MPI_STATUS_IGNORE);
for (j = 0; j < n / 2; j++)
M_BL[i * n / 2 + j] += v[j];
}
B_BL = M_BL;
for (i = 0; i < n / 2; i++) {
MPI_Recv(v, n / 2, MPI_INT, 0, 0, common, MPI_STATUS_IGNORE);
MPI_Send(B_BL + n / 2 * i, n / 2, MPI_INT, 0, 0, common);
for (j = 0; j < n / 2; j++)
B_BL[i * n / 2 + j] = v[j] - B_BL[i * n / 2 + j];
}
C_BL = B_BL;
a = C_BL; // for verification later
break;
case 3:
M_BR = malloc(sizeof(int) * n / 2 * n / 2);
generate_array(M_BR, n, current_rank);
for (i = 0; i < n / 2; i++) {
MPI_Recv(v, n / 2, MPI_INT, 2, 0, common, MPI_STATUS_IGNORE);
MPI_Send(M_BR + n / 2 * i, n / 2, MPI_INT, 2, 0, common);
for (j = 0; j < n / 2; j++)
M_BR[i * n / 2 + j] = v[j] - M_BR[i * n / 2 + j];
}
B_BR = M_BR;
for (i = 0; i < n / 2; i++) {
MPI_Recv(v, n / 2, MPI_INT, 1, 0, common, MPI_STATUS_IGNORE);
MPI_Send(B_BR + n / 2 * i, n / 2, MPI_INT, 1, 0, common);
for (j = 0; j < n / 2; j++)
B_BR[i * n / 2 + j] = v[j] - B_BR[i * n / 2 + j];
}
C_BR = B_BR;
a = C_BR;
break;
}
finish = MPI_Wtime();
elapsed = finish - start;
MPI_Reduce(&elapsed, &Elapsed, 1, MPI_DOUBLE, MPI_MAX, 0, common);
M = malloc(sizeof(int) * n * n);
MPI_Gather(a, n / 2 * n / 2, MPI_INT, M, n / 2 * n / 2, MPI_INT, 0, common);
if (current_rank == 0) {
printf("Elapsed time: %lf.\n", Elapsed);
printf("CTL: \n");
for (i = 0; i < n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", M[j + i * n / 2]);
printf("\n");
}
printf("CTR: \n");
for (i = 0; i < n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", M[j + n / 4 * n + i * n / 2]);
printf("\n");
```

```
}
printf("CBL: \n");
for (i = n; i < 3 * n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", M[j + i * n / 2]);
printf("\n");
}
printf("CBR: \n");
for (i = n; i < 3 * n / 2; i++) {
for (j = 0; j < n / 2; j++)
printf("%d\t", M[j + n / 4 * n + i * n / 2]);
printf("\n");
}
}
free(M);
free(a);
free(v);
MPI_Finalize();
return 0;
}
```

```
Elapsed time: 0.044031.
CTL:
10      10
10      10
CTR:
-2      -2
-2      -2
CBL:
-4      -4
-4      -4
CBR:
0       0
0       0
```

Exercise – 3 [3 points] [MPI implementation]

We ask you to generate two versions of the parallel program using:
a) Blocking communications
b) Non-blocking communication, where communication and computation overlap.

Answer:

The blocking version was implemented for the first code, Because of the blocking nature of those functions, no barriers were required to establish process synchronization.
For the second the non-blocking implementation, an MPI_Barrier was necessary after computing the grid, before deleting the receiving buffers.

Code:

```
#include <math.h>
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define DEBUG 0
#define THRESHOLD (5e-3)

void get_inputs(int argc, char *argv[], int *gn_p, int current_rank,
```

```
int common_sz, MPI_Comm common) {
if (current_rank == 0) {
if (argc != 2) {
fprintf(stderr, "Usage: %s <n>.\n", argv[0]);
exit(-1);
}
*gn_p = atoi(argv[1]);
}
}

