ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

Project 3 – Multiple Sort Merge Join

**Όνομα:** Ελευθέριος Δημητράς **ΑΜ:** 1115201600042

**Όνομα:** Μιχαήλ Ξανθόπουλος **ΑΜ:** 1115201600119

Εντολή Εκτέλεσης: ./main -w ./workloads/small/ -q small.work < ./workloads/small/small.init

Αν προστεθεί -s τότε εκτελούμε το πρόγραμμα με την χρήση στατιστικών

Σύνοψη:

Γενικά στο πρόγραμμα υλοποιούμε το SortMergeJoin διαφόρων πινάκων που δίνονται με βάση

κάποια δοθέντα queries . Ο αλγόριθμος μπορεί να τρέξει βασιζόμενος σε κάποια στατιστικά

που παράγονται κατά την διάρκεια εκτέλεσης του προγράμματος . Ο αλγόριθμος χρησιμοποιεί σε μεγάλο βαθμό την “ενδιάμεση μνήμη” όπου αποθηκεύονται όλα τα αποτελέσματα των joins που γίνοται με σκοπό να χρησιμοποιηθούν αργότερα . Η ενδιάμεση μνήμη δουλεύει έχοντας μια λίστα όπου κάθε της κόμβος περιέχει μία λίστα με πίνακες που συμμετέχουν στα ενδιάμεσα αποτελέσματα της εκάστοτε πράξεις . Αυτοί οι πίνακες-κόμβοι έχουν αποθηκευμένα τα rowIDs του σε υλοποιημένο vector. Τέλος τα αποτελέσματα του κάθε batch επιστρέφονται στην main από τα threads και εκτυπώνονται . Έπειτα προχωρούμε στην εκτέλεση του επόμενου batch. Τέλος αποδεσμεύουμε την μνήμη που κάναμε allocate και τερματίζουμε το πρόγραμμα.

Διευκρινήσεις Merge:

H merge παίρνει σαν ορίσματα τους δύο ταξινομημένους πίνακες όπως αυτοί επεστράφησαν από την TableSort(). Χρησιμοποιείται ένα δείκτης στον πίνακα Α και δύο δείκτες στον πίνακα Β. Ο ένας εκ των δύο του Β είναι pinned στο 1ο από μια λίστα με ίδια κλειδιά και ο 2ος κάνει το traversal στη λίστα αυτή. Μόλις ο δείκτης του Α και ο 2ος δείκτης του Β δείχνουν σε ίδιο κλειδί, το εισάγουμε στη λίστα. Αν το κλειδί του Α είναι μεγαλύτερο από του Β, μετακινούμε τον pinned δείκτη στη λίστα με τα αμέσως μεγαλύτερα κλειδιά του Β. Τέλος, αν μετακινήσουμε το δείκτη του Α στο επόμενο κλειδί και αυτό είναι ίδιο με το προηγούμενο, επαναφέρουμε το 2ο δείκτη του Β στη θέση του pinned δείκτη, αλλιώς μετακινούμε τον pinned στη θέση του 2ου.

Διευκρινήσεις Λίστας:

Η λίστα έχει υλοποιηθεί με templates και η λειτουργικότητά της είναι η εξής. Κάθε κόμβος, αποτελείται από ένα Bucket, το οποίο έχει ένα σταθερό μέγεθος χώρου, καθορισμένο από το χρήστη, στο οποίο θα αποθηκεύονται τα δεδομένα. Τα δεδομένα μπορούν να είναι οποιουδήποτε τύπου επιλέξει ο χρήστης και θα πρέπει να είναι σταθερού μεγέθους.

Η αποθήκευση των στοιχείων γίνεται σε έναν δυναμικά δεσμευμένο πίνακα που εξυπηρετεί στη γρήγορη προσπέλαση των στοιχείων. Θα πρέπει το μέγεθος του Bucket να είναι τουλάχιστον όσο το μέγεθος ενός από τα στοιχεία που πρόκειται να αποθηκευτούν σε αυτήν.

