Εργασία Παράλληλη Επεξεργασία 2023-2024

Παντελής Φλουρής, 1093507

Άγγελος Μενεγάτος, 1093426

Χρυσάφης Κολτσάκης, 1084671

Γιώργος Αμαξόπουλος, 1093311

Α. Εισαγωγή

Η εργασία αυτή αποσκοπεί στην παραλληλοποίηση της εφαρμογής multistart_mds_seq.c με χρήση OpenMP, OpenMP tasks και MPI. Στην περίπτωση των tasks, ζητείται η παραλληλοποίηση μέσα στην μέθοδο mds (torczon_tasks.c).

Αναμενόμενα αποτελέσματα: Η παραλληλοποίηση έχει ως αποτέλεσμα την μείωση του χρόνου κατά λίγο λιγότερο από n φορές, όπου n τα νήματα/διεργασίες, με την υλοποίηση με tasks να ειναι η γρηγορότερη και οι αλλες δυο να ειναι παρομοιας ταχύτητας.

Περιβάλλον συλλογής αποτελεσμάτων:

CPU: Ryzen 7 3700x 8c 16th

RAM: 16GB DDR4 3200MHz

OS: Fedora Linux 40

Β. Παράλληλες Υλοποιήσεις

OpenMP (multistart_mds_omp.c)

Για την παραλληλοποίηση του προγράμματος χρησιμοποιήσαμε την βιβλιοθήκη omp.h, και θέσαμε μεσα στην main τον αριθμό των threads. Δημιουργήσαμε ενα struct (calc) που χρησιμοποιείται για αποθήκευση των καλύτερων αποτελεσμάτων του καθε thread, ελαχιστοποιώντας ετσι την αναγκη για critical sections, και βελτιώνοντας την ταχύτητα. Την θέση του srand48 πήρε το erand48, το οποίο ειναι thread safe.

Η κυρια παραλληλοποίηση γίνεται μεσα στο for loop, οπου σε κάθε επανάληψη το κάθε νήμα ενημερώνει την firstprivate μεταβλητή του local_best, εαν αυτή ειναι καλύτερη απο την ηδη αποθηκευμένη, και επειτα ελεγχει εαν ειναι καλύτερη απο την global μεταβλητη best. Εαν ειναι, την αντικαθιστά μεσα σε ενα critical section.

Επίσης βαλαμε atomic στην ενημέρωση της global μεταβλητής funevals στην συναρτηση f.

Το κύριο πρόβλημα που αντιμετωπίσαμε σε αυτό το υποερώτημα ειχε να κανει με το τελικό αποτέλεσμα, οπου περίπου 1 στις 10 φορές ηταν διαφορετικό. Εν τέλει αυτο το πρόβλημα λυθηκε με την χρηση στατικού scheduling με 1 επανάληψη μεσα στο for loop.

OpenMP tasks (multistart_mds_omp_tasks.c && torczon_tasks.c)

- **multistart_mds_omp_tasks.c**
  Η υλοποίηση αυτού του παράλληλου κώδικα ηταν πολύ απλή και απαιτούσε μόνο μικρές αλλαγές απο το προηγούμενο υποερώτημα μεσα στο for loop.
  Η *#pragma parallel for* αντικαταστάθηκε με δυο εντολές:

  #pragma omp parallel - > Δημιουργία των νημάτων
  #pragma omp single nowait - > το for το τρέχει μονο ενα νήμα, τα αλλα περιμένουν καποιο task.

  Δημιουργείται μονο ενα task, πριν την εκτέλεση της mds, και ολοκληρώνεται πριν την εκτύπωση των αποτελεσμάτων. Μέσα στο task, περαν της αλλαγης οτι το buffer ειναι πλεον firstprivate, αφου το εχουμε αρικοποιήσει, δεν αλλαζει τιποτα απο το προηγούμενο υποερώτημα.

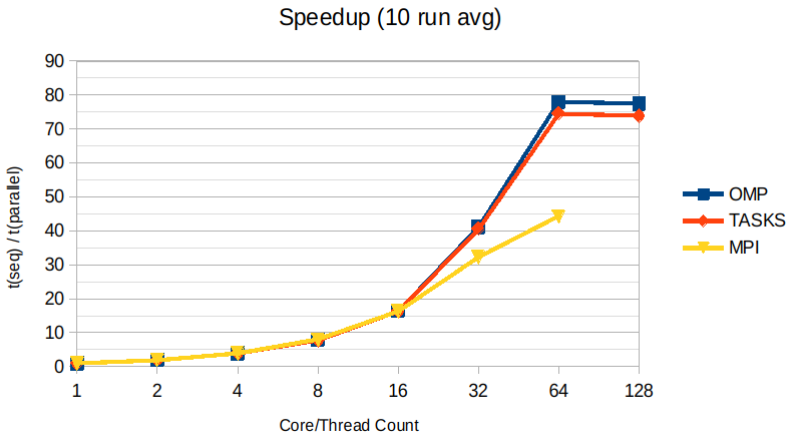
- **torczon_tasks.c**
  Σε αυτόν τον παράλληλο κώδικα οι παραλληλοποιήσεις ηταν πολυ περισσοτερες, λόγω των πολλών εμφωλευμένων for loops. Η παραλληλοποίηση γίνεται μεσα στο *while (terminate == 0 && iter < maxiter)*, καθώς εκει εχουμε την πλειοψηφία του φορτου εργασίας. Δημιουργούμε τα νήματα και συνεχίζουμε χρησιμοποιώντας ενα απο αυτά. Για κάθε εμφωλευμένο for loop δημιουργήσαμε ενα taskgroup, το οποίο περιέχει ενα task για το εξωτερικό loop και ενα για το εσωτερικό. Οι εγγραφές γινονται με χρηση atomic, και στην θέση των break οταν βρισκεται ο εκαστοτε στοχος, εχουμε χρησιμοποιήσει cancel taskgroup.

MPI (multistart_mds_mpi.c)

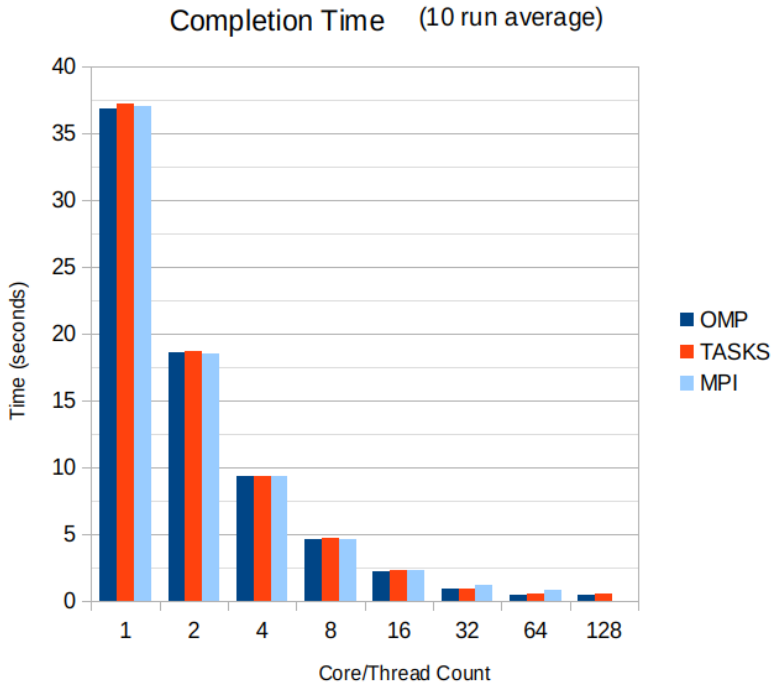
Για αυτό το υποερώτημα χρειαστήκαμε το struct των προηγούμενων υποερωτημάτων μαζι με την βιβλιοθηκη mpi.h.
Στον σειριακό κωδικα, στην αρχη της main, αρχικοποιήσαμε το MPI περιβαλλον και δημιουργήσαμε εναν νεο MPI τυπο δεδομένων για το struct calc.
Δημιουργήσαμε τοπικές μεταβλητές για τα ntrials, start ετσι ωστε ο καθε πυρήνας να εκτελεί μονο την δουλεία που του αναλογεί.
Μεσα στο for loop, υπολογιζεται η τοπική καλύτερη τιμή στον κάθε πυρήνα και αποθηκεύεται.
Οταν τελειώσει, η καλύτερη τοπική τιμή απο κάθε πυρήνα (περαν του 0) αποστελλεται στον 0 μεσω MPI_Send, MPI_Recv και χρήσης του νέου τύπου δεδομένων.
Στον πυρήνα 0 γινεται η συγκριση και καταλήγει στην καλύτερη λύση.
Τελος, προστείθονται τα τοπικά funevals μεσω MPI_Reduce, και τοποθετούμε barrier.
Ο πυρήνας 0 τυπώνει τα τελικά στοιχεία, αποδεσμεύει την μνημη για τον τυπο δεδομένων που δημιουργήσαμε, και κλείνει το περιβαλλον MPI.

