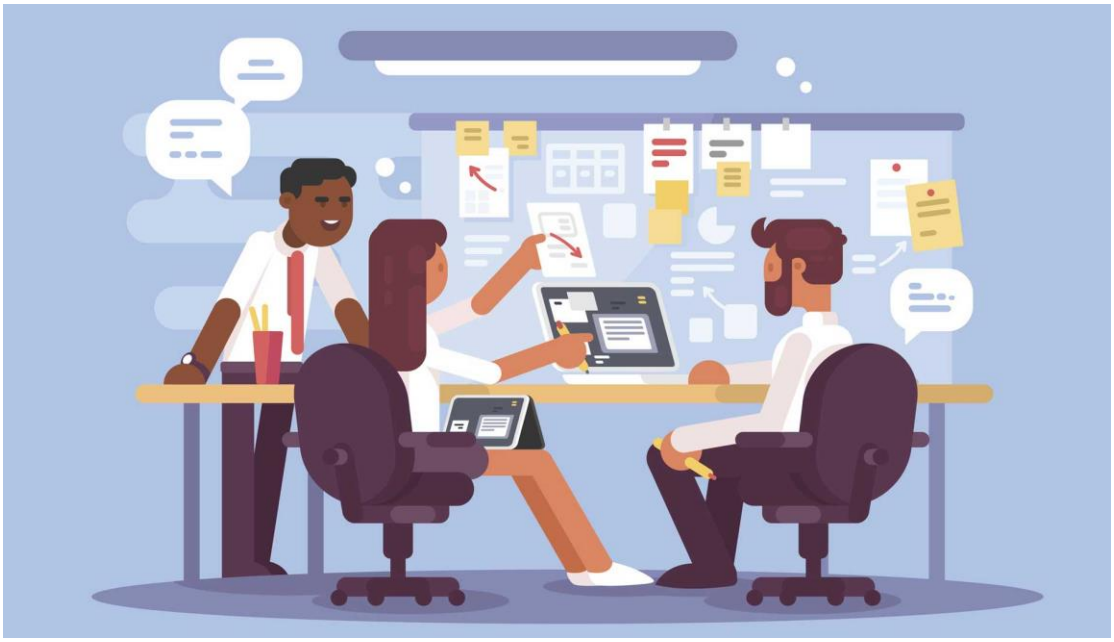
Project: Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα
"Join Implementation"



Ομάδα

Στεφανία Πάτσου 1115201400156

Παναγιώτης-Ορέστης Γαρμπής 1115201400025

Ανδρέας Τσόλκας 1115201400212

Περιεχόμενα

Join Implementation: Part 1

Δομές

Στο πρώτο μέρος της εργασίας, κληθήκαμε να υλοποιήσουμε τον τελεστή ζεύξης με την βοήθεια του Radix Hash Join. Για τον λόγο αυτό, δημιουργήσαμε τις παρακάτω δομές, οι οποίες για την συγκεκριμένη έκδοση στο gitHub, περιέχονται στο αρχείο radixHashJoin.h:

- Tuple: είναι ένα ζευγάρι από το id της γραμμής και την τιμή της συγκεκριμένης γραμμής.
- Relation: αποτελείται από πολλά ζευγάρια tuple και έναν θετικό αριθμό για το μέγεθος του.
- ResultElement: αποτελείται από 2 ids, τα οποία δείχνουν πως για διαφορετικές σχέσεις, το καθένα στο κάθε id, έχουν ίδια τιμή. Χρησιμοποιείται για την αποθήκευση των τελικών αποτελεσμάτων.
- ResultNode: αποτελείται από πίνακα από ResultElement structs, απεικονίζοντας τα τελικά αποτελέσματα του Radix Hash Join με συγκεκριμένη χωρητικότητα.
- Result: μία λίστα αποτελούμενη από ResultNode structs, την οποία χρησιμοποιούμε για τα τελικά αποτελέσματα του Radix Hash Join.

Στην συνάρτηση main.c, εκτός από διάφορα παραδείγματα για το πως εκτελούνται κάποιες συναρτήσεις, κύρια υλοποίηση ήταν η δημιουργία δύο σχέσεων με τυχαίες τιμές, μέσω του rand. Έπειτα, υπήρχε επιλογή εκτύπωσής τους για να φαίνεται πως δημιουργήθηκαν χωρίς προβλήματα. Για την κάθε σχέση επιλέγαμε την κολώνα με την οποία θα εκτελούσαμε το Radix Hash Join. Στην συνέχεια, και για κάθε σχέση, δημιουργούσαμε το ιστόγραμμα, το αθροιστικό ιστόγραμμα από το ιστόγραμμα και τέλος την ανακατανεμημένη σχέση. Πριν την εκτέλεση του Radix Hash Join, αρχικοποιούσαμε την λίστα για τα αποτελέσματα «result», την οποία δίναμε σαν όρισμα, έτσι ώστε να αποθηκευτούν σε αυτή τα ευρεθέντα row ids με την ίδια τιμή. Έπειτα, καθαρίζουμε την σωρό μας, για να μην έχουμε πρόβλημα με την μνήμη μας.

Αρχεία δομών και συναρτήσεων

Πέρα από την main.c , υλοποιήσαμε και τα εξής αρχεία:

- auxMethods.c : Περιέχει την συνάρτηση επιλογής κολώνας προς Radix Hash Join από μία σχέση (το συγκεκριμένο αρχείο αλλά και συνάρτηση διαγράφονται σε επόμενες εκδόσεις) (getColumnOfArray).
- radixHashJoin.c : Περιέχει όλες τις συναρτήσεις παραγωγής tuples για την δημιουργία ιστογράμματος, αθροιστικού και ταξινομημένου πίνακα σχέσης, αλλά και συναρτήσεις για την αποθήκευση των τελικών αποτελεσμάτων, στην κατάλληλη λίστα «Result».

