



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ

Συστήματα Παράλληλης Επεξεργασίας 1^η Άσκηση
Ακ. έτος 2012-2013

Ομάδα 8^η

Μαυρογιάννης Αλέξανδρος	A.M.: 03109677
Λύρας Γρηγόρης	A.M.: 03109687

12 Νοεμβρίου 2012

Πηγαίος κώδικας

Κοινή βιβλιοθήκη

```
1  /* -.-.-.-.-.
2  * File Name : common.h
3  * Creation Date : 06-11-2012
4  * Last Modified : Mon 12 Nov 2012 10:06:15 AM EET
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * Created By : Alex Maurogiannis <nalfemp@gmail.com>
7  -.-.-.-.-.*/
8
9  #ifndef DEBUG_FUNC
10 #define DEBUG_FUNC
11
12 #if main_DEBUG
13 #define debug(fmt,arg...)    fprintf(stdout, "%s: " fmt, __func__ , ##arg)
14 #else
15 #define debug(fmt,arg...)    do { } while(0)
16 #endif /* main_DEBUG */
17
18 #endif /* DEBUG_FUNC */
19
20 #ifndef COMMON_H
21 #define COMMON_H
22
23 #include <stdlib.h>
24 #include <stdio.h>
25 #include <mpi.h>
26
27 double *allocate_2d(int N, int M);
28 double *allocate_2d_with_padding(int N, int M, int max_rank);
29 double *parse_matrix_2d(FILE *fp, int N, int M, double *A);
30 void fprintf_matrix_2d(FILE *fp, int N, int M, double *A);
31 void print_matrix_2d(int N, int M, double *A);
32 double timer(void);
33 void usage(int argc, char **argv);
34
35 #ifdef USE_MPI /* USE_MPI */
36 void propagate_with_send(void *buffer, int count , MPI_Datatype datatype, \
37     int root, MPI_Comm comm);
38 void propagate_with_flooding(void *buffer, int count , MPI_Datatype datatype, \
39     int root, MPI_Comm comm);
40 #endif /* USE_MPI */
41
42 #endif /* COMMON_H */
43
44 1  /* -.-.-.-.-.
45 2  * File Name : common.c
46 3  * Creation Date : 06-11-2012
47 4  * Last Modified : Mon 12 Nov 2012 10:06:02 AM EET
48 5  * Created By : Greg Liras <gregliras@gmail.com>
49 6  * Created By : Alex Maurogiannis <nalfemp@gmail.com>
50 7  -.-.-.-.-.*/
51 8
52 9  #include "common.h"
53 10 #include <sys/time.h>
54 11
55 12 double *allocate_2d(int N, int M)
56 13 {
57 14     double *A;
58 15     A = malloc(N * M * sizeof(double));
59 16     return A;
60 17 }
61 18
62 19 double *allocate_2d_with_padding(int N, int M, int max_rank)
63 20 {
64 21     double *A;
65 22     A = allocate_2d(N + max_rank, M);
66 23     return A;
67 24 }
68 25
69 26 double *parse_matrix_2d(FILE *fp, int N, int M, double *A)
70 27 {
71 28     int i,j;
72 29     double *p;
73 30     p = A;
74 31     for (i = 0; i < N; i++) {
75 32         for (j = 0; j < M; j++) {
76 33             if(!fscanf(fp, "%lf", p++)) {
77 34                 return NULL;
78 35             }
79 36         }
80 37     }
81 38     return A;
