GPU Accelerated data management. A closer look

Βασίλης Πίσχος και Βαλσάμης Γεώργιος

Εξόρυξη Δεδομένων 2023-24
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας, Βόλος
{vpischos, gvalsamis}@e-ce.uth.gr

Περίληψη Την προηγούμενη δεκαετία, οι ειδικοί επεξεργαστές γραφικών (GPUs) εξελίχθηκαν σε γενικού σκοπού υπολογιστές, με την εμφάνιση γενικών προγραμματιστικών μοντέλων παράλληλης επεξεργασίας, όπως το CUDA [4] και το OpenCL [5]. Εξαιτίας της τεράστιας υπολογιστικής δύναμης τους πλέον χρησιμοποιούνται για εφαρμογές deep learning και high performance computing. Υπάρχουν σημαντικές προοπτικές και στον τομέα των βάσεων δεδομένων καθώς μπορούν να επιταχύνουν σημαντικά εφαρμογές περιορισμένης μνήμης (memory bound applications). Οι GPUs χρησιμοποιούν την High-Bandwidth Memory (HBM), μια νέα κατηγορία μνήμης RAM που έχει σημαντικά υψηλότερο εύρος ζώνης σε σύγκριση με την παραδοσιακή DDR RAM που χρησιμοποιείται με τους επεξεργαστές CPU. Μια μοντέρνα GPU μπορεί να έχει έως και 32 GB HBM που μπορεί να παρέχει έως και 1,2 TBps εύρος ζώνης μνήμης και 14 Tflops υπολογισμού. Σε αντίθεση, μία μοναδική CPU μπορεί να έχει εκατοντάδες GB μνήμης με έως και 100 GBps εύρος ζώνης μνήμης και 1 TFlop υπολογισμού. Η αύξηση αυτής της χωρητικότητας μνήμης, σε συνδυασμό με τη δυνατότητα εξοπλισμού ενός μοντέρνου διακομιστή με αρκετές GPUs (έως και 20), σημαίνει ότι είναι δυνατόν να έχουμε εκατοντάδες GB μνήμης GPU σε έναν μοντέρνο διακομιστή. Η βελτίωση του εύρους ζώνης των GPUs οδήγησε πολλούς ερευνητές στο να τις χρησιμοποιήσουν ως co-processors (συνεπεξεργαστή) [6], μία τεχνική που έχει δημιουργήσει αρκετά αναπάντητα ερωτήματα σχετικά με την αποδοτικότητά της.

Λέξεις Κλειδιά: GPU High-Bandwidth Memory, εύρος ζώνης, co-processors

1 Τα ερωτήματα που θα ερευνήσουμε

1)Τα συστήματα βάσεων δεδομένων που βασίζονται σε GPU έχουν αναφέρει ευρύ φάσμα βελτίωσης απόδοσης σε σύγκριση με συστήματα βάσεων δεδομένων που βασίζονται σε CPU, κυμαινόμενη από 2x έως 100x. Δεν υπάρχει ομοφωνία σχετικά με το πόσο μεγάλη βελτίωση της απόδοσης μπορεί να επιτευχθεί χρησιμοποιώντας GPUs. Οι προηγούμενες εργασίες συχνά συγκρίνονται με αναποτελεσματικές βάσεις, όπως το MonetDB, το οποίο είναι γνωστό ότι είναι ανεπαρκές [8]. Η εμπειρική φύση των προηγούμενων εργασιών καθιστά δύσκολο τον γενικευμένο σχολιασμό των αποτελεσμάτων σε διαφορετικές πλατφόρμες υλικού.

2)Σε πολλές από τις προηγούμενες χρονολογικά έρευνες οι GPUs χρησιμοποιούνται αυστηρά ως συνεπεξεργαστές. Κάθε ερώτηση (query) καταλήγει να μεταφέρει δεδομένα από τη CPU στη GPU μέσω του PCIe. Η μεταφορά δεδομένων μέσω PCIe είναι

σημαντικά πιο αργή από το εύρος ζώνης μνήμης της GPU και συνήθως λιγότερη από το εύρος ζώνης μνήμης της CPU. Ως αποτέλεσμα, ο χρόνος μεταφοράς μέσω PCIe γίνεται ο φραγμός (bottleneck) και περιορίζει τα κέρδη. Σε μεγάλο βαθμό, προηγούμενες έρευνες στον τομέα δείχνουν βελτιώσεις απόδοσης οι οποίες κυρίως οφείλονται σε αξιολόγηση έναντι ανεπαρκών βάσεων.

3)Υπήρξε σημαντική βελτίωση στο υλικό των GPU τα τελευταία χρόνια. Η πιο πρόσφατη έρευνα σε βάσεις δεδομένων που βασίζεται σε GPU αξιολογεί σε GPU που διαθέτουν χωρητικότητα μνήμης και εύρος ζώνης 4 GB και 150 GBps αντίστοιχα, ενώ οι πιο πρόσφατες γενιές των GPU έχουν 8 φορές υψηλότερη χωρητικότητα και εύρος ζώνης. Αυτά τα κέρδη βελτιώνουν σημαντικά την ελκυστικότητα των GPU για την επεξεργασία ερωτημάτων.