Για βελτίωση της χρήσης της μνήμης, δε δεσμεύεται το ακριβές μέγεθος Bucket που επιλέγει ο χρήστης, αλλά το μέγιστο δυνατό πολλαπλάσιο μνήμης των στοιχείων που πρόκειται να αποθηκευτούν. Για παράδειγμα, αν ο χρήστης ζητήσει Bucket size 100 Bytes και ο χρήστης θέλει να αποθηκεύσει μια δομή με μέγεθος 51 Bytes, θα δεσμευτούν 51 Bytes και όχι 100. Έτσι, σώζουμε 49 Bytes ανά Bucket.

Διευκρινήσεις Sort:

Το sorting δουλεύει χρησιμοποιώντας δυο πίνακες οι οποίοι εναλλάσονται μεταξύ των αναδρομικών κλήσεων της συνάρτησης SimpleSortRec(). Η SimpleSortRec() αρχικά δημιουργεί το ιστόγραμμα και τον πίνακα psum όπως αναγράφεται στην εκφώνηση. Έπειτα χρησιμοποιεί το table1 ως R και το table2 ως R’, δηλαδή σε κάθε αναδρομική κλήση ταξινομεί τον πίνακα R χρησιμοποιώντας το psum και γράφει τα αποτελέσματα στον R’ . Αφού τελειώσει αυτή η διαδικασία αντιγράφουμε τα αποτελέσματα (που είναι ταξινομημένα) στον R. Όταν σε κάποια αναδρομική κλήση τα δεδομένα του δοθέντος R (ο οποίος αποτελεί ένα από τα buckets της αρχικής κλήσης), γίνουν στο πλήθος μικρότερα από 8.192 δηλαδή 64KB τότε ο R ταξινομείται με quicksort. Έτσι τελικά, μόλις τελειώσει η SimpleSortRec(), έχουμε ταξινομημένο τον πίνακα που δόθηκε στην αρχική κλήση της συνάρτησης.

(Είναι ταξινομημένοι και ο R και ο R’)

Διευκρινήσεις ComparisonPredicate:

Το **comparision** **predicate** δουλεύει χρησιμοποιώντας ένα ***struct compPred*** το οποίο περιέχει το comparison query. Αν ο πίνακας υπάρχει στην ενδιάμεση μνήμη με κάποια μορφή τότε χρησιμοποιείται αυτή η μορφή για το predicate, αλλιώς χρησιμοποιείται ο αρχικός πίνακας.

Διευκρινήσεις JoinPredicate:

Το **join predicate** δουλεύει χρησιμοποιώντας τις συναρτήσεις της προηγούμενης άσκησης. Αν οι δύο πίνακες που δίνονται υπάρχουν στην ενδιάμεση μνήμη με κάποια μορφή τότε χρησιμοποιείται αυτή η μορφή για το predicate, αλλιώς χρησιμοποιούνται οι αρχικοί πίνακες.

Διευκρινήσεις JoinSelf:

Το **join self** χρησιμοποιείται όταν πρέπει να γίνει ***join*** μεταξύ δύο στηλών ενος πίνακα. Το join γίνεται συγκρίνοτας ανα γραμμή οι δοθείσες στείλες και κρατώντας μόνο αυτές που έχουν ίδιο κλειδί. Αν αυτός ο πίνακας υπάρχει στην ενδιάμεση μνήμη με κάποια μορφή τότε χρησιμοποιείται αυτή η μορφή για το predicate, αλλιώς χρησιμοποιείται ο αρχικός πίνακας.