Radix Hash Join

Η υλοποίηση της Radix Hash Join βασίζεται στην ενδεδειγμένη από την εκφώνηση. Ενώ η δημιουργία του Histogram και του Psum είναι σχετικά απλή, η αντίστοιχη του Rordered είναι κάπως πιο σύνθετη. Για βελτιστοποίηση της πολυπλοκότητας, έτσι ώστε να σαρώνουμε τον πίνακα R μόνο μια φορά, και κάθε στοιχείο του να το αντιστοιχούμε απευθείας στην θέση που πρέπει να μπει στον Rordered, χρησιμοποιούμε έναν βοηθητικό πίνακα RemainHist, ο οποίος και είναι αντίγραφο του Hist. Κάθε φορά που βρίσκουμε ένα στοιχείο με συγκεκριμένο hash value, μειώνουμε τον δείκτη του RemainHist που δείχνει στο αντίστοιχο bucket κατά 1, έτσι ώστε να ξέρουμε πόσα τέτοιου hash value στοιχεία έχουν περάσει στον Rordered, και να πετυχαίνουμε με την βοήθεια του Hist και του Psum το κατάλληλο offset στην λίστα του Rordered. Κατά το join, παίρνουμε το μικρότερο bucket από τους δύο πίνακες R και S, έτσι ώστε τα ευρετήρια που θα δημιουργήσουμε να είναι μικρότερα, και να κερδίσουμε τόσο σε χώρο, όσο και σε χρόνο.

Unit Testing

Προκειμένου να ελέγξουμε τη λειτουργικότητα των συναρτήσεών μας, υλοποιήσαμε μερικά Unit Tests που αφορούν τα δύο αρχεία .c που προαναφέρθηκαν. Πιο συγκεκριμένα, ελέγχθηκε αν ο τρόπος επιλογής κολώνας σχέσης είναι σωστός, αλλά και αν οι συναρτήσεις κατακερματισμού έδιναν σωστά αποτελέσματα κι αν κόμβος «ResultNode» δημιουργείται χωρίς λάθη. Για την υλοποίηση των συγκεκριμένων αλλά και επόμενων Unit Tests, χρησιμοποιήσαμε το framework CuTest. Τα αρχεία που περιέχουν τα δικά μας Unit Tests βρίσκονται σε αυτά με όνομα «CuTest<όνομα αρχείου c>.c». Υπάρχει ξεχωριστό Makefile για τα Unit Tests αλλά και το κυρίως εκτελέσιμο πρόγραμμα.

Εκτέλεση & Στατιστικά

Η εκτέλεση γίνεται με την εντολή: **./joinProgram**

Για την εκτέλεση, δεν υπήρχαν ακόμη κάποια datasets που θα μπορούσαμε να εκτελέσουμε, έτσι ώστε να ελέγξουμε την ορθότητα του Radix Hash Join. Για τον λόγο αυτό, χρησιμοποιήσαμε κάποιες προκαθορισμένες τιμές, οι οποίες είναι αποθηκευμένες στο «radixHashJoin.h» αρχείο και είναι οι εξής:

- BUCKETS = 4, αριθμός των κάδων (2^n, όπου n = το πλήθος των τελευταίων bits για το hash function 1 (H1).
- HEXBUCKETS = 0x3, επιλογή 2 τελευταίων bits για την συνάρτηση κατακερματισμού 1 (H1).
- HASH2 = 8, αριθμός θέσεων πίνακα bucket με την συνάρτηση κατακερματισμού (H2).
- HEXHASH2 = 0x7, για την επιλογή θέσης στον πίνακα bucket.
- ARRAYSIZE = ((1024 * 1024) / 64), μέγεθος κάθε «ResultNode».

- PRINT = 1, λαμβάνει Boolean τιμή, αν είναι 1 εκτυπώνει αυτό που θέλουμε (πιο πολύ για debugging).

Για τον έλεγχο, δημιουργούσαμε σχέσεις, είτε με διαφορετικές τιμές στις γραμμές τους, έτσι ώστε κάποια από αυτά να έχουν αποτελέσματα, είτε καμία τιμή ίδια μεταξύ τους, οπότε και τα τελικά αποτελέσματα ήταν 0 ή όλες οι γραμμές να έχουν ίδιες τιμές, έτσι ώστε στα τελικά αποτελέσματα να εγγραφούν όλες οι γραμμές ως συνδυασμοί. Όσον αφορά τα στατιστικά, δεν υπάρχουν κάποια για το συγκεκριμένο μέρος της εργασίας, διότι δεν δόθηκαν κάποια datasets. Παρ' όλα αυτά, και με βάση τις προκαθορισμένες τιμές, έχουμε:

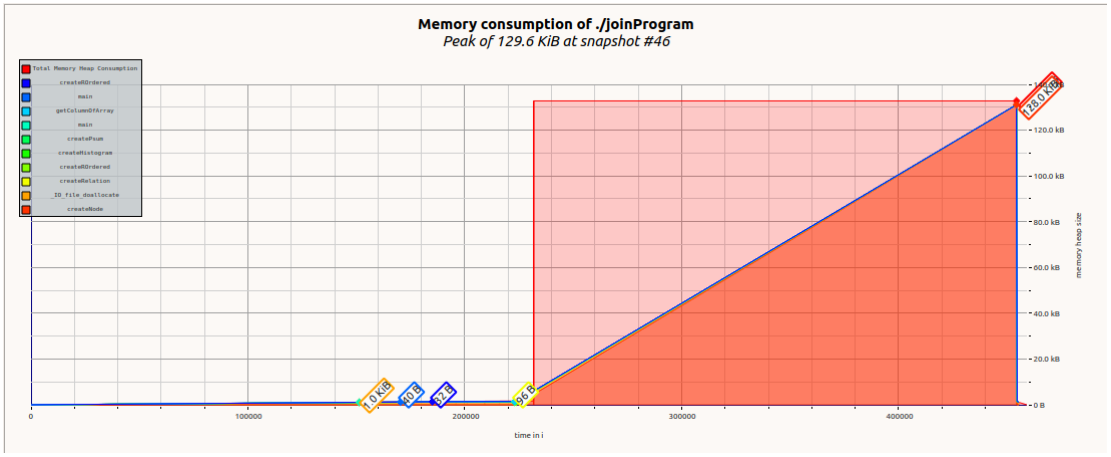
ΑΡΧΕΙΑ ΕΙΣΟΔΟΥ	ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ	ΟΡΘΟΤΗΤΑ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (✓)
κανένα	0.006 sec	✓