82 39 }
83 40
84 41 void fprintf_matrix_2d(FILE *fp, int N, int M, double *A)
85 42 {
86 43     for (i = 0; i < N; i++) {
87 44         for (j = 0; j < M; j++) {
88 45             fprintf(fp, "%lf ", A[i*M+j]);
89 46         }
90 47         fprintf(fp, "\n");
91 48     }
92 49 }
```

```

43     int i,j;
44     double *p;
45     p = A;
46     for (j = 0; j < M; j++) {
47         fprintf(fp, "=");
48     }
49     fprintf(fp, "\n");
50     for (i = 0; i < N; i++) {
51         for (j = 0; j < M; j++) {
52             fprintf(fp, "%lf\t", *p++);
53         }
54         fprintf(fp, "\n");
55     }
56     for (j = 0; j < M; j++) {
57         fprintf(fp, "=");
58     }
59     fprintf(fp, "\n");
60 }
61
62 void print_matrix_2d(int N, int M, double *A)
63 {
64     fprintf_matrix_2d(stdout, N, M, A);
65 }
66
67 double timer(void)
68 {
69     static double seconds = 0;
70     static int operation = 0;
71     struct timeval tv;
72     gettimeofday(&tv, NULL);
73     if (operation == 0) {
74         seconds = tv.tv_sec + (((double) tv.tv_usec)/1e6);
75         operation = 1;
76         return 0;
77     }
78     else {
79         operation = 0;
80         return tv.tv_sec + (((double) tv.tv_usec)/1e6) - seconds;
81     }
82 }
83
84 void usage(int argc, char **argv)
85 {
86     if(argc != 3) {
87         printf("Usage: %s <matrix file> <output file>\n", argv[0]);
88         exit(EXIT_FAILURE);
89     }
90 }
91
92 #ifdef USE_MPI /* USE_MPI */
93 void propagate_with_send(void *buffer, int count, MPI_Datatype datatype, \
94     int root, MPI_Comm comm)
95 {
96     int rank;
97     int i;
98     int max_rank;
99
100     MPI_Comm_rank(comm, &rank);
101     MPI_Comm_size(comm, &max_rank);
102     if(rank == root) {
103         for(i = 0; i < max_rank; i++) {
104             if(i == rank) {
105                 continue;
106             }
107             else {
108                 debug("%d\n", i);
109                 MPI_Send(buffer, count, datatype, i, root, comm);
110             }
111         }
112     }
113     else {
114         MPI_Status status;
115         MPI_Recv(buffer, count, datatype, root, root, comm, &status);
116     }
117 }
118
119 void propagate_with_flooding(void *buffer, int count, MPI_Datatype datatype, \
120     int root, MPI_Comm comm)
121 {
122     int rank;
123     int max_rank;
124     int cur;
125
126     MPI_Comm_rank(comm, &rank);
127     MPI_Comm_size(comm, &max_rank);
128
129     if(rank != 0) {
130         if(rank == root) {
131             MPI_Send(buffer, count, datatype, 0, root, comm);