Διευκρινήσεις JoinInSameBucket:

Αυτή η συνάρτηση χρησιμοποιείται όταν θέλουμε να κάνουμε ***join*** μεταξύ δύο πινάκων οι οποίοι υπάρχουν ήδη στο **ίδιο** bucket της ενδιάμεσης μνήμης. Το ***join*** γίνεται θεωρώντας ότι έχουμε έναν ενιαίο ενδιάμεσο πίνακα από τον οποίο αρχικά παίρνουμε την πρώτη γραμμή όπου σε αυτήν υπάρχουν τα **rowIDs** του καθένα από τους δύο πίνακες. Έτσι, αν τα στοιχεία στα οποία οδηγούν τα **rowIDs** είναι ίδια, κρατάμε αυτήν την γραμμή, αλλιώς την σβήνουμε.

*Οι βοηθητικές συναρτήσεις* ***TableExistsInMidStruct*** *και η* ***CreateTableForJoin*** *χρησιμοποιούνται για την εύρεση ή δημιουργία πινάκων για την εκτέλεση του join.*

*Τέλος, οι συναρτήσεις που καθορίζουν ποια συνάρτηση* ***comparion*** *ή* ***join*** *θα κληθεί είναι οι* ***DoAllJoinPred*** *και* ***DoAllCompPred****.*

***Structs που χρησιμοποιήσαμε :***

Stats→ αποθηκεύουμε όλα τα στατιστικά και τον bool πίνακα από τον οποίο υπολογίζονται τα distinct values

RelationTable→ αποθηκεύουμε τον εκάστοτε πίνακα που μας δίνεται και τα στατιστικά στοιχεία για αυτόν . Επίσης αποθηκεύονται και βασικές πληροφορίες για τον πίνακα.

JoinPred→ αποθηκεύουμε τα στοιχεία των join predicates που μας δίνονται . (IDs των δύο πινάκων και στηλών)

CompPred→ αποθηκεύουμε τα στοιχεία των comparison predicates που μας δίνονται.(ID πίνακα , στήλη , τύπος σύγκρισης “<,>,=” και αριθμού συγκρίσεως)

TableStats→ αποθηκεύουμε στατιστικά στοιχεία για έναν πίνακα που μας δίνεται

JoinHashEntry→ χρησιμοποιείται από τις συναρτήσεις στατιστικών για την αποθήκευση των predicates , tableIDs , στατιστικών και κόστους

Projection→ αποθηκεύει τον πίνακα και στήλη

Query→ αποθηκεύουμε δείκτες σε όλους τους πίνακες που χρησιμοποιούνται στο εκάστοτε query και δύο λίστες μία για τα join predicates και μια για τα comparison predicates

FullResList → αποτελούν κόμβους για την ενδιάμεση μνήμη

***Δομές με χρήση templates:***

Λίστα , vector , hashmap

***Γενικές παρατηρήσεις :***

Αρχική υλοποίηση → Αρχικά χρησιμοποιούσαμε λίστες αντί για vectors στην ενδιάμεση μνήμη και το small τελείωνε σε 12 περίπου λεπτά . Οι λίστες έκαναν πολύ αργή την υλοποίηση και έτσι αλλάξαμε σε vectors.

Υλοποίηση με vectors → Αλλάζοντας τις λίστες σε vectors το small τελείωνε σε περίπου 11 δευτερόλεπτα ενώ το medium σε 13:30 περίπου

Προσθήκη παραλληλοποίησης → Μετά την προσθήκη threads είδαμε σημαντικές βελτιώσεις καθώς το small τελείωνε σε 7-8 δευτερόλεπτα και το medium τελειώνε

σε 8:30-9 λεπτά

Προσθήκη στατιστικών → Προσθέτοντας στατιστικά δεν είδαμε σημαντικές βελτιώσεις και πρέπει να πούμε ότι και στις δύο περιπτώσεις workload είχαμε

αύξηση στον χρόνο εκτέλεσης. Γενικά το small εκτελείται σε 9:30-10 δευτερόλεπτα ενώ το medium έφτασε τα 12 λεπτά

***Παρατηρήσεις σχετικά με την χρήση threads:***

Γενικά με την χρήση threads είδαμε σημαντική βελτίωση στην απόδοση του προγράμματος πιο συγκεκριμένα :

2 threads για queries , 2 για sort , 2 για merge