Παράλληλα Συστήματα – Εργασία 2η

Εισαγωγή

Για την εκπόνηση της εργασίας πραγματοποιήθηκε παραλληλοποίηση του αλγορίθμου kNN-Search με συνδυασμένη χρήση mpi και openmp, τόσο με τη χρήση blocking επικοινωνίας, αλλά και non-blocking. Τα αποτελέσματα που παρουσιάζονται πάρθηκαν από το hellasgrid. Ο κώδικας της εργασίας βρίσκεται στο «https://github.com/NikosPaschos/kNN-Search-MPI-OpenMP».

Σειριακός Αλγόριθμος

Για την σειριακή εκδοχή του αλγορίθμου, η οποία παραλληλοποιήθηκε στη συνέχεια, υλοποιήθηκαν τα παρακάτω:

- **read_data.c:** Στο αρχείο αυτό υπάρχουν οι κατάλληλες συναρτήσεις για την ανάγνωση των δεδομένων από τα αρχεία *.bin. Για την εξαγωγή των αρχείων αυτών, έγινε χρήση συναρτήσεων Matlab με σκοπό την εύκολη ανάγνωση των δεδομένων. Η συνάρτηση read_data() επιστρέφει έναν double* και έχει διαβάσει τα δεδομένα του αρχείου με την εξής λογική: Ο πίνακας είναι 1D και ανά 30 στοιχεία δημιουργούνται οι 30 διαστάσεις του κάθε στοιχείου. Τα 30 δηλαδή πρώτα στοιχεία αφορούν τις διαστάσεις [1,2,…,30] του σημείου με αύξων αριθμό 1 κοκ. Η συνάρτηση read_labels() επιστρέφει έναν double* επίσης, όμως κάθε στοιχείο είναι ένας double που περιέχει το label του στοιχείου με index i.
- **calc_dist():** Η συνάρτηση αυτή δέχεται σαν όρισμα 2 double* που δείχνουν στην αρχή των σημείων των εκάστοτε πινάκων προς τον υπολογισμό της απόστασης. Στην κλήση της συνάρτησης έγινε χρήση pointer arithmetic για τον υπολογισμό της κατάλληλης θέσης. Επιστρέφει έναν double που περιέχει το τετράγωνο της απόστασης των 2 σημείων. Αναλυτικότερα, υπολογίζει τη διαφορά των συντεταγμένων για κάθε ζευγάρι διαστάσεων και στη συνέχεια την υψώνει στο τετράγωνο. Αυξάνει στη συνέχεια το αποτέλεσμα dist που συμβολίζει την απόσταση, και συνεχίζει με την επόμενη διάσταση μέχρι να συμπληρωθούν 30. Δεν έγινε χρήση της βιβλιοθήκης math.h επειδή μετά από δοκιμές ο κώδικας ήταν πιο αργός. Έτσι παραλείπεται η pow(). Εδώ σημειώνεται ότι δεν έγινε υπολογισμός της ρίζας του αριθμού αυτού, που αποτελεί τον αυστηρό μαθηματικό ορισμό της Ευκλείδειας απόστασης 2 σημείων, επειδή μας ενδιαφέρει να την συγκρίνουμε με άλλες αποστάσεις, άρα δεν υπάρχει πρακτική διαφορά. Έτσι επιτυγχάνεται καλύτερη απόδοση.
- **update_neighbors():** Δέχεται σαν όρισμα έναν πίνακα που περιέχει τις μέχρι τώρα υπολογισμένες ελάχιστες αποστάσεις σε αύξουσα σειρά, έναν πίνακα που περιέχει τα αντίστοιχα index σε σχέση με τα 60.000 σημεία, την υπολογισμένη απόσταση, τον αριθμό των γειτόνων που υπολογίζει ο αλγόριθμος καθώς και το index του σημείου. Εκτελούνται κατάλληλοι έλεγχοι μόνο για το πρώτο και το τελευταίο σημείο του πίνακα αποστάσεων. Αρχικά ελέγχεται εάν η δοθείσα απόσταση είναι μικρότερη από το τελευταίο στοιχείο του

Πάσχος Νικόλαος 8360
npaschos@ece.auth.gr

πίνακα, ενώ εάν δεν είναι τότε ελέγχει και την πρώτη θέση. Αυτό συμβαίνει γιατί μετά την εισαγωγή ενός στοιχείο ως ελάχιστο, πραγματοποιείται SelectionSort στον πίνακα των αποστάσεων και όταν πραγματοποιείται κάποιο swap αλλάζουν και οι ίδιες θέσεις στον πίνακα των index ώστε να υπάρχει αντιστοιχία αποστάσεων με index. SelectionSort εκτελείται μόνο εάν εισαχθεί ένα σημείο, και μόλις εισαχθεί σταματάει η εκτέλεση του for statement που εξετάζει τα σημεία στον πίνακαneighbors_matrix.

- **SelectionSort():** Εκτελείται SelectionSort στον πίνακα που δίνεται σαν όρισμα στην πρώτη θέση, ελέγχει n στοιχεία, 3^{ο} όρισμα της συνάρτησης, και δέχεται και τον πίνακα neighbors_index σαν 2^{ο} όρισμα ώστε να μετακινήσει και τα στοιχεία του neighbors_index κατάλληλα, ώστε να συνεχίσει να υπάρχει αντιστοιχία μεταξύ neighbors_matrix (αποστάσεις) και neighbors_index (index σημείου). Ως βοηθητική συνάρτηση χρησιμοποιείται η swap() που δέχεται 2 pointers για να αλλάξει τα δεδομένα μεταξύ τους.
-

Έλεγχος ορθότητας αποτελεσμάτων

Για έλεγχο ορθότητας των αποτελεσμάτων του αλγορίθμου υλοποιήθηκε η συνάρτηση

void test(int** neighbors_index, int world_size, int world_rank),

η οποία για κάθε γραμμή του neighbors_index βρίσκει με ποιο label είναι αυτό που εμφανίζεται ποιο συχνά για το κάθε σημείο. Δηλαδή, ελέγχει τον πίνακα neighbors_index και στη συνέχεια για κάθε index αποθηκεύει το label του σημείου αυτού. Στη συνέχεια υποθέτει ότι το classification του κάθε σημείου αποτελεί η πιο συχνή εμφάνιση κλάσης. Τέλος ελέγχει εάν το σημείο ανήκει όντως σε αυτή την κλάση από τον πίνακα labels. Έτσι παράγεται ένα ποσοστό επιτυχίας για τον αλγόριθμο. Κάθε σημείο που έχει ίδιο προβλεπόμενο label με το δοθέν, θεωρείται ως επιτυχία.

Παραλληλοποίηση BLOCKING

Για την ανάγνωση δεδομένων σε κάθε process, έγινε υλοποίηση μιας ακόμα συνάρτησης, της read_data_custom() που περιέχεται στο read_data.c αρχείο. Επιστρέφει έναν double* δεν δείχνει σε όλο τον πίνακα των δεδομένων, αλλά έχει διαβάσει αποσπασματικά το αρχείο .bin μόνο για LEN/world_size στοιχεία, όπου LEN ο αριθμός των στοιχείων.

Για την blocking εκδοχή του αλγορίθμου ακολουθήθηκε η εξής λογική: Αρχικοποιήθηκε το περιβάλλον μέσω της MPI_Init(NULL,NULL); και αποθηκεύτηκαν τα world_size και world_rank μέσω των MPI_Comm_size(MPI_COMM_WORLD, &world_size); και MPI_Comm_rank(MPI_COMM_WORLD, &world_rank); αντίστοιχα. Έγινε χρήση MPI_Barrier(MPI_COMM_WORLD); για την σωστή εκκίνηση χρονομέτρησης του αλγορίθμου από το process με world_rank==0. Στη συνέχεια υπολογίστηκαν οι αποστάσεις και ανανεώθηκαν κατάλληλα τα αποτελέσματα στους πίνακες για τα κομμάτια του πίνακα που έχει διαβάσει το κάθε process. Μετά την ολοκλήρωση της διαδικασίας αυτής πραγματοποιήθηκε επικοινωνία των processes με σκοπό την ανταλλαγή των «κομματιών» του αρχικού πίνακα με τα σημεία. Για την αποφυγή deadlock τα process με world_rank%2==0 πρώτα στέλνουν και μετά

λαμβάνουν δεδομένα, ενώ εκείνα με world_rank%2==1 ανάποδα. Στη συνέχεια υπολογίστηκαν οι νέες αποστάσεις και ανανεώθηκαν οι πίνακες, όπου χρειάζεται, με τα νέα σημεία και index. Αυτό υλοποιείται σε for statement όσες φορές χρειάζεται ώστε κάθε process να έχει διαβάσει και συγκρίνει στο τέλος όλα τα «κομμάτια» του πίνακα με το «κομμάτι» που είχε διαβάσει αρχικά. Μετά την ολοκλήρωση της διαδικασίας αυτής, γίνεται χρήση ενός ακόμα MPI_Barrier(MPI_COMM_WORLD); και το process με world_rank==0 τυπώνει τον χρόνο εκτέλεσης. Υπολογίζεται στη συνέχεια η ακρίβεια του αλγορίθμου για κάθε process και ολοκληρώνεται ο αλγόριθμος με χρήση MPI_Finalize();. Για την αποστολή των δεδομένων έγινε χρήση της