Χρονικό διάγραμμα

Παρακάτω φαίνονται πληροφορίες για τα χρονικά ποσοστά που καταλαμβάνει η κάθε συνάρτηση.

Χωρικό διάγραμμα

Παρακάτω παρουσιάζεται η κατανομή του χώρου όσων αφορά τις συναρτήσεις.



Σχόλια Εξεταστή & Διορθώσεις

Τα σχόλια του εξεταστή ήταν θετικά, οπότε δεν υπήρξε κάποια περαιτέρω διόρθωση/βελτίωση.

Join Implementation: Part 2

Δομές

Στο δεύτερο μέρος της εργασίας, υλοποιήσαμε μία λίστα για την καταγραφή των ενδιάμεσων αποτελεσμάτων για κάθε σχέση. Αυτή η λίστα ανανεωνόταν συνεχώς σε κάθε εκτέλεση κατηγορήματος για τις σχέσεις που έπαιρναν μέρος στο Radix Hash Join. Επίσης, δημιουργήθηκαν πολλές άλλες βοηθητικές δομές, οι οποίες είναι:

- implementation.h: περιέχει τις δομές για την αποθήκευση των ενδιάμεσων – τελικών αποτελεσμάτων των κατηγορημάτων μίας γραμμής.
  - RowIdNode: αποτελεί τον κόμβο της λίστας, αποτελείται από ένα rowId, μία τιμή που δείχνει αν είναι κενός και έναν δείκτη προς τον επόμενο κόμβο.
  - RowIdsList: αποτελεί την λίστα από RowIdNode κόμβους, την τιμή της σχέσης που εκπροσωπεί αλλά και μία μεταβλητή για το μέγεθός της.
- queryMethods.h: περιέχει μία δομή για τον χαρακτηρισμό ενός κατηγορήματος. Πιο συγκεκριμένα:
  - Predicate: αποτελεί την εκπροσώπηση ενός κατηγορήματος, περιέχει πληροφορίες για το δεξί και το αριστερό μέρος (ζευγάρια/tuple) που περιέχουν τον αριθμό της σχέσης και την κολώνα, τον τύπο της σύγκρισης («=», «>», «<») αλλά και τι τύπος κατηγορήματος είναι (φίλτρου/0, ζεύξης/1).
- radixHashJoin.h: δεν τροποποιήθηκε, άρα ισχύουν ό,τι και στην προηγούμενη έκδοση.
- relationMethods.h: περιέχει δομές για την αποθήκευση των σχέσεων:
  - MetadataCol: αποτελείται από τα στατιστικά(max,min,discrete values, number of rows) που πρέπει να υπολογιστούν κατά το διάβασμα και την αποθήκευση των σχέσεων. Στην συγκεκριμένη έκδοση, δεν χρησιμοποιείται.
  - RelationsInfo: περιέχει τις πληροφορίες μία σχέσης, όπως το όνομά της, τον αριθμό των γραμμών και κολόνων, την ίδια την σχέση και μία δομή metadata η οποία για αυτήν την έκδοση, δεν χρησιμοποιείται.
  - StringNode: αποτελεί μία λίστα για τα ονόματα των σχέσεων.

Αρχεία δομών και συναρτήσεων

Τα αρχεία που τροποποιήθηκαν σε αυτό το μέρος ήταν:

- auxMethods.c : προσθήκη συνάρτησης (isNumeric) η οποία επιστρέφει 0 ή 1 ανάλογα αν μία συμβολοσειρά είναι αριθμός.
- implementation.c : περιέχει την κύρια υλοποίηση της εργασίας (queriesImplementation/joinColumns), την δημιουργία, προσθήκη και διαγραφή

της κύριας δομής rowIdsList αλλά και συναρτήσεις βοηθητικές για την ορθή υλοποίηση του join.

- queryMethods.c : περιέχει συναρτήσεις για την επεξεργασία μίας γραμμής εκτέλεσης κατηγορημάτων, με την ανάθεση σε συγκεκριμένες μεταβλητές, τις σχέσεις, τις προβολές και το κάθε κατηγόρημα ξεχωριστά.
- relationMethods.c : περιέχει συναρτήσεις για την επεξεργασία ενός relationsInfo struct, δηλαδή, για την δημιουργία των σχέσεων για χρήση στα κατηγορήματα.
- main.c: δημιουργεί τις σχέσεις με βάση το αρχείο εισόδου και έπειτα εκτελεί το αρχείο work.

Unit Testing

Όσων αφορά το Unit Testing, συνεχίσαμε να χρησιμοποιούμε το framework CuTest. Πρέπει να αναφερθεί ότι δεν υλοποιήσαμε όλες τις συναρτήσεις ξεχωριστά, καθώς ορισμένες καλούνταν η μία από την άλλη.

