Distribute By Median

Fotiou Dimitrios
AEM 9650
Aristotle University of Thessaloniki
Department of Electrical and Computer Engineering
https://github.com/dimfot3/MPI_Proj

***Abstract*—**Το παρών αρχείο αποτελεί την αναφορά στη δεύτερη εργασία στο μάθημα Παράλληλα και Διανεμημένα συστήματα του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Αριστοτελείου Πανεπιστημίου**.** Σκοπός αυτής της εργασίας ήταν η μελέτη και η ανάπτυξη μιας εφαρμογής με χρήση **Message Passing Interface(MPI).** Η αναφορά ξεκινά με την παρουσίαση της εφαρμογής**,** στη συνέχεια παρουσιάζεται η υλοποίησή της και τέλος τα αποτελέσματα και χρόνοι εκτέλεσης**.** Όλοι οι κώδικες και τα δεδομένα που χρησιμοποιήθηκαν βρίσκονται στο **github https://github.com/dimfot3/MPI_Proj.**

***Index Terms*—MPI, parallelism**

I. Introduction

Το Message Passing Interface(MPI) είναι ένα πρωτόκολλο επικοινωνίας μεταξύ διαδικασιών ενός ή περισσοτέρων υπολογιστικών κόμβων. Αποτελεί σημαντικό εργαλείο σε περιπτώσεις που θέλουμε να κατανέμουμε τον υπολογιστικό φόρτο ώστε να επωφεληθούμε από την παράλληλη διεκπεραίωσή του.
Το MPI δεν στηρίζεται σε κοινή μνήμη και για αυτό χρειάζεται να αντιγραφούν τα δεδομένα από τη μια διαδικασία στην άλλη. Σε περίπτωση που τα δεδομένα καταλαμβάνουν λίγο χώρο οι διαφορές μιας MPI υλοποίησης και μιας που στηρίζεται σε κοινή μνήμη (pthreads, OpenMP, OpenCilk) είναι μικρές. Παρόλα αυτά όταν θέλουμε να εντάξουμε στον υπολογισμό και επεξεργαστές που βρίσκονται σε διαφορετικούς υπολογιστικούς κόμβους οι MPI υλοποιήσεις αποτελούν πιο ελκυστική και εφικτή λύση.
Στα πλαίσια αυτής της εργασίας καλούμαστε να υλοποιήσουμε μια εφαρμογή όπου να κατανέμει κάποια σημεία με βάση την απόστασή τους από ένα σταθερό σημείο. Πιο συγκεκριμένα θα πρέπει σε k διεργασίες να κατανέμουμε N σημεία έτσι ώστε σε μια διεργασία η μικρότερη απόσταση από ένα σημείο αναφοράς να είναι μεγαλύτερη από τη μέγιστη της προηγούμενης.
Για την υλοποίηση αυτής της κατανομής θα πρέπει να χρησιμοποιηθεί MPI. Κάποιες παραδοχές που γίνονται είναι ότι τόσο ο αριθμός των διεργασιών όσο και ο αριθμός των σημείων είναι ζυγός και η διαίρεση του αριθμού των σημείων με τον αριθμό των διαδικασιών δεν αφήνει υπόλοιπο. Επιπλέον η διάταξη των διεργασιών γίνεται με βάση το id του στις συνολικές MPI διεργασίες που συμμετέχουν σε ένα υπολογισμό.