MPI_Send(data_send,LEN/world_size *sizeof(double),MPI_DOUBLE,send_dest,0,MPI_COMM_WORLD);

ενώ για το receive της

MPI_Recv(data_proc,LEN/world_size *sizeof(double),MPI_DOUBLE,rec_from,0,MPI_COMM_WORLD, MPI_STATUS_IGNORE);.

Παραλληλοποίηση NON-BLOCKING

Για την ανάγνωση των δεδομένων ακολουθήθηκε η διαδικασία που περιγράφηκε παραπάνω, όπως και η αρχικοποίηση και κλείσιμο του mpi environment.

Για την σωστή εκτέλεση της λογικής non-blocking επικοινωνίας, άλλαξε η σειρά εκτέλεσης των tasks που περιγράφηκαν στο blocking κομμάτι. Μετά την ανάγνωση των δεδομένων ξεκίνησε η αποστολή και λήψη δεδομένων μεταξύ των processes. Στη συνέχεια πραγματοποιήθηκε ο υπολογισμός αποστάσεων και ενημέρωση των πινάκων. Ο υπολογισμός έγινε σε διαφορετικό pointer και πίνακα από αυτόν στον οποίο διαβάζονται και στέλνονται δεδομένα. Μετά την ολοκλήρωση του υπολογισμού έγινε χρήση

MPI_Wait(&send_request,&status);

MPI_Wait(&recv_request,&status);

για να ολοκληρωθεί η διαδικασία αποστολής και λήψης δεδομένων. Τέλος έγινε κατάλληλη αλλαγή των pointers στους πίνακες ώστε να συνεχίσει η διαδικασία ομαλά στις επόμενες επαναλήψεις. Η επικοινωνία επιτεύχθηκε με τη χρήση των συναρτήσεων

MPI_Isend(data_send,LEN/world_size *sizeof(double),MPI_DOUBLE,send_dest,0,MPI_COMM_WORLD, &send_request);

για την αποστολή δεδομένων, ενώ για την λήψη

MPI_Irecv(tmpnb,LEN/world_size *sizeof(double),MPI_DOUBLE,rec_from,MPI_ANY_TAG,MPI_COMM_WORLD, &recv_request);.

Πάσχος Νικόλαος 8360
npaschos@ece.auth.gr

Συνδυασμένη χρήση με OpenMP

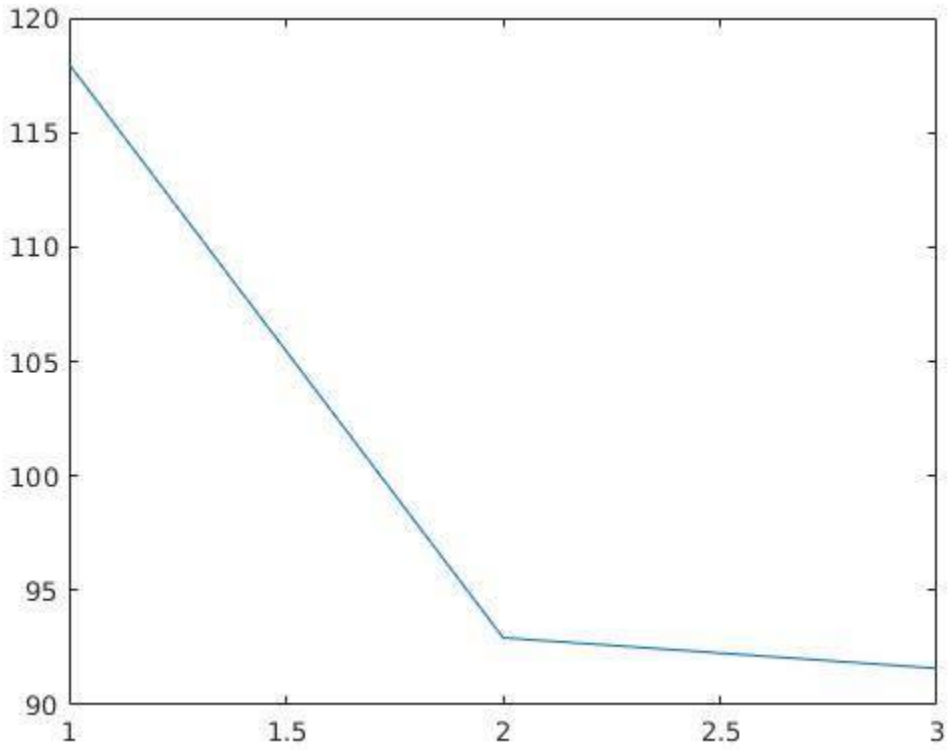
Και στις 2 υλοποιήσεις πραγματοποιήθηκε συνδυασμένη χρήση mpi και openmp για την επιτάχυνση των υπολογισμών. Εφαρμόσθηκε μόνο στο κομμάτι υπολογισμού των αποστάσεων και μόνο στην εξωτερική επανάληψη αφού μόνο εκεί παρατηρήθηκε ανεξαρτησία δεδομένων. Έγινε χρήση μέσω της εντολής

