



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 |

Πηγαίος κώδικας

Κοινή βιβλιοθήκη

```
1  /* -.-.-.-.-.
2  * File Name : common.h
3  * Creation Date : 06-11-2012
4  * Last Modified : Wed 12 Dec 2012 08:37:45 PM 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
24 /* Operation Mode */
25 enum OPMODE { CONTINUOUS, CYCLIC, OPMODE_SIZE };
26 typedef enum OPMODE OPMODE;
27
28 #define MIN(a,b) ((a) < (b)) ? (a) : (b)
29 #define MAX(a,b) ((a) > (b)) ? (a) : (b)
30
31 #include <stdlib.h>
32 #include <stdio.h>
33
34 struct time_struct {
35     struct timeval latest_timestamp;
36     struct timeval current_duration;
37 };
38
39 typedef struct time_struct time_struct;
40
41 void time_struct_init(time_struct *ts);
42 void time_struct_set_timestamp(time_struct *ts);
43 void time_struct_add_timestamp(time_struct *ts);
44 double get_seconds(time_struct *ts);
45
46 struct Matrix {
47     int N;
48     double *A;
49 };
50
51 typedef struct Matrix Matrix;
52
53 Matrix *get_matrix(char *filename, int max_rank, OPMODE operation);
54 double **appoint_2D(double *A, int N, int M);
55 void fprintf_matrix_2d(FILE *fp, int N, int M, double *A);
56 void print_matrix_2d(int N, int M, double *A);
57 double timer(void);
58 void usage(int argc, char **argv);
59 void * get_propagation(int argc, char **argv);
60
61 void upper_triangularize(int N, double **Ap2D);
62
63 #ifdef USE_MPI /* USE_MPI */
64 #include <mpi.h>
65 void propagate_with_send(void *buffer, int count , MPI_Datatype datatype, \
66     int root, MPI_Comm comm);
67 void propagate_with_flooding(void *buffer, int count , MPI_Datatype datatype, \
68     int root, MPI_Comm comm);
69 void gather_to_root_cyclic(double **Ap2D, int max_rank, int rank, int root, double **A2D, int N, int M);
70 void get_counts(int max_rank, int N, int *counts);
71 void get_displs(int *counts, int max_rank, int *displs);
72 #endif /* USE_MPI */
73
74 #endif /* COMMON_H */
75
76
77 1  /* -.-.-.-.-.
78 2  * File Name : common.c
79 3  * Creation Date : 06-11-2012
80 4  * Last Modified : Wed 12 Dec 2012 08:51:50 PM EET
81 5  * Created By : Greg Liras <gregliras@gmail.com>
82 6  * Created By : Alex Maurogiannis <nalfemp@gmail.com>
83 7  -.-.-.-.-.*/
84
85 #include "common.h"
```

```

10 #include <sys/time.h>
11 #include <string.h>
12
13
14
15
16 static double *allocate_2d(int N, int M)
17 {
18     double *A;
19     A = malloc(N * M * sizeof(double));
20     return A;
21 }
22
23 static double *allocate_2d_with_padding(int N, int M, int max_rank)
24 {
25     return allocate_2d(N+max_rank, M);
26 }
27
28 static double *parse_matrix_2d_cyclic(FILE *fp, unsigned int N, unsigned int M, double *A, int max_rank)
29 {
30     int i,j;
31     double *p;
32     int workload = N / max_rank + 1;
33     int remainder = N % max_rank;
34     double **A2D = appoint_2D(A, N + max_rank, M);
35
36     for(i = 0; i < workload - 1; i++) {
37         for(j = 0; j < max_rank; j++) {
38             p = A2D[j*workload + i];
39
40             if(fread(p, sizeof(double), M, fp) != M) {
41                 return NULL;
42             }
43         }
44     }
45
46
47
48     /* this loop reads any remaining data from the file */
49     for(i = 1; i <= remainder; i++) {
50         p = A2D[i*workload] - M;
51         if(fread(p, sizeof(double), M, fp) != M) {
52             return NULL;
53         }
54     }
55
56     /* this loop memsets the final line of the bottom parts */
57     for(i = max_rank - remainder + 1; i < max_rank; i++) {
58         p = A2D[i*workload] - M;
59         memset(p, 0, M*sizeof(double));
60     }
61
62     free(A2D);
63     return A;
64 }
65
66
67 static double *parse_matrix_2d(FILE *fp, int N, int M, double *A, int max_rank, OPMODE operation)
68 {
69     switch(operation) {
70         case CONTINUOUS:
71             return parse_matrix_2d_cyclic(fp, N, M, A, 1);
72         case CYCLIC:
73             return parse_matrix_2d_cyclic(fp, N, M, A, max_rank);
74         default:
75             return NULL;
76     }
77 }
78
79 /* Turns a 2D matrix to upper triangular */
80 void upper_triangularize(int N, double **Ap2D)
81 {
82
83     int i,j;
84     for (i=1; i < N; i++) {
85         for (j=0; j < i; j++) {
86             Ap2D[i][j] = 0;
87         }
88     }
89 }
90 void fprintf_matrix_2d(FILE *fp, int N, int M, double *A)
91 {
92     int i,j;
93     double *p;
94     p = A;
95     for (j = 0; j < M; j++) {
96         fprintf(fp, "=");
97     }
98     fprintf(fp, "\n");

