



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

11 Νοεμβρίου 2012

Πηγαίος κώδικας

Κοινή βιβλιοθήκη

```

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

```

```

31     for (i = 0; i < N; i++) {
32         for (j = 0; j < M; j++) {
33             if(!fscanf(fp, "%lf", p++)) {
34                 return NULL;
35             }
36         }
37     }
38     return A;
39 }
40
41 void fprintf_matrix_2d(FILE *fp, int N, int M, double *A)
42 {
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
68 double timer(void)
69 {
70     static double seconds = 0;
71     static int operation = 0;
72     struct timeval tv;
73     gettimeofday(&tv, NULL);
74     if (operation == 0) {
75         seconds = tv.tv_sec + (((double) tv.tv_usec)/1e6);
76         operation = 1;
77         return 0;
78     }
79     else {
80         operation = 0;
81         return tv.tv_sec + (((double) tv.tv_usec)/1e6) - seconds;
82     }
83 }
84
85 void usage(int argc, char **argv)
86 {
87     if(argc != 3) {
88         printf("Usage: %s <matrix file> <output file>\n", argv[0]);
89         exit(EXIT_FAILURE);
90     }
91 }
92
93 #ifdef USE_MPI /* USE_MPI */
94 void propagate_with_send(void *buffer, int count, MPI_Datatype datatype, 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
120 }
121
122 void propagate_with_flooding(void *buffer, int count , MPI_Datatype datatype, int root, MPI_Comm comm)
123 {
124     int rank;
125     int max_rank;
126     int cur;
127
128     MPI_Comm_rank(comm, &rank);
129     MPI_Comm_size(comm, &max_rank);
130
131     if(root != 0) {
132         if(rank == root) {
133             MPI_Send(buffer, count, datatype, 0, root, comm);
134         }
135         if(rank == 0) {
136             MPI_Status status;
137             MPI_Recv(buffer, count, datatype, root, root, comm, &status);
138         }
139     }
140
141
142     if(rank != 0) {
143         MPI_Status status;
144         MPI_Recv(buffer, count, datatype, (rank-1)/2, root, comm, &status);
145     }
146     cur = 2*rank+1;
147     if(cur < max_rank) {
148         MPI_Send(buffer, count, datatype, cur, root, comm);
149     }
150     if(++cur < max_rank) {
151         MPI_Send(buffer, count, datatype, cur, root, comm);
152     }
153 }
154 #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 Παράλληλισμός Αλγορίθμου

Ζητούμενο 3 Μοντέλο κοινού χώρου διευθύνσεων (OpenMP)

Ζητούμενο 4 Μοντέλο ανταλλαγής μηνυμάτων (MPI)

Ζητούμενο 4.1 Point to Point

```

1  /* ---
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Sun 11 Nov 2012 06:40:33 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 2
20
21
22 void process_rows(int k, int rank, int N, int max_rank, double *A){
23     /* performs the calculations for a given set of rows.
24     * In this hybrid version each thread is assigned blocks of
25     * continuous rows in a cyclic manner.
26     */

```

```

27  int i, j, w;
28  double l;
29  /* For every cyclic repetition of a block */
30  for (i = (rank + ((BLOCK_ROWS * max_rank) * (k / (BLOCK_ROWS * max_rank)))); i < N ; i+=(max_rank * BLOCK_ROWS)) {
31      if (i > k) {
32          /* Calculate each continuous row in the block*/
33          for (w = i; w < (i + BLOCK_ROWS) && w < (N * N); w++){
34              l = A[(w * N) + k] / A[(k * N) + k];
35              for (j = k; j < N; j++) {
36                  A[(w * N) + j] = A[(w * N) + j] - l * A[(k * N) + j];
37              }
38          }
39      }
40  }
41  }
42
43  int main(int argc, char **argv)
44  {
45      int k;
46      int N;
47      int rank;
48      int max_rank;
49      int last_rank;
50      double *A = NULL;
51      double sec = 0;
52
53      int ret = 0;
54      FILE *fp = NULL;
55      usage(argc, argv);
56
57      MPI_Init(&argc, &argv);
58      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
59      MPI_Comm_size(MPI_COMM_WORLD, &max_rank);
60
61      if (rank == 0) {
62          debug("rank: %d opens file: %s\n", rank, argv[1]);
63          fp = fopen(argv[1], "r");
64          if(fp) {
65              if(!fscanf(fp, "%d\n", &N)) {
66                  MPI_Abort(MPI_COMM_WORLD, 1);
67              }
68          }
69          else {
70              MPI_Abort(MPI_COMM_WORLD, 1);
71          }
72      }
73
74      MPI_Barrier(MPI_COMM_WORLD);
75      propagate_with_flooding(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
76
77      /* Everyone allocates the whole table */
78      debug("Max rank = %d\n", max_rank);
79      if((A = allocate_2d_with_padding(N, N, max_rank)) == NULL) {
80          MPI_Abort(MPI_COMM_WORLD, 1);
81      }
82
83      /* Root Parses file */
84      if (rank == 0) {
85          if(parse_matrix_2d(fp, N, N, A) == NULL) {
86              MPI_Abort(MPI_COMM_WORLD, 1);
87          }
88          fclose(fp);
89          fp = NULL;
90      }
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++) {

```

