



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 fprint_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
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

29     double *p;
30     p = A;
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 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(root != 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     /*

```

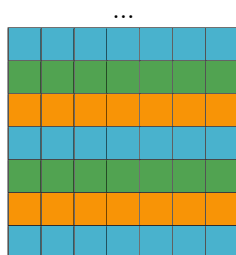
```

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 }

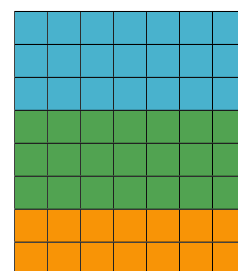
```

Ζητούμενο 2 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.



...
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 09:58:12 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 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[(k * 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_with_padding(N, N, max_rank)) == 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 09:56:29 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
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 *counts, 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      *      In this hybrid version each thread is assigned blocks of
47      *      continuous rows in a cyclic manner.
48      */
49     int j, w;
50     double l;
51     int start = max(displs[rank], k+1);
52     for (w = start; w < (start + block_rows) && w < N; w++){
53         l = A[(w * N) + k] / A[(k * N) + k];
54         for (j = k; j < N; j++) {
55             A[(w * N) + j] = A[(k * N) + j] - l * A[(k * N) + j];
56         }
57     }
58 }
59
60 /* distributes the rows in a continuous fashion */
61 void distribute_rows(int max_rank, int N, int *counts)
62 {
63     int j, k;
64     int rows = N;
65
66     /* Initialize counts */
67     for (j = 0; j < max_rank ; j++) {
68         counts[j] = (rows / max_rank);
69     }
70
71     /* Distribute the indivisible leftover */
72     if (rows / max_rank != 0) {
73         j = rows % max_rank;
74         for (k = 0; k < max_rank && j > 0; k++, j--) {
75             counts[k] += 1;
76         }
77     } else {
78         for (k = 0; k < max_rank; k++){
79             counts[k] = 1;
80         }
81     }

```



```

82 }
83
84
85
86 int main(int argc, char **argv)
87 {
88     int i, j, k;
89     int N;
90     int rank;
91     int max_rank;
92     int block_rows;
93     int *counts;
94     int *displs;
95     int *ccounts;
96     int ret = 0;
97     int bcaster = 0;
98     double l;
99     double sec;
100     double *A = NULL;
101     FILE *fp = NULL;
102
103
104     usage(argc, argv);
105
106     MPI_Init(&argc, &argv);
107     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
108     MPI_Comm_size(MPI_COMM_WORLD, &max_rank);
109
110     if (rank == 0) {
111         debug("rank: %d opens file: %s\n", rank, argv[1]);
112         fp = fopen(argv[1], "r");
113         if (fp) {
114             if (!fscanf(fp, "%d\n", &N)) {
115                 MPI_Abort(MPI_COMM_WORLD, 1);
116             }
117         }
118         else {
119             MPI_Abort(MPI_COMM_WORLD, 1);
120         }
121     }
122
123     MPI_Barrier(MPI_COMM_WORLD);
124     propagate_with_flooding(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
125
126     counts = malloc(max_rank * sizeof(int));
127     displs = malloc(max_rank * sizeof(int));
128     ccounts = malloc(max_rank * sizeof(int));
129
130     distribute_rows(max_rank, N, counts);
131     get_displs(counts, max_rank, displs);
132     memcpy(ccounts, counts, max_rank * sizeof(int));
133
134     #if main_DEBUG
135     printf("CCounts is :\n");
136     for (j = 0; j < max_rank ; j++) {
137         printf("%d\n", ccounts[j]);
138     }
139     #endif
140
141     /* Everybody Allocates the whole table */
142     if ((A = allocate_2d_with_padding(N, N, max_rank)) == NULL) {
143         MPI_Abort(MPI_COMM_WORLD, 1);
144     }
145     if (rank == 0) {
146         if (parse_matrix_2d(fp, N, N, A) == NULL) {
147             MPI_Abort(MPI_COMM_WORLD, 1);
148         }
149         fclose(fp);
150         fp = NULL;
151     }
152
153     MPI_Barrier(MPI_COMM_WORLD);
154     propagate_with_flooding(A, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
155
156     /* Start Timing */

```



```

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_with_padding(N, N, max_rank)) == 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
93     /* And distributes the table */
94     MPI_Barrier(MPI_COMM_WORLD);
95     MPI_Bcast(A, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
96
97     last_rank = (N - 1) % max_rank;
98
99     if (rank == 0) {
100         sec = timer();
101     }
102
103     for (k = 0; k < N - 1; k++) {
104         /* The owner of the row for this k broadcasts it*/
105         MPI_Barrier(MPI_COMM_WORLD);