```

```

99     for (i = 0; i < N; i++) {
100         for (j = 0; j < M; j++) {
101             fprintf(fp, "%lf\t", *p++);
102         }
103         fprintf(fp, "\n");
104     }
105     for (j = 0; j < M; j++) {
106         fprintf(fp, "=");
107     }
108     fprintf(fp, "\n");
109 }
110
111 void print_matrix_2d(int N, int M, double *A)
112 {
113     fprintf_matrix_2d(stdout, N, M, A);
114 }
115
116
117 /* Initialize ts to zero */
118 void time_struct_init(time_struct *ts)
119 {
120     ts->latest_timestamp.tv_sec = 0;
121     ts->latest_timestamp.tv_usec = 0;
122     ts->current_duration.tv_sec = 0;
123     ts->current_duration.tv_usec = 0;
124 }
125
126 /* Set ts timestamp to current time */
127 void time_struct_set_timestamp(time_struct *ts)
128 {
129     struct timeval tv;
130     gettimeofday(&tv, NULL);
131     ts->latest_timestamp.tv_sec = tv.tv_sec;
132     ts->latest_timestamp.tv_usec = tv.tv_usec;
133 }
134
135 /* Set ts timestamp to current time and add the diff to current_duration */
136 void time_struct_add_timestamp(time_struct *ts)
137 {
138     struct timeval tv;
139     gettimeofday(&tv, NULL);
140
141     ts->current_duration.tv_sec += tv.tv_sec - ts->latest_timestamp.tv_sec;
142     ts->current_duration.tv_usec += tv.tv_usec - ts->latest_timestamp.tv_usec;
143
144     ts->latest_timestamp.tv_sec = tv.tv_sec;
145     ts->latest_timestamp.tv_usec = tv.tv_usec;
146 }
147
148 double get_seconds(time_struct *ts)
149 {
150     return ts->current_duration.tv_sec + (((double) ts->current_duration.tv_usec)/1e6);
151 }
152
153
154
155 double timer(void)
156 {
157     static double seconds = 0;
158     static int operation = 0;
159     struct timeval tv;
160     gettimeofday(&tv, NULL);
161     if (operation == 0) {
162         seconds = tv.tv_sec + (((double) tv.tv_usec)/1e6);
163         operation = 1;
164         return 0;
165     }
166     else {
167         operation = 0;
168         return tv.tv_sec + (((double) tv.tv_usec)/1e6) - seconds;
169     }
170 }
171
172 void usage(int argc, char **argv)
173 {
174     #ifdef USE_MPI /* USE_MPI */
175     if (argc > 4 || argc < 3) {
176         printf("Usage: %s <matrix file> <output file> [propagation mode: default=0 (ptp)]\n", argv[0]);
177         exit(EXIT_FAILURE);
178     }
179     #else
180     if (argc != 3) {
181         printf("Usage: %s <matrix file> <output file>\n", argv[0]);
182         exit(EXIT_FAILURE);
183     }
184     #endif
185 }
186
187 Matrix *get_matrix(char *filename, int max_rank, OPMODE operation)