```

```

132     }
133     if(rank == 0) {
134         MPI_Status status;
135         MPI_Recv(buffer, count, datatype, root, root, comm, &status);
136     }
137 }
138
139 if(rank != 0) {
140     MPI_Status status;
141     MPI_Recv(buffer, count, datatype, (rank-1)/2, root, comm, &status);
142 }
143 cur = 2*rank+1;
144 if(cur < max_rank) {
145     MPI_Send(buffer, count, datatype, cur, root, comm);
146 }
147 if(++cur < max_rank) {
148     MPI_Send(buffer, count, datatype, cur, root, comm);
149 }
150 }
151 #endif /* USE_MPI */

```

Ζητούμενο 1 Σειριακό Πρόγραμμα

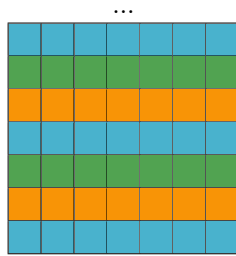
```

1  /* -.-.-.-.-.
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Thu 08 Nov 2012 09:50:55 AM EET
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * Created By : Alex Maurogiannis <nalfemp@gmail.com>
7  -.-.-.-.-.*/
8
9  #include <stdio.h>
10 #include <stdlib.h>
11
12 #include "common.h"
13
14 int main(int argc, char **argv)
15 {
16     int i,j,k;
17     int N;
18     double *A;
19     double l;
20     double sec;
21
22     FILE *fp = NULL;
23     usage(argc, argv);
24     /*
25      * Allocate me!
26      */
27     fp = fopen(argv[1], "r");
28     if(fp) {
29         if(!fscanf(fp, "%d\n", &N)) {
30             exit(EXIT_FAILURE);
31         }
32     }
33
34     if((A = allocate_2d(N, N)) == NULL) {
35         exit(EXIT_FAILURE);
36     }
37     if(parse_matrix_2d(fp, N, N, A) == NULL) {
38         exit(EXIT_FAILURE);
39     }
40
41
42     sec = timer();
43     for (k = 0; k < N - 1; k++)
44     {
45         for (i = k + 1; i < N; i++)
46         {
47             l = A[i * N + k] / A[k * N + k];
48             for (j = k; j < N; j++)
49             {
50                 A[i * N + j] = A[i * N + j] - l * A[k * N + j];
51             }
52         }
53     }
54     sec = timer();
55     printf("Calc Time: %lf\n", sec);
56
57     fp = fopen(argv[2], "w");
58     fprintf_matrix_2d(fp, N, N, A);
59     fclose(fp);
60     free(A);
61
62     return 0;
63 }

```

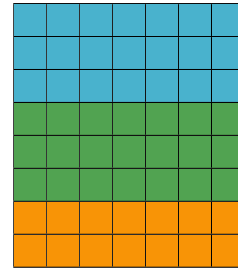
Ζητούμενο 2 Παραλληλισμός Αλγορίθμου

Ο παραλληλισμός του αλγορίθμου εντοπίζεται στο γεγονός ότι υπάρχει ανεξαρτησία του υπολογισμού κατά γραμμές για δεδομένο k . Καθ' όλη την εκτέλεση του προγράμματος κρατάμε σταθερό τον τρόπο διαμοιρασμού των γραμμών του πίνακα, μοιράζοντας σε κάθε επανάληψη την k^{th} γραμμή.



...

Circular



Continuous

Ζητούμενο 3 Μοντέλο κοινού χώρου διευθύνσεων (OpenMP)

Ζητούμενο 4 Μοντέλο ανταλλαγής μηνυμάτων (MPI)

Ζητούμενο 4.1 Point to Point

Η υλοποίηση **point-to-point** με χρήση flooding για λογαριθμικό propagation και κυκλική κατανομή των γραμμών (*Circular*).

```
1  /* -.-.-.-.-.
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Mon 12 Nov 2012 01:25:21 PM EET
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * Created By : Alex Maurogiannis <nalfemp@gmail.com>
7  -.-.-.-.-.*/
8
9  #include <mpi.h>
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <signal.h>
13 #include <signal.h>
14 #include <unistd.h>
15 #include <string.h>
16
17 #include "common.h"
18
19 #define BLOCK_ROWS 1
20
21
22 void process_rows(int k, int rank, int N, int max_rank, double *A)
23 {
24     /* performs the calculations for a given set of rows.
25     * In this hybrid version each thread is assigned blocks of
26     * continuous rows in a cyclic manner.
27     */
28     int i, j, w;
29     double l;
30     /* For every cyclic repetition of a block */
31     for (i = (rank + ((BLOCK_ROWS * max_rank) * (k / (BLOCK_ROWS * max_rank)))); \
32          i < N ; i += (max_rank * BLOCK_ROWS)) {
33         if (i > k) {
34             /* Calculate each continuous row in the block */
35             for (w = i; w < (i + BLOCK_ROWS) && w < (N * N); w++) {
36                 l = A[(w * N) + k] / A[(k * N) + k];
37                 for (j = k; j < N; j++) {
38                     A[(w * N) + j] = A[(w * N) + j] - l * A[(k * N) + j];
39                 }
40             }
41         }
42     }
43 }
44
45 int main(int argc, char **argv)
46 {
47     int k;
48     int N;
49     int rank;
50     int max_rank;
51     int last_rank;
52     double *A = NULL;
53     double sec = 0;
54
55     int ret = 0;
56     FILE *fp = NULL;
57     usage(argc, argv);
58
59     MPI_Init(&argc, &argv);
60     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
61     MPI_Comm_size(MPI_COMM_WORLD, &max_rank);
62
63     if (rank == 0) {
64         debug("rank: %d opens file: %s\n", rank, argv[1]);
65         fp = fopen(argv[1], "r");
66         if (fp) {
67             if (!fscanf(fp, "%d\n", &N)) {
68                 MPI_Abort(MPI_COMM_WORLD, 1);
69             }
70         }
71         else {
72             MPI_Abort(MPI_COMM_WORLD, 1);
73         }
74     }
75
76     MPI_Barrier(MPI_COMM_WORLD);
77     propagate_with_flooding(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
78
79     /* Everyone allocates the whole table */
80     debug("Max rank = %d\n", max_rank);
81     if ((A = allocate_2d(N, N)) == NULL) {
```

```

82     MPI_Abort(MPI_COMM_WORLD, 1);
83 }
84 /* Root Parses file */
85 if (rank == 0) {
86     if(parse_matrix_2d(fp, N, N, A) == NULL) {
87         MPI_Abort(MPI_COMM_WORLD, 1);
88     }
89     fclose(fp);
90     fp = NULL;
91 }
92 /* And distributes the table */
93 MPI_Barrier(MPI_COMM_WORLD);
94 propagate_with_flooding(A, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
95
96 last_rank = (N - 1) % max_rank;
97
98 if(rank == 0) {
99     sec = timer();
100 }
101
102 for (k = 0; k < N - 1; k++) {
103     /* The owner of the row for this k broadcasts it*/
104     MPI_Barrier(MPI_COMM_WORLD);
105     propagate_with_flooding(&A[k * N], N, MPI_DOUBLE, \
106         ((k % (max_rank * BLOCK_ROWS)) / BLOCK_ROWS), MPI_COMM_WORLD);
107     process_rows(k, rank, N, max_rank, A);
108 }
109
110 MPI_Barrier(MPI_COMM_WORLD);
111 if (rank == 0) {
112     sec = timer();
113     printf("Calc Time: %lf\n", sec);
114 }
115 ret = MPI_Finalize();
116
117 if(ret == 0) {
118     debug("%d FINALIZED!!! with code: %d\n", rank, ret);
119 }
120 else {
121     debug("%d NOT FINALIZED!!! with code: %d\n", rank, ret);
122 }
123
124 /* Last process has table */
125 if (rank == last_rank) {
126     //print_matrix_2d(N, N, A);
127     fp = fopen(argv[2], "w");
128     fprintf_matrix_2d(fp, N, N, A);
129     fclose(fp);
130 }
131 free(A);
132
133 return 0;
134 }

```

Η υλοποίηση **point-to-point** με χρήση flooding για λογαριθμικό propagation και συνεχή κατανομή των γραμμών (*Continuous*).

```

1  /* .....
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Mon 12 Nov 2012 01:34:38 PM EET
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * Created By : Alex Maurogiannis <nalfemp@gmail.com>
7  .....*/
8
9  #include <mpi.h>
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <signal.h>
13 #include <signal.h>
14 #include <unistd.h>
15 #include <string.h>
16
17 #include "common.h"
18
19
20 int get_bcaster(int *ccounts, int bcaster)
21 {
22     if (ccounts[bcaster]-- > 0 ){
23         return bcaster;
24     } else {
25         return bcaster+1;
26     }
27 }
28
29 void get_displs(int *ccounts, int max_rank, int *displs)
30 {
31     int j;

