Project part 3

Ορέστης Γιαννούκος sdi1600035
Γιώργος Λιακόπουλος sdi1600091
Κωσταντίνος Χασιαλής sdi1600195

1) Σχεδιαστικές επιλογές:

Κύριες Δομές :

Stretchy Buffer: Ο stretchy buffer αποτελεί μια δομή η οποία λειτουργεί όπως ένας vector στην cpp. Για την αρχικοποίηση ενός stretchy buffer ορίζουμε έναν δείκτη στον τυπο που θέλουμε να έχει ο πίνακας και του αναθέτουμε την τιμή null. Στην συνέχεια με την συνάρτηση buf_push μπορούμε να εισάγουμε στοιχεία στην τελευταία θέση του πίνακα, ενώ με την χρήση του τελεστή [ ] μπορούμε με index να πάρουμε το i-οστο στοιχείο του πίνακα. Ακόμα υπάρχουν οι συναρτήσεις:

- buf_free για την αποδέσμευση της μνήμης του πίνακα
- buf_len η οποία επιστρέφει το μέγεθος του πίνακα
- buf_remove η οποία αφαιρεί βάση του index κάποιο στοιχείο του πίνακα
- buf_cap η οποία επιστρέφει το μέγιστο πλήθος στοιχείων που μπορεί να κρατήσει ο πίνακας την συγκεκριμένη στιγμή

Best Tree: Το Best Tree ειναι μια δομή που λειτουργεί σαν dictionary. Το key είναι της μορφής char* και το value είναι ένα δέντρο μέσα στο οποίο στα leaf nodes βρίσκονται relations ενώ στους εσωτερικούς κόμβους βρίσκονται τα joins μεταξύ των relation. Για την υλοποίηση του χρησιμοποιούμε ένας hash table. Οι συναρτήσεις του best tree είναι:

- BestTree η οποία βάση του σετ το οποίο έχει κωδικοποιηθεί σε ένα string επιστρεφει το καλυτερο δυνατο δεντρο
- BestTree_set η οποία αντιστοιχεί σε ένα σετ ένα δέντρο
- BestTree_delete η οποία διαγραφεί το dictionary

Αλγοριθμικά:

Sort: Για την υλοποίηση της σορτ χρησιμοποιήσαμε μια ουρά. Σε αυτή την ουρά περνάμε τους πίνακες με τα ριλεσιον, το τρέχων ιστόγραμμα και psum, το πλήθος των στοιχείων που προκειται να ταξινομηθουν καθώς και ένα οφσετ βάση του οποίου μπορούμε να γράψουμε στους πίνακες με τα relation. Η παραπάνω εισαγωγή αποτελεί ένα bucket.

Στην συνέχεια μπαίνουμε σε ένα loop μέσα στο οποίο για κάθε byte ταξινόμησης ακολουθείται η παρακάτω διαδικασία. Για κάθε μπακετ που έχει σχηματιστεί από το προηγούμενο βήμα αν το πλήθος των στοιχείων είναι μικρότερο των 32kb τότε εκτελείται quick sort και βάση του offset τοποθέτειται τα relation στην κατάλληλη θέση του πίνακα αλλιώς σπάσε το bucket σε μικρότερα buckets βάση του byte ταξινόμησης, τοποθέτησε τα στην ουρά και κάνε την αρχική διαδικασία. Προκειμένου να μην γίνει λάθος εγγραφή των δεδομένων στον πίνακα του relation κατα τον υπολογισμό του psum χρησιμοποιούμε δυο πίνακες για την αποθήκευση των relation στους οποίους γράφουμε ανάλογα με το αν το byte ταξινομησης ειναι μονο ή ζυγο.

2) Multithreading:

Έχουμε μια ουρά που σαν members εχει εναν δείκτη σε μία συνάρτηση (το job που έχει αναλάβει), έχει έναν δείκτη στο επόμενο job και τα ορίσματα που απαιτούνται για την κάθε συνάρτηση (void*).

Τα threads μένουν idle οταν δεν υπάρχει job στην ουρά τους και απελευθερώνονται όταν γίνει το pool_destroy.

'Οταν καλείται η thr_pool_create δεν δημιουργούνται αύτοματα τα threads παρά αρχικοποιεί ο scheduler τις δομές που χρησιμοποιεί (ουρα, mutex, τα conditional variables, πίνακα από threads). Threads δημιουργούνται μόνο όταν γίνει push στην ουρά.