1.1 Στόχοι της εργασίας

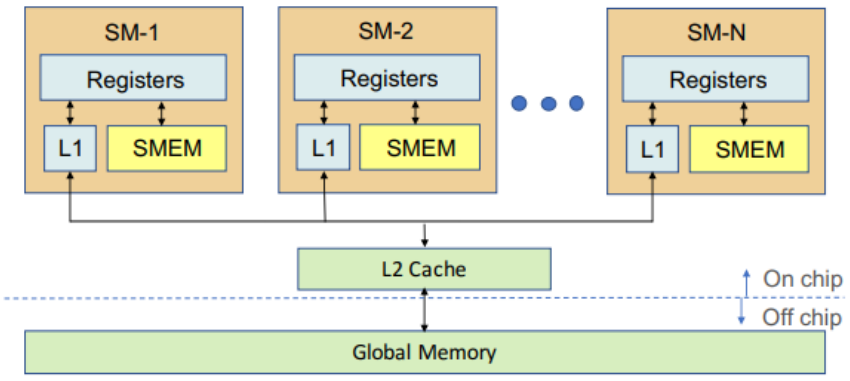
1)Θα δείξουμε πώς πως η χρήση των GPUs ως co-processors με PCIe interconnect δέν προσφέρει καλύτερη επίδοση από ένα state of art CPU baseline.

2)Θα δείξουμε πώς η διασύνδεση μέσω NVLink [7] είναι πολύ πιο γρήγορη και επίσης θα παρουσιάσουμε transfer optimizations (βελτιστοποιήσεις μεταφοράς).

3)Θα δείξουμε πως η χρήση της αυξημένης χωρητικότητας μνήμης των σύγχρονων GPU για την αποθήκευση του εργατικού συνόλου (working set) απευθείας στη μονάδα GPU αποτελεί καλύτερο σχεδιασμό.

2 Background

2.1 Ιεραρχία της μνήμης της GPU



Εικ. 1. Ιεραρχία μνήμης της GPU

Πολλές λειτουργίες βάσεων δεδομένων που εκτελούνται στη μονάδα GPU περιορίζονται από το υποσύστημα μνήμης (είτε κοινόχρηστη είτε παγκόσμια μνήμη). Για

να χαρακτηριστεί η απόδοση διαφορετικών αλγορίθμων στη μονάδα GPU, είναι, συνεπώς, κρίσιμο να κατανοήσουμε σωστά την ιεραρχία της μνήμης της. Η Εικόνα 1 δείχνει μια απλοποιημένη ιεραρχία ενός σύγχρονου GPU. Η χαμηλότερη και μεγαλύτερη μνήμη στην ιεραρχία είναι η παγκόσμια μνήμη. Ένα σύγχρονο GPU μπορεί να έχει χωρητικότητα παγκόσμιας μνήμης έως και 32 GB με εύρος ζώνης μνήμης έως και 1200 GBps. Κάθε GPU έχει έναν αριθμό μονάδων υπολογισμού που ονομάζονται Streaming Multiprocessors (SMs). Κάθε SM έχει έναν αριθμό πυρήνων και ένα σταθερό σύνολο καταχωρητών. Κάθε SM έχει επίσης μια κοινόχρηστη μνήμη που λειτουργεί ως προσωρινή αποθήκευση και μπορεί να προσπελαστεί από όλους τους πυρήνες στο SM. Οι προσπελάσεις στην παγκόσμια μνήμη από ένα SM αποθηκεύονται στην κρυφή μνήμη L2 (η οποία είναι κοινόχρηστη μεταξύ όλων των SMs) και προαιρετικά επίσης στην κρυφή μνήμη L1 (η οποία είναι τοπική για κάθε SM). Η επεξεργασία στη μονάδα GPU γίνεται από ένα μεγάλο αριθμό νημάτων οργανωμένων σε μπλοκ νημάτων (κάθε ένα από τα οποία εκτελείται από ένα SM). Τα μπλοκ νημάτων διαιρούνται περαιτέρω σε ομάδες νημάτων που ονομάζονται warps (συνήθως αποτελούμενα από 32 νήματα). Τα νήματα ενός warp εκτελούνται σύμφωνα με ένα μοντέλο Single Instruction Multiple Threads (SIMT), όπου κάθε νήμα εκτελεί την ίδια ακολουθία εντολών σε διαφορετικά δεδομένα. Η μονάδα ομαδοποιεί προσπελάσεις σε παγκόσμια μνήμη από νήματα σε ένα warp έτσι ώστε πολλαπλές προσπελάσεις/αποθηκεύσεις στην ίδια γραμμή μνήμης cache να συγχωνεύονται σε μία μόνο αίτηση. Η μέγιστη χωρητικότητα μπορεί να επιτευχθεί όταν η προσπέλαση ενός warp στην παγκόσμια μνήμη έχει ως αποτέλεσμα να προσπελαστούν γειτονικές θέσεις. Το μοντέλο προγραμματισμού επιτρέπει στους χρήστες να δεσμεύουν απευθείας παγκόσμια μνήμη και κοινόχρηστη μνήμη σε κάθε μπλοκ νημάτων. Η κοινόχρηστη μνήμη έχει υψηλότερο εύρος ζώνης σε σύγκριση με την παγκόσμια μνήμη, αλλά έχει πολύ μικρότερη χωρητικότητα (μερικά MB έναντι πολλαπλών GB). Τέλος, οι καταχωρητές αποτελούν το ταχύτερο επίπεδο της ιεραρχίας μνήμης. Αν ένα μπλοκ νημάτων χρειάζεται περισσότερους καταχωρητές από όσους είναι διαθέσιμοι, οι τιμές των καταχωρητών μεταφέρονται στην παγκόσμια μνήμη.

2.2 Εκτέλεση queries στην GPU

Με την εδραίωση του Νόμου του Moore, η απόδοση των CPU έχει στασιμοποιηθεί. Τα τελευταία χρόνια, οι ερευνητές άρχισαν να εξερευνούν τον ομογενή υπολογισμό για να ξεπεράσουν τα προβλήματα κλιμάκωσης των CPU και να συνεχίσουν να παρέχουν διαδραστική απόδοση για εφαρμογές βάσεων δεδομένων. Σε ένα τέτοιο υβριδικό σύστημα CPU-GPU, οι δύο επεξεργαστές είναι συνδεδεμένοι μέσω PCIe. Το εύρος ζώνης του PCIe ενός μοντέρνου υπολογιστή είναι έως και 16 GBps, πολύ χαμηλότερο από το εύρος ζώνης μνήμης τόσο της CPU όσο και της GPU. Ως εκ τούτου, η μεταφορά δεδομένων μεταξύ CPU και GPU αποτελεί σοβαρό φραγμό απόδοσης (performance bottleneck). Προηγούμενες έρευνες στην κοινότητα των βάσεων δεδομένων επικεντρώθηκαν στη χρήση της GPU ως συνεπεξεργαστή, τον οποίο αποκαλούμε μοντέλο συνεπεξεργαστή. Σε αυτό το μοντέλο, τα δεδομένα βρίσκονται κυρίως στην κύρια μνήμη της CPU. Για την εκτέλεση των ερωτημάτων, τα δεδομένα μεταφέρονται από τη CPU στη GPU μέσω PCIe, έτσι ώστε (κάποια) επεξεργασία ερωτημάτων να μπορεί να γίνει στη GPU. Τα αποτελέσματα μεταφέρονται στη συνέχεια πίσω στη CPU.

3 Ανάλυση του μοντέλου συνεπεξεργαστή

3.1 Οι αποτυχίες

Παρόλο που προηγούμενες μελέτες έχουν δηλώσει βελτιώσεις στην ταχύτητα χρησιμοποιώντας GPU στο μοντέλο συνεπεξεργαστή, υπάρχει ασυμφωνία ανάμεσα στις προηγούμενες εργασίες σχετικά με το ποσοστό βελτίωσης της απόδοσης που επιτυγχάνεται καθώς υπάρχει μεγάλη απόκληση στα αποτελέσματα απο 2x έως και 100x.

```
SELECT SUM(lo_extendedprice * lo_discount) AS revenue
FROM lineorder
WHERE lo_quantity < 25
AND lo_orderdate >= 19930101 AND lo_orderdate <= 19940101
AND lo_discount >= 1 AND lo_discount <= 3;
```

Εικ. 2. Star Schema Benchmark

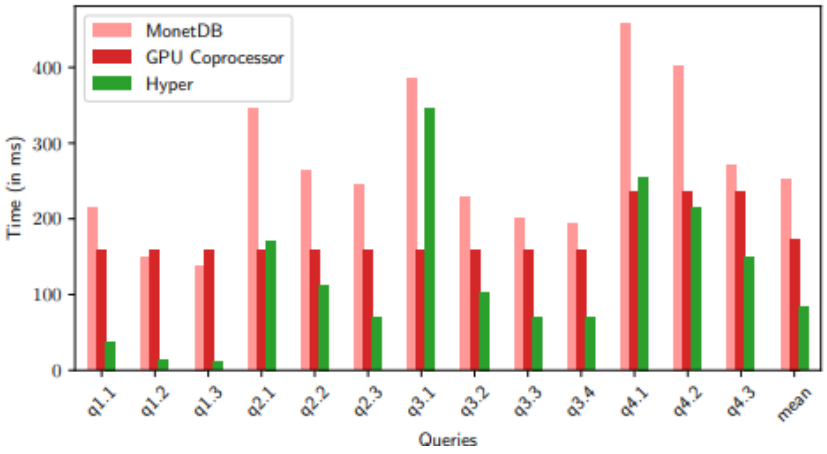
Στην ερώτηση Q1.1 του Star Schema Benchmark (SSB), συγκρίνεται η απόδοση μιας υλοποίησης σε CPU σε σχέση με μια σε μοντέλο με συνεπεξεργαστή GPU, βασιζόμενοι στα όρια εύρους μνήμης τους.

Για μια υλοποίηση σε CPU, ο βέλτιστος χρόνος εκτέλεσης (RC) είναι άνω φράγμενος από $\frac{16L}{B_c}$, όπου B_c είναι το εύρος μνήμης της CPU. Αυτό το όριο προκύπτει από την υπόθεση ότι μια μόνο διέλευση πάνω από τις 4 στήλες δεδομένων είναι αρκετή. Ωστόσο, ο πραγματικός χρόνος εκτέλεσης μπορεί να είναι μικρότερος αν οι προσκείμενες συνθήκες είναι επιλεκτικές, επιτρέποντας την παράκαμψη ολόκληρων γραμμών cache κατά την πρόσβαση στη στήλη lo_extendedprice..

Αντίθετα, για ένα μοντέλο με συνεπεξεργαστή GPU, ο χρόνος εκτέλεσης του ερωτήματος στη GPU (RG) είναι κάτω φραγμένος από $\frac{16L}{B_p}$, όπου B_p είναι το εύρος ζώνης PCIe. Αυτό το κάτω φράγμα επιτυγχάνεται αν καταφέρουμε να επικαλυφθεί απόλυτα η μεταφορά δεδομένων και η εκτέλεση του ερωτήματος στη GPU.

Ωστόσο, αφού $B_c > B_p$ σε σύγχρονες ρυθμίσεις CPU-GPU, $RC < RG$, δηλαδή η εκτέλεση του ερωτήματος στη CPU οδηγεί σε μικρότερο χρόνο εκτέλεσης από τον χρόνο εκτέλεσης με συνεπεξεργαστή GPU.

Για να το αποδείξουμε πειραματικά, εκτελέσαμε ολόκληρο το SSB με κλίμακα παράγοντα 20 σε ένα περιβάλλον όπου το εύρος μνήμης της CPU είναι 54 GBps, το εύρος μνήμης της GPU είναι 880 GBps και το εύρος ζώνης PCIe είναι 12,8 GBps.Πλήρεις λεπτομέρειες του συστήματος μπορούν να βρεθούν στην εικόνα 4. Συγκρίνουμε την απόδοση του συνεπεξεργαστή GPU με δύο συστήματα διαχείρισης βάσεων δεδομένων OLAP: το MonetDB και το Hyper.Η προηγούμενη έρευνα σχετικά με τη χρήση των GPU ως συνεπεξεργαστή συγκρίθηκε κυρίως με το MonetDB, το οποίο είναι γνωστό ότι είναι ανεπαρκές. Η Εικόνα 3 δείχνει τα αποτελέσματα. Κατά μέσο όρο, ο συνεπεξεργαστής GPU εκτελείται $1,5\times$ ταχύτερα απότο MonetDB, αλλά είναι $1,4\times$ πιο αργό



Εικ. 3. Αξιολόγηση του Start Schema Benchmark

από το Hyper.Για όλες τις ερωτήσεις(queries), ο χρόνος εκτέλεσης του ερωτήματος στον συνεπεξεργαστή GPU είναι περιορισμένος από τον χρόνο μεταφοράς PCIe. Συμπεραίνουμε ότι ο λόγος που η προηγούμενη έρευνα μπόρεσε να δείξει βελτίωση της απόδοσης με έναν συνεπεξεργαστή GPU είναι επειδή οι βελτιστοποιημένες υλοποιήσεις τους συγκρίθηκαν με ανεπαρκείς βάσεις (π.χ., MonetDB) στη CPU.

Platform	CPU	GPU
Model	Intel i7-6900	Nvidia V100
Cores	8 (16 with SMT)	5000
Memory Capacity	64 GB	32 GB
L1 Size	32KB/Core	16KB/SM
L2 Size	256KB/Core	6MB (Total)
L3 Size	20MB (Total)	-
Read Bandwidth	53GBps	880GBps
Write Bandwidth	55GBps	880GBps
L1 Bandwidth	-	10.7TBps
L2 Bandwidth	-	2.2TBps
L3 Bandwidth	157GBps	-

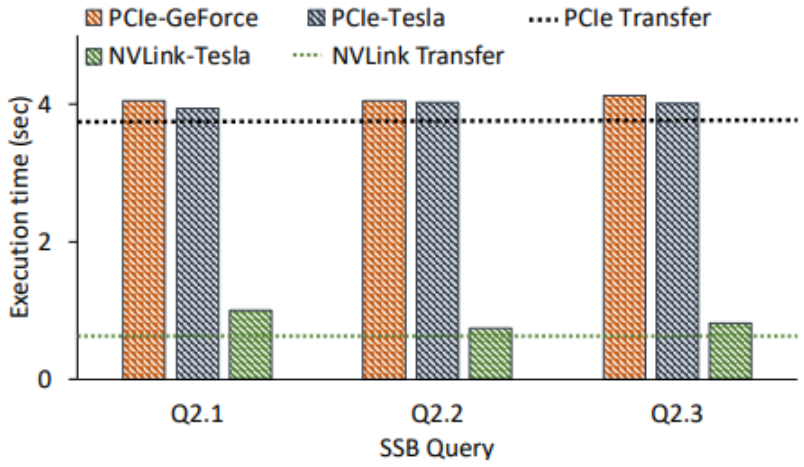
Εικ. 4. Λεπτομέρειες συστήματος

3.2 PCIe vs NVLink για το μοντέλο συνεπεξεργαστή

Τα H2TAP συστήματα ενώνουν τον επιταχυνόμενο αναλυτικό κινητήρα με την συναλλαγματική αποθήκευση μέσω του ενδιάμεσου συνδέσμου(interconnect). Ο ενδιάμεσος συνδεσμος είναι το κεντρικό στοιχείο του H2TAP και ο πιο κρίσιμος παράγοντας στο προφίλ απόδοσης των δύο συνιστωσών κινητήρων. Καταρχάς, το εύρος μνήμης του

ενδιάμεσου συνδέσμου επιβάλλει έναν αυστηρό περιορισμό στην αναλυτική απόδοση, επειδή ο κινητήρας χρειάζεται να μεταφέρει τις στήλες προς σάρωση στη GPU για κάθε αναλυτική επεξεργασία. Δεύτερον, η χρήση του ενδιάμεσου συνδέσμου καταναλώνει εύρος μνήμης και επομένως λειτουργεί ως μετρική για τον κοινό διαμοιρασμό πόρων μεταξύ του συναλλαγματικού και του αναλυτικού κινητήρα.

Η εικόνα 5 παρουσιάζει την επίδραση του ενδιάμεσου συνδέσμου στην αναλυτική επεξεργασία ερωτήσεων μέσω της εμφάνισης των χρόνων εκτέλεσης της οικογένειας ερωτήσεων 2 από το Star Schema Benchmark σε συνδυασμό με καταναλωτικές και εξυπηρετικές GPU. Οι ράβδοι αντιπροσωπεύουν τον χρόνο εκτέλεσης για τις ερωτήσεις Q2.1, Q2.2 και Q2.3, οι οποίες έχουν το ίδιο πρότυπο ερώτησης και φθίνουσα επιλεκτικότητα από 1 έως 3, με συνδυασμό εξυπηρετητικών GPU με PCIe και NVLink, και καταναλωτικής κατηγορίας GPU με PCIe. Οι παύλες δείχνουν το χρόνο που απαιτείται για τη μεταφορά του working set των queries(9.2GB) στη GPU χρησιμοποιώντας διαφορετικούς μηχανισμούς ενδιάμεσου συνδέσμου συγκεκριμένα το PCIe(μαύρο χρώμα) και το NVLink(πράσινο χρώμα).



Εικ. 5. Αποτελέσματα εκτέλασης

3.3 Παρατηρήσεις/Συμπεράσματα

Από το παραπάνω πείραμα συλλέγουμε τα εξής γενικά συμπεράσματα:

1) Ο ενδιάμεσος σύνδεσμος έχει εξαιρετική σημασία για την ταχύτητα εκτέλεσης. Με την χρήση του NVLink πετυχαίνουμε αισθητά καλύτερη απόδοση απ ότι με το PCIe.

2) Η Εικόνα 5 δείχνει ότι ο χρόνος εκτέλεσης για όλα τα ερωτήματα είναι το πολύ 11% υψηλότερος από τον χρόνο μεταφοράς που απαιτεί ο ενδιάμεσος συνδεσμος για την συγκεκριμένη διαμόρφωση. Για διαμορφώσεις βασισμένες σε PCIe, η αναβάθμιση

των GPUs από καταναλωτικού βαθμού (consumer grade) GTX 1080 (320 GB/sec) σε εξυπηρετικού βαθμού (server grade) V100 (900 GB/sec) φέρνει μια μικρή βελτίωση απόδοσης της τάξης του 0,5−2,7%. Αντίθετα, η αναβάθμιση του ενδιάμεσου συνδέσμου από PCIe σε NVLink σε μια διαμόρφωση βασισμένη σε V100 οδηγεί σε επιτάχυνση της τάξης του 3,94−5,42. Το εύρος μνήμης του ενδιάμεσου συνδέσμου καθορίζει τον χρόνο εκτέλεσης του ερωτήματος, ενώ οι δυνατότητες επεξεργασίας της GPU υπο- χρησιμοποιούνται, ειδικά για πιο αργούς ενδιάμεσους συνδέσμους.

3) Το μέγεθος του εργασιακού συνόλου καθορίζει τον χρόνο εκτέλεσης του ερωτήματος. Η Εικόνα 5 δείχνει ότι ο χρόνος εκτέλεσης του ερωτήματος είναι περίπου σταθερός, ακόμα και όταν η επιλεκτικότητα μειώνεται. Οι χρήστες περιμένουν ότι τα ερωτήματα με πιο επιλεκτικά φίλτρα θα έχουν μικρότερο χρόνο μέχρι το αποτέλεσμα. Η προσδοκία ισχύει για τα αναλυτικά συστήματα που βασίζονται σε CPU τα οποία έχουν πρόσβαση στην κύρια μνήμη απευθείας, αλλά όχι για τα συστήματα επιτάχυνσης με GPU που μεταφέρουν ολόκληρο το εργασιακό σύνολο του ερωτήματος μέσω του ενδιάμεσου συνδέσμου, ένα διάνυσμα κάθε φορά.

Οι δύο τελευταίες ιδιότητες των κινητήρων OLAP που επιταχύνονται από GPU είναι ανεπιθύμητες. Υποδεικνύουν ότι ο αναλυτικός κινητήρας αποτυγχάνει να εκμεταλλευτεί πλήρως τα χαρακτηριστικά του hardware και του φορτίου εργασίας(workload) στο περιβάλλον εκτέλεσης. Συνεπώς θα συνεχίσουμε την έρευνα μας προτείνωντας τροποποιήσεις στο παράδειγμα εκτέλεσης που επιτρέπουν στον κινητήρα μας να αντιμετωπίσει αυτές τις προκλήσεις μειώνοντας τηη πίεση στον ενδιάμεσο συνδέσμου (interconnect) και την κύρια μνήμη.

4 Βελτιστοποιήσεις μεταφοράς (data transfers)

4.1 Μεθοδολογία

Αυτή η ενότητα αξιολογεί την απόδοση διαφορετικών τεχνικών μεταφοράς δεδομένων κατά μήκος δύο αξόνων: τα χαρακτηριστικά του ενδιάμεσου συνδέσμου και την επιλεκτικότητα του ερωτήματος. Πρώτα, δείχνουμε πώς το εύρος ζώνης του ενδιάμεσου συνδέσμου επηρεάζει την εκτέλεση του ερωτήματος και στη συνέχεια αξιολογούμε την απόδοση που επιτυγχάνεται με την αργότερη πρόσβαση στην είσοδο. Κάθε ερώτημα ανακτά τα απαιτούμενα δεδομένα μέσω του ενδιάμεσου συνδέσμου και κάθε κοινή μεταφορά ή αποθήκευση απενεργοποιείται, για να προσομοιώσουμε την περίπτωση ανάγνωσης φρέσκων δεδομένων από τη CPU. Το Eager προφορτώνει όλα τα δεδομένα στη μνήμη του GPU. Στη μέθοδο Lazy, όλα τα δεδομένα προσπελαύνονται απευθείας από τα νήματα του GPU κατά τη διάρκεια της εκτέλεσης, χωρίς έναν ρητό ανιχνευτή. Ως αποτέλεσμα, κατά τη διάρκεια της εκτέλεσης του πυρήνα, τα νήματα του GPU αντιμετωπίζουν υψηλότερη καθυστέρηση κατά τις εντολές φόρτωσης για δεδομένα εισόδου, σε σύγκριση με τη μέθοδο Eager, όπου τα δεδομένα προσπελαύνονται από την τοπική μνήμη του GPU. Για να μειώσουμε τις αιτήσεις για απομακρυσμένα δεδομένα και, συνεπώς, το πραγματικό όγκο μεταφοράς, στον παραγόμενο κώδικα, όλες οι αιτήσεις ανάγνωσης ωθούνται όσο το δυνατόν ψηλότερα στο σχέδιο ερωτήματος (query plan). Για τη μέθοδο SemiLazy, προφορτώνουμε την πρώτη προσπελαύνουσα στήλη του πίνακα

γεγονότων για κάθε ερώτημα και προσπελαύνουμε το υπόλοιπο του εργασιακού συνόλου χρησιμοποιώντας τη μέθοδο Lazy.

4.2 Eager Method

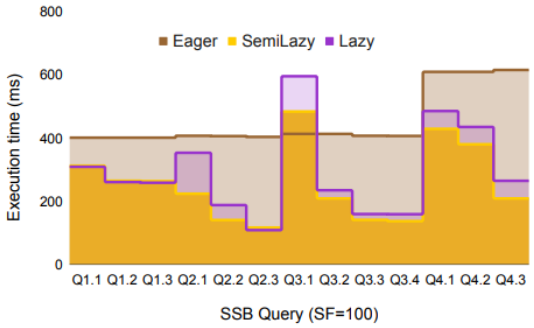
Οι μέθοδοι πρόσβασης Eager επιτυγχάνουν εύρος ζώνης (εργασιακό σύνολο πάνω στο χρόνο εκτέλεσης) πολύ κοντά στο εύρος ζώνης του ενδιάμεσου συνδέσμου για όλες τις τρεις διαμορφώσεις υλικού. Η μοναδική εξαίρεση είναι το ερώτημα Q3.1 στο NT, το οποίο έχει τη χαμηλότερη επιλεκτικότητα και σε συνδυασμό με τις πολλαπλές συζεύξεις που επιβαρύνουν τις μνήμες του GPU, προκαλεί υψηλές εμπλοκές μνήμης. Το εύρος ζώνης του ενδιάμεσου συνδέσμου PCIe είναι αρκετά χαμηλό ώστε αυτές οι εμπλοκές να έχουν ελάχιστη επίδραση στον χρόνο εκτέλεσης, καθώς κρύβονται από τον χρόνο μεταφοράς. Αντίθετα, το περίπου 5 φορές υψηλότερο εύρος ζώνης του NVLink 2 καθιστά τους χρόνους μεταφοράς πιο αργούς από τους χρόνους εκτέλεσης του πυρήνα, προκαλώντας τις εμπλοκές να γίνουν το νέο σημείο που περιορίζει την απόδοση. Για όλα τα άλλα ερωτήματα, η εκτέλεση του πυρήνα επικαλύπτεται και κρύβεται από τις μεταφορές δεδομένων.

4.3 Lazy Method

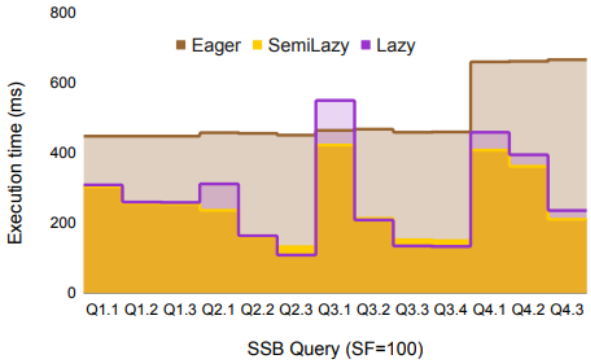
Οι μέθοδοι αργού προσβάσεως(Lazy) επιτρέπουν τη μείωση του όγκου των μεταφερόμενων δεδομένων. Τα ερωτήματα SSB είναι υψηλά επιλεκτικά και, συνεπώς, η απόδοση βελτιώνεται σχεδόν σε όλα τα ερωτήματα, εκτός από το ερώτημα 3.1. Καθώς το ερώτημα Q3.1 απαιτεί πολύ υπολογιστική ισχύ, η απόδοσή του υποβαθμίζεται λόγω της αυξημένης καθυστέρησης που εκδηλώνεται ως memory stall. Εντός κάθε ομάδας ερωτημάτων, το όφελος σε σύγκριση με τη μέθοδο Eager αυξάνεται, καθώς τα ερωτήματα γίνονται πιο επιλεκτικά.

4.4 Semi-Lazy Method

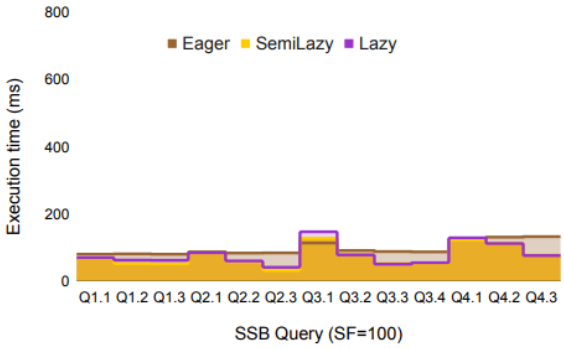
Η μέθοδος SemiLazy βελτιώνει τη μέθοδο Lazy με το να μειώνει το κύριο κόστος για τις αργές μεθόδους πρόσβασης. Ενώ το laziness μειώνει τα μεταφερόμενα δεδομένα, το κάνει με το κόστος υψηλότερης καθυστέρησης μνήμης. Παρ' όλα αυτά, κάποιες στήλες προσπελαίνονται(prefetched) σχεδόν ολόκληρες. Η προεπιλογή αυτών των στηλών μειώνει τον υπερβολικό φόρτο(overhead) της lazy μεθόδου κάνοντας όμως fetch επιπλέον πλειάδες. Αυτή η προσέγγιση φέρνει την απόδοση πιο κοντά στους αναμενόμενους χρόνους εκτέλεσης που προβλέπονται λαμβάνοντας υπόψη την επιλεκτικότητα κάθε μεμονωμένης λειτουργίας. Το μεγαλύτερο αποτέλεσμα της SemiLazy είναι στη GPU χαμηλής απόδοσης, όπου οι υπολογιστικοί πόροι είναι περιορισμένοι σε σύγκριση με τις GPU ποιότητας εξυπηρετητή. Επιπλέον, ενώ η Lazy βελτιώνει κυρίως την απόδοση πολύ επιλεκτικών ερωτημάτων, οι μέθοδοι πρόσβασης SemiLazy βελτιώνουν τα ερωτήματα που είναι λιγότερο επιλεκτικά και έχουν πολλαπλές εξαρτώμενες λειτουργίες, όπως το Q2.1. Η μικρότερη καθυστέρηση (latency) του NVLink, σε σύγκριση με το PCIe, μειώνει την επίδραση αυτής της μεθόδου.



(a) PG (consumer-grade GPU & PCIe 3)



(b) PT (server-grade GPU & PCIe 3)



(c) NT (server-grade GPU & NVLink 2.0)

4.5 Συμπεράσματα

Οι μέθοδοι Eager μειώνουν τις εμπλοκές μνήμης κατά τη διάρκεια της εκτέλεσης και επιτρέπουν μεγαλύτερες ευκαιρίες για την κρυφή αποθήκευση και κοινή χρήση δεδομένων που μεταφέρονται στις μονάδες επεξεργασίας γραφικών (GPUs). Οι μέθοδοι μεταφοράς eager μεταφέρουν περιττά δεδομένα και έτσι η εκτέλεση επιλεκτικών ερωτημάτων δεν γίνεται με την αναμενώμενη ταχύτητα. Από την άλλη πλευρά, οι μέθοδοι αργής πρόσβασης μειώνουν τον χρόνο εκτέλεσης των υψηλά επιλεκτικών ερωτημάτων, αποφεύγοντας περιττές μεταφορές, αυξάνοντας όμως τις εμπλοκές μνήμης (memory stalls) και τις εξαρτήσεις δεδομένων που συνδυάζονται με μερικώς μεταφερμένες στήλες και εμποδίζουν την επαναχρησιμοποίηση των στηλών. Η SemiLazy προσαρμόζεται μεταξύ των δύο για να επιτύχει το καλύτερο από τους δύο κόσμους, χρησιμοποιώντας μια διαφορετική μέθοδο για κάθε στήλη: αν ένα ερώτημα πρόκειται να αγγίξει την πλειονότητα μιας στήλης, θα προεπιλεγεί ενεργά, ενώ οι στήλες που προσπελάζονται μόνο ανάλογα με σιγκεκριμένες συνθήκες προσπελάζονται με αργή πρόσβαση. Αυτό έχει ως αποτέλεσμα η SemiLazy να προσαρμόζεται μεταξύ των δύο μεθόδων και να βελτιώνει την απόδοση των ερωτημάτων που είναι επιλεκτικά αλλά και απαιτητικά σε υπολογισμούς.

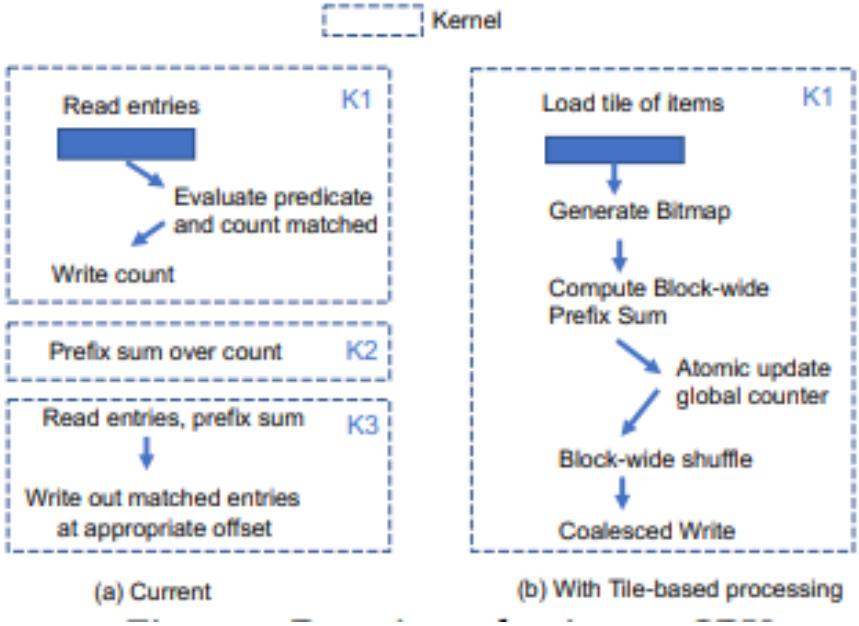
5 Χρησιμοποιώντας την GPU ως primary execution engine

5.1 Tile-based Execution model

Ενώ μία σύγχρονη CPU μπορεί να έχει δεκάδες πυρήνες, μία σύγχρονη GPU όπως η Nvidia V100 μπορεί να έχει 5000 πυρήνες. Η τεράστια αύξηση του παραλληλισμού εισάγει μερικές μοναδικές προκλήσεις για την επεξεργασία δεδομένων. Για να το δούμε αυτό, ας εξετάσουμε την εκτέλεση του ακόλουθου απλού ερωτήματος επιλογής ως μια μικρο-δοκιμή σε ένα CPU και ένα GPU:

$$Q0 : \text{SELECT } y \text{ FROM } R \text{ WHERE } y > v;$$

Στο CPU, το ερώτημα μπορεί να εκτελεστεί αποτελεσματικά ως εξής. Τα δεδομένα διαμορφώνονται ισόποσα ανάμεσα στους πυρήνες. Ο στόχος είναι να γραφούν τα αποτελέσματα παράλληλα σε ένα συνεχές πίνακα εξόδου. Το σύστημα διατηρεί ένα γενικό ατομικό μετρητή που λειτουργεί ως δείκτης και λέει σε κάθε νήμα πού να γράψει το επόμενο αποτέλεσμα. Κάθε πυρήνας επεξεργάζεται το διαμορφωμένο του μέρος επαναλαμβάνοντας τις εγγραφές στο διαμορφωμένο μέρος ένα διάνