ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ - ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ

ΑΛΓΟΡΙΘΜΙΚΕΣ ΤΕΧΝΙΚΕΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

ΑΝΑΦΟΡΑ ΠΡΩΤΗΣ ΕΡΓΑΣΙΑΣ

ΟΜΑΔΑ ΑΝΑΠΤΥΞΗΣ:

- ΝΙΚΟΛΑΣ ΠΑΤΕΡΑΣ – **ΜΠΣΠ21043**
- ΒΑΣΙΛΕΙΟΣ ΖΑΡΤΗΛΑΣ ΠΑΠΑΧΑΡΑΛΑΜΠΟΥΣ – **ΜΠΣΠ21015**

Πειραιάς, Ιανουάριος 2022

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

1. ΕΙΣΑΓΩΓΗ

Σκοπός της παρούσας υπολογιστικής εργασίας είναι η χρήση ενός αλγόριθμου χρησιμοποιώντας τις παράλληλες αλγοριθμικές τεχνικές και την υλοποίησή τους σε συστήματα κατανεμημένης μνήμης. Συγκεκριμένα, γίνεται υλοποίηση του αλγόριθμου Canopy με χρήση του προγραμματιστικού μοντέλου MPI σε C++ συγκεκριμένα την υλοποίηση MPICH. Στην συνέχεια θα αναφέρουμε τους βασικούς ορισμούς και ακολούθως θα γίνει τεκμηρίωση κώδικα.

1.1. ΑΛΓΟΡΙΘΜΟΣ CANOPY

Ο αλγόριθμος ομαδοποίησης Canopy είναι ένας αλγόριθμος προ-ομαδοποίησης χωρίς επίβλεψη. Ο αλγόριθμος παρέχει έναν πολύ απλό, γρήγορο και ακριβή τρόπο για την εκχώρηση των αντικειμένων σε ομάδες. Ο αλγόριθμος κάνει χρήση των διανυσμάτων στα οποία έχουν μοντελοποιηθεί τα δεδομένα και υπολογίζει την απόσταση μεταξύ τους σύμφωνα με κάποιο από τα μέτρα απόστασης. Επίσης δέχεται σαν είσοδο και δύο τιμές T1 και T2, για τις οποίες ισχύει η σχέση T1 > T2, οι οποίες αποτελούν τιμές κατωφλιών απόστασης. Στη συνέχεια παρουσιάζονται τα βήματα που ακολουθούνται κατά την εκτέλεση του αλγορίθμου [1].

- **Βήμα 1**: Η διαδικασία ομαδοποίησης ξεκινά με την τυχαία επιλογή ενός διανύσματος από τα συνολικά δεδομένα και τη δημιουργία μιας πρώιμης ομάδας (canopy) που αποτελείται από αυτό.
- **Βήμα 2**: Εξετάζεται η απόσταση των υπόλοιπων διανυσμάτων από το αρχικά επιλεγόμενο και αν αυτή είναι μικρότερη της τιμής T1 τότε αυτά εκχωρούνται στην ομάδα του.
- **Βήμα 3**: Ελέγχεται αν η τιμή της απόστασης, για ένα διάνυσμα, είναι μικρότερη και από την τιμή T2, τότε αυτό αφαιρείται τελείως από το σύνολο των υπό εξέταση δεδομένων, ώστε να αποφευχθεί η περαιτέρω επεξεργασία του.
- **Βήμα 4**: Επιστροφή στο Βήμα 1 και επανάληψη της διαδικασίας δημιουργώντας ένα σύνολο από ομάδες (canopies) όπου η κάθε μία αποτελείται από ένα ή περισσότερα στοιχεία.
- **Βήμα 5**: Ο αλγόριθμος τερματίζει όταν όλα τα στοιχεία αφαιρεθούν από το αρχικό σύνολο.

