Παράλληλα και Διανεμημένα Συστήματα

Κωτούλας Εμμανουήλ 9697

Εργασία 2

Ανακατανομή σημείων σε διαδικασίες ανάλογα με την απόσταση τους από ένα τυχαία επιλεγμένο σημείο από τον «αρχηγό» με χρήση MPI

Κώδικας

Ο κώδικας μου βρίσκεται εδώ. Υπάρχει bash script το οποίο μπορείτε να τρέξετε το οποίο κάνει compile τα αρχεία και τρέχει κάποια benchmark για να αξιολογηθεί ότι λειτουργεί σωστά.

Υλοποίηση MPI

Αρχικά έγραψα τα αρχεία makeMnist.c και makeRand.c τα οποία παίρνουν στοιχεία από το MNIST και δημιουργούν τυχαία N σημεία d διαστάσεων με τιμές float 0-100 αντίστοιχα και τα αποθηκεύουν σε αρχεία bin για να τα διαβάσουν τα Mpi.c και MpiMnist.c.

Main

Αρχικά κάθε process διαβάζει με τις ρουτίνες MPI_File από το ίδιο αρχείο τα σημεία που της αναλογούν από την κατάλληλη θέση μνήμης και τα αποθηκεύει σε έναν «δισδιάστο» πίνακα ο οποίος ορίστηκε ως μονοδιάστατος για διευκόλυνση στις αποστολές/λήψεις και στην ανάγνωση των σημείων.

Distribute by Median

Έπειτα όλα τα process εισέρχονται στην DistributeByMedian. Εκεί ο αρχηγός ελέγχει άμα αυτή είναι η πρώτη φόρα που καλείται η συνάρτηση και αν είναι επιλέγει τυχαία ένα από τα σημεία του που θα είναι το Pivot και για τις υπόλοιπες αναδρομικές κλήσεις της συνάρτησης. Ο αρχηγός στέλνει το Pivot και στις υπόλοιπες διεργασίες και όλοι μαζί υπολογίζουν τις αποστάσεις των σημείων τους από το Pivot που επιλέχθηκε. Ο αρχηγός αρχικοποιεί έναν πίνακα meanD όπου και αποθηκεύει τις τιμές που του στέλνουν ένα ένα τα υπόλοιπα process. Μετά είναι εύκολο για τον αρχηγό να καλέσει μια υλοποιήση της quickSelect, να βρει την διάμεσο και να την ανακοινώσει σε όλα τα process.

Trading

Πριν ωστόσο ξεκινήσουν οι ανταλλαγές όλα τα process βρίσκουν πόσα σημεία έχουν μικρότερα και πόσα μεγαλύτερα της διαμέσου και κάνουν partition τους πινακές με τα σημεία τους έτσι ώστε οι πρώτες p/2 να έχουν πρώτα τα μεγάλα τους σημεία και οι υπόλοιπες να έχουν πρώτα τα μικρότερα σημεία ώστε να γίνουν και ευκολότερα οι ανταλλαγές. Έπειτα όλα τα process στέλνουν στον αρχηγό πόσα σημεία έχουν να ανταλλάξουν, τα μισά τα σημεία με απόσταση από το Pivot μικρότερης της διάμεσης και αντίστοιχα τα άλλα μισά. Ο αρχηγός πλέον ξέρει πόσα σημεία πρέπει το κάθε process συμπεριλαμβανομένου αυτού να ανταλλάξει.

Έτσι παίρνει με την σειρά από το πρώτο process στα πρώτα p/2 και από το πρώτο process στα δεύτερα p/2 και ανταλλάσσει όσα περισσότερα στοιχεία μπορεί με μια ανταλλαγή. Όταν κάποιος δώσει/λάβει όλα τα στοιχεία που χρειάζεται ώστε να έχει μόνο σημεία με μικρότερη/μεγαλύτερη απόσταση ο αντίστοιχος counter πάει στην επόμενη διεργασία. Αυτό γίνεται μέσω μιας while στον αρχηγό ο οποίος κάνει τις παραπάνω επιλογές και στέλνει στις διεργασίες με ποιον θα ανταλλάξουν στοιχεία και πόσα στοιχεία θα ανταλλάξουν. Αν ο αρχηγός είναι αυτός που θα συμμετάσχει στην ανταλλαγή τότε επιλέγει να ανταλλάξει στοιχεία. Οι ανταλλαγές θα έχουν τελειώσει όταν οι δείκτες για το ποιο process ανταλλάσσει φτάσουν στο αντίστοιχο τέλος τους.

