Παράλληλη Επεξεργασία - Εργασία Εξαμήνου

Ιωαννης Καλδίρης 1080428
Αντωνία Καρρά 1072468
Άρτεμις Νικολάου 1067496
Αικατερίνη Σταυροπούλου 1064282

Περιεχόμενα

I Εισαγωγή 2

II Υλοποίηση 2

III Παρατηρήσεις 6

IV Εκτέλεση 6

I Εισαγωγή

Ο κώδικας και οι μετρήσεις υλοποιήθηκαν σε λειτουργικό σύστημα Zorin OS. Το σύστημα χρησιμοποιεί αρχιτεκτονική CPU Coffee Lake στα 2,80 GHz και μνήμη cache 9MB σε επεξεργαστή με 4 cores/8 threads.

MDS

Μια σημαντική μέθοδος στο πεδίο της παράλληλης επεξεργασίας είναι ο Αλγόριθμος Πολυδιάστατης Αναζήτησης (MDS), ο οποίος λειτουργεί σε ένα ημιαμφίδρομο σημείων εντός του χώρου αναζήτησης. Ο αλγόριθμος MDS περιλαμβάνει μια σειρά λειτουργιών για να διασφαλιστεί η σύγκλιση στο τοπικό ελάχιστο και βασίζεται αποκλειστικά σε κλήσεις συναρτήσεων που μπορούν να εκτελεστούν ταυτόχρονα.

Σε αυτήν την εργασία εστιάζουμε στην υλοποίηση και την παραλληλοποίηση του αλγόριθμου MDS, χρησιμοποιώντας διαφορετικά μοντέλα παράλληλου προγραμματισμού για τη βελτίωση της απόδοσης και τη μείωση του χρόνου υπολογισμού. Τα επιλεγμένα μοντέλα για παραλληλοποίηση περιλαμβάνουν OpenMP, OpenM tasks και MPI, καθένα από τα οποία προσφέρει μοναδικές προσεγγίσεις για τη διανομή και τη διαχείριση υπολογιστικού φόρτου εργασίας.

II Υλοποίηση

OpenMP

Για να επιτευχθεί η παραλληλοποίηση στην έκδοση OpenMP, χρησιμοποιήθηκε ένας κυκλικός διαχωρισμός των διεργασιών σε πολλαπλά νήματα. Με τον τρόπο αυτό εξασφαλίστηκε η εξισορρόπηση του φορτίου μεταξύ των διεργασιών. Εφόσον, για οποιοδήποτε τυχαίο σημείο, μπορούμε να βρούμε τη λύση, χρησιμοποιούμε πολλαπλά νήματα για να εκτελέσουμε τον παρακάτω βρόχο, όπου κάθε νήμα εκτελείται ανεξάρτητα για να βρεθεί μια λύση και στη συνέχεια ελέγχεται εάν είναι η βέλτιστη. Τέλος αυτή η λύση διατηρείται στο αρχείο της βέλτιστης λύσης.

```
1          #pragma omp parallel for firstprivate(startpt, endpt) private(
               trial, i, nt, nf, fx) shared(best_trial, best_nt, best_nf,
               best_fx, best_pt)
```

- pragma omp parallel for: Χρησιμοποιείται για την παραλληλοποίηση του βρόχου for. Έτσι, ο βρόχος for εκτελείται από πολλαπλά νήματα.
- firstprivate: Είναι η διατήρηση της αρχικής τιμής της μεταβλητής στην παράλληλη περιοχή, αλλά δημιουργείται ένα τοπικό αντίγραφο για κάθε νήμα. Κάθε νήμα στην παράλληλη περιοχή έχει ένα τοπικό αντίγραφο του πίνακα έναρξης και τερματισμού.
- private: Διατηρεί τη μεταβλητής ιδιωτική σε κάθε νήμα, έτσι ώστε κάθε νήμα να μπορεί να έχει το αντίγραφό της.
- shared: Διατηρεί τη μεταβλητή κοινή σε όλα τα νήματα, έτσι ώστε όλα τα νήματα να έχουν πρόσβαση στη μεταβλητή. Η τροποποίηση της κοινής μεταβλητής θα πρέπει να εκτελείται σε ένα κρίσιμο τμήμα για να αποφευχθεί ένα ενδεχόμενο race condition.