1.2. MPI (MESSAGE PASSING INTERFACE)

Το MPI αποτελεί ένα πρωτόκολλο επικοινωνίας που χρησιμοποιείται για τον προγραμματισμό παράλληλων υπολογιστικών συστημάτων. Υποστηρίζει και τα δύο μοντέλα επικοινωνίας που μπορούν να εφαρμοστούν σε αυτά τα συστήματα, τα οποία είναι το μοντέλο επικοινωνίας σημείο-προς-σημείο (point-to-point communication) και το συλλογικό μοντέλο επικοινωνίας (collective communication). Στόχος του MPI αποτελεί η επίτευξη υψηλής απόδοσης, κλιμάκωσης και φορητότητας και μέχρι και σήμερα αποτελεί κυρίαρχο μοντέλο στον τομέα του [2].

2. ΥΛΟΠΟΙΗΣΗ

Στην τεκμηρίωση κώδικα τα εξής χρώματα συμβολίζουν τα εξής:

➔ Συνάρτηση

➔ Μεταβλητή

Καταρχάς, αρχίζουμε με την συνάρτηση **main**/2 όπου η κάθετος συμβολίζει τον αρ. των ορισμάτων της συνάρτησης και καλούμε την συνάρτηση **MPI_Init()** της βιβλιοθήκης **MPI** για να ξεκινήσουμε τον υπολογισμό. Τοποθετούμε τις τιμές NULL στα ορίσματα της διότι παίρνουμε δύο ορίσματα από την **main** που δεν θα χρειαστούμε στην συγκεκριμένη περίπτωση. Στην συνέχεια αρχίζουμε το χρονόμετρο για να μετρήσουμε τον χρόνο εκτέλεσης του προγράμματος μας μαζί με το **srand**/1 που εκτελείται στην αρχή για να αρχικοποιηθεί το PRNG (Pseudo-Random Number Generator) το οποίο δημιουργεί μια ντετερμινιστική ακολουθία αριθμών. Στην συνέχεια με την εντολή assert η οποία θα τερματίσει το πρόγραμμα εάν δεν ισχύει η σχέση T1 > T2, όπου T1 και T2 αποτελούν τιμές κατωφλιών απόστασης.

Οι συναρτήσεις **MPI_Comm_rank**(MPI_COMM_WORLD, &world_rank) και **MPI_Comm_size**(MPI_COMM_WORLD, &world_size) που βρίσκονται στην main δίνουν το ID της διεργασίας και το πλήθος των διεργασιών αντίστοιχα. Το rank γενικά συμβολίζει το rank του worker οπόταν ο πρώτος worker έχει rank 0 και ο δεύτερος rank 1, κλπ. Υπάρχουν κομμάτια κώδικα που δεν θέλουμε να εκτελεστούν από όλους τους workers οπόταν απλά κάνουμε έλεγχο εάν το rank είναι 0 τότε θα είναι ο πρώτος worker γι' αυτό έτσι θα εκτελεστεί μόνο από τον πρώτο και όχι από τους άλλους N-ιστούς workers.

2.1. ΓΕΝΝΗΤΡΙΑ ΑΡΙΘΜΩΝ

το πρώτο βήμα ήταν να δημιουργήσουμε την γεννήτρια αριθμών που θα παράξει τα σημεία. Η συνάρτηση **generate_points**/2 παίρνει ένα διάνυσμα τύπου «Point» σε δείκτη (Το σύμβολο «*» είναι ο τελεστής έμμεσης κατεύθυνσης και κάθε φορά που χρησιμοποιείται, υποδεικνύει ότι η μεταβλητή δίπλα του είναι ένας δείκτης που περιέχει τη διεύθυνση μιας άλλης μεταβλητής.) αλλά ταυτόχρονα περιέχει και το σύμβολο «**&**» που σημαίνει ότι η μεταβλητή μεταβιβάζεται με αναφορά. Επίσης, στο δεύτερο όρισμα, ζητά το πλήθος των δειγμάτων που θα δημιουργήσουμε από την μεταβλητή το οποίο το ορίζουμε εμείς στην αρχή του προγράμματος στην παρούσα περίπτωση αν και θα δοκιμάσουμε διάφορες τιμές στην συνέχεια. Στην main αρχικοποιούμε το διάνυσμα «**all_points**» με τύπο τον δείκτη της κλάσης «**Point**» και αφού το rank του παρόν worker είναι 0 θα αρχίσουμε να γεμίζουμε το διάνυσμα με τα σημεία. Να σημειωθεί ότι ο χρόνος που απαιτείται για την εκτέλεση αυτής της διαδικασίας αφαιρείται από τον χρόνο που μετριέται για εκτέλεση της παράλληλης διαδικασίας όπως αναφέρει η εκφώνηση της εργασίας. Με την συνάρτηση **MPI_Wtime()**/0 μπορούμε να μετρήσουμε τον χρόνο που θα χρειαστεί και μετά να τον αφαιρέσουμε.