```

```

32     displs[0] = 0;
33     for (j = 1; j < max_rank ; j++) {
34         displs[j] = displs[j - 1] + counts[j - 1];
35     }
36 }
37
38 int max(int a, int b)
39 {
40     return a > b ? a : b;
41 }
42
43 void process_rows(int k, int rank, int N, int max_rank, int block_rows, int *displs, double *A)
44 {
45     /*      performs the calculations for a given set of rows.
46     */
47     int j, w;
48     double l;
49     int start = max(displs[rank], k+1);
50     for (w = start; w < (start + block_rows) && w < N; w++){
51         l = A[(w * N) + k] / A[(k * N) + k];
52         for (j = k; j < N; j++) {
53             A[(w * N) + j] = A[(w * N) + j] - l * A[(k * N) + j];
54         }
55     }
56 }
57
58 /* distributes the rows in a continuous fashion */
59 void distribute_rows(int max_rank, int N, int *counts)
60 {
61     int j, k;
62     int rows = N;
63
64     /* Initialize counts */
65     for (j = 0; j < max_rank ; j++) {
66         counts[j] = (rows / max_rank);
67     }
68
69     /* Distribute the indivisible leftover */
70     if (rows / max_rank != 0) {
71         j = rows % max_rank;
72         for (k = 0; k < max_rank && j > 0; k++, j--) {
73             counts[k] += 1;
74         }
75     } else {
76         for (k = 0; k < max_rank; k++){
77             counts[k] = 1;
78         }
79     }
80 }
81
82
83
84 int main(int argc, char **argv)
85 {
86     int k;
87     int N;
88     int rank;
89     int max_rank;
90     int block_rows;
91     int *counts;
92     int *displs;
93     int *ccounts;
94     int ret = 0;
95     int bcaster = 0;
96     double sec;
97     double *A = NULL;
98     FILE *fp = NULL;
99
100
101     usage(argc, argv);
102
103     MPI_Init(&argc, &argv);
104     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
105     MPI_Comm_size(MPI_COMM_WORLD, &max_rank);
106
107     if (rank == 0) {
108         debug("rank: %d opens file: %s\n", rank, argv[1]);
109         fp = fopen(argv[1], "r");
110         if(fp) {
111             if(!fscanf(fp, "%d\n", &N)) {
112                 MPI_Abort(MPI_COMM_WORLD, 1);
113             }
114         }
115         else {
116             MPI_Abort(MPI_COMM_WORLD, 1);
117         }
118     }
119 }
120

```



```

121 MPI_Barrier(MPI_COMM_WORLD);
122 propagate_with_flooding(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
123
124 counts = malloc(max_rank * sizeof(int));
125 displs = malloc(max_rank * sizeof(int));
126 ccounts = malloc(max_rank * sizeof(int));
127
128 distribute_rows(max_rank, N, counts);
129 get_displs(counts, max_rank, displs);
130 memcpy(ccounts, counts, max_rank * sizeof(int));
131 #if main_DEBUG
132 printf("CCounts is :\n");
133 for (j = 0; j < max_rank ; j++) {
134     printf("%d\n", ccounts[j]);
135 }
136 #endif
137
138 /* Everybody Allocates the whole table */
139 if((A = allocate_2d(N, N)) == NULL) {
140     MPI_Abort(MPI_COMM_WORLD, 1);
141 }
142 if (rank == 0) {
143     if(parse_matrix_2d(fp, N, N, A) == NULL) {
144         MPI_Abort(MPI_COMM_WORLD, 1);
145     }
146     fclose(fp);
147     fp = NULL;
148 }
149
150 MPI_Barrier(MPI_COMM_WORLD);
151 propagate_with_flooding(A, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
152
153 /* Start Timing */
154 if(rank == 0) {
155     sec = timer();
156 }
157
158
159 for (k = 0; k < N - 1; k++) {
160     block_rows = counts[rk];
161     bcaster = get_bcaster(ccounts, bcaster);
162
163     debug(" broadcaster is %d\n", bcaster);
164     MPI_Barrier(MPI_COMM_WORLD);
165     propagate_with_flooding(&A[k * N], N, MPI_DOUBLE, bcaster, MPI_COMM_WORLD);
166
167     process_rows(k, rank, N, max_rank, block_rows, displs, A);
168 }
169
170 MPI_Barrier(MPI_COMM_WORLD);
171 if(rank == 0) {
172     sec = timer();
173     printf("Calc Time: %lf\n", sec);
174 }
175 ret = MPI_Finalize();
176
177 if(ret == 0) {
178     debug("%d FINALIZED!!! with code: %d\n", rank, ret);
179 }
180 else {
181     debug("%d NOT FINALIZED!!! with code: %d\n", rank, ret);
182 }
183
184 if(rank == (max_rank - 1)) {
185     fp = fopen(argv[2], "w");
186     fprintf_matrix_2d(fp, N, N, A);
187     fclose(fp);
188 }
189 free(A);
190 free(counts);
191 free(ccounts);
192
193 return 0;
194 }