Στην παρακάτω περιοχή, ελέγχουμε την τιμή, εάν είναι μεγαλύτερη από την καλύτερη λύση μέχρι στιγμής. Αποθηκεύουμε το σημείο στην κρίσιμη ενότητα. Είναι σημαντικό αυτό το βήμα, διαφορετικά θα προκληθεί ένα race condition και το αποτέλεσμα δεν θα είναι ακριβές.

```
        if (fx < best_fx) {
                        // We will update the best solution based on the
                           best_fx value
                        #pragma omp critical
                        {
                                // the recheck ensures that the entered
                                   thread actually do has the best result
                                if(fx < best_fx) {
                                        best_trial = trial;
                                        best_nt = nt;
                                        best_nf = nf;
                                        best_fx = fx;
                                        for (i = 0; i < nvars; i++)
                                                best_pt[i] = endpt[i];
                                }
                        }
                }
        }
```

Αποτελέσματα

Number of Threads	Execution Time (s)	Speed Up
1	58.617 (serial)	-
2	29.803	1.9668
4	15.565	3.7659
8	9.128	6.4217

OpenMP Task

Σε αυτήν την έκδοση, πρώτα δημιουργούμε μια παράλληλη περιοχή, όπου πολλαπλά νήματα θα εκτελούν την εργασία. Εκτελούμε την επανάληψη του βρόχου for με ένα μόνο νήμα. Στη συνέχεια, δημιουργούμε πολλαπλές διεργασίες όπου καθεμία εκτελείται από το διαθέσιμο νήμα. Εδώ OpenMP pragmas χρησιμοποιούνται για την παραλληλοποίηση αλγορίθμων που βασίζονται σε διεργασίες. Κάθε ερώτημα αντιμετωπίζεται ως ατομική διαδικασία και το pragma omp atomic χρησιμοποιείται για να διασφαλίσει ότι μια συγκεκριμένη λειτουργία μνήμης (συνήθως μια ενημέρωση σε μια κοινόχρηστη μεταβλητή) εκτελείται ατομικά ως μια ενιαία λειτουργία.

```
#pragma omp parallel
{
    // Only one process will run and create a task - each task is then run
       by the thread that is available
    #pragma omp single
    {
        for (trial = 0; trial < ntrials; trial++) {

            // Creating a task for execution of a thread worker
            // firstprivate -> keep the initial value of the variable in
               the parallel region, but a local copy is created for each
               thread
            // private -> keep the variable private to each thread, so
               that each thread can have it's own copy of the variable
```

```
            // shared -> keep the variable shared among all the threads,
               so that all the threads can access the variable
            #pragma omp task firstprivate(startpt, endpt) private(nt, nf,
               fx) shared(best_trial, best_nt, best_nf, best_fx, best_pt)
            {
                // it will then not produce the same as the serial
                   implementation
                /* starting guess for rosenbrock test function, search
                   space in [-2, 2) */
                srand48(trial);
                for (int i = 0; i < nvars; i++) {
                    startpt[i] = lower[i] + (upper[i]-lower[i])* drand48()
                       ;
                }


                int term = -1;
                mds(startpt, endpt, nvars, &fx, eps, maxfevals, maxiter,
                   mu, theta, delta,
                &nt, &nf, lower, upper, &term);
                #if DEBUG
                        printf("\n\n\nMDS %d USED %d ITERATIONS AND %d
                           FUNCTION CALLS, AND RETURNED\n", trial, nt, nf)
                           ;
                        for (i = 0; i < nvars; i++)
                            printf("x[%3d] = %15.7le \n", i, endpt[i]);

                        printf("f(x) = %15.7le\n", fx);
                #endif

                /* keep the best solution */
                if (fx < best_fx) {
                    // We will update the best solution based on the
                       best_fx value
                    #pragma omp critical
                    {
                        // recheck ensures that the entered thread has the
                            best result
                        if(fx < best_fx) {
                            best_trial = trial;
                            best_nt = nt;
                            best_nf = nf;
                            best_fx = fx;
                            for (int i = 0; i < nvars; i++)
                                best_pt[i] = endpt[i];
                        }
                    }
                }
            }

        }
    }
```

```
}
```

Αποτελέσματα

Number of Threads	Execution Time (s)	Speed Up
1	58.617 (serial)	-
2	28.374	2.0659
4	16.321	3.5915
8	9.178	6.3867