Ο ψευδοκώδικας της συνάρτησης είναι ο εξής,

Αλγόριθμος: Γεννήτρια Αριθμών
Είσοδος: Σημεία και συνολικός αρ. σημείων. **Έξοδος:** Τυχαία σημεία.
1. Άρχισε με τον αρ. των σημείων S. 2. Επανέλαβε μέχρι να φτάσεις στον αρ. των σημείων: 3. Δημιούργησε ένα πίνακα V διάστασης D ανάλογα με τα σημεία. 4. Επανέλαβε μέχρι να φτάσεις στον αρ. των διαστάσεων D: 5. Τοποθέτησε ένα τυχαίο σημείο μέσα στον πίνακα V κάθε φορά στην θέση που 6. βρίσκεσαι.

2.2. ΑΛΓΟΡΙΘΜΟΣ CANOPY ΚΑΙ ΣΥΣΤΑΔΟΠΟΙΗΣΗ

Στην κλάση Canopy [class Canopy{}] βρίσκεται ότι αφορά τον αλγόριθμό όπου θα βοηθήσει στην υλοποίηση του.

Αρχικά έχουμε μία μέθοδο **Canopy**/1 όπου δέχεται κάθε φορά το εκάστοτε κέντρο (**center**), η μέθοδος αυτή καταγράφει τα σημεία που είναι κεντρικά (**center**), δηλαδή χρησιμοποιούνται για την συσταδοποίηση. Τα σημεία καταχωρούνται σε ένα vector μέσω της μεθόδου **push_back**/1 όπου αποτελεί μέρος του διανύσματος (vector) και προσθέτει κάθε φόρα ένα σημείο στο τέλος του διανύσματος. Η ίδια λειτουργεία που περιγράψαμε συμβαίνει και την **add_point**/1.

Μέσω της μεθόδου **get_data_points**/0 παίρνουμε τα σημεία που καταχωρήσαμε μέσα στο διάνυσμα.

Μέσω της **printCenter**/0 τυπώνουμε τα κέντρα τα οποία χρησιμοποιήθηκαν για την συσταδοποίηση.

Μέσω της **printElements**/0 τυπώνουμε τα σημεία που καταχωρήθηκαν σε κάθε συστάδα.

Τα κύρια μέρη του αλγορίθμου βρίσκονται στην συνάρτηση **canopy_mpi**/1, η συνάρτηση φυσικά παίρνει όλα τα σημεία ως είσοδο (**all_points**) τα οποία δημιουργήθηκαν μέσω της γεννήτριας παραγωγής τυχαίων σημείων (**generate_points**/1*)* και επιστρέφει τους canopies που δημιουργήθηκαν μέσα σε ένα διάνυσμα τύπου **Canopy** (*vector<Canopy>*). Στη συνέχεια παραθέτουμε ψευδοκώδικα για την καλύτερη κατανόηση της διαδικασίας μέσω αυτού αντί της πραγματικής γλώσσας προγραμματισμού.

Ο ψευδοκώδικας του αλγόριθμου βρίσκεται στην επόμενη σελίδα.