Εκτέλεση & Στατιστικά

Η εκτέλεση γίνεται με την εντολή: **./joinProgram -i <path_init_file> -w <path_work_file>**

Για την εκτέλεση, χρησιμοποιήθηκε το αρχείο workload/small από το submission.tar.gz. Τα αποτελέσματα δεν ήταν σωστά, διότι δεν είχε υλοποιηθεί ορθά η ανανέωση των σχέσεων σε κατηγορήματα της μορφής : «0.1=2.1&2.1=3.1», όπου θα έπρεπε να ενημερωθεί η σχέση 0 μετά την εκτέλεση του τελεστή ζεύξης του δεύτερου κατηγορήματος. Ακόμη, δημιουργήθηκε ένα πιο μικρό dataset small1 το οποίο είναι μέρος του small. Το συγκεκριμένο χρησιμοποιήθηκε μόνο για λόγους debugging.

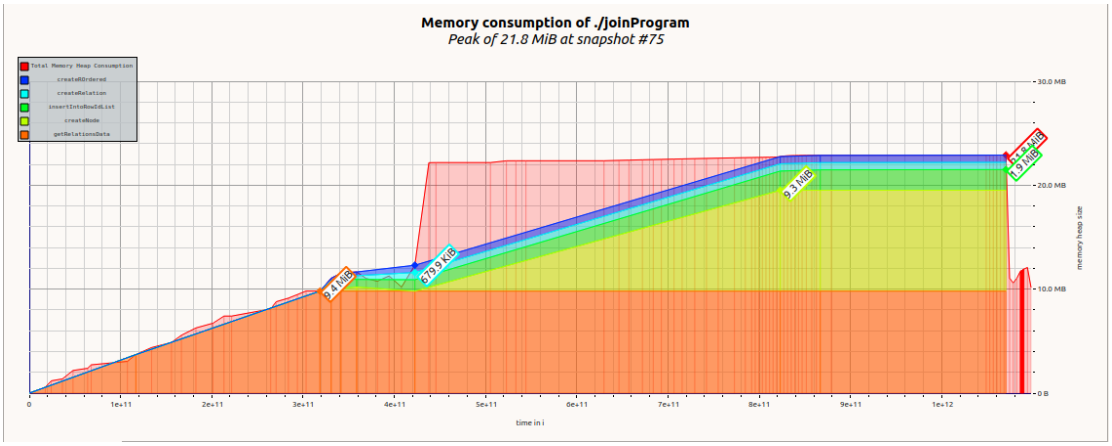
ΑΡΧΕΙΑ ΕΙΣΟΔΟΥ	ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ	ΟΡΘΟΤΗΤΑ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (✓)
small	~5m (Segmentation)	**X**

Χρονικό διάγραμμα

Περαιτέρω πληροφορίες σχετικά με τη χρονική διάρκεια των συναρτήσεων φαίνονται στο παρακάτω διάγραμμα.

Χωρικό διάγραμμα

Το διάγραμμα χώρου μνήμης φαίνεται παρακάτω.



Σχόλια Εξεταστή & Διορθώσεις

Ζητήσαμε κάποιες συμβουλές, ώστε να υλοποιήσουμε σωστά την διαδικασία της ενημέρωσης. Στην επόμενη έκδοση, υλοποιήσαμε κομμάτι κώδικα που τελικά παράγει σωστά αποτελέσματα για το small dataset. Άλλη μία διόρθωση που θέλαμε να πραγματοποιήσουμε ήταν να μετατραπεί η κύρια δομή από λίστα σε πίνακα, η οποία θα διπλασιάζεται όταν γεμίζει. Αυτό θα ήταν πιο βέλτιστο, διότι η αναζήτηση, αλλά πιο πολύ η δημιουργία και η χρήση malloc, που είναι χρονοβόρα, θα μειωνόταν σημαντικά, κάνοντας το πρόγραμμα πιο βέλτιστο. Το συγκεκριμένο υλοποιήθηκε επιτυχώς στο επόμενο μέρος της εργασίας. Αυτό που υλοποιήθηκε ήταν το εξής:

Η βασική ιδέα για να ενημερώσουμε, πέραν των ενδιάμεσων σχέσεων που παίρνουν μέρος σε ένα join, και τα άλλα rowIdArrays που εξαρτώνται από αυτά, είναι να κρατάμε την θέση στον ενδιάμεσο πίνακα των σχέσεων του join, και να την περνάμε κάθε φορά στο join και στην λίστα που δημιουργείται με τα νέα αποτελέσματα. Έτσι, όταν θα πρέπει να περάσουμε τα αποτελέσματα του resultList στους ενδιάμεσους πίνακες, έχουμε την θέση που αντιστοιχεί κάθε καινούργια εγγραφή στους παλιούς πλέον ενδιάμεσους, άρα κάθε φορά προσθέτουμε το rowId που αντιστοιχούσε στην συγκεκριμένη σειρά, κι έτσι είναι εύκολο να ανανεώσεις και τους άλλους ενδιάμεσους πίνακες. Κάθε φορά ψάχνουμε πιο relation χρειάζεται ενημέρωση, και θα είναι κάποιο που σχετίζεται αναγκαστικά με κάποιο από τα relation που πήραν μέρος στο join, λόγω κάποιου προηγούμενου join προφανώς.