```

```

188 {
189     FILE *fp;
190     double *A;
191     int N;
192     Matrix *mat;
193
194     if(NULL == (mat = malloc(sizeof(struct Matrix)))) {
195         debug("Could not allocate empty Matrix\n");
196         exit(EXIT_FAILURE);
197     }
198     fp = fopen(filename, "rb");
199     if(fp) {
200         if(fread(&N, sizeof(int), 1, fp) != 1) {
201             debug("Could not read N from file\n");
202             exit(EXIT_FAILURE);
203         }
204     }
205     if((A = allocate_2d_with_padding(N, N, max_rank)) == NULL) {
206         debug("Could not allocate enough contiguous memory\n");
207         exit(EXIT_FAILURE);
208     }
209     if(parse_matrix_2d(fp, N, N, A, max_rank, operation) == NULL) {
210         debug("Could not parse matrix\n");
211         exit(EXIT_FAILURE);
212     }
213     fclose(fp);
214     mat->N = N;
215     mat->A = A;
216
217     return mat;
218 }
219
220 double **appoint_2D(double *A, int N, int M)
221 {
222     int i;
223     double **A2D = (double **) malloc(N*sizeof(double *));
224     /* sanity check */
225     if(NULL == A2D) {
226         return NULL;
227     }
228     for(i = 0; i < N; i++) {
229         A2D[i] = &A[i*M];
230     }
231     return A2D;
232 }
233
234 #ifdef USE_MPI /* USE_MPI */
235
236 /* get operation mode from the third argument.
237  * 1 for continuous, 0 for ptp */
238 void * get_propagation(int argc, char **argv)
239 {
240     if (argc > 3) {
241         if (argv[3][0] == '1') {
242             return &MPI_Bcast;
243         }
244     }
245     return &propagate_with_flooding;
246 }
247
248 void propagate_with_send(void *buffer, int count, MPI_Datatype datatype, \
249     int root, MPI_Comm comm)
250 {
251     int rank;
252     int i;
253     int max_rank;
254     MPI_Status status;
255
256     MPI_Comm_rank(comm, &rank);
257     MPI_Comm_size(comm, &max_rank);
258     if(rank == root) {
259         for(i = 0; i < max_rank; i++) {
260             if(i == rank) {
261                 continue;
262             }
263             else {
264                 debug("%d\n", i);
265                 MPI_Send(buffer, count, datatype, i, root, comm);
266             }
267         }
268     }
269     else {
270         MPI_Recv(buffer, count, datatype, root, root, comm, &status);
271     }
272 }
273
274 void propagate_with_flooding(void *buffer, int count, MPI_Datatype datatype, \
275     int root, MPI_Comm comm)
276 {

```

```

277     int rank;
278     int max_rank;
279     int cur;
280
281     MPI_Comm_rank(comm, &rank);
282     MPI_Comm_size(comm, &max_rank);
283     MPI_Status status;
284
285     if(root != 0) {
286         if(rank == root) {
287             MPI_Send(buffer, count, datatype, 0, root, comm);
288         }
289         if(rank == 0) {
290             MPI_Recv(buffer, count, datatype, root, root, comm, &status);
291         }
292     }
293
294     if(rank != 0) {
295         MPI_Status status;
296         MPI_Recv(buffer, count, datatype, (rank-1)/2, root, comm, &status);
297     }
298     cur = 2*rank+1;
299     if(cur < max_rank) {
300         MPI_Send(buffer, count, datatype, cur, root, comm);
301     }
302     if(++cur < max_rank) {
303         MPI_Send(buffer, count, datatype, cur, root, comm);
304     }
305 }
306
307 /* Returns the displacements table in rows */
308 void get_displs(int *counts, int max_rank, int *displs)
309 {
310     int j;
311     displs[0] = 0;
312     for (j = 1; j < max_rank; j++) {
313         displs[j] = displs[j - 1] + counts[j - 1];
314     }
315 }
316
317 /* distributes the rows in a continuous fashion */
318 void get_counts(int max_rank, int N, int *counts)
319 {
320     int j, k;
321     int rows = N;
322
323     /* Initialize counts */
324     for (j = 0; j < max_rank; j++) {
325         counts[j] = (rows / max_rank);
326     }
327
328     /* Distribute the indivisible leftover */
329     if (rows / max_rank != 0) {
330         j = rows % max_rank;
331         for (k = 0; k < max_rank && j > 0; k++, j--) {
332             counts[k] += 1;
333         }
334     }
335     else {
336         for (k = 0; k < max_rank; k++) {
337             counts[k] = 1;
338         }
339     }
340 }
341
342
343 /* Gather everything to root */
344 void gather_to_root_cyclic(double **Ap2D, int max_rank, int rank, int root, double **A2D, int N, int M)
345 {
346     int i;
347     int bcaster;
348     int current_row;
349     MPI_Status status;
350     for(i = 0; i < N; i++) {
351         bcaster = i % max_rank;
352         current_row = i / max_rank;
353         MPI_Barrier(MPI_COMM_WORLD);
354         if(rank == bcaster) {
355             if(bcaster == root) {
356                 memcpy(A2D[i], Ap2D[current_row], M*sizeof(double));
357             }
358             else {
359                 MPI_Send(Ap2D[current_row], M, MPI_DOUBLE, 0, i, MPI_COMM_WORLD);
360             }
361         }
362         else if (rank == root) {
363             MPI_Recv(A2D[i], M, MPI_DOUBLE, bcaster, i, MPI_COMM_WORLD, &status);
364         }
365     }
366 }