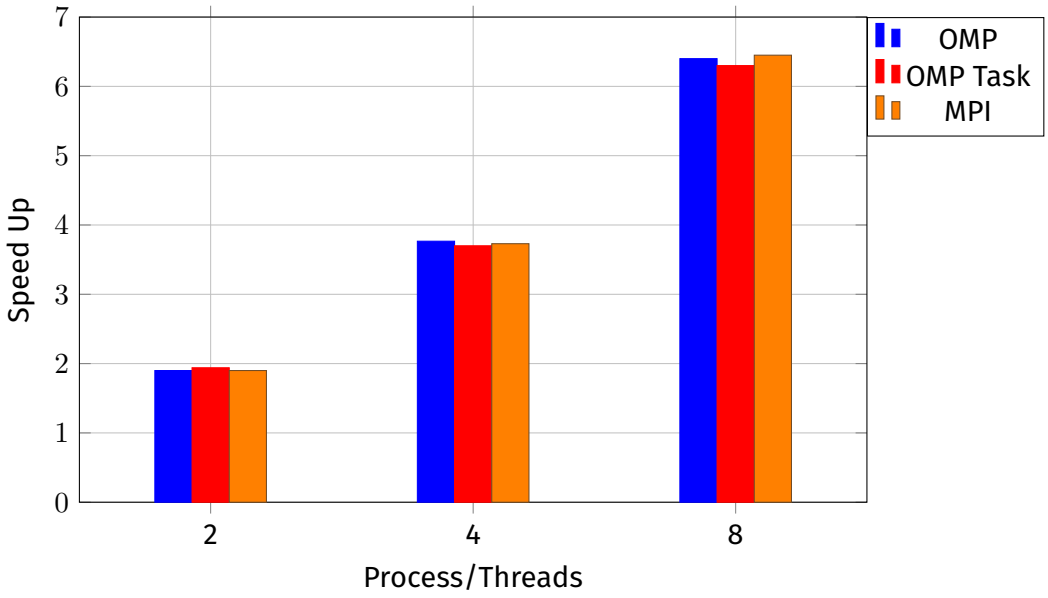
MPI

Στην υλοποίηση MPI, διαιρούμε σε πολλαπλές διεργασίες. Κάθε διεργασία παίρνει ένα κομμάτι των επαναλήψεων για να εργαστεί ανεξάρτητα. Ακολουθεί μια εξήγηση βήμα προς βήμα:
Αρχικοποίηση: Προσδιορισμός του συνολικού αριθμού δοκιμών και του αριθμού των διεργασιών.
Διαίρεση: Κάθε διεργασία υπολογίζει τους δείκτες έναρξης και λήξης για τις δοκιμές που θα χειριστεί.
Κατανομή Εργασίας: Κάθε διεργασία λειτουργεί στο τμήμα δοκιμών που της έχει ανατεθεί.
Συνδυασμός αποτελεσμάτων: Αφού όλες οι διεργασίες ολοκληρωθούν, τα αποτελέσματα συνδυάζονται για να βρεθεί η βέλτιστη λύση. Πιο συγκεκριμένα:

1. Διεργασία 0:
   Start = rank * (trial / size) = 0 * (64 /4) = 0
   End = (rank+1) * (trial / size) = (0+1) * (64 /4) = 16
   Έτσι, η διεργασία 0 θα λειτουργήσει για δοκιμές 0-15.
2. Διεργασία 1:
   Start = rank * (trial / size) = 1 * (64 /4) = 16
   End = (rank+1) * (trial / size) = (1+1) * (64 /4) = 32
   Έτσι, η διεργασία 1 θα λειτουργήσει για δοκιμές 16-31.
3. Διεργασία 2:
   Start = rank * (trial / size) = 2 * (64 /4) = 32
   End = (rank+1) * (trial / size) = (2+1) * (64 /4) = 48
   Έτσι, η διεργασία 2 θα λειτουργήσει για δοκιμές 32-47.
4. Διεργασία 3:
   Start = rank * (trial / size) = 3 * (64 /4) = 48
   End = (rank+1) * (trial / size) = (3+1) * (64 /4) = 64
   Έτσι, η διεργασία 3 θα λειτουργήσει για δοκιμές 48-63.

```
int start= rank*(ntrials/size);
        int end = (rank+1)*(ntrials/size);
        if(rank==size-1){
                end = ntrials;
        }

        for (trial = start; trial < end; trial++) ...
```