Recursion

Τέλος υπάρχει μια if που διαχωρίζει πως θα συνεχίσουν οι διεργασίες, καθαυτόν τον τρόπο οι πρώτες p/2 και οι υπόλοιπες p/2 καλούν την κατάλληλη συνάρτηση ώστε να υπάρχει το σωστό διάστημα διεργασιών σε κάθε κλήση, δηλαδή κάθε φορά συνεχίζουν στην επόμενη κλήση οι μισές και δημιουργείται μια δομή παρόμοια με αυτή ενός δέντρου. Κάθε process βγαίνει από την συνάρτηση όταν κληθεί μόνο με τον ίδιο μέσα.

Self-Check

Αφού επιστρέψουν στην main όλα τα process περιμένουν στο Barrier ώστε ο αρχηγός να πάρει σωστή μέτρηση για τον χρόνο που χρειάστηκε για να τρέξουν όλα τα process. Τέλος το κάθε process υπολογίζει εκ νέου την απόσταση των σημείων του από το pivot και στέλνει την μέγιστη και την ελάχιστη στον αρχηγό. Αυτός ελέγχει κατά πόσο είναι σωστή η κατανομή αυτών και το τυπώνει το κατάλληλο μήνυμα.

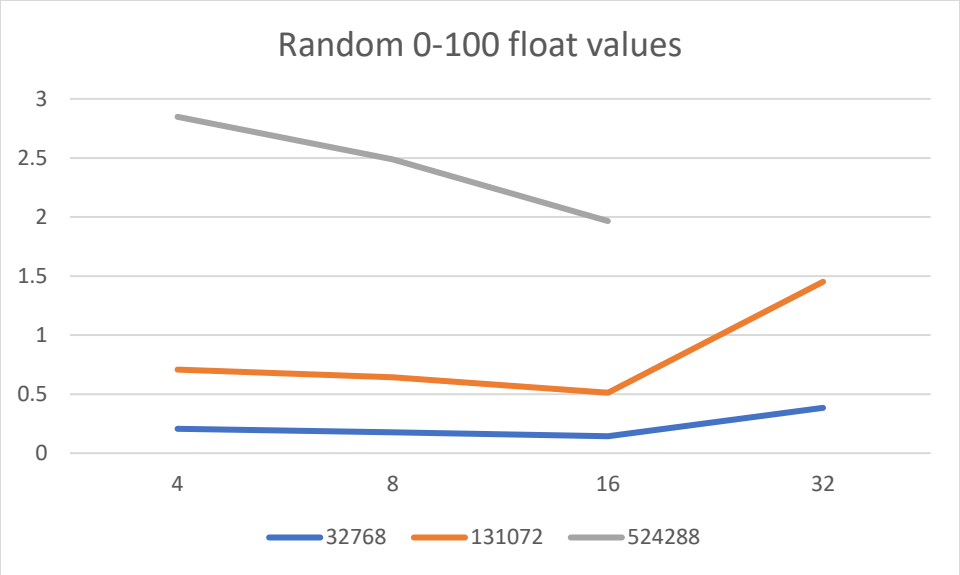
Τρέχοντας στον Αριστοτέλη:

Να σημειωθεί ότι πειράματα έτρεξαν και στο μηχάνημά μου και ο χρόνος αυξάνονταν ελαφρώς όσο περισσότερα process χρησιμοποιούνται και αυξάνονταν πολύ με την χρήση 16 process και πάνω, ωστόσο θεωρώ τα αποτελέσματα ασήμαντα μιας και δεν είναι αυτή η χρησιμότητα του MPI, θα χρησιμοποιούσαμε κάτι σαν τις υλοποιήσεις της προηγούμενης εργασίας.