Join Implementation: Part 3

Δομές

Στο τρίτο μέρος δημιουργήθηκαν πολλές δομές, άλλες τροποποιήθηκαν ή διαγράφτηκαν. Αυτό συνέβη είτε για βελτιστοποίηση, μείωση χώρου αλλά και για την επιτυχή υλοποίηση των νημάτων με job scheduler. Άρα, έχουμε τα εξής:

- implementation.h: Διαγράφτηκαν ό,τι δομές υπήρχαν.
- queryMethods.h: Τροποποιήθηκε η δομή του κατηγορήματος, προσθέτοντας άλλη μία μεταβλητή, η οποία δείχνει αν το συγκεκριμένο κατηγόρημα πρέπει να παραλειφθεί από το σύνολο των κατηγορημάτων, λόγω της ύπαρξης ίδιου.
- radixHashJoin.h: Τροποποιήθηκαν κάποιες δομές:
  - Tuple: προσθήκη μεταβλητής rArrayRow, για την σωστή υλοποίηση της ενημέρωσης.
  - ResultElement: προσθήκη 2 μεταβλητών, αντίστοιχες με αυτές του rArrayRow στο tuple, βοηθητικές για την ενημέρωση των σχέσεων.
  - HistArgs: κόμβος που χρησιμοποιείται για την δημιουργία ιστογράμματος από ένα νήμα, περιέχει τις παραμέτρους που λαμβάνει η κανονική συνάρτηση createHistogram.
  - ROrderedArgs: κόμβος που χρησιμοποιείται για την δημιουργία R', από ένα νήμα, παρ' όλα αυτά, δεν υλοποιήθηκε κατάλληλη συνάρτηση για να ανατίθεται στο νήμα.
  - IndexCompareJoinArgs: κόμβος που χρησιμοποιείται για την εκτέλεση του Radix Hash Join από ένα νήμα.
- relationMethods.h: παρέμεινε η ίδια.
- rowIdArrayMethods.h: καινούριο αρχείο για την απεικόνιση των ενδιάμεσων και τελικών αποτελεσμάτων σε πίνακα αντί για λίστα. Περιέχει:
  - RowIdsArray: περιέχει τον αριθμό της σχέσης, το μέγεθος αλλά και το πραγματικό μέγεθος (position) που χρησιμοποιείται για τον διπλασιασμό της. Ακόμη, περιέχει έναν πίνακα από row ids.
- threadPool.h: περιέχει όλες τις σημαντικές δομές για την υλοποίηση ενός Job Scheduler με Thread Pool:
  - Job: περιέχει έναν δείκτη στην επόμενη εργασία, ένα πεδίο για την συνάρτηση και ένα άλλο για την παράμετρο της συνάρτησης (πρέπει να είναι μία).
  - JobPool: περιέχει δύο δείκτες, το πρώτο και τον τελευταίο κόμβο της ουράς, το μέγεθος, ένα mutex για το κλείδωμα της και ένα condition variable που δείχνει πότε είναι κενή.
  - Thread: είναι ένας κόμβος που περιέχει το id του thread που δημιουργείται αλλά και έναν δείκτη προς το thread pool για να μπορεί να βλέπει πληροφορίες της ουράς εργασιών.
  - ThreadPool: είναι η συνολική δομή των νημάτων, περιέχει τα νήματα, την ουρά εργασιών, τον αριθμό των νημάτων, πόσα νήματα ζουν και πόσα εργάζονται, έναν mutex που κλειδώνει το threadPool, ένα condition variable που χρησιμοποιείται για τον συγχρονισμό αλλά και ένα barrier για τον ορθό συγχρονισμό μεταξύ main και child threads.

Αρχεία δομών και συναρτήσεων

Τα αρχεία αναδιοργανώθηκαν σε φακέλους για την ορθή εκτέλεση του Harness αλλά και για την καλύτερη ανάγνωση των αρχείων. Πιο συγκεκριμένα:

- implementation.c: ό,τι και πριν, μόνο που αφαιρέθηκαν συναρτήσεις που έχουν να κάνουν με την υλοποίηση των ενδιάμεσων αποτελεσμάτων.
- main.c: όπως και πριν, μόνο που δημιουργεί και ένα threadPool και το καταστρέφει στο τέλος της.
- queryMethods.c: όπως και πριν, αλλά τώρα, διαγράφει ένα κατηγόρημα από το σύνολο κατηγορημάτων, αν αυτό ή παρόμοιο, υπάρχει ήδη στο σύνολο που αναφέρθηκε.
- radixHashJoin.c: όπως και πριν με την μικρή διαφορά ότι δημιουργήθηκαν ακόμη 2 συναρτήσεις που ανατίθενται ως jobs στο threadPool (createHistogramThread, indexCompareJoinThread). Ακόμη, ανατίθεται τιμή στην προστιθέμενη μεταβλητή rArrayRow σε κάθε περίπτωση που χρειάζεται.
- relationMethods.c: έγινε πιο δυναμική η ανάθεση μονοπατιού προς ανάγνωση σχέσεων. Επίσης, δημιουργείται ο κόμβος metadataCol για κάθε κολώνα κάθε σχέσης.
- rowIdArrayMethods.c: καινούριο αρχείο για την επεξεργασία της rowIdsArray δομής. Περιέχει συναρτήσεις για δημιουργία, διπλασιασμό, διαγραφή και την δημιουργία σχέσης από μία τέτοια δομή.
- statisticsMethods.c: περιέχει συναρτήσεις για τον υπολογισμό στατιστικών και την δημιουργία BestTree.
- threadPool.c: περιέχει τις συναρτήσεις για δημιουργία, προσθήκη, εξαγωγή και διαγραφή της ουράς εργασιών, την δημιουργία και διαγραφή του thread pool, την γενική συνάρτηση εκτέλεσης του νήματος (executeJob) και συναρτήσεις για τον συνδυασμό των δομών που παράγουν τα νήματα.