Σχήμα 1: Speed Up Graph (Serial time / Parallel Time)

III Παρατηρήσεις

Δεν υπάρχει μεγάλη διαφορά μεταξύ του OpenMP και OpenMP task, καθώς η διεργασία που παρέχεται δεν διαφέρει σημαντικά από την παράλληλη περιοχή OMP. Επιπλέον, οι εργασίες εκτελούν σχεδόν τον ίδιο αριθμό επαναλήψεων στη συνάρτηση. Γενικά οι διεργασίες στο OpenMP είναι ιδιαίτερα χρήσιμες όταν οι επαναλήψεις δεν είναι ομοιόμορφες, με αποτέλεσμα ορισμένα νήματα να χρειάζονται περισσότερο χρόνο για να ολοκληρωθούν από άλλα. Στην περίπτωσή μας, οι επαναλήψεις είναι σχεδόν πανομοιότυπες. Η εκτέλεση του προγράμματος με μεγαλύτερο αριθμό μεταβλητών και δοκιμών μπορεί να αποκαλύψει περισσότερες διαφορές.

Για μικρό αριθμό διαδικασιών, το MPI ήταν λιγότερο αποτελεσματικό. Ωστόσο, καθώς ο αριθμός των διεργασιών αυξάνεται, το MPI γίνεται πιο αποτελεσματικό σε σύγκριση με το OpenMP. Αυτό συμβαίνει επειδή η υλοποίηση MPI δεν απαιτεί κρίσιμα τμήματα, τα οποία χρειάζονται στο OpenMP για να αποφευχθούν τα race conditions κατά τη διαδικασία εύρεσης της καλύτερης λύσης. Καθώς ο αριθμός των νημάτων ή των διεργασιών αυξάνεται, το MPI τείνει να αποδίδει ελαφρώς καλύτερα λόγω της μειωμένης επιβάρυνσης των κρίσιμων τμημάτων.

IV Εκτέλεση

Στις επόμενες σελίδες ακολουθούν στιγμιότυπα εκτέλεσης

Serial

```
user@zorin:~/Desktop/parpro/code$ ls
Makefile              multistart_mds_omp        multistart_mds_omp_tasks.c  readme
multistart_mds_mpi    multistart_mds_omp.c      multistart_mds_seq          torczon.c
multistart_mds_mpi.c  multistart_mds_omp_tasks  multistart_mds_seq.c
user@zorin:~/Desktop/parpro/code$ make
gcc -Wall -O3 -fopenmp -DDEBUG -o multistart_mds_seq multistart_mds_seq.c torczon.c -lm
gcc -Wall -O3 -fopenmp -DDEBUG -o multistart_mds_omp multistart_mds_omp.c torczon.c -lm
gcc -Wall -O3 -fopenmp -DDEBUG -o multistart_mds_omp_tasks multistart_mds_omp_tasks.c torczon.c -lm
mpicc -Wall -O3 -fopenmp -DDEBUG -o multistart_mds_mpi multistart_mds_mpi.c torczon.c -lm
user@zorin:~/Desktop/parpro/code$ make run_serial
./multistart_mds_seq



MDS 0 USED 1250 ITERATIONS AND 10005 FUNCTION CALLS, AND RETURNED
x[  0] =   7.9659339e-01
x[  1] =   6.3535011e-01
x[  2] =   4.0208818e-01
x[  3] =   1.5764216e-01
f(x) =   5.3378104e-01
```

```
MDS 63 USED 1251 ITERATIONS AND 10005 FUNCTION CALLS, AND RETURNED
x[  0] =   7.9346268e-01
x[  1] =   6.3062366e-01
x[  2] =   3.9759651e-01
x[  3] =   1.5443142e-01
f(x) =   5.4342900e-01


FINAL RESULTS:
Elapsed time = 41.116 s
Total number of trials = 64
Total number of function evaluations = 640292
Best result at trial 13 used 1251 iterations, 10005 function calls and returned
x[  0] =   1.0194756e+00
x[  1] =   1.0393283e+00
x[  2] =   1.0802491e+00
x[  3] =   1.1671989e+00
f(x) =   8.3729430e-03
```