II. Implementation

A. Simple Implemenation

Η εφαρμογή ξεκινά με k διαδικασίες να φορτώνουν N/k σημεία διάστασης d. Τα σημεία αρχικά είναι στιβαγμένα σε float32 και έπειτα φορτόνονται σε δισδιάστακο πίνακα. Η διαδικασία 0 είναι ο αρχηγός που διαλέγει αρχικά ένα από τα σημεία του ως pivot. Το ανακοινώνει σε όλους τους άλλους με απλό blocking bcast.
Στη συνέχεια γίνεται κλήση της αναδρομικής ρουτίνας distributeByMedian. Στη ρουτίνα αυτή μπαίνουν όλες οι διαδικασίες. Σαν όρισμα παίρνει το id του αρχηγού, τα σημεία της εκάστοτε διαδικασίας, τον αριθμό των διαδικασιών που συμμετέχουν στην ανακατανομή και το pivot σημείο.
Η συνάρτηση distributeByMedian σε κάθε διαδικασία μετράει την ευκλείδεια απόσταση των σημείων της διαδικασία από το pivot. Στη συνεχεια τα στέλνει στον αρχηγό που έχει οριστεί για την κάθε διαδικασία. Ο αρχηγός χρησιμοποιεί την quickselect για να βρει το ενδιάμεσο και έπειτα το στέλνει σε όλες τις διαδικασίες που έχει αναλάβει. Η quickselect είναι υλοποιημένη με επαναληπτική και όχι αναδρομική αρχιτεκτονική και επειδή το σύνολο των σημείων είναι ζυγός αριθμός επιστρέφει το μέσο των δύο ενδιάμεσων σημείων.
Ακολούθως, οι διαδικασίες με βάση αυτή την ενδιάμεση τιμή διαχωρίζουν τα σημεία που έχουν μεγαλύτερη ή μικρότερη απόσταση. Η κάθε διαδικασία βρίσκει τη σχετική θέση της στο σύνολο των διαδικασιών όπου γίνεται η ανακατανομή κάθε φορά, λαμβάνοντας υπόψη που βρίσκεται και ο αρχηγός της ανακατανομής. Αν η σχετική της θέση είναι μικρότερη από το μέσο του συνόλου τότε πρέπει να στείλει όλα τα σημεία με απόσταση μεγαλύτερη του μέσου, αλλιώς αυτά με μικρότερη.
Για το σκοπό αυτό, ανακοινώνει στις υπόλοιπες διαδικασίες πόσα στοιχεία πρέπει να στείλει. Έτσι γνωρίζοντας η κάθε διαδικασία πόσα στοιχεία έχουν να στείλουν οι υπόλοιπες που συμμετέχουν στην κάθε ανακατανομή μπορεί να δημιουργήσει ένα πίνακα. Η λογική είναι ότι στη πρώτη ανταλλαγή η κάθε διαδικασία βρίσκει το μέγιστο που μπορεί να ανταλλάξει με την διαδικασία που έχει id μεγαλύτερο κατά το μισό των συνολικών διαδικασιών σε κάθε ανακατανομή. Στη δεύτερη ανταλλαγή ασχολείται με την ακριβώς επόμενη διαδικασία. Έτσι γνωρίζουν μεταξύ τους πως να επικοινωνούν ασύγχρονα χωρίς περαιτέρω επικοινωνίες και χωρίς παρεμβολή του αρχηγού.
Για να γίνει καλύτερα αντιληπτός ο τρόπος με τον οποίο

Table I
CHANGE TABLE FOR THE FIRST THREE STEPS

Step 1	Step 2	Step 3
$0 \leftrightarrow 4$	$0 \leftrightarrow 5$	$0 \leftrightarrow 6$
$1 \leftrightarrow 5$	$1 \leftrightarrow 6$	$1 \leftrightarrow 7$
$2 \leftrightarrow 6$	$2 \leftrightarrow 7$	$2 \leftrightarrow 4$
$3 \leftrightarrow 7$	$3 \leftrightarrow 4$	$3 \leftrightarrow 5$

γίνονται οι ανταλλαγές μπορούμε να δούμε και το πίνακα I όπου συμμετέχουν 8 διαδικασίες. Παρουσιάζονται 3 βήματα αλλά με την ίδια λογική προκύπτουν και τα υπόλοιπα.

Σε κάθε ανταλλαγή ο πίνακας ανανεώνεται αυτόματα και έτσι γνωρίζουν μεταξύ τους πόσα στοιχεία μένει να στείλει η κάθε διαδικασία χωρίς περαιτέρω επικοινωνίες. Όταν τελειώσει με τις επαναλήψεις της η κάθε διαδικασία ενώνει πάλι όλα της τα σημεία και υπολογίζει τις αποστάσεις από το pivot.

Στο τέλος της distributedByMedian γίνεται πάλι κλήση της αλλά αυτή τη φορά με $k/2$ διεργασίες, και αρχηγό τον ίδιο αν το id της διαδικασίας είναι μικρότερο από $k/2$ αλλιώς αρχηγός είναι ο $leader\_id + k/2$.