```

102     /* The owner of the row for this k broadcasts it*/
103     MPI_Barrier(MPI_COMM_WORLD);
104     propagate_with_flooding(&A[k * N], N, MPI_DOUBLE, ((k % (max_rank * BLOCK_ROWS)) / BLOCK_ROWS), MPI_COMM_WORLD);
105
106     process_rows(k, rank, N, max_rank, A);
107 }
108
109 MPI_Barrier(MPI_COMM_WORLD);
110 if (rank == 0) {
111     sec = timer();
112     printf("Calc Time: %lf\n", sec);
113 }
114 ret = MPI_Finalize();
115
116 if (ret == 0) {
117     debug("%d FINALIZED!!! with code: %d\n", rank, ret);
118 }
119 else {
120     debug("%d NOT FINALIZED!!! with code: %d\n", rank, ret);
121 }
122
123 /* Last process has table */
124 if (rank == last_rank) {
125     //print_matrix_2d(N, N, A);
126     fp = fopen(argv[2], "w");
127     fprintf_matrix_2d(fp, N, N, A);
128     fclose(fp);
129 }
130 free(A);
131
132 return 0;
133 }

```

Ζητούμενο 4.2 Collective

```

1  /* .....
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Thu 08 Nov 2012 09:49:47 AM 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 2
20
21
22 void process_rows(int k, int rank, int N, int max_rank, double *A){
23     /* performs the calculations for a given set of rows.
24     * In this hybrid version each thread is assigned blocks of
25     * continuous rows in a cyclic manner.
26     */
27     int i, j, w;
28     double l;
29     /* For every cyclic repetition of a block */
30     for (i = (rank + ((BLOCK_ROWS * max_rank) * (k / (BLOCK_ROWS * max_rank)))); i < N ; i+=(max_rank * BLOCK_ROWS)) {
31         if (i > k) {
32             /* Calculate each continuous row in the block*/
33             for (w = i; w < (i + BLOCK_ROWS) && w < (N * N); w++){
34                 l = A[(w * N) + k] / A[(k * N) + k];
35                 for (j = k; j < N; j++) {
36                     A[(w * N) + j] = A[(w * N) + j] - l * A[(k * N) + j];
37                 }
38             }
39         }
40     }

```

```

41 }
42
43 int main(int argc, char **argv)
44 {
45     int k;
46     int N;
47     int rank;
48     int max_rank;
49     int last_rank;
50     double *A = NULL;
51     double sec = 0;
52
53     int ret = 0;
54     FILE *fp = NULL;
55     usage(argc, argv);
56
57     MPI_Init(&argc, &argv);
58     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
59     MPI_Comm_size(MPI_COMM_WORLD, &max_rank);
60
61     if (rank == 0) {
62         debug("rank: %d opens file: %s\n", rank, argv[1]);
63         fp = fopen(argv[1], "r");
64         if(fp) {
65             if(!fscanf(fp, "%d\n", &N)) {
66                 MPI_Abort(MPI_COMM_WORLD, 1);
67             }
68         }
69         else {
70             MPI_Abort(MPI_COMM_WORLD, 1);
71         }
72     }
73
74     MPI_Barrier(MPI_COMM_WORLD);
75     MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
76
77     /* Everyone allocates the whole table */
78     debug("Max rank = %d\n", max_rank);
79     if((A = allocate_2d_with_padding(N, N, max_rank)) == NULL) {
80         MPI_Abort(MPI_COMM_WORLD, 1);
81     }
82
83     /* Root Parses file */
84     if (rank == 0) {
85         if(parse_matrix_2d(fp, N, N, A) == NULL) {
86             MPI_Abort(MPI_COMM_WORLD, 1);
87         }
88         fclose(fp);
89         fp = NULL;
90     }
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) {
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_Bcast(&A[k * N], N, MPI_DOUBLE, ((k % (max_rank * BLOCK_ROWS)) / BLOCK_ROWS), MPI_COMM_WORLD);
105
106         process_rows(k, rank, N, max_rank, A);
107     }
108
109     MPI_Barrier(MPI_COMM_WORLD);
110     if (rank == 0) {
111         sec = timer();
112         printf("Calc Time: %lf\n", sec);
113     }
114     ret = MPI_Finalize();
115

```



```

116     if (ret == 0) {
117         debug("%d FINALIZED!!! with code: %d\n", rank, ret);
118     }
119     else {
120         debug("%d NOT FINALIZED!!! with code: %d\n", rank, ret);
121     }
122
123     /* Last process has table */
124     if (rank == last_rank) {
125         //print_matrix_2d(N, N, A);
126         fp = fopen(argv[2], "w");
127         fprintf_matrix_2d(fp, N, N, A);
128         fclose(fp);
129     }
130     free(A);
131
132     return 0;
133 }

```