Ο συγχρονισμός γίνεται ως εξής:
Αρχικά με την δημιουργία του κάθε νήματος μπαίνει σε while(1) βρόγχο και περιμένει, αν η ουρά του έχει αδειάσει.
Επίσης περιμένει αν η κατάσταση του scheduler είναι "wait" και δεν έχει δουλειά να εκτελέσει.
Ο scheduler μπορεί να έχει 3 καταστάσεις, "destroy", "wait" και "running"

Η κατάσταση destroy σημαίνει ότι πρέπει αφού όλα τα threads ολοκληρώσουν τις δουλείες τους να γίνει cancel αυτών και να απελευθερωθεί ομαλά η μνήμη που είχε δεσμευτεί.

Η κατάσταση wait σημαίνει ότι ο scheduler περιμένει όλα τα threads να ολοκληρώσουν τις δουλειές τους για να πάρει τα αποτελέσματα στην σειρά που τα θέλει.

Όταν είναι σε κατάσταση wait τα threads που δεν έχουν δουλειά περιμένουν πάνω σε ενα conditional variable pool_barrier.

Γίνεται επίσης η χρήση ενός mutex για access πάνω στις δομές, και αλλά 2 conditional variables.
Το πρώτο (pool_queue_empty) δηλώνει ότι η ουρά με τις δουλείες είναι άδεια και πρέπει τα threads να περιμένουν.
Το δεύτερο (pool->pool_cleanup) δηλώνει ότι πρέπει για να καταστρέψει ο scheduler το pool να περιμένει να κάνουν cleanup όλοι οι workers.

Η ουρά με τις δουλειές είναι μια απλά συνδεδεμένη λίστα με δείκτες στο τέλος και στην αρχή για O(1) εισαγωγή και διαγραφή.
Σαν μέλη της έχει έναν δείκτη σε συνάρτηση (το job), έναν δείκτη στα ορίσματα του job και έναν δείκτη στο επόμενο job.

Multithread sort-merge : Εφόσον χρησιμοποιούμε μη-αναδρομική sort και κρατάμε σε μια ουρά τα base, size των ιστογραμμάτων και τον psum είναι εύκολο να γίνει ο διαχωρισμός των buckets και η επιστροφή αυτών στην merge. Παρόλαυτα, μπορεί η sort να σπάσει σε 2 buckets τους αριθμούς και να είναι μικρή η σχέση οπότε να τα κάνει quicksort απευθείας. Σε τέτοιες περιπτώσεις αποφεύγουμε την δημιουργία των threads καθώς το overhead απλά θα υπερκάλπυτε μια quicksort με δεδομένα που χωράνε στην cache.

Επίσης παρατηρήσαμε ότι στα δεδομένα που δόθηκαν το σπάσιμο σε παραπάνω από 1 bucket γίνεται μετά από το k-byte, k != 1. Οπότε προχωράμε με 1 thread όσο σπάει σε 1 bucket και αν την πρώτη φορά σπάσει σε έναν ικανοποιητικό αριθμό από buckets (π.χ. > 20) τότε δημιουργούμε threads και αναλαμβάνουν αυτά να εκτελέσουν το sort, το καθένα ένα πλήθος από buckets. Μετά η merge χρησιμοποιεί αυτά τα ίδια buckets για να εκτελέσει και η ίδια παράλληλα το join.
Συνεπώς ο πολυνηματισμός της sort-merge εξαρτάται από το πλήθος των δεδομένων.
Αν γίνει multi-threaded η sort, θα γίνει και η join αλλιώς καμία από τις 2.

Εισάγαμε επίσης μια νέα μορφή πολυνηματισμού, το να παραλληλοποιήσουμε τις κλήσεις των 2 sort πριν από κάθε join.

Έχουμε επίσης την μορφή πολυνηματισμού για queries.

Για κάθε μία από αυτές τις 3 μορφές δεσμεύουμε 2-3 threads εκτός αν δεν οριστούν κάποιες μορφές οπότε ορίζουμε 7 threads για τα queries μόνο.

Για παράδειγμα : Έστω ότι κάνουμε πολυνηματισμό στην sort-merge (εσωτερικά) και στα queries. Τότε δεσμεύουμε 3 threads για τα queries και 2 για την sort-merge.