Αλγόριθμος: Canopy
Είσοδος: Πλήθος σημείων που δημιουργήθηκαν. **Έξοδος:** Canopies.
1. Μέχρι το σύνολο των δεδομένων **δεν** είναι κενό: 2. Αναζήτηση ενός τυχαίου σημείου για κέντρο. 3. Δημιουργία ενός νέου canopy με το τυχαίο σημείο. 4. Αφαίρεση κέντρου από τα διαθέσιμα σημεία για να μην επιλεχθεί ξανά. 5. Δημιουργία vector για καταχώρηση σημείων προς αφαίρεση εφόσον 6. συσταδοποιηθούν. 7. Για κάθε σημείο που βρίσκεται στο σύνολο των δεδομένων: 8. Υπολόγισε την ευκλείδεια απόσταση. 9. Εάν η ευκλείδεια απόσταση είναι μικρότερη του T1: 10. Αποθήκευση του σημείου στον canopy. 11. Εάν η ευκλείδεια απόσταση είναι μικρότερη και του T2 **: 12. Τότε σημείωση του σημείου για αφαίρεση από το σύνολο των δεδομένων. 13. Για κάθε σημείο του vector: καταχώρηση σημείων προς αφαίρεση: 14. Τότε αφαίρεση σημείου από σύνολο δεδομένων. 15. Προσθήκη θόλου στο διάνυσμα καταχώρησης θόλου

*Εφόσον η ευκλείδεια απόσταση είναι μικρότερη από το T1 (και T1 > T2) τότε το σημείο δεν βρίσκεται αρκετά κοντά στο κέντρο και μπορεί να ανήκει και σε άλλο θόλο, έτσι τοποθετούμε [**new_canopy.add_point**/1] απλά το σημείο στον θόλο χωρίς να το αποθηκεύσουμε στο διάνυσμα προς αφαίρεση από το σύνολο των δεδομένων μας.

Εφόσον όμως η ευκλείδεια απόσταση είναι μικρότερη **κι από το T2 τότε θα προσθέσουμε το σημείο στο διάνυσμα προς αφαίρεση από το σύνολο των δεδομένων μας. Το σημείο έχει προστεθεί στον θόλο από τον πρώτο έλεγχο (if) εφόσον εξυπακούεται ότι εάν είναι μικρότερη η ευκλείδεια απόσταση του T2 είναι σίγουρα και μικρότερη του T1 (ευκλείδεια απόσταση < T2 < T1).

Σημείωση: Τοποθετήσαμε τα σημεία μας μέσα σε ένα unordered_set το οποίο είναι ένας πίνακας κατακερματισμού όπου τα κλειδιά του κατακερματίζονται σε δείκτες (στον πίνακα κατακερματισμού) έτσι ώστε η εισαγωγή τους να γίνεται πάντα τυχαία. Έτσι όταν παίρνουμε τα σημεία μας πίσω θα γίνεται η επιλογή τυχαία.

2.3. MPI_SCATTERV

Η συνάρτηση **MPI_Scatterv**/9 που βρίσκεται στην συνάρτηση **canopy_mpi**/1 διασκορπίζει ένα buffer σε μέρη, σε όλες τις διεργασίες σε έναν επικοινωνητή.

Οι παράμετροι της συνάρτησης ορίζονται ως εξής:

- buffer_send: Η προσωρινή μνήμη που περιέχει τα δεδομένα προς αποστολή από τη κύρια διαδικασία. Για τις μη-κύριες διεργασίες, οι παράμετροι αποστολής όπως αυτή αγνοούνται.
- counts_send: Ένας πίνακας που περιέχει τον αριθμό των στοιχείων προς αποστολή σε κάθε διεργασία, όχι τον συνολικό αριθμό στοιχείων στο buffer αποστολής. Για τις μη-κύριες διεργασίες, οι παράμετροι αποστολής όπως αυτή αγνοούνται.
- displacements: Ένας πίνακας που περιέχει τη μετατόπιση για εφαρμογή στο μήνυμα που αποστέλλεται σε κάθε διεργασία. Οι μετατοπίσεις εκφράζονται σε αριθμό στοιχείων, όχι σε byte. Για τις μη-κύριες διεργασίες, οι παράμετροι αποστολής όπως αυτή αγνοούνται.
- datatype_send: Ο τύπος ενός στοιχείου προσωρινής αποθήκευσης αποστολής. Για τις μη-κύριες διεργασίες, οι παράμετροι αποστολής όπως αυτή αγνοούνται.
- buffer_recv: Το buffer στο οποίο αποθηκεύονται τα δεδομένα που αποστέλλονται.
- count_recv: Ο αριθμός των στοιχείων στο buffer λήψης.
- datatype_recv: Ο τύπος ενός στοιχείου buffer λήψης.
- root: Η κατάταξη της κύριας διαδικασίας, η οποία θα αποστείλει τα δεδομένα για διασπορά.
- communicator: Ο επικοινωνητής στον οποίο λαμβάνει χώρα η διασπορά.