Join Enumeration

Τα στατιστικά υπολογίζονται για κάθε predicate με βάση τους τύπους που δίνονται. Οι τύποι αυτοί δεν είναι και πολύ ακριβείς και πέφτουν αρκετές φορές πολύ έξω από τα πραγματικά αποτελέσματα, αλλά σε κάθε περίπτωση υπάρχει μία εξοικονόμηση, έστω και μικρή. Ο αλγόριθμος του joinEnumeration υπολογίζει τον καλύτερο συνδυασμό με έναν greedy τρόπο. Πάντοτε γίνονται πρώτα τα predicates που συγκρίνουν στήλες σχέσεων με κάποιες τιμές, καθώς είναι σίγουρα πιο αποδοτικά. Έπειτα από τα διαθέσιμα join predicates, επιλέγουμε αυτό με το μικρότερο κόστος, όπως αυτό υπολογίζεται από τους τύπους που δίνονται. Ενημερώνουμε τα στατιστικά για να περάσουν σε επόμενες διερευνήσεις, και κρατάμε την θέση στην οποία επιλέχθηκε το τρέχον predicate, έτσι ώστε να αλλάξει θέση εν τέλει. Αν παραδείγματος χάριν το 3ο join predicate, που σε μία σχέση με ένα compare predicate θα εκτελεστεί τέταρτο, βρεθεί από τους τύπους ότι θα βγάλει τους λιγότερους ενδιάμεσους συνδυασμούς, επιλέγεται να εκτελεστεί ως πρώτο από τα join

predicates, άρα και 2ο συνολικά μετά το compare predicate. Αφότου ενημερώνουμε τον πίνακα που κρατάει αυτήν την θέση, όσο και το συνολικό κόστος έως τώρα, προχωράμε στην διερεύνηση των εναπομεινάντων join predicates. Ο υπολογισμός αυτός του κόστους και της θέσης, μιας και χρησιμοποιούμε άπληστο αλγόριθμο, έχει πραγματοποιηθεί κατά βάση αναδρομικά.

Thread Pool

Η υλοποίηση του thread pool είναι απλή. Αποτελείται από έναν πίνακα από threads, μία ουρά από jobs, mutexes, condition variables, barrier και τον αριθμό των νημάτων που δουλεύουν κι αυτά που είναι ζωντανά. Όταν ένα thread δημιουργείται, του αναθέτουμε μία συνάρτηση, στην συγκεκριμένη περίπτωση την executeJob που έχει ως όρισμα το ίδιο το thread. Μέσω αυτής της συνάρτησης, ο αριθμός των ζωντανών νημάτων αυξάνεται και στην συνέχεια, το νήμα μπαίνει σε ατέρμων βρόγχο, έως ότου η στατική volatile μεταβλητή keepAlive γίνει 0. Η συγκεκριμένη μεταβλητή, είναι volatile για να μένει σε συγκεκριμένο σημείο στο stack, χωρίς να αλλάζει θέση, κι άρα να μην υπάρχει πρόβλημα στην αλλαγή της τιμής της από το main thread. Τα νήματα, λοιπόν, περιμένουν να λάβουν κάποια εργασία. Όταν μία εργασία δημιουργείται, εισέρχεται στο τέλος της ουράς, ενώ για να αναθέσουμε μία εργασία σε ένα νήμα, εξάγουμε τον αρχικό κόμβο της ουράς. Όταν εισαχθεί μία εργασία στην ουρά, γίνεται signal στα νήματα, ότι η ουρά δεν είναι άδεια. Κάποιο από τα νήματα, θα λάβει την εργασία, θα την εκτελέσει ως function(args) και θα περιμένει πάλι για νέα εργασία. Στην δική μας υλοποίηση, επειδή κάθε νήμα λαμβάνει εργασία κι το main thread πρέπει να περιμένει να τελειώσουν όλα, έχουμε χρησιμοποιήσει το pthread_barrier. Η μεταβλητή αυτή, λειτουργεί ως threshold, κάνοντας τον συγχρονισμό των νημάτων για αυτή την περίπτωση πιο εύκολη. Περιμένει, δηλαδή, όλα τα νήματα να γίνουν wait, και μετά συνεχίζουν όλα από εκεί που έμειναν. Όταν ένα thread εκτελεί μία λειτουργία, ο αριθμός των νημάτων που δουλεύουν αυξάνεται. Όλες οι τροποποιήσεις των μεταβλητών προστατεύονται από mutexes.

Όσον αφορά την παραλληλοποίηση της δημιουργίας του ROrdered, δεν έχει υλοποιηθεί στο συγκεκριμένο σημείο με threads, διότι η συνάρτηση αυτή είναι ήδη βελτιστοποιημένη, οπότε, η παραλληλοποίηση είναι πολύ δύσκολη σε αυτή την περίπτωση. Ακόμη, για την δημιουργία ενός thread, παραλείπεται η εκτέλεσή του για παραλληλοποίηση, και εκτελείται κανονικά το main thread.

Unit Testing

Υλοποιήσαμε και τα υπόλοιπα αρχεία για έλεγχο συναρτήσεων στο φάκελο cutest-1.5.

Εκτέλεση & Στατιστικά

Η εκτέλεση μπορεί να γίνει με 2 τρόπους, είτε μέσω harness με την εκτέλεση του αρχείου **./runTestharness.sh <path of workload>**,

είτε με την εκτέλεση της εντολής:

./run.sh <workload type> <function type: time/valgrind> <type of function: leak-check,callgrind,massif,…> <log out file for callgrind or massif>.

Για την μεταγλώττιση του προγράμματος, πρέπει πριν την εκτέλεση, να εκτελεστεί το πρόγραμμα **./compile.sh**.

Η εκτέλεση του harness παράγει τον χρόνο που χρειάζεται για να εκτελεστεί το συγκεκριμένο dataset, μόνο στο σημείο των κατηγορημάτων, σε milliseconds. Για τον υπολογισμό των δευτερολέπτων, αρκεί ο υπολογισμός milliseconds / 1000.