```

Ζητούμενο 4.2 Collective

H collective υλοποίηση για κυκλική κατανομή των γραμμών (*Circular*).

```

1  /* -.-.-.-.-.
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Mon 12 Nov 2012 01:25:30 PM EET
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * Created By : Alex Maurogiannis <nalfemp@gmail.com>
7  -.-.-.-.-.*/
8
9  #include <mpi.h>

```

```

10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <signal.h>
13 #include <signal.h>
14 #include <unistd.h>
15 #include <string.h>
16
17 #include "common.h"
18
19 #define BLOCK_ROWS 1
20
21
22 void process_rows(int k, int rank, int N, int max_rank, double *A)
23 {
24     /* performs the calculations for a given set of rows.
25      * In this hybrid version each thread is assigned blocks of
26      * continuous rows in a cyclic manner.
27      */
28     int i, j, w;
29     double l;
30     /* For every cyclic repetition of a block */
31     for (i = (rank + ((BLOCK_ROWS * max_rank) * (k / (BLOCK_ROWS * max_rank)))); i < N ; i+=(max_rank * BLOCK_ROWS)) {
32         if (i > k) {
33             /* Calculate each continuous row in the block */
34             for (w = i; w < (i + BLOCK_ROWS) && w < (N * N); w++){
35                 l = A[(w * N) + k] / A[(k * N) + k];
36                 for (j = k; j < N; j++) {
37                     A[(w * N) + j] = A[(w * N) + j] - l * A[(k * N) + j];
38                 }
39             }
40         }
41     }
42 }
43
44 int main(int argc, char **argv)
45 {
46     int k;
47     int N;
48     int rank;
49     int max_rank;
50     int last_rank;
51     double *A = NULL;
52     double sec = 0;
53
54     int ret = 0;
55     FILE *fp = NULL;
56     usage(argc, argv);
57
58     MPI_Init(&argc, &argv);
59     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
60     MPI_Comm_size(MPI_COMM_WORLD, &max_rank);
61
62     if (rank == 0) {
63         debug("rank: %d opens file: %s\n", rank, argv[1]);
64         fp = fopen(argv[1], "r");
65         if(fp) {
66             if(!fscanf(fp, "%d\n", &N)) {
67                 MPI_Abort(MPI_COMM_WORLD, 1);
68             }
69         }
70         else {
71             MPI_Abort(MPI_COMM_WORLD, 1);
72         }
73     }
74
75     MPI_Barrier(MPI_COMM_WORLD);
76     MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
77
78     /* Everyone allocates the whole table */
79     debug("Max rank = %d\n", max_rank);
80     if((A = allocate_2d(N, N)) == NULL) {
81         MPI_Abort(MPI_COMM_WORLD, 1);
82     }
83
84     /* Root Parses file */
85     if (rank == 0) {
86         if(parse_matrix_2d(fp, N, N, A) == NULL) {
87             MPI_Abort(MPI_COMM_WORLD, 1);
88         }
89         fclose(fp);
90         fp = NULL;
91     }
92     /* And distributes the table */
93     MPI_Barrier(MPI_COMM_WORLD);
94     MPI_Bcast(A, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
95
96     last_rank = (N - 1) % max_rank;
97
98     if(rank == 0) {