Στην δική μας υλοποίηση MPI η Scatterv έχει τα εξής ορίσματα:

```
1.  MPI_Scatterv(data, scatter_counts, scatter_displs, MPI_FLOAT,
2.                rec_buff, rec_buff_cnt, MPI_FLOAT,
3.                0, MPI_COMM_WORLD);
```

- data: Ο χώρος που βρίσκονται τα δεδομένα που θα σταλούν. Ο χώρος δηλώνεται μέσω της εντολής,

  ```
  1.  float* data = NULL;
  ```

  Είναι ένα διάνυσμα και βρίσκεται στον επεξεργαστή root. Το μέγεθος του διανύσματος data είναι το γινόμενο ανάμεσα στο πλήθος των σημείων που θα δημιουργηθούν και το πλήθος των διαστάσεων κάθε σημείου.

- scatter_counts: Διάνυσμα με το πλήθος των δεδομένων που θα σταλούν για κάθε διεργασία. Υπολογίζεται με το πηλίκο ανάμεσα του πλήθους των σημείων προς συσταδοποίηση και το πλήθος των διεργασιών επί τις διαστάσεις του σημείου.

- scatter_displs: Διάνυσμα για τις μετατοπίσεις που θα χρειαστεί να γίνουν στον χώρο των δεδομένων μας (data) για να διαμοιραστεί ισάξια «η εργασία» στις διεργασίες μας. Υπολογίζεται κατά την διάρκεια και της scatter_counts.

- MPI_FLOAT: Το τέταρτο και το έβδομο όρισμα ορίζουν τον τύπο δεδομένων που αποστέλλονται και τον τύπο που παραλαμβάνονται αντίστοιχα. Στην περίπτωση μας τα δεδομένα που αποστέλλονται και παραλαμβάνονται είναι του ίδιου τύπου και είναι τύπου float.

- rec_buff: Είναι η διεύθυνση στην οποία θα αποθηκευτούν τα δεδομένα του εκάστοτε επεξεργαστή όταν τα παραλάβει. Υπολογίζεται με την εξής εντολή όπου θα αποθηκευτούν δεδομένα τύπου float και κάθε διεύθυνση θα έχει μέγεθος το πλήθος των στοιχείων που θα αποθηκευτούν σε αυτήν,

  ```
  1.  float* rec_buff = new float[rec_buff_cnt];
  ```

- rec_buff_cnt: Είναι το πλήθος των στοιχείων που θα παραλάβει κάθε επεξεργαστής και υπολογίζεται με την εξής εντολής:

```
int rec_buff_cnt = (number0fPoints / world_size + (world_rank < number0fPoints % world_size ? 1 : 0)) * dimensionsOfPoint;
```

- 0: Είναι ο επεξεργαστής που αναλαμβάνει να στείλει τα δεδομένα στους υπόλοιπους επεξεργαστές. Στην περίπτωση μας θα τα στείλει ο επεξεργαστής μηδέν (0).
- MPI_COMM_WORLD: Η διασπορά των δεδομένων μας θα γίνει σε επίπεδο World άρα θα λάβουν όλοι μέρος. Αυτό δηλώνεται με την MPI_COMM_WORLD.

2.4. MPI_GATHERV

Η συνάρτηση **MPI_Gatherv**/9 που βρίσκεται στην συνάρτηση **canopy_mpi**/1 είναι μια παραλλαγή του *MPI_Gather*. Συλλέγει δεδομένα από όλες τις διεργασίες σε έναν δεδομένο επικοινωνητή και τα ενώνει στο δεδομένο buffer στην καθορισμένη διεργασία. Ωστόσο, σε αντίθεση με το *MPI_Gather*, το **MPI_Gatherv** επιτρέπει στα μηνύματα που λαμβάνονται να έχουν διαφορετικό μήκος και να αποθηκεύονται σε αυθαίρετες θέσεις στο buffer της κύριας διεργασίας. Το **MPI_Gatherv** είναι μια συλλογική λειτουργία. Όλες οι διεργασίες στον επικοινωνιακό φορέα πρέπει να επικαλούνται αυτήν τη ρουτίνα.

Οι παράμετροι της συνάρτησης ορίζονται ως εξής:

- buffer_send: Το buffer που περιέχει τα δεδομένα προς αποστολή. Η επιλογή 'in place' για ενδοεπικοινωνίες καθορίζεται περνώντας το MPI_IN_PLACE ως την τιμή του buffer_send στη κύρια διεργασία. Σε μια τέτοια περίπτωση, τα count_send και datatype_send αγνοούνται και η συμβολή της στην κύρια διεργασία στο συλλεγμένο διάνυσμα θεωρείται ότι βρίσκεται ήδη στη σωστή θέση στο buffer λήψης.
- count_send: Ο αριθμός των στοιχείων στο buffer αποστολής.
- datatype_send: Ο τύπος ενός στοιχείου προσωρινής αποθήκευσης αποστολής.
- buffer_recv: Το buffer στο οποίο αποθηκεύονται τα συγκεντρωμένα δεδομένα για τη διαδικασία root. Για άλλες διεργασίες, οι παράμετροι λήψης όπως αυτή αγνοούνται.
- counts_recv: Ένας πίνακας που περιέχει τον αριθμό των στοιχείων του μηνύματος που πρέπει να ληφθούν από κάθε διεργασία, όχι τον συνολικό αριθμό στοιχείων που θα ληφθούν από όλες τις διεργασίες συνολικά. Για τις μη-κύριες διεργασίες, οι παράμετροι λήψης όπως αυτή αγνοούνται.

- displacements: Ένας πίνακας που περιέχει τη μετατόπιση για εφαρμογή στο μήνυμα που λαμβάνεται από κάθε διεργασία. Οι μετατοπίσεις εκφράζονται σε αριθμό στοιχείων, όχι σε byte. Για διεργασίες χωρίς ρίζα, οι παράμετροι λήψης όπως αυτή αγνοούνται.
- datatype_recv: Ο τύπος ενός στοιχείου buffer λήψης. Για διεργασίες χωρίς ρίζα, οι παράμετροι λήψης όπως αυτή αγνοούνται.
- root: Η κατάταξη της διαδικασίας ρίζας, η οποία θα συλλέξει τα δεδομένα που συγκεντρώθηκαν.
- communicator: Ο επικοινωνιολόγος στον οποίο γίνεται η συγκέντρωση.

```
1.  MPI_Gatherv(canopy_id_send_data, gather_counts[world_rank], MPI_INT,canopy_id_data, gather_counts,
    gather_displs, MPI_INT,0, MPI_COMM_WORLD);
```

Στην Gatherv η διαδικασία είναι αντίστοιχη της Scatterv. Εδώ τα δεδομένα αποστέλνονται πίσω στον επεξεργαστή που αναλαμβάνει να την δουλειά αυτή. Στην περίπτωση μας είναι ο επεξεργαστής μηδέν (0) και ορίζεται μέσω του όγδοου ορίσματος της MPI_Gatherv.

Σημείωση: Χρησιμοποιήσαμε Scatterv και Gatherv αντί των Scatter και Gather επειδή μέσω των Scatterv και Gatherv μπορούμε να αποστέλλουμε και να παραλαμβάνουμε vectors.

2.5. MPI_BCAST

Μέσω της **MPI_Bcast**/5 αποστέλνονται δεδομένα στους υπόλοιπους επεξεργαστές. Στην περίπτωση εδώ ο αποστέλνεται το νέο κέντρο canopy προς τους επεξεργαστές.