Τέλος ζητείται από την εκφώνηση να αυτοελέγχεται η ορθότητα του κώδικα. Για τον σκοπό αυτό στο τέλος όλες οι διαδικασίες στέλνουν στην διαδικασία 0 τις αποστάσεις από το pivot. Ο αρχηγός ελέγχει αν η ελάχιστη κάθε διαδικασίας είναι μεγαλύτερη από την μέγιστη της προηγούμενης και αν η μέγιστη είναι μικρότερη από τις αποστάσεις των επόμενων διαδικασιών. Σε περίπτωση που δεν ισχύει αυτή η συνθήκη το εκτυπώνει στην κονσόλα και δεν αποθηκεύει τους χρόνους μιας και έχει αποτύχει ο αλγόριθμος.

B. Complex Implementation

Η παραπάνω υλοποίηση δουλεύει σωστά για πολλά διαφορετικά dataset όταν έχουμε κυρίως μέχρι 4 διαδικασίες. Όσο αυξάνουμε τον αριθμό διαδικασιών παρατηρούμε ότι ο αλγόριθμος αποτυγχάνει. Η αιτία είναι επειδή προκύπτουν φορές που το μέσο υπάρχει μέσα στο πίνακα των αποστάσεων.

Κάτι τέτοιο δημιουργεί πρόβλημα καθώς τα σημεία αυτά μπορούν να ανήκουν και στις δύο μεριές των ανταλλαγών και δημιουργούνται καταστάσεις όπου η μία από τις δύο μεριές των διαδικασιών έχουν περισσότερα στοιχεία να στείλουν από την άλλη.

Για να αντιμετωπιστεί αυτό το ζήτημα χωρίς να αυξηθεί πολύ η πολυπλοκότητα αλλά ούτε και ο χρόνο εκτέλεσης, στις περιπτώσεις που παρατηρείται ότι οι δύο πλευρές έχουν άνισο αριθμό στοιχείων, τα σημεία με απόσταση ίση με τη ενδιάμεση αντιμετωπίζονται ώς μπαλαντέρ. Έτσι δημιουργείται ένας πίνακας ανταλλαγών ακριβώς όπως παραπάνω και ένας πίνακας μπαλαντερ. Ο πίνακας μπαλαντέρ απλά δηλώνει ποιες διαδικασίες δεν έχουν να στείλουν άλλα στοιχεία εκτός από αυτά με απόσταση ίσης της ενδιάμεσης. Οπότε χρειάζεται να στείλει η κάθε διαδικασία πόσα στοιχεία μπορεί να στείλει. Τέλος γίνεται ανταλλαγή των στοιχείων όπως και πριν απλά με τον περιορισμό ότι δύο μπαλαντέρ δεν μπορούν να στείλουν μεταξύ τους, καθώς σκοπός μας είναι να κατανέμουν τα υπόλοιπα στοιχεία που έχουν μεγαλύτερη ή μικρότερη τιμή από την ενδιάμεση.

Γενικά αυξάνεται η πολυπλοκότητα και ο χρόνος αλλά γίνεται μόνο τις φορές που προκύπτει άνισος αριθμός στοιχείων μεταξύ των δύο ομάδων των διεργασιών. Παρόλα αυτά οι μόνες επιπλέον επικοινωνίες είναι ο πίνακας ανταλλαγών και τα στοιχεία που δεν μπορούσαν να ανταλλαχθούν στην πρώτη φάση ανταλλαγών. Συνήθως αυτά είναι $2 - 4$ σε αριθμό διαδικασιών μέχρι και 64 οπότε η τελική αύξηση χρόνου είναι ανεπαίσθητη, αλλά φυσικά κάτι τέτοιο επηρεάζεται από την κατανομή που ακολουθούν τα σημεία και από το εύρως τιμών που πέρνουν.

III. CODE AND VALIDATION

Η υλοποίηση της παραπάνω εφαρμογής έγινε σε C. Από τις διάφορες MPI υλοποιήσεις χρησιμοποιήθηκε η OpenMPI [1] ενώ σαν εργαλείο κατασκευής της εφαρμογής χρησιμοποιήθηκε το CMake [3].