```



```

51     double l;
52     int start = max(displs[rank], k+1);
53     for (w = start; w < (start + block_rows) && w < N; w++){
54         l = A[(w * N) + k] / A[(k * N) + k];
55         for (j = k; j < N; j++) {
56             A[(w * N) + j] = A[(w * N) + j] - l * A[(k * N) + j];
57         }
58     }
59 }
60
61 /* distributes the rows in a continuous fashion */
62 void distribute_rows(int max_rank, int N, int *counts)
63 {
64     int j, k;
65     int rows = N;
66
67     /* Initialize counts */
68     for (j = 0; j < max_rank ; j++) {
69         counts[j] = (rows / max_rank);
70     }
71
72     /* Distribute the indivisible leftover */
73     if (rows / max_rank != 0) {
74         j = rows % max_rank;
75         for (k = 0; k < max_rank && j > 0; k++, j--) {
76             counts[k] += 1;
77         }
78     }
79     else {
80         for (k = 0; k < max_rank; k++) {
81             counts[k] = 1;
82         }
83     }
84 }
85
86
87
88 int main(int argc, char **argv)
89 {
90     int k;
91     int N;
92     int rank;
93     int max_rank;
94     int block_rows;
95     int *counts;
96     int *displs;
97     int *ccounts;
98     int ret = 0;
99     int bcaster = 0;
100     double sec;
101     double *A = NULL;
102     FILE *fp = NULL;
103
104
105     usage(argc, argv);
106
107     MPI_Init(&argc, &argv);
108     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
109     MPI_Comm_size(MPI_COMM_WORLD, &max_rank);
110
111     if (rank == 0) {
112         debug("rank: %d opens file: %s\n", rank, argv[1]);
113         fp = fopen(argv[1], "r");
114         if (fp) {
115             if (!fscanf(fp, "%d\n", &N)) {
116                 MPI_Abort(MPI_COMM_WORLD, 1);
117             }
118         }
119         else {
120             MPI_Abort(MPI_COMM_WORLD, 1);
121         }
122     }
123
124     MPI_Barrier(MPI_COMM_WORLD);
125     MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
126
127     counts = malloc(max_rank * sizeof(int));
128     displs = malloc(max_rank * sizeof(int));
129     ccounts = malloc(max_rank * sizeof(int));
130
131     distribute_rows(max_rank, N, counts);
132     get_displs(counts, max_rank, displs);
133     memcpy(ccounts, counts, max_rank * sizeof(int));
134
135
136     #if main_DEBUG
137     printf("CCounts is :\n");
138     for (j = 0; j < max_rank ; j++) {
139         printf("%d\n", ccounts[j]);

```

```

140     }
141 #endif
142
143 /* Everybody Allocates the whole table */
144 if((A = allocate_2d(N, N)) == NULL) {
145     MPI_Abort(MPI_COMM_WORLD, 1);
146 }
147 if (rank == 0) {
148     if(parse_matrix_2d(fp, N, N, A) == NULL) {
149         MPI_Abort(MPI_COMM_WORLD, 1);
150     }
151     fclose(fp);
152     fp = NULL;
153 }
154
155 MPI_Barrier(MPI_COMM_WORLD);
156 MPI_Bcast(A, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
157
158 /* Start Timing */
159 if(rank == 0) {
160     sec = timer();
161 }
162
163
164 for (k = 0; k < N - 1; k++) {
165     block_rows = counts[rank];
166     bcaster = get_bcaster(ccounts, bcaster);
167
168     MPI_Barrier(MPI_COMM_WORLD);
169     debug(" broadcaster is %d\n", bcaster);
170     MPI_Barrier(MPI_COMM_WORLD);
171     MPI_Bcast(&A[k * N], N, MPI_DOUBLE, bcaster, MPI_COMM_WORLD);
172
173     process_rows(k, rank, N, max_rank, block_rows, displs, A);
174 }
175
176 MPI_Barrier(MPI_COMM_WORLD);
177 if(rank == 0) {
178     sec = timer();
179     printf("Calc Time: %lf\n", sec);
180 }
181 ret = MPI_Finalize();
182
183 if(ret == 0) {
184     debug("%d FINALIZED!!! with code: %d\n", rank, ret);
185 }
186 else {
187     debug("%d NOT FINALIZED!!! with code: %d\n", rank, ret);
188 }
189
190 if(rank == (max_rank - 1)) {
191     fp = fopen(argv[2], "w");
192     fprintf_matrix_2d(fp, N, N, A);
193     fclose(fp);
194 }
195 free(A);
196 free(counts);
197 free(ccounts);
198
199 return 0;
200 }

```