Λογική υλοποίησης

Για την υλοποίηση της άσκησης χρησιμοποίησα τον δοσμένο κώδικα με ορισμένες αλλαγές για την διευκόλυνση αλλά και για την προσαρμογή του σε περιβάλλον MPI.

Η βασική ιδέα για την υλοποίηση σε MPI ήταν να χωριστούν τα σημεία (δηλαδή ο πίνακας data_in.dataset) σε τόσα κομμάτια, όσος και ο αριθμός των MPI διεργασιών που θα επιλέγει ο χρήστης. Δηλαδή, στην περίπτωση που έχουμε 50 σημεία και 4 MPI διεργασίες τότε κάθε διεργασία θα πάρει απο 50/4 = 12 σημεία. Τα 2 σημεία που περισσεύουν τα λαμβάνει η τελευταία διεργασία (στην περίπτωση αυτή η 4η) ώστε να μην χάνονται σημεία όταν είναι ατελής η διαίρεση του αριθμού των σημείων με τον αριθμό των διεργασιών MPI. Για αυτό το διαχωρισμό είναι υπεύθυνη η root process η οποία έχει σκοπό πέρα από την εκτέλεση των υπολογισμών των δικών της σημείων, το διαμοιρασμό των κατάλληλων buffer στις υπόλοιπες διεργασίες. Τα σημεία αυτά προέρχονται είτε απο random initialization όπως δινόταν (για testing) είτε απο το αρχείο .mat που δόθηκε.

Έτσι λοιπόν σε κάθε επανάληψη εκτελούνται παράλληλα οι υπολογισμοί σε κάθε διεργασία. Με το πέρας κάθε επανάληψης, η συνολική απόσταση (

new_SumOfDist), τα buffers των memberships (data_in.members , clusters_members) και το buffer των νέων κέντρων (newC) αθροίζονται σε ένα το οποίο και λαμβάνει η root process. Από κει και πέρα αυτή υπολογίζει τα νέα κέντρα και διαμοιράζει στις υπόλοιπες διεργασίες και ελέγχει αν η μεταβολή της απόστασης ήταν μικρότερη από το threshold. Αν ναι, τότε οι υπολογισμοί έχουν ολοκληρωθεί, άρα οι επαναλήψεις σταματούν και γίνεται αποθήκευση και εμφάνιση των αποτελεσμάτων από την root process. Αν η μεταβολή της απόστασης ήταν μεγαλύτερη του threshold τότε η διαδικασία επαναλαμβάνεται με τον ίδιο τρόπο (μέγιστος αριθμός επαναλήψεων).

Πιο αναλυτικά στον κώδικα

int main(int argc, char **argv) : Η κύρια συνάρτηση του προγράμματος στην οποία έχουν ενσωματωθεί και οι συναρτήσεις υπολογισμού.

Στην αρχή γίνεται έλεγχος για τα arguments από την κονσόλα. Εάν είναι λανθασμένα τυπώνεται το αντίστοιχο μήνυμα.

Με την εντολή MPI_Init(&argc, &argv) γίνεται η εκκίνηση του περιβάλλοντος MPI με τον καταλληλο αριθμό MPI διεργασιών όπως δόθηκε από την κονσόλα. Ακολουθεί η δήλωση των δομών data_in και clusters και η δυναμική παραχώρηση μνήμης σε αυτές ανάλογα με το μέγεθος τους το οποίο καθορίζεται ανάλογα με τον αριθμό στοιχείων, clusters, διαστάσεων και διεργασιών MPI.

Στην root process γίνεται η προσπέλαση του αρχείου και τα δεδομένα του αποθηκεύονται στη μνήμη της. Με χρήση της εντολής MPI_Bcast και των MPI_Send

η root process στέλνει τα απαραίτητα δεδομένα στις υπόλοιπες διεργασίες. Αυτές χάρη στην MPI_Recv λαμβάνουν το σωστό κομμάτι του buffer που αντιστοιχεί στα σημεία τα οποία πρόκειται να επεξεργαστούν. Πριν το κομμάτι των υπολογισμών ορίζονται 3 νέα buffers (dp, cp, newC) τα οποία θα χρησιμοποιηθούν στις συναρτήσεις MPI_Reduce.

Οι υπολογισμοί είναι ίδιοι με αυτούς του αρχείου kmeans.c με μερικές διαφορές οι οποίες οφείλονται στο ότι επεξεργαζόμαστε μέρος των συνολικών buffers παράλληλα και έτσι πρέπει να συγκεντρώνουμε σε κάθε επανάληψη τα αποτελέσματα στην root process και αν χρειαστεί να μεταφέρουμε κάποια και στις υπόλοιπες διεργασίες.

- Υπολογίζετε αντίτης new_SumOfDist μια part_distance που αναφέρεται στην συνολική απόσταση που απέχουν τα σημεία της συγκεκριμένης διεργασίας από τα κέντρα τους.