Η δημιουγεία δεδομένων, η επεξεργασία και η οπτικοποίηση των αποτελεσμάτων έγινε σε Python χρησιμοποιόντας βιβλιοθήκες όπως numpy, pandas, pytorch, matplotlib.

Όσον αφορά την επαλήθευση της ορθότητας χρησιμοποιήθηκε ο αυτόματος έλεγχος που εξηγήθηκε παραπάνω ώστε να επαληθεύεται ότι σε κάθε περίπτωση η εφαρμογή πετυχαίνει το στόχο της, κατανέμοντας σωστά τα σημεία. Ο επιτυχής έλεγχος αποτελεί προϋπόθεση ώστε η εφαρμογή να βγάζει αποτελέσματα χρόνου. Επιπρόσθετα χρησιμοποιήθηκαν κάποιοι έλεγχοι μονάδας από το έργο ανοιχτού κώδικα GoogleTest [4] για να βεβαιώνεται εύκολα και γρήγορα η σωστή λειτουργία βασικών συναρτήσεων που χρησιμοποιήθηκαν. Για περετέρω εξακρίβωση ότι τα αποτελέσματα της κατανομής με χρήση ενδιάμεσου είναι σωστά, χρησιμοποιείται και ένα python script, eval.py το οποίο ταξινομεί τα δεδομένα και βρίσκει ποιο θα έπρεπε να ήταν το min/max κάθε διαδικασίας για δεδομένο dataset και pivot.

Τα αποτελέσματα χρόνου πάρθηκαν από τη συστηχεία του ΑΠΘ για να μπορούν εύκολα να συγκριθούν πάνω σε κοινή βάση με τις υλοποιήσεις των συμφοιτητών. Χρησιμοποιήθηκαν εώς και 64 διεργασίες, επομένως 4 κόμβοι με 16 πυρήνες ο καθένας.

IV. DATASET

Απαραίτητα για την εφαρμογή ήταν ένα σύνολο πολυδιάστατων σημείων τα οποία θα κατανέμονται στις διάφορες διαδικασίες. Προκειμένου να δοκιμάσουμε τον αλγόριθμο σε διαφορετικές συνθήκες δημιουργήθηκαν τυχαία σημεία από τρεις διαφορετικές κατανομές, ομοιόμορφη, κανονική και εκθετική με ποικιλία στον συνολικό αριθμό των σημείων αλλά και τον αριθμό της διάστασης των σημείων. Επίσης χρησιμοποιήθηκαν δύο γνωστά σύνολα δεδομένων αυτό του MNIST με τα ψηφία [5] και το MNIST με τα ρούχα μόδας [6] προκειμένου τα δοκιμαστεί η εφαρμογή σε άγνωστες κατανομές.

Τα σημεία είναι αποθηκευμένα σε binary αρχεία. Τα πρώτα δύο float32 είναι ο αριθμός των σημείων και η διάσταση.

Και έπειτα ακολουθούν τα σημεία σε float32. Υπάρχουν διαθέσιμα κάποια python scripts στο git repo [2] όπου δημιουργούν τέτοια δεδομένα από στατιστικές κατανομές και από γνωστές συλλογές δεδομένων όπως το MNIST. Επίσης υπάρχουν διαθέσιμα κάποια δείγματα τέτοιων δεδομένων στο προσωπικό drive και μπορούν να κατέβουν από το σύνδεσμο [7].

V. Results

Θα εξετάσουμε την επίδραση διαφόρων παραγόντων στο χρόνο διεκπεραίωσης της εφαρμογής. Μπορούμε να πούμε ότι οι παράγοντες που επηρεάζουν άμεσα τον χρόνο εκτέλεσης είναι τέσσερις. Συγκεκριμένα είναι ο αριθμός των σημείων, ο αριθμός της διάστασης, ο αριθμός των εργασιών και τέλος η στατιστική κατανομή των σημείων.
Αρχικά εξετάζουμε την επίδραση του αριθμού των σημείων σε διαφορετικές κατανομές. Όπως βλέπουμε από το σχήμα 1 αυξάνοντας τα σημεία αυξάνεται και ο χρόνος εκτέλεσης. Κάθε σημείο στο γράφημα αποτελεί μία ξεχωριστή εκτέλεση. Οι τρεις κατανομές ομοιόμορφη, κανονική και εκθετική φαίνεται να μην έχουν πολύ επιρροή στο τελικό χρόνο εκτέλεσης, αλλά το dataset που δημιουργήθηκε από το Mnist γενικά αποδείχθηκε πιο δαπανηρό. Κάτι τέτοιο μας φανερώνει ότι σε τυχαίες κατανομές η εφαρμογή είναι πιο αργή.