void init_phi(double phi[], int ln, int gn, int current_rank, int common_sz) {
int i, j;
if (current_rank == 0) {
for (i = 1; i < ln; i++)
for (j = 1; j < gn - 1; j++)
phi[i * gn + j] = 50.0;
for (j = 0; j < gn; j++)
phi[j] = 100.0;
for (i = 0; i < ln; i++) {
phi[i * gn + (gn - 1)] = 100.0;
phi[i * gn + 0] = 100.0;
}
}
else if (current_rank == common_sz - 1) {
for (i = 0; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
phi[i * gn + j] = 50.0;
for (i = 0; i < gn; i++)
phi[(ln - 1) * gn + i] = 0.0;
for (i = 0; i < ln - 1; i++) {
phi[i * gn + (gn - 1)] = 100.0;
phi[i * gn + 0] = 100.0;
}
}
else {
for (i = 0; i < ln; i++)
for (j = 1; j < gn - 1; j++)
phi[i * gn + j] = 50.0;
for (i = 0; i < ln; i++) {
phi[i * gn + (gn - 1)] = 100.0;
phi[i * gn + 0] = 100.0;
}
}
}

void update(double *cur, double *next, int gn, int ln, int current_rank,
int common_sz, MPI_Comm common) {
int i, j;
double *vectT, *vectB;
vectT = malloc(sizeof(double) * gn - 2);
vectB = malloc(sizeof(double) * gn - 2);
if (current_rank != 0 && current_rank != common_sz - 1) {
MPI_Recv(vectT, gn - 2, MPI_DOUBLE, current_rank - 1, 0, common,
MPI_STATUS_IGNORE);
MPI_Ssend(cur + 1, gn - 2, MPI_DOUBLE, current_rank - 1, 0, common);
MPI_Ssend(cur + gn * (ln - 1) + 1, gn - 2, MPI_DOUBLE, current_rank + 1, 0,
common);
MPI_Recv(vectB, gn - 2, MPI_DOUBLE, current_rank + 1, 0, common,
MPI_STATUS_IGNORE);
for (i = 1; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
next[i * gn + j] = (cur[(i - 1) * gn + j] + cur[i * gn + (j - 1)] +
cur[i * gn + (j + 1)] + cur[(i + 1) * gn + j]) /
```

```
4;
for (j = 1; j < gn - 1; j++)
next[j] = (cur[j + 1] + cur[j - 1] + cur[j + gn] + vectT[j]) / 4;
for (j = 1; j < gn - 1; j++)
next[gn * (ln - 1) + j] =
(cur[gn * (ln - 1) + j - 1] + cur[gn * (ln - 1) + j + 1] +
cur[gn * (ln - 1) + j - gn] + vectB[j]) /
4;
}
else if (current_rank == 0) {
MPI_Ssend(cur + gn * (ln - 1) + 1, gn - 2, MPI_DOUBLE, current_rank + 1, 0,
common);
MPI_Recv(vectB, gn - 2, MPI_DOUBLE, current_rank + 1, 0, common,
MPI_STATUS_IGNORE);
for (i = 1; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
next[i * gn + j] = (cur[(i - 1) * gn + j] + cur[i * gn + (j - 1)] +
cur[i * gn + (j + 1)] + cur[(i + 1) * gn + j]) /
4;
for (j = 1; j < gn - 1; j++)
next[gn * (ln - 1) + j] =
(cur[gn * (ln - 1) + j - 1] + cur[gn * (ln - 1) + j + 1] +
cur[gn * (ln - 1) + j - gn] + vectB[j]) /
4;
}
else {
MPI_Recv(vectT, gn - 2, MPI_DOUBLE, current_rank - 1, 0, common,
MPI_STATUS_IGNORE);
MPI_Ssend(cur + 1, gn - 2, MPI_DOUBLE, current_rank - 1, 0, common);
for (i = 1; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
next[i * gn + j] = (cur[(i - 1) * gn + j] + cur[i * gn + (j - 1)] +
cur[i * gn + (j + 1)] + cur[(i + 1) * gn + j]) /
4;
for (j = 1; j < gn - 1; j++)
next[j] = (cur[j + 1] + cur[j - 1] + cur[j + gn] + vectT[j]) / 4;
}
free(vectT);
free(vectB);
}

int converged(double *cur, double *next, int gn, int ln) {
int i, j;
for (i = 1; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
if (fabs(next[i * gn + j] - cur[i * gn + j]) > THRESHOLD)
return 0;
return 1;
}