OpenMP

```
user@zorin:~/Desktop/parpro/code$ make run_omp
./multistart_mds_omp 8



MDS 24 USED 1249 ITERATIONS AND 10001 FUNCTION CALLS, AND RETURNED
x[  0] =  -5.8203195e-01
x[  1] =   3.5232862e-01
x[  2] =   1.3356043e-01
x[  3] =   1.8354589e-02
f(x) =   3.7003380e+00



MDS 16 USED 1251 ITERATIONS AND 10001 FUNCTION CALLS, AND RETURNED
x[  0] =   7.7794115e-01
x[  1] =   6.0609295e-01
x[  2] =   3.6768231e-01
x[  3] =   1.3066106e-01
f(x) =   6.0644217e-01
```

```
MDS 47 USED 1251 ITERATIONS AND 10005 FUNCTION CALLS, AND RETURNED
x[  0] =  -1.0433538e+00
x[  1] =   1.0984795e+00
x[  2] =   1.2107499e+00
x[  3] =   1.4677895e+00
f(x) =   4.2412206e+00


FINAL RESULTS:
Elapsed time = 4.733 s
Total number of trials = 64
Total number of function evaluations = 640288
Best result at trial 13 used 1251 iterations, 10005 function calls and returned
x[  0] =   1.0194756e+00
x[  1] =   1.0393283e+00
x[  2] =   1.0802491e+00
x[  3] =   1.1671989e+00
f(x) =   8.3729430e-03
```

OpenMP Task

```
user@zorin:~/Desktop/parpro/code$ make run_omp_tasks
./multistart_mds_omp_tasks 8


MDS 6 USED 1250 ITERATIONS AND 10005 FUNCTION CALLS, AND RETURNED
x[  0] =  -1.0093514e+00
x[  1] =   1.0287976e+00
x[  2] =   1.0628310e+00
x[  3] =   1.1311668e+00
f(x) =   4.0544690e+00


MDS 5 USED 1251 ITERATIONS AND 10005 FUNCTION CALLS, AND RETURNED
x[  0] =   7.8685832e-01
x[  1] =   6.1973752e-01
x[  2] =   3.8389006e-01
x[  3] =   1.4383947e-01
f(x) =   5.7090636e-01


MDS 63 USED 1251 ITERATIONS AND 10005 FUNCTION CALLS, AND RETURNED
x[  0] =   7.9346268e-01
x[  1] =   6.3062366e-01
x[  2] =   3.9759651e-01
x[  3] =   1.5443142e-01
f(x) =   5.4342900e-01


FINAL RESULTS:
Elapsed time = 4.713 s
Total number of trials = 64
Total number of function evaluations = 640292
Best result at trial 13 used 1251 iterations, 10005 function calls and returned
x[  0] =   1.0194756e+00
x[  1] =   1.0393283e+00
x[  2] =   1.0802491e+00
x[  3] =   1.1671989e+00
f(x) =   8.3729430e-03
```

MPI

```
user@zorin:~/Desktop/parpro/code$ make run_mpi
mpirun --oversubscribe -np 8 ./multistart_mds_mpi


MDS 56 USED 1249 ITERATIONS AND 10001 FUNCTION CALLS, AND RETURNED
x[  0] =   8.8658397e-01
x[  1] =   7.8612387e-01
x[  2] =   6.1809341e-01
x[  3] =   3.8010284e-01
f(x) =   2.0483580e-01


MDS 32 USED 1251 ITERATIONS AND 10005 FUNCTION CALLS, AND RETURNED
x[  0] =   7.5733578e-01
x[  1] =   5.7488266e-01
x[  2] =   3.3132332e-01
x[  3] =   1.0563308e-01
f(x) =   6.8869989e-01
```

```
MDS 55 USED 1251 ITERATIONS AND 10005 FUNCTION CALLS, AND RETURNED
x[  0] =   7.8716380e-01
x[  1] =   6.1986162e-01
x[  2] =   3.8452601e-01
x[  3] =   1.4546103e-01
f(x) =   5.6920267e-01


FINAL RESULTS:
Elapsed time = 4.100 s
Total number of trials = 64
Total number of function evaluations = 640292
Best result at trial 13 used 1251 iterations, 10005 function calls and returned
x[  0] =   1.0194756e+00
x[  1] =   1.0393283e+00
x[  2] =   1.0802491e+00
x[  3] =   1.1671989e+00
f(x) =   8.3729430e-03
```