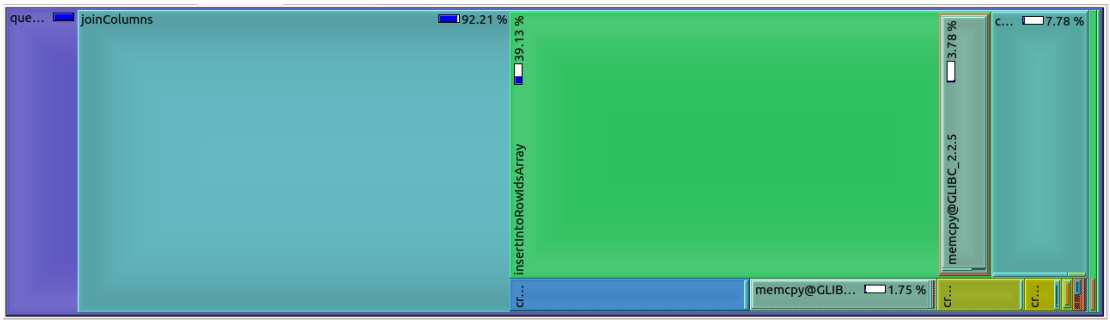
	ΑΡΧΕΙΑ ΕΙΣΟΔΟΥ	**ΧΡΟΝΟΙ ΕΚΤΕΛΕΣΗΣ**	**ΝΗΜΑΤΑ**	**ΟΡΘΟΤΗΤΑ ΑΠΟΤΕΛΕΣΜΑΤΩΝ (✓)**
harness	*small*	15min	1	✓
	small	55,64 s	2	✓
	small	53,061 s	4	✓
	small	23,127 s	8	✓

Χρονικά διαγράμματα

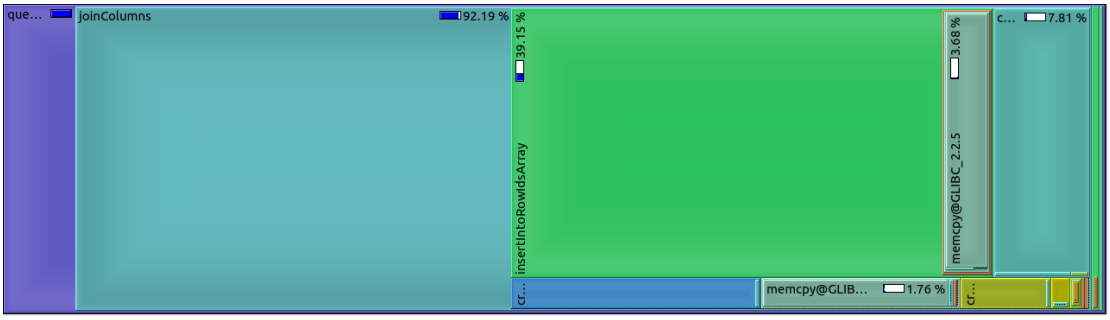
Παρακάτω φαίνονται πληροφορίες για τα χρονικά ποσοστά που καταλαμβάνει η κάθε συνάρτηση.