- Όλες αυτές οι part_distance αθροίζονται μέσω της MPI_Reduce με σκοπό η root process να συγκρίνει την μεταβολή στην απόσταση σε αυτό το βήμα.

- To buffer dp το οποίο έχει σε όλες τις διεργασίες το ίδιο μήκος, περιέχει τιμές μόνο στις θέσεις που αντιπροσωπεύουν τα σημεία της εκάστοτε διεργασίας. Με την MPI_Reduce σχηματίζεται στην root process το συνολικό buffer με τα memberships των στοιχείων.
- To buffer cp με μέγεθος όσα και τα clusters στα οποία χωρίζουμε τα σημεία. Αυτό περιέχει τον αριθμό των στοιχείων που ανήκουν στο κάθε cluster. Με την MPI_Reduce σχηματίζεται στην root process το συνολικό buffer με τον αριθμό των μελών του κάθε cluster.

- Το buffer newC με μέγεθος όσα και τα clusters επίτον αριθμό των διαστάσεων των σημείων. Χρησιμοποιείται για το μερικό άθροισμα των συντεταγμένων και με την MPI_Reduce στέλνεται στην root process η οποία στη συνέχεια υπολογίζει τις νέες συντεταγμένες των κέντρων των clusters. Αυτές βέβαια τις χρειάζονται και οι υπόλοιπες διεργασίες στην περίπτωση που χρειαστεί κι άλλη επανάληψη. Με την MPI_Bcast επιτυγχάνεται η αποστολή του buffer των νέων συντεταγμένων από την root process στις υπόλοιπες διεργασίες.

- Η συνθήκη if(fabs(SumOfDist - new_SumOfDist)<threshold) ελέγχει αν η συνολική απόσταση των σημείων από τα κέντρα τους έχει μειωθεί κάτω απο μια τιμή threshold. Αν ναι όπως είπαμε και παραπάνω σταματάει τους υπολογισμούς και προχωράει στην εμφάνιση των αποτελεσμάτων. Αν όχι επαναλαμβάνει τους υπολογισμούς για την μείωση της απόστασης αυτής.

Είτε σταματήσουν οι υπολογισμοί λόγω της μικρής μείωσης της απόστασης είτε εάν εκτελεστεί ο μέγιστος αριθμός επαναλήψεων, τα τελικά αποτελέσματα βρίσκονται στην root process η οποία και τα εμφανίζει και στη συνέχεια τα αποθηκεύει σε αρχεία. Τέλος, με την MPI_Finalize τερματίζεται το περιβάλλον MPI και τελειώνει το προγράμμα μας.

Μετατροπή των Δεδομένων

Να σημειωθεί ότι τροποποιήθηκε η συνάρτηση mat2bin.m για την μετατροπή των δεδομένων με σκοπό να παραχθεί το .bin αρχείο με το buffer της μορφής που εξυπηρετείστο πρόγραμμάμας, όλα τα σημεία και οι διαστάσεις τους σε μία σειρά, [1x(812900x34)]. Η νέα mat2bin.m επισυνάπτεται στην εργασία.

Έλεγχος Ορθότητας

Για μια σχετική επαλήθευση των αποτελεσμάτων δοκιμάστηκε το clustering των σημείων του δοσμένου αρχείου με τον σειριακό κώδικα του kmeansTest.c. Τα σημεία φαίνεται και στις δύο υλοποιήσεις να συγκεντρώνονται σε ένα cluster ωστόσο υπάρχει μια μικρή διαφοροποίηση στον αριθμό των σημείων που συγκεντρώνονται στο ‘κύριο’ cluster ανάμεσα στο σειριακό και το παράλληλο πρόγραμμα.

Απόδοση

Η ταχύτητα εκτέλεσης δοκιμάστηκε στον προσωπικόμου υπολογιστή MacBook Pro με 2 πυρήνες 2.66 GHz σε περιβάλλον Mac OSX 10.8. Επίσης δοκιμάστηκε και στον Διάδηαλλάμάλλον λόγω φόρτου εργασίας το Σάββατο και την Κυριακή αργούσε υπερβολικά και τα αποτελέσματα ήταν σίγουρα λανθασμένα.