Έστω ότι κάνουμε πολυνηματισμό μόνο στα queries, τότε δίνουμε 7 threads στα queries, κλπ.

Παρακάτω παρουσιάζουμε σχέσεις χρόνου εκτέλεσης - μνήμης.

Περίπτωση 1 - Όλα single-threaded

Χρόνος εκτέλεσης small : 1.205s
Συνολική μνήμη small : 2,043mb
Max μνημη small: 404mb

Χρόνος εκτέλεσης medium : 133.8s
Μνήμη medium : 77gb
Max μνημη medium: 9gb

Περίπτωση 2 - Παραλληλία μόνο σε επίπεδο queries

Χρόνος εκτέλεσης small : 1.231s
Συνολική Μνήμη small : 2,050mb
Max μνήμη small: 410mb

Χρόνος εκτέλεσης medium : 177,83s
Μνήμη medium : (bytes)
Max μνημη medium
(σε 4 threads τελειωνει η μνημη, μετρησεις σε)

Περίπτωση 3 - Παραλληλία σε επίπεδο queries, sort-merges
Χρόνος εκτέλεσης small : 1.352s
Συνολική Μνήμη small : 2,267gb
Max μνήμη small: 410mb

Χρόνος εκτέλεσης medium : 315.53s
Μνήμη medium : (bytes)
Max μνημη medium:

Περίπτωση 4 - Παραλληλία σε επίπεδο queries, sort-merge, sort_call (εκτέλεση 2 παράλληλων sort)

Χρόνος εκτέλεσης small : 1,523s
Συνολική Μνήμη small : 2,340mb
Max μνήμη small: 410mb

Χρόνος εκτέλεσης medium : 305.54s
Μνήμη medium : (bytes)
Max μνημη medium :
(τελείωσε η μνήμη με όλους τους συνδυασμούς)

Περίπτωση 5 - Παραλληλία σε επίπεδο sort-merge, sort_call

Χρόνος εκτέλεσης small : 1,776s
Συνολική μνήμη small: 2,390mb
Max μνημη small: 404mb

Χρόνος εκτέλεσης medium : 357.15s
Μνήμη medium : (bytes)
Max μνημη medium :

Περίπτωση 6 - Παραλληλία μόνο σε επίπεδο sort_merge

Χρόνος εκτέλεσης small : 1,633s
Συνολική μνήμη small : 2,390mb
Max μνημη small: 410mb

Χρόνος εκτέλεσης medium : 316.16s
Μνήμη medium : (bytes)
Max μνημη medium :

Περίπτωση 7 - Παραλληλία μόνο σε επίπεδο sort_call

Χρόνος εκτέλεσης small : 1,460s
Συνολική μνήμη small : 2,030mb
Max μνημη small: 404 mb

Χρόνος εκτέλεσης medium :293.31
Μνήμη medium : (bytes)
Max μνημη medium :

Περίπτωση 8 - Παραλληλία σε επίπεδο queries, sort_call

Χρόνος εκτέλεσης small : 1,491
Συνολική μνήμη small : 2,030
Max μνημη small: 410mb

Χρόνος εκτέλεσης medium : 189.54s
Μνήμη medium : (bytes)
Max μνημη medium :

Ο υπολογισμός της μνήμης με valgrind ήταν πολύ χρονοβόρος για αυτό δεν υπάρχουν οι μετρήσεις μνήμης στα medium.

Αυτά τα test cases ήταν εύκολο να υλοποιηθούν με ελάχιστα παραπάνω κώδικα (δεν αντιγράψαμε τις συναρτήσεις) χρησιμοποιώντας conditional compilation.

Πιο συγκεκριμένα, ορίζουμε με -D κάποιες σταθερές και κατά την διάρκεια της μεταγλώτισης παράγεται ο κατάλληλος κώδικας.

Για παράδειγμα, για να εκτελεστεί παραλληλία μόνο σε επίπεδο queries αρκεί να ορίσουμε στο Makefile
-DMULTITHREAD_QUERIES και καμία άλλη σταθερά.

Αν θέλουμε παραλληλία σε queries και sort-merge ορίζουμε την -DMULTITHREAD_SORT και την -DMULTITHREAD_QUERIES κλπ.

Βελτιώσεις από την φάση 2 : Στην εργασία της φάσης 2 είχαμε πολλές καθυστερήσεις και χρησιμοποιούσαμε πολύ παραπάνω μνήμη. Για παράδειγμα, το small έκανε 42 δευτερόλεπτα ένω τώρα 1.2 δευτερόλεπτα.