Το πρώτο όρισμα είναι η διεύθυνση στην οποία βρίσκονται τα δεδομένα προς αποστολή, το δεύτερο όρισμα είναι το πλήθος των δεδομένων που θα αποσταλούν, στην περίπτωση μας εφόσον μιλάμε για το νέο κέντρο canopy θα στείλουμε το πλήθος των διαστάσεων του σημείου μας. Στο τρίτο όρισμα είναι ο τύπος των δεδομένων μας, είναι τύπου float εφόσον μιλάμε για σημεία. Στο τέταρτο όρισμα είναι ο επεξεργαστής που θα αναλάβει να αποστείλει τα δεδομένα. Εδώ να σημειώσουμε ότι εφόσον ορίσουμε τον επεξεργαστή που θα αποστείλει τα δεδομένα οι υπόλοιποι στον communicator θα παραλάβουν. Τέλος το πέμπτο όρισμα είναι το που θα λάβει χώρα η διαδικασία, στην περίπτωση μας θα λάβει χώρα σε όλο το communicator.

```
1.  MPI_Bcast(new_canopy_centre_data, dimensionsOfPoint, MPI_FLOAT, root_process, MPI_COMM_WORLD);
```

2.6. ΑΞΙΟΛΟΓΗΣΗ ΕΠΙΔΟΣΗΣ

```
-------------------------------------------
The performance of the algorithm:

-Points: 7443233
-Dimensions Of Point: 2
-Processes: 8

-Measured Work Took: 5.87986183 sec
-Measured Generation of Points Took: 0.53587675 seconds to run.

-Measured Whole Work Took: 5.34399 seconds.
-------------------------------------------
zartilas@zartilas:~/Documents/Github/CppProjects/Parallel-Canopy-Algorithm$
```

Εικόνα 2.6.1: Απλή Εκτέλεση

2.6.1. ΔΙΑΦΟΡΕΤΙΚΟ ΠΛΗΘΟΣ ΔΙΕΡΓΑΣΙΩΝ

```
-------------------------------------------
The performance of the algorithm:

-Points: 7443233
-Dimensions Of Point: 2
-Processes: 6

-Measured Work Took: 6.51106882 sec
-Measured Generation of Points Took: 0.76667166 seconds to run.

-Measured Whole Work Took: 5.7444 seconds.
-------------------------------------------
zartilas@zartilas:~/Documents/Github/CppProjects/Parallel-Canopy-Algorithm$
```

Εικόνα 2.6.1.1: Διαφορετικό Πλήθος Διεργασιών

2.6.2. ΔΙΑΦΟΡΕΤΙΚΟ ΠΛΗΘΟΣ ΣΗΜΕΙΩΝ ΠΡΟΣ ΣΥΣΤΑΔΟΠΟΙΗΣΗ

```
-------------------------------------------
The performance of the algorithm:

-Points: 769666
-Dimensions Of Point: 2
-Processes: 8

-Measured Work Took: 0.58519435 sec
-Measured Generation of Points Took: 0.11923122 seconds to run.

-Measured Whole Work Took: 0.465963 seconds.
-------------------------------------------
zartilas@zartilas:~/Documents/Github/CppProjects/Parallel-Canopy-Algorithm$
```

Εικόνα 2.6.2.1: Διαφορετικό Πλήθος Σημείων Προς Συσταδοποίηση

2.6.3. ΔΙΑΦΟΡΕΤΙΚΟ ΠΛΗΘΟΣ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΕΝΟΣ ΣΗΜΕΙΟΥ (ΠΑΡΑΜΕΤΡΟΣ Ν)

```
-------------------------------------------
The performance of the algorithm:

-Points: 7443233
-Dimensions Of Point: 3
-Processes: 8

-Measured Work Took: 35.55745721 sec
-Measured Generation of Points Took: 1.12615943 seconds to run.

-Measured Whole Work Took: 34.4313 seconds.
-------------------------------------------
zartilas@zartilas:~/Documents/Github/CppProjects/Parallel-Canopy-Algorithm$
```