```

```

366 }
367
368
369 #endif /* USE_MPI */

```

Ζητούμενο 1 Σειριακό Πρόγραμμα

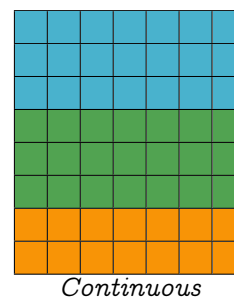
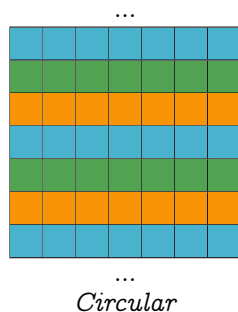
```

1  /* .....
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Thu 29 Nov 2012 03:19:28 PM 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     Matrix *mat = get_matrix(argv[1],0, CONTINUOUS);
28     N = mat->N;
29     A = mat->A;
30     fp = fopen(argv[1], "rb");
31
32
33     sec = timer();
34     for (k = 0; k < N - 1; k++)
35     {
36         for (i = k + 1; i < N; i++)
37         {
38             l = A[i * N + k] / A[k * N + k];
39             for (j = k; j < N; j++)
40             {
41                 A[i * N + j] = A[i * N + j] - l * A[k * N + j];
42             }
43         }
44     }
45     sec = timer();
46     printf("Calc Time: %lf\n", sec);
47
48     fp = fopen(argv[2], "w");
49     fprintf_matrix_2d(fp, N, N, A);
50     fclose(fp);
51     free(A);
52
53     return 0;
54 }

```

Ζητούμενο 2 Παράλληλισμός Αλγορίθμου

Ο παράλληλισμός 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.



Ζητούμενο 3 Μοντέλο κοινού χώρου διευθύνσεων (OpenMP)

Η υλοποίηση μοντέλου κοινού χώρου διευθύνσεων βασίζεται στην δομή `pragma omp for`, με χρήση `private` μεταβλητών `divisor` και `A2` για κάθε `thread` ώστε να αποφεύγονται όσο γίνεται οι προσβάσεις στην κοινή μνήμη. Επιπλέον, έχουν πραγματοποιηθεί βελτιστοποιήσεις μέσω `flags` του `gcc`, όπως φαίνεται στο `Makefile`.

```

1  /* .....*/
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Thu 20 Dec 2012 01:27:37 PM 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 #include <omp.h>
12
13
14 #include "common.h"
15
16 int main(int argc, char **argv)
17 {
18     int i,j,k;
19     int N;
20     double *A;
21     double **A2D;
22     int flag = 1;
23
24     double l;
25     double sec;
26
27     FILE *fp = NULL;
28     usage(argc, argv);
29     /*
30      * Allocate me!
31      */
32
33     Matrix *mat = get_matrix(argv[1],0, CONTINUOUS);
34     N = mat->N;
35     A = mat->A;
36     A2D = appoint_2D(A, N, N);
37
38
39
40     int chunk = N/omp_get_max_threads();
41     double *Ai;
42     double *Ak;
43     chunk = 1;
44
45     sec = timer();
46
47     for (k = 0; k < N - 1; k++)
48     {
49         #pragma omp parallel private(Ak)
50         {
51             if(flag)
52             {
53                 printf("%d %d\n", omp_get_num_threads(), omp_get_max_threads());
54                 flag=0;
55             }
56             Ak = A2D[k];
57             #pragma omp for schedule(static, chunk) private(l,j, Ai)
58             for (i = k + 1; i < N; i++)
59             {
60                 Ai = A2D[i];
61
62                 l = Ai[k] / Ak[k];
63                 for (j = k; j < N; j++)
64                 {
65                     Ai[j] = Ai[j] - l * Ak[j];
66                 }
67             }
68         }
69
70         sec = timer();
71         printf("Calc Time: %lf\n", sec);
72
73         fp = fopen(argv[2], "w");
74         fprintf_matrix_2d(fp, N, N, A);
75         fclose(fp);
76         free(A);
77
78         return 0;
79     }

```


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*).

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

Ζητούμενο 4.2 Collective

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

Η **collective** υλοποίηση για συνεχή κατανομή των γραμμών (*Continuous*).