Γ. Αποτελέσματα

Speedup (t(seq) / t)			
Core /Thread #	**OMP**	**TASKS**	**MPI**
1	0.999	0.99	0.994
2	1.978	1.97	1.985
4	3.954	3.933	3.955
8	7.951	7.822	7.991
16	16.45	16.337	16.372
32	41.182	40.654	32.311
64	77.943	74.623	44.269
128	77.392	73.854	DNF



Completion Time (10 run average)			
Core /Thread #	**OMP**	**TASKS**	**MPI**
1	36.815	37.166	37.021
2	18.591	18.669	18.528
4	9.302	9.352	9.3
8	4.627	4.704	4.605
16	2.236	2.252	2.248
32	0.893	0.905	1.138
64	0.472	0.493	0.831
128	0.475	0.498	DNF



Παραδείγματα λειτουργίας

multistart_mds_omp (8th)

```
paflou@fedora:~/Documents/CEID/Parallel-Programming-Practice/multistart$ ./multistart_mds_omp


 FINAL RESULTS:
 Elapsed time = 4.634 s
 Total number of trials = 64
 Total number of function evaluations = 640308
 Best result at trial 40 used 1250 iterations, 10005 function calls and returned
 x[  0] =   1.0074912e+00
 x[  1] =   1.0150409e+00
 x[  2] =   1.0303266e+00
 x[  3] =   1.0617248e+00
 f(x) =   1.2043904e-03
```

multistart_mds_omp_tasks (8th)

```
paflou@fedora:~/Documents/CEID/Parallel-Programming-Practice/multistart$ ./multistart_mds_omp_tasks


 FINAL RESULTS:
 Elapsed time = 4.698 s
 Total number of trials = 64
 Total number of function evaluations = 640308
 Best result at trial 40 used 1250 iterations, 10005 function calls and returned
 x[  0] =   1.0074912e+00
 x[  1] =   1.0150409e+00
 x[  2] =   1.0303266e+00
 x[  3] =   1.0617248e+00
 f(x) =   1.2043904e-03
```

multistart_mds_mpi (8c)

```
paflou@fedora:~/Documents/CEID/Parallel-Programming-Practice/multistart$ mpirun -n 8 ./multistart_mds_mpi


FINAL RESULTS:
Elapsed time = 4.544 s
Total number of trials = 64
Total number of function evaluations = 640308
Best result at trial 40 used 1250 iterations, 10005 function calls and returned
x[  0] =   1.0074912e+00
x[  1] =   1.0150409e+00
x[  2] =   1.0303266e+00
x[  3] =   1.0617248e+00
f(x) =   1.2043904e-03
```

Δ. Συμπεράσματα

Παρατηρούμε οτι η πιο γρήγορη υλοποίηση ειναι αυτή του πρώτου υποερωτήματος (multistart_mds_omp), με τις αλλες δυο υλοποιήσεις να ειναι σε αντιστοιχη ταχυτητα για τιμές πυρηνων / νημάτων εως 16 (απο 32 και ανω το MPI μενει πίσω λογω ελλειψης πυρήνων στο συστημα).

Να σημειωθεί οτι η ιδια υλοποίηση εχει και την καλύτερη χρονοβελτίωση απο ολες.

Βλέπουμε οτι η υλοποίηση με tasks ειναι πιο αργή απο την υλοποίηση με απλο parallel for, αν και εχουμε παραλληλοποιήσει και την μεθοδο mds.

Υποθέτουμε οτι αυτο συμβαινει λογω ενος συνδυασμου overhead δημιουργίας και χρησης tasks και λογω false sharing μεσα στους κλειστούς εμφολευμενους βρογχους της μεθοδου mds. Μάλιστα η παραλληλοποίηση της mds επιβράδυνε ελαφρώς το προγραμμα.

Καθώς το MPI χρησιμοποιεί πυρήνες αντι για νήματα, η χρονοβελτίωση του ειναι σημαντικά χειροτερη, και για 128 πυρήνες το προγραμμα εληξε μετα απο 100 δευτερόλεπτα.

Συμπεραίνουμε οτι η παραλληλοποίηση με χρήση της οδηγίας parallel for ειναι η πιο γρήγορη για την συγκεκριμένη περίπτωση, καθώς εχει τα καλύτερα πειραματικά αποτελέσματα και η υλοποίηση της ειναι πιο απλη.