void fprintf_grid(int current_rank, int common_sz, double phi_cur[], int gn,
int ln, FILE *fp, MPI_Comm common, int niters) {
int p, i, j;
double *recv_buf;
recv_buf = (double *)malloc(ln * gn * sizeof(double));
if (current_rank != 0)
MPI_Send(phi_cur, gn * ln, MPI_DOUBLE, 0, 0, common);
if (current_rank == 0) {
fprintf(fp, "Iteration: #%d\n", niters);
for (i = 0; i < ln; i++) {
for (j = 0; j < gn; j++)
fprintf(fp, "%lf,", phi_cur[i * gn + j]);
fprintf(fp, "\n");
```

```
}
for (p = 1; p < common_sz; p++) {
MPI_Recv(recv_buf, gn * ln, MPI_DOUBLE, p, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < ln; i++) {
for (j = 0; j < gn; j++)
fprintf(fp, "%lf,", recv_buf[i * gn + j]);
fprintf(fp, "\n");
}
}
}
free(recv_buf);
}

int main(int argc, char *argv[]) {
int common_sz, current_rank, i, j;
int gn, ln, niters, conv, convT;
double *phi_cur, *phi_next, *tmp;
MPI_Comm common = MPI_COMM_WORLD;
FILE *fp, *exp;
double start, finish, elapsed, elapsed_f;
fp = fopen("poisson_results.csv", "w+");
exp = fopen("perf_eval.csv", "a+");
if (fp == NULL)
printf("Couldn't open the file poisson_results.csv.\n");
if (exp == NULL)
printf("Couldn't open the file perf_eval.csv.\n");
MPI_Init(&argc, &argv);
MPI_Comm_size(common, &common_sz);
MPI_Comm_rank(common, &current_rank);
get_inputs(argc, argv, &gn, current_rank, common_sz, common);
MPI_Bcast(&gn, 1, MPI_INT, 0, common);
ln = gn / common_sz;
phi_cur = (double *)malloc(ln * gn * sizeof(double));
phi_next = (double *)malloc(ln * gn * sizeof(double));
init_phi(phi_cur, ln, gn, current_rank, common_sz);
init_phi(phi_next, ln, gn, current_rank, common_sz);
MPI_Barrier(common);
start = MPI_Wtime();
niters = 0;
while (1) {
niters++;
#if DEBUG
fprintf_grid(current_rank, common_sz, phi_cur, gn, ln, fp, common, niters);
#endif
update(phi_cur, phi_next, gn, ln, current_rank, common_sz, common);
conv = converged(phi_cur, phi_next, gn, ln);
MPI_Allreduce(&conv, &convT, 1, MPI_INT, MPI_SUM, common);
if (convT == common_sz)
break;
tmp = phi_cur;
phi_cur = phi_next;
phi_next = tmp;
}
finish = MPI_Wtime();
elapsed = finish - start;
MPI_Reduce(&elapsed, &elapsed_f, 1, MPI_DOUBLE, MPI_MAX, 0, common);
if (current_rank == 0)
fprintf(exp, "%d, %d, %d, %lf\n", common_sz, gn, niters, elapsed_f);
#if DEBUG
fprintf_grid(current_rank, common_sz, phi_cur, gn, ln, fp, common, niters);
fprintf(fp, "last phi NEXT:\n");
fprintf_grid(current_rank, common_sz, phi_next, gn, ln, fp, common, niters);
if (current_rank == 0)
```

```
fprintf(fp, "Converged after %d iterations\n", niters);
#endif
free(phi_cur);
free(phi_next);
MPI_Finalize();
fclose(fp);
return 0;