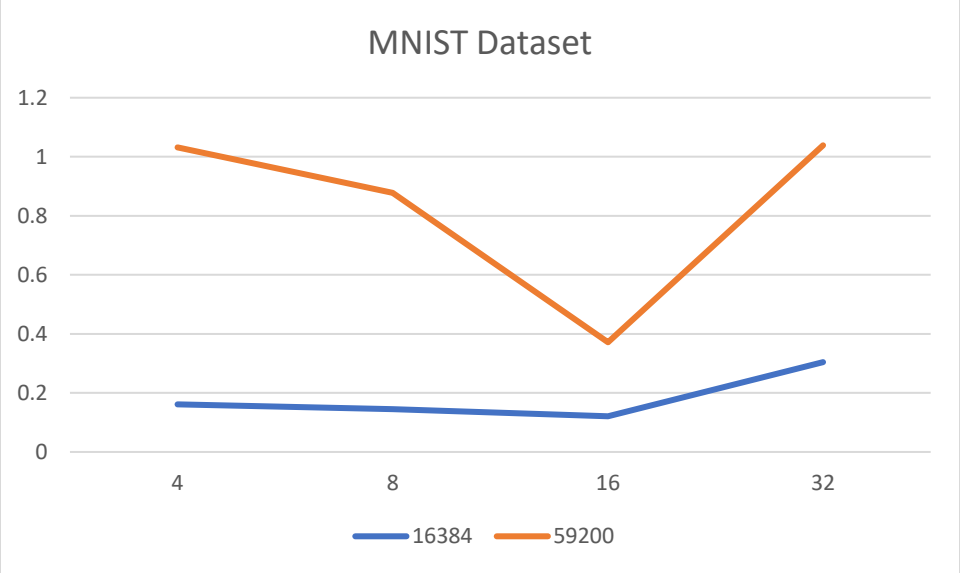
Τα πειράματα που έγιναν στην συστοιχία ήταν περιορισμένα καθώς υπήρχε περιορισμένος χρόνος και υπήρξαν διάφορα τεχνικά προβλήματα.

Αρχικά και στα δυο πειράματα έτρεξα με 4,8,16,32 cpus. Αρχικά με 16χιλλιάδες και 59χιλλιάδες σημεία του MNIST και έπειτα με 30χιλλιάδες, 130χιλλιάδες και 520χιλλιάδες σημεία 512 διαστάσεων με τυχαίες float τιμές στο range(0-100).

Και στα δύο πειράματα παρατηρούμε το αναμενόμενο, ότι μειώνεται ο χρόνος όσο αυξάνουμε τα process. Υπάρχει ωστόσο μια αύξηση χρόνου για 32 processes που υποθέτω ότι οφείλεται το γεγονός ότι οι μεταφορές δεδομένων κοστίζουν πολύ χρονικά και έτσι υπερτερούν του κέρδους της παραλληλοποίησης. Επίσης να σημειωθεί ότι το μεγαλύτερο κέρδος απόδοσης φαίνεται στο πείραμα των τυχαίων τιμών στα 520χιλλιάδες σημεία με την βελτίωση με λιγότερα σημεία να είναι πολύ μικρή (δυστυχώς δεν κατάφερα να τρέξω το πρόγραμμα για 520χιλλιάδες σημεία με 32 process, συνεχώς κολλούσε.)



Στο πείραμα με τα δεδομένα MNIST παρατηρείται σημαντική βελτίωση για 16 παράλληλα process, με καλύτερη από γραμμική βελτίωση (8 process 0.877s - 16 process 0.3714)



Επίλογος

Από τα δύο παραπάνω πειράματα γίνεται εύκολα αντιληπτή η ανάγκη και η χρησιμότητα του συστήματος MPI το οποιο είναι αναγκαίο για μεγάλο όγκο δεδομένων και εξαιρετικά χρήσιμο για την επιτάχυνση ενός προγράμματος σαν αυτό που τρέξαμε στην συστοιχία Αριστοτέλης. Τέλος θα ήθελα να αναφέρω την υπόθεση ότι η καθυστέρηση που βλέπουμε για 32 πυρήνες σε σχέση με 16 μπορεί να οφείλεται στο γεγονός ότι το κάθε node της συστοιχίας έχει 20 πυρήνες και έτσι για να πάρουμε 32 θα πρέπει να χρησιμοποιήσουμε παραπάνω nodes. Υποθέτω λοιπόν ότι για την καθυστέρηση αυτή οφείλεται το latency που υπάρχει ανάμεσα στα δύο αυτά συστήματα, αν και μετά την πρώτη κλήση της συνάρτησης τα 2 συστήματα θα μπορούσαν να κάνουν ανεξάρτητα τις πράξεις τους. Δεν γνωρίζω κατά πόσο αυτό αληθεύει. Όλα τα τεστ έγιναν στο batch.