Με 1 Νήμα

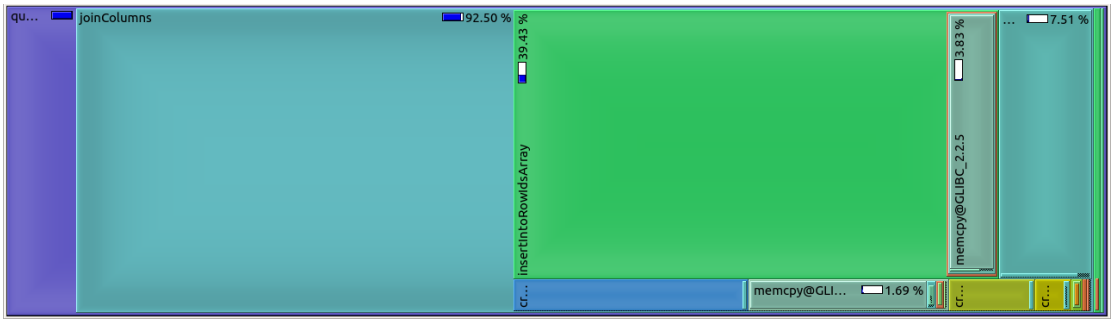
Με 2 Νήματα



Με 4 Νήματα



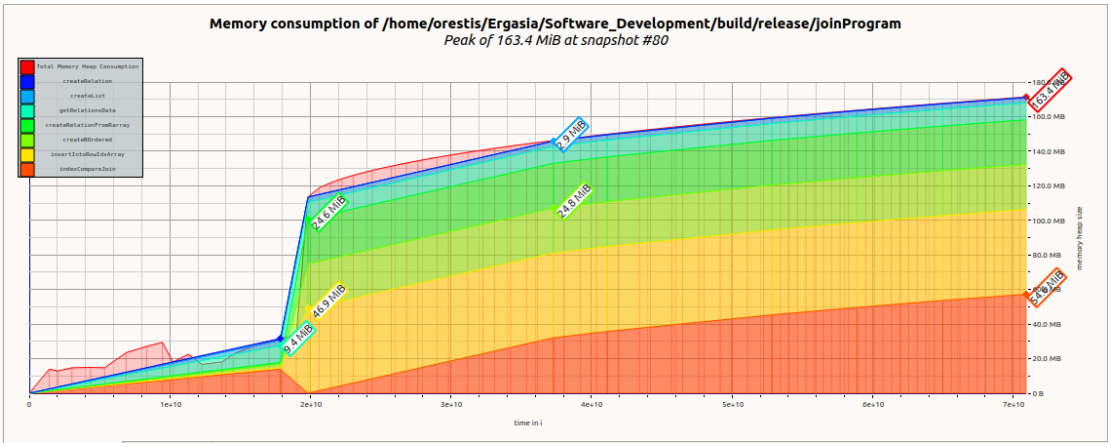
Με 8 Νήματα



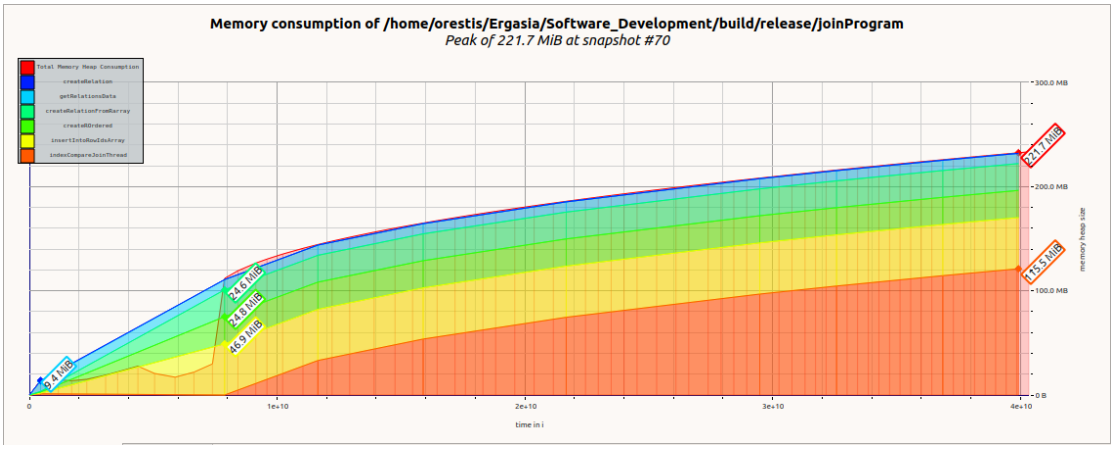
Χωρικά διαγράμματα

Παρακάτω παρουσιάζεται η κατανομή του χώρου όσων αφορά τις συναρτήσεις.

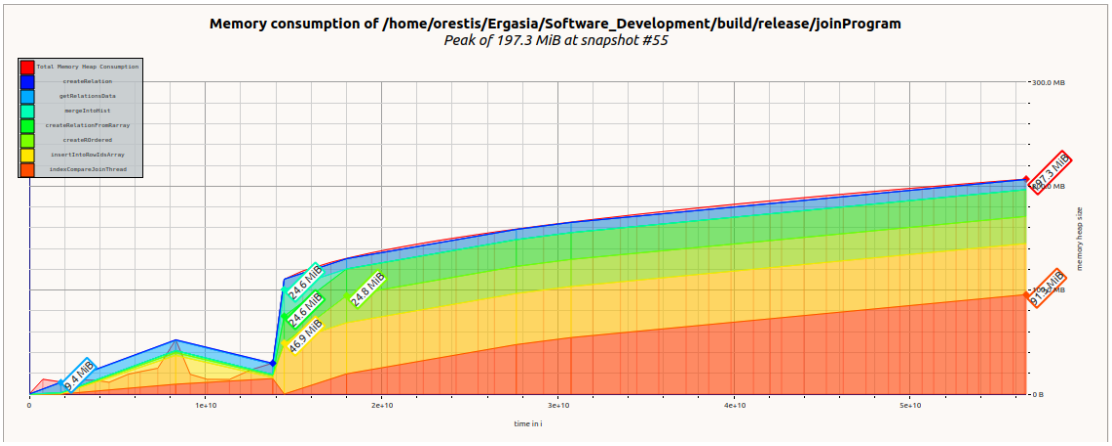
Με 1 Νήμα



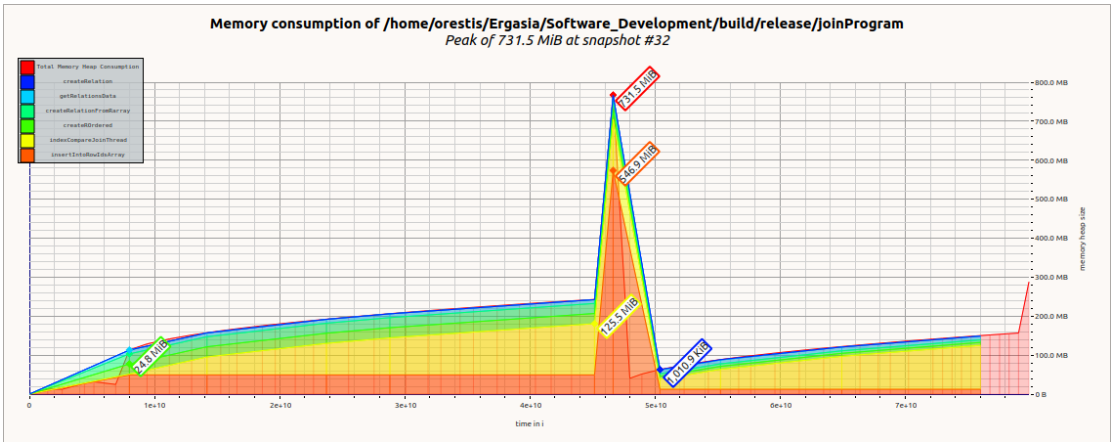
Με 2 Νήματα



Με 4 Νήματα



Με 8 Νήματα



Επίλογος/Σχόλια

Για τη δημιουργία των διαγραμμάτων χρησιμοποιήσαμε τις εντολές callgrind και massif, ενώ για τη χρονομέτρηση των προγραμμάτων, την εντολή time. Όλες οι εκδόσεις ελέγχθηκαν με το –leak-check=full για την τυχόν εύρεση leaks. Τα αποτελέσματα ήταν ορθά. Για την χρονομέτρηση και εκτέλεση των αποτελεσμάτων, χρησιμοποιήθηκαν οι παρακάτω:

	Υπολογιστής 1	Υπολογιστής 2
Μέρος εργασίας	1,2	3
Λειτουργικό Σύστημα	Opensuse Leap 15.0	Ubuntu 64-bit
Επεξεργαστής	i7 2.67 GHz	Intel Core i5 2.6 GHz
Πυρήνες	8	4
Μνήμη	6 GB RAM DDR3 3 slots	4 GB RAM