}


#include <math.h>
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define DEBUG 0
#define THRESHOLD (5e-3)

void get_inputs(int argc, char *argv[], int *gn_p, int current_rank,
int common_sz, MPI_Comm common) {
if (current_rank == 0) {
if (argc != 2) {
fprintf(stderr, "Usage: %s <n>.\n", argv[0]);
exit(-1);
}
*gn_p = atoi(argv[1]);
}
}

void init_phi(double phi[], int ln, int gn, int current_rank, int common_sz) {
int i, j;
if (current_rank == 0) {
for (i = 1; i < ln; i++)
for (j = 1; j < gn - 1; j++)
phi[i * gn + j] = 50.0;
for (j = 0; j < gn; j++)
phi[j] = 100.0;
for (i = 0; i < ln; i++) {
phi[i * gn + (gn - 1)] = 100.0;
phi[i * gn + 0] = 100.0;
}
}
else if (current_rank == common_sz - 1) {
for (i = 0; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
phi[i * gn + j] = 50.0;
for (i = 0; i < gn; i++)
phi[(ln - 1) * gn + i] = 0.0;
for (i = 0; i < ln - 1; i++) {
phi[i * gn + (gn - 1)] = 100.0;
phi[i * gn + 0] = 100.0;
}
}
else {
for (i = 0; i < ln; i++)
for (j = 1; j < gn - 1; j++)
phi[i * gn + j] = 50.0;
for (i = 0; i < ln; i++) {
phi[i * gn + (gn - 1)] = 100.0;
phi[i * gn + 0] = 100.0;
}
}
}
```

```
void update(double *cur, double *next, int gn, int ln, int current_rank,
int common_sz, MPI_Comm common) {
int i, j;
double *vectT, *vectB;
vectT = malloc(sizeof(double) * gn - 2);
vectB = malloc(sizeof(double) * gn - 2);
MPI_Request req1, req2;
if (current_rank != 0 && current_rank != common_sz - 1) {
MPI_Irecv(vectT, gn - 2, MPI_DOUBLE, current_rank - 1, 0, common, &req1);
MPI_Isend(cur + 1, gn - 2, MPI_DOUBLE, current_rank - 1, 0, common, &req2);
MPI_Isend(cur + gn * (ln - 1) + 1, gn - 2, MPI_DOUBLE, current_rank + 1, 0,
common, &req1);
MPI_Irecv(vectB, gn - 2, MPI_DOUBLE, current_rank + 1, 0, common, &req2);
for (i = 1; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
next[i * gn + j] = (cur[(i - 1) * gn + j] + cur[i * gn + (j - 1)] +
cur[i * gn + (j + 1)] + cur[(i + 1) * gn + j]) /
4;
for (j = 1; j < gn - 1; j++)
next[j] = (cur[j + 1] + cur[j - 1] + cur[j + gn] + vectT[j]) / 4;
for (j = 1; j < gn - 1; j++)
next[gn * (ln - 1) + j] =
(cur[gn * (ln - 1) + j - 1] + cur[gn * (ln - 1) + j + 1] +
cur[gn * (ln - 1) + j - gn] + vectB[j]) /
4;
}
else if (current_rank == 0) {
MPI_Isend(cur + gn * (ln - 1) + 1, gn - 2, MPI_DOUBLE, current_rank + 1, 0,
common, &req1);
MPI_Irecv(vectB, gn - 2, MPI_DOUBLE, current_rank + 1, 0, common, &req2);
for (i = 1; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
next[i * gn + j] = (cur[(i - 1) * gn + j] + cur[i * gn + (j - 1)] +
cur[i * gn + (j + 1)] + cur[(i + 1) * gn + j]) /
4;
for (j = 1; j < gn - 1; j++)
next[gn * (ln - 1) + j] =
(cur[gn * (ln - 1) + j - 1] + cur[gn * (ln - 1) + j + 1] +
cur[gn * (ln - 1) + j - gn] + vectB[j]) /
4;
}
else {
MPI_Irecv(vectT, gn - 2, MPI_DOUBLE, current_rank - 1, 0, common, &req1);
MPI_Isend(cur + 1, gn - 2, MPI_DOUBLE, current_rank - 1, 0, common, &req2);
for (i = 1; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
next[i * gn + j] = (cur[(i - 1) * gn + j] + cur[i * gn + (j - 1)] +
cur[i * gn + (j + 1)] + cur[(i + 1) * gn + j]) /
4;
for (j = 1; j < gn - 1; j++)
next[j] = (cur[j + 1] + cur[j - 1] + cur[j + gn] + vectT[j]) / 4;
}
MPI_Wait(&req1, MPI_STATUS_IGNORE);
MPI_Wait(&req2, MPI_STATUS_IGNORE);
free(vectT);
free(vectB);
}
int converged(double *cur, double *next, int gn, int ln) {
int i, j;
for (i = 1; i < ln - 1; i++)
for (j = 1; j < gn - 1; j++)
if (fabs(next[i * gn + j] - cur[i * gn + j]) > THRESHOLD)
```

```
return 0;
return 1;
}
void fprintf_grid(int current_rank, int common_sz, double phi_cur[], int gn,
int ln, FILE *fp, MPI_Comm common, int niters) {
int p, i, j;
double *recv_buf;
recv_buf = (double *)malloc(ln * gn * sizeof(double));
if (current_rank != 0)
MPI_Send(phi_cur, gn * ln, MPI_DOUBLE, 0, 0, common);
if (current_rank == 0) {
fprintf(fp, "Iteration: %d\n", niters);
for (i = 0; i < ln; i++) {
for (j = 0; j < gn; j++)
fprintf(fp, "%lf,", phi_cur[i * gn + j]);
fprintf(fp, "\n");
}
for (p = 1; p < common_sz; p++) {
MPI_Recv(recv_buf, gn * ln, MPI_DOUBLE, p, 0, common, MPI_STATUS_IGNORE);
for (i = 0; i < ln; i++) {
for (j = 0; j < gn; j++)
fprintf(fp, "%lf,", recv_buf[i * gn + j]);
fprintf(fp, "\n");
}
}
}
free(recv_buf);
}

int main(int argc, char *argv[]) {
int common_sz, current_rank, i, j;
int gn, ln, niters, conv, convT;
double *phi_cur, *phi_next, *tmp;
MPI_Comm common = MPI_COMM_WORLD;
FILE *fp, *exp;
double start, finish, elapsed, elapsed_f;
fp = fopen("poisson_results.csv", "w+");
exp = fopen("perf_eval.csv", "a+");
if (fp == NULL)
printf("Couldn't open the file poisson_results.csv.\n");
if (exp == NULL)
printf("Couldn't open the file perf_eval.csv.\n");
MPI_Init(&argc, &argv);
MPI_Comm_size(common, &common_sz);
MPI_Comm_rank(common, &current_rank);
get_inputs(argc, argv, &gn, current_rank, common_sz, common);
MPI_Bcast(&gn, 1, MPI_INT, 0, common);
ln = gn / common_sz;
phi_cur = (double *)malloc(ln * gn * sizeof(double));
phi_next = (double *)malloc(ln * gn * sizeof(double));
init_phi(phi_cur, ln, gn, current_rank, common_sz);
init_phi(phi_next, ln, gn, current_rank, common_sz);
MPI_Barrier(common);
start = MPI_Wtime();
niters = 0;
while (1) {
niters++;
#if DEBUG
fprintf_grid(current_rank, common_sz, phi_cur, gn, ln, fp, common, niters);
#endif
update(phi_cur, phi_next, gn, ln, current_rank, common_sz, common);
conv = converged(phi_cur, phi_next, gn, ln);
MPI_Allreduce(&conv, &convT, 1, MPI_INT, MPI_SUM, common);
```

```
if (convT == common_sz)
break;
tmp = phi_cur;
phi_cur = phi_next;
phi_next = tmp;
}
finish = MPI_Wtime();
elapsed = finish - start;
MPI_Reduce(&elapsed, &elapsed_f, 1, MPI_DOUBLE, MPI_MAX, 0, common);
if (current_rank == 0)
fprintf(exp, "%d, %d, %d, %lf\n", common_sz, gn, niters, elapsed_f);
#if DEBUG
fprintf_grid(current_rank, common_sz, phi_cur, gn, ln, fp, common, niters);
fprintf(fp, "last phi NEXT:\n");
fprintf_grid(current_rank, common_sz, phi_next, gn, ln, fp, common, niters);
if (current_rank == 0)
fprintf(fp, "Converged after %d iterations\n", niters);
#endif
free(phi_cur);
free(phi_next);
MPI_Finalize();
fclose(fp);
return 0;
}
```