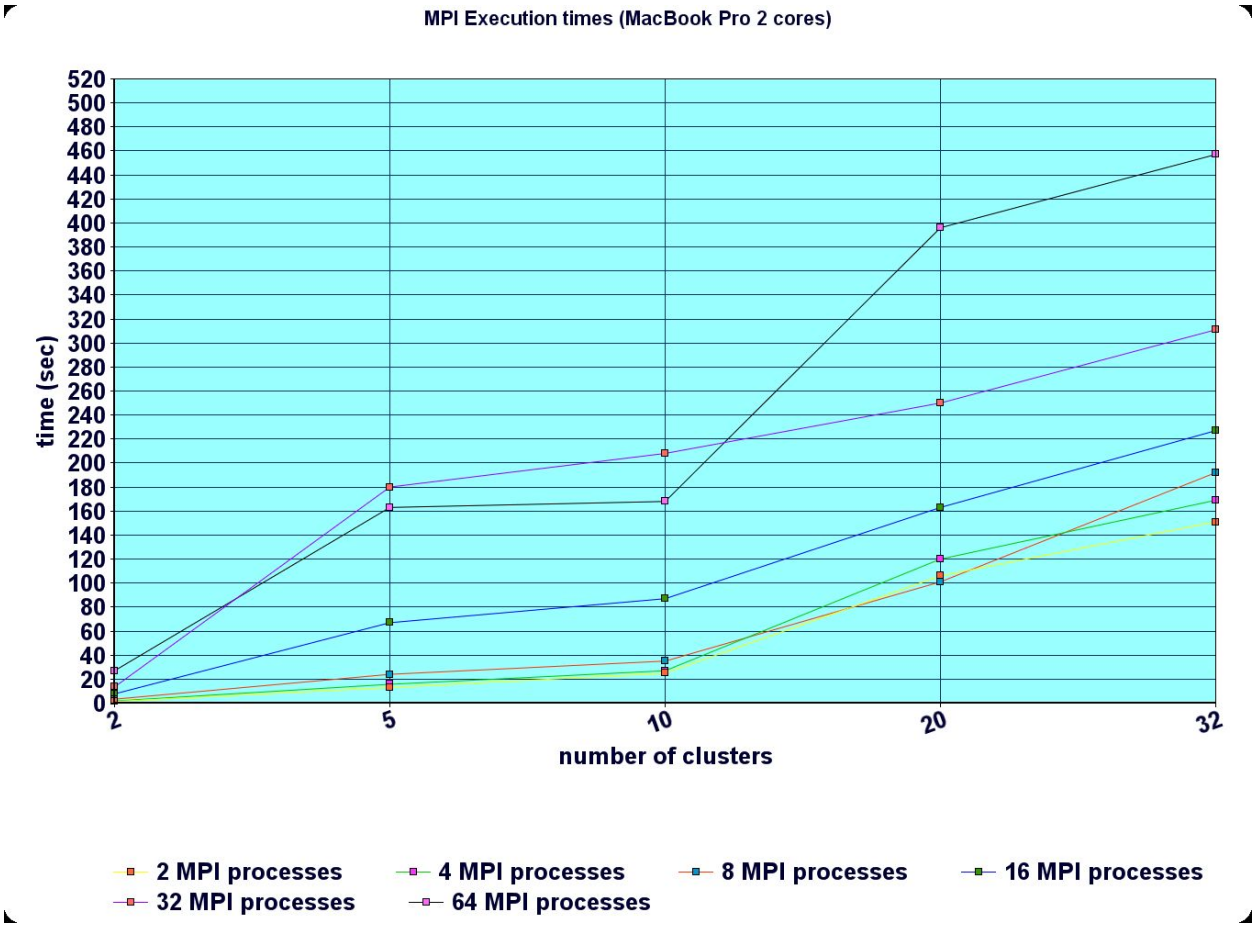
Ένας πίνακας με τους χρόνους (Οριζόντια ο αριθμός των MPI διεργασιών και κάθετα ο αριθμός των clusters):

\	2	4	8	16	32	64
2	1.4	1.8	3.3	7.7	13.5	27

5	13	15.7	24	67	180	163
10	25	27	35	87	208	168
20	106	111	103	163	250	396
32	167	169	192	227	311	457

Οι τιμές στα κελιά είναι οι χρόνοι εκτέλεσης των υπολογισμών σε δευτερόλεπτα.

Και ένα συγκεντρωτικό διάγραμμα με τους χρόνους:



Σχόλια:

Παρατηρούμε ότι με την παράλληλη υλοποίηση του k-means clustering κερδίζουμε σε χρόνο έναντι της σειριακής υλοποίησης καθώς η δεύτερη για 2 clusters απαιτεί χρόνο πάνω απο 5 δευτερόλεπτα και για 32 clusters πάνω απο 11 λεπτά (660 δευτ). Βλέπουμε όμως ότι η παράλληλη υλοποίηση δεν συμφέρει πάντα αφού για 2 custers με χρήση 64 MPI διεργασιών απαιτεί 27 (!) δευτερόλεπτα, πολύ περισσότερο από την σειριακή.
Συμπεραίνουμε λοιπόν ότι έχει σημασία και ο αριθμός των MPI διεργασιών που χρησιμοποιούμε ανάλογα με το σύστημα και τους πόρους που έχουμε στη διάθεσή μας. Για την παρούσα προσωμοίωση φαίνεται ότι τα βέλτιστα αποτελέσματα καταγράφηκαν για 2 MPI διεργασίες (πιθανότατα λόγω των 2 πυρήνων στο σύστημα).