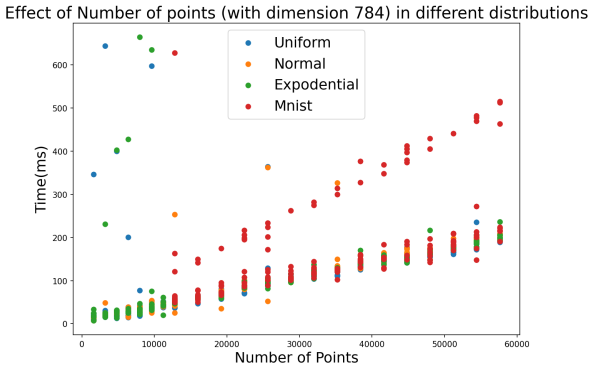

Figure 1. Number of points effect in different distributions(16 processes)

Στη συνέχεια κρατάμε τον αριθμό των σημείων σταθερό και πειράζουμε τον αριθμό της διάστασης. Όπως βλέπουμε στο 2, αυξάνοντας την διάσταση αυξάνεται και ο χρόνος σχεδόν γραμμικά. Το mnist είχε σταθερή διάσταση καθώς είναι $28\chi28$ και επομένως δεν είχε πολύ αξία στη συγκεκριμένη σύγκριση.

Η επόμενη επίδραση που θα εξετάσουμε είναι αυτή των αριθμών των διαδικασιών. Για το σκοπό αυτό χρησιμοποιούμε 10 dataset διαφόρων μεγεθών, διάστασης σημείων και παίρνουμε το μέσο χρόνο σε 100 επαναλήψεις για αριθμό διαδικασιών $2, 4, 8, 16, 32, 64$. Όπως βλέπουμε στο 3 έχουμε εκθετική αύξηση του χρόνου.
Θα εξετάσουμε και τη διασπορά των χρόνων εκτέλεσης ώστε να δούμε κατά πόσο μπορούμε να εμπιστευτούμε μεμονωμένους χρόνους. Για τον σκοπό αυτό τρέχουμε 100 επαναλήψεις σε ένα dataset με ίδιες παραμέτρους και pivot.

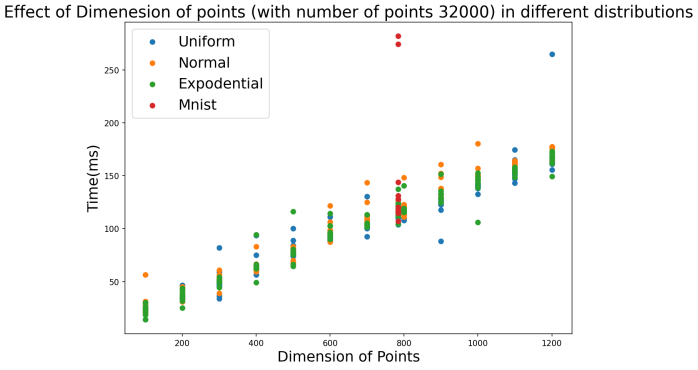

Figure 2. Dimension of points effect in different distributions (16 processes)

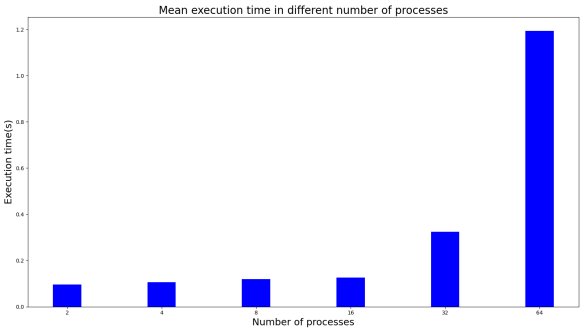

Figure 3. Number of process effect

Η κατανομή του χρόνου φαίνεται στο 4 και μας δείχνει ότι γενικά οι χρόνοι είναι σταθεροί.