Εικόνα 2.6.3.1: Διαφορετικό Πλήθος Χαρακτηριστικών Ενός Σημείου

3. ΣΥΜΠΕΡΑΣΜΑ

Χρησιμοποιώντας οκτώ διεργασίες και δημιουργώντας 7443133 σημεία τα οποία χαρακτηρίζονται από δύο στοιχεία βλέπουμε ότι χρειάζονται κάτι παραπάνω από πέντε δευτερόλεπτα για να συσταδοποιηθούν (Εικόνα 2.6.1: Απλή Εκτέλεση). Όταν μειώσαμε τις διεργασίες κατά δύο και έχοντας 6 διεργασίες στην διάθεση μας με τον ίδιο πλήθος σημείων και χαρακτηριστικών παρατηρούμε και χωρίς να αμφιβάλαμε ότι ο χρόνος συσταδοποίησης είναι μεγαλύτερος κατά κλάσματα δευτερολέπτων (Εικόνα 2.6.1.1: Διαφορετικό Πλήθος Διεργασιών). Αν είχαμε μειώσει κατά 4 διεργασίες και είχαμε τις μισές από την αρχική μας εκτέλεση τότε θα βλέπαμε ότι θα χρειαζόταν περισσότερος χρόνος εκτέλεσης. Στην περίπτωση που μειώσουμε τα σημεία προς συσταδοποίηση και κρατήσουμε το πλήθος των διεργασιών στις οκτώ και το πλήθος των χαρακτηριστικών στα δύο, βλέπουμε ότι χρειάζεται λιγότερο από μισό δευτερόλεπτο για την συσταδοποίηση (Εικόνα 2.6.2.1: Διαφορετικό Πλήθος Σημείων Προς Συσταδοποίηση). Τέλος, όταν αυξήσουμε το πλήθος των χαρακτηριστικών κατά ένα (Εικόνα 2.6.3.1: Διαφορετικό Πλήθος Χαρακτηριστικών Ενός Σημείου) βλέπουμε ότι ο χρόνος αυξήθηκε εκθετικά. Χρειάστηκε λίγο λιγότερο από 35 δευτερόλεπτα για να γίνει η συσταδοποίηση των σημείων. Συμπεραίνουμε λοιπόν ότι θα δούμε ραγδαία αλλαγή του χρόνου εκτέλεσης όταν αυξήσουμε το πλήθος των χαρακτηριστικών των σημείων μας.

4. ΒΙΒΛΙΟΓΡΑΦΙΑ ΚΑΙ ΠΗΓΕΣ

4.1. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] A. Mccallum, K. Nigam, and L. H. Ungar, “Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching”.

[2] S. Sur, M. J. Koop, and D. K. Panda, “High-performance and scalable MPI over InfiniBand with reduced memory usage: An in-depth performance analysis,” *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC’06*, 2006, doi: 10.1145/1188455.1188565.

4.2. ΗΛΕΚΤΡΟΝΙΚΕΣ ΠΗΓΕΣ

[1] https://medium.com/geekculture/configuring-mpi-on-windows-10-and-executing-the-hello-world-program-in-visual-studio-code-2019-879776f6493f

[2] https://www.reddit.com/r/learnprogramming/comments/3agesw/can_someone_explain_to_me_what_an_mpi_rank_is/

[3] https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report/node303.htm

[4] https://www.mpich.org/static/docs/v3.1/www3/MPI_Scatterv.html

[5] http://www2.cs.uh.edu/~johnson2/labs.html

[6] https://people.math.sc.edu/Burkardt/cpp_src/mpi/mpi.html

[7] https://stackoverflow.com/questions/46312468/c-vector-pushback-popback-functions

[8] https://en.cppreference.com/w/cpp/container/unordered_set

[9] https://www.cplusplus.com/reference/cassert/assert/

[10] https://www.codingame.com/playgrounds/349/introduction-to-mpi/measuring-time

5. ΒΙΒΛΙΟΘΗΚΕΣ ΚΑΙ ΕΡΓΑΛΕΙΑ

Όνομα	*Έκδοση*	*Τύπος*
cassert	Core	Βιβλιοθήκη
cfloat	Core	Βιβλιοθήκη
cmath	–	Βιβλιοθήκη
cstring	Core	Βιβλιοθήκη
ctime	Core	Βιβλιοθήκη
iostream	–	Βιβλιοθήκη
MPICH	3.3.2	Βιβλιοθήκη
unordered_set	–	Βιβλιοθήκη
vector	–	Βιβλιοθήκη
Visual Studio Code	1.63.2	Εργαλείο

- Core: Κύρια βιβλιοθήκη της C++.

6. ΠΙΝΑΚΑΣ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ

Συντομογραφία	*Λέξη*
MPI	Message Passing Interface
PRNG	Pseudo-Random Number Generator
Κλπ.	Και τα λοιπά