#pragma omp parallel for num_threads(4) default(shared) private(i)

όπου i ο δείκτης της εξωτερικής επανάληψης.

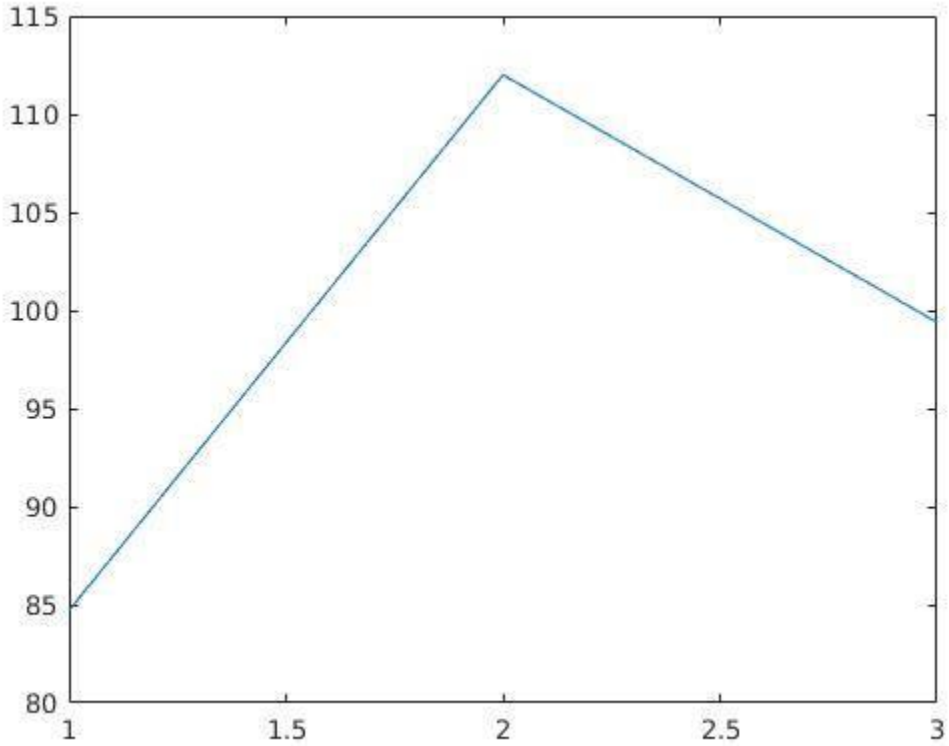
Παρουσίαση Αποτελεσμάτων

- Blocking



Πάσχος Νικόλαος 8360
npaschos@ece.auth.gr

- NON-BLOCKING



Σχολιασμός Αποτελεσμάτων

Παρατηρώ πως οι χρόνοι της blocking και non-blocking υλοποίησης είναι σχεδόν ίδιοι. Αυτό οφείλεται στο γεγονός ότι οι επικοινωνίες στο hellasgrid γίνονται σε μεγάλες ταχύτητες. Ακραίες τιμές παρουσιάζονται πιθανότατα λόγω φόρτου της συστοιχίας. Σε κάθε περίπτωση τα αποτελέσματα κυμαίνονται στο διάστημα [85,117]. Συγκριτικά με το σειριακό αλγόριθμο αυτό αποτελεί σημαντική βελτίωση, αφού εκτελείται σε 428 δευτερόλεπτα κατά μέσο όρο.

Πάσχος Νικόλαος 8360
npaschos@ece.auth.gr