Πιο συγκεκριμένα, αλλάξαμε τον τρόπο που τροποποιούμε τα ενδιάμεσα αποτελέσματα μετά από κάθε join και χρησιμοποιήσαμε διαφορετικό generic array από την φάση 2 διότι το generic array της φάσης 2 χρησιμοποιούσε void * για να αποθηκεύει τα δεδομένα και απαιτούσε παραπάνω χώρο διότι κρατούσε πολλές παραπάνω πληροφορίες και επειδή οι δείκτες του ήταν διάσκορποι στην μνήμη και όχι σε συνεχόμενη σειρά είχαμε πολλά cache miss.

Το νέο array που χρησιμοποιήσαμε αποθηκεύει τα δεδομένα όπως τα συνηθισμένα arrays (σε συνεχόμενη σειρά).

Τα queries εκτελούνται ανά batch.

3) Join enumeration:

Για την υλοποίηση του χρησιμοποιήσαμε τον ψευδοκώδικα που δόθηκε στην εκφώνηση.

Για την κατασκευή των set χρησιμοποιήσαμε συμβολοσειρές στις οποίες μέσω των κατάλληλων συναρτήσεων μπορούμε να εισάγουμε στοιχεία, ταξινομώντας το παράλληλα set από το μικρότερο relation στο μεγαλύτερο, να ελέγξουμε αν ένα relation ανήκει ήδη στο set καθώς και αν αντιγράψουμε κάποιο υπάρχων set.

Για την κατασκευή του δέντρου χρησιμοποιούμε τις συναρτήσεις to_tree και create_join_tree.

Στην πρώτη συνάρτηση δίνουμε ένα relation και αρχικοποιεί βάση αυτού τα στοιχεία του κόμβου του δέντρου που θα χρησιμοποιηθούν αργότερα στην create_join_treeαν ο συγκεκριμένος κόμβος αποτελέσει αριστερό κομβο σε κάποιο δέντρο τζοιν. Στην περίπτωση που αποτελέσει δεξιό κόμβο τότε κατα την κατασκευή του δέντρου τζοιν τα στοιχεία του συγκεκριμένου κόμβου θα διαγραφθουν για εξοικονόμηση μνήμης. Τα στοιχεία που θα έχει ένας κόμβος είναι το πλήθος των relation, predicate, οι πίνακες για τα relation, τα predicates και τα στατιστικά καθώς και κάποιες άλλες μεταβλητές που βοηθούν στην υλοποίηση της δομής καθώς και του υπολογισμού των στατιστικών .

Στην δεύτερη συνάρτηση κατασκευάζουμε έναν εσωτερικό κόμβο ο οποίος κατα την αρχικοποιηση των μεταβλητών του αντιγράφει τα στοιχεία του αριστερού κόμβου και τα ενημερώνει με το καινούριο relation που βρίσκεται στον δεξιά κόμβο. Στο συγκεκριμένο κομμάτι του αλγορίθμου γίνεται έλεγχος μεταξύ όλων των πιθανών συνδυασμών ριλεσιον που βρίσκονται ηδη στο αριστερό κόμβο και θα κάνουν join με relation του δεξιά κόμβου καθώς και κάθε δυνατός συνδυασμός μεταξύ των column των relation αν υπάρξει η περίπτωση δυο ριλεσιον να συνδέονται με παραπάνω από ενα join. Στην συνέχεια ενημερώνονται τα στατιστικά του καλύτερου συνδυασμού καθώς και των υπολοίπων στηλών και διαγράφεται ο κόμβος δεξιού δέντρου αφού όλη η πληροφορία που χρειαζόμαστε βρίσκεται στο τρέχων root node .

Με αυτόν τον τροπο κατασκευάζεται το δέντρο για την επιλογή της καλύτερης σειράς εκτέλεσης των join. Στην συνέχεια διαγράφονται τα δέντρα και η συνάρτηση επιστρέφει το επιθυμητό αποτελεσμα. Για τον υπολογισμό των στατιστικών ακολουθούμε τους τύπους που βρίσκονται στην εκφώνηση.

Για την εκτέλεση του προγράμματος κάνουμε make
και στην συνεχεια (cat small.init && echo “done” && cat small.work) | ./queries
αντίστοιχα για medium.