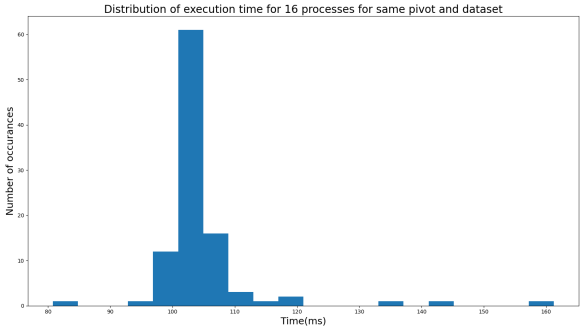

Figure 4. Execution time distribution

Στα αποτελέσματα αξίζει να αναφέρουμε ένα αρκετά ενδιαφέρον φαινόμενο. Γενικά όπως και στην προηγούμενη εργασία, παρατηρούνται φαινόμενα cold cache και οι πρώτοι χρόνοι ήταν αρκετά μεγαλύτεροι. Σε 4-8 διεργασίες η διαφορά ήταν $\pm 300ms$ όμως όταν εξετάζουμαι 32 ή 64 διεργασίες, η διαφορά ξέφευγε σε $6-8$ δευτερόλεπτα. Οι

χρόνοι μετά από κάποιο αριθμό επαναληψεων έπεφταν σε κανονικά επίπεδα κοντά στο ένα δευτερόλεπτο (στις 64 διεργασίες).
Αυτό ίσως είναι ένα άλλο σημαντικό πλεονέκτημα του MPI. Δεδομένου ότι χρησιμοποιούνται περισσότερα cores, υπάρχει περισσότερη cache διαθέσιμη και μπορούμε να επωφεληθούμε από αυτή ακόμα και για μεγάλα dataset (60k σημεία με 1200 διάσταση), καθώς μοιράζονται τα σημεία στις διάφορες διαδικασίες. Αν εκτελούταν σειριακά η εφαρμογή, η cache θα φόρτωνε συνεχώς καινούργια δεδομένα και ο χρόνος εκτέλεσης θα ήταν πολύ μεγαλύτερος.

VI. Conclusion

Κλείνοντας θα λέγαμε ότι το MPI αποτελεί πολύ χρήσιμο εργαλείο όταν έχουμε να επεξεργαστούμε πολλά δεδομένα, καθώς επιτρέπει την συνεργασία πολλών διαφορετικών πυρήνων. Συγεκριμένα για την εφαρμογή κατανομής με τη χρήση της ενδιάμεσης τιμής προσπαθήσαμε όσο το δυνατόν περισσότερο να δουλεύουν ασύγχρονα οι διαδικασίες ώστε να μη χάνεται χρόνος σε άσκοπες επικοινωνίες και μεταφορές δεδομένων. Και τέλος εξετάσαμε τους διαφορετικούς παράγοντες που επιρεάζουν τον χρόνο εκτέλεσης.

Acknowledgment

Τα αποτελέσματα που παρουσιάζονται έχουν παραχθεί αξιοποιώντας την Υπολογιστική Συστοιχία και τις παρεχόμενες υπηρεσίες υποστήριξης του Κέντρου Ηλεκτρονικής Διακυβέρνησης του Α.Π.Θ..

References

[1] OpenMPI: Open Source High Performance Computing https://www.open-mpi.org/
[2] This project's github repository https://github.com/dimfot3/MPI_Proj
[3] CMake: Open Source bulding and testing tool https://cmake.org/
[4] GoogleTest - Google Testing and Mocking Framework - GitHub https://github.com/google/googletest
[5] MNIST - handwritten digits dataset https://en.wikipedia.org/wiki/MNIST_database
[6] MNIST - fashion dataset https://www.kaggle.com/zalando-research/fashionmnist
[7] Data samples on personal drive: https://drive.google.com/file/d/1GoOx96Bppyylp6Pe3CGnyaywA24NJDJd/view?usp=sharing