```

```

105     MPI_Bcast(&A[k * N], N, MPI_DOUBLE, ((k % (max_rank * BLOCK_ROWS)) / BLOCK_ROWS), MPI_COMM_WORLD);
106
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 }

```

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

```

1  /* -.-.-.-.-.
2  * File Name : main.c
3  * Creation Date : 30-10-2012
4  * Last Modified : Mon 12 Nov 2012 10:03:03 AM EET
5  * Created By : Greg Liras <greyliras@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 <unistd.h>
14 #include <string.h>
15
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     }
25     else {
26         return bcaster+1;
27     }
28 }
29
30 void get_displs(int *ccounts, int max_rank, int *displs)
31 {
32     int j;
33     displs[0] = 0;
34     for (j = 1; j < max_rank ; j++) {
35         displs[j] = displs[j - 1] + ccounts[j - 1];
36     }
37 }
38
39 int max(int a, int b)
40 {
41     return a > b ? a : b;
42 }
43

```

```

44 void process_rows(int k, int rank, int N, int max_rank, int block_rows, int *displs, double *A)
45 {
46     /*      performs the calculations for a given set of rows.
47      *      In this hybrid version each thread is assigned blocks of
48      *      continuous rows in a cyclic manner.
49      */
50     int j, w;
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 i, j, 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 l;
101     double sec;
102     double *A = NULL;
103     FILE *fp = NULL;
104
105
106     usage(argc, argv);
107
108     MPI_Init(&argc, &argv);
109     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
110     MPI_Comm_size(MPI_COMM_WORLD, &max_rank);
111
112     if (rank == 0) {
113         debug("rank: %d opens file: %s\n", rank, argv[1]);
114         fp = fopen(argv[1], "r");
115         if(fp) {
116             if(!fscanf(fp, "%d\n", &N)) {
117                 MPI_Abort(MPI_COMM_WORLD, 1);
118             }

```

```

119     }
120     else {
121         MPI_Abort(MPI_COMM_WORLD, 1);
122     }
123
124 }
125
126 MPI_Barrier(MPI_COMM_WORLD);
127 MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
128
129 counts = malloc(max_rank * sizeof(int));
130 displs = malloc(max_rank * sizeof(int));
131 ccounts = malloc(max_rank * sizeof(int));
132
133 distribute_rows(max_rank, N, counts);
134 get_displs(counts, max_rank, displs);
135 memcpy(ccounts, counts, max_rank * sizeof(int));
136
137 #if main_DEBUG
138 printf("CCounts is :\n");
139 for (j = 0; j < max_rank ; j++) {
140     printf("%d\n", ccounts[j]);
141 }
142 #endif
143
144 /* Everybody Allocates the whole table */
145 if((A = allocate_2d_with_padding(N, N, max_rank)) == NULL) {
146     MPI_Abort(MPI_COMM_WORLD, 1);
147 }
148 if (rank == 0) {
149     if(parse_matrix_2d(fp, N, N, A) == NULL) {
150         MPI_Abort(MPI_COMM_WORLD, 1);
151     }
152     fclose(fp);
153     fp = NULL;
154 }
155
156 MPI_Barrier(MPI_COMM_WORLD);
157 MPI_Bcast(A, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
158
159 /* Start Timing */
160 if(rank == 0) {
161     sec = timer();
162 }
163
164
165 for (k = 0; k < N - 1; k++) {
166     block_rows = counts[rank];
167     bcaster = get_bcaster(ccounts, bcaster);
168
169     MPI_Barrier(MPI_COMM_WORLD);
170     debug(" broadcaster is %d\n", bcaster);
171     MPI_Barrier(MPI_COMM_WORLD);
172     MPI_Bcast(&A[k * N], N, MPI_DOUBLE, bcaster, MPI_COMM_WORLD);
173
174     process_rows(k, rank, N, max_rank, block_rows, displs, A);
175 }
176
177 MPI_Barrier(MPI_COMM_WORLD);
178 if(rank == 0) {
179     sec = timer();
180     printf("Calc Time: %lf\n", sec);
181 }
182 ret = MPI_Finalize();
183
184 if(ret == 0) {
185     debug("%d FINALIZED!!! with code: %d\n", rank, ret);
186 }
187 else {
188     debug("%d NOT FINALIZED!!! with code: %d\n", rank, ret);
189 }
190
191 if(rank == (max_rank - 1)) {
192     fp = fopen(argv[2], "w");
193     fprintf_matrix_2d(fp, N, N, A);

```

```
194         fclose(fp);
195     }
196     free(A);
197     free(counts);
198     free(ccounts);
199
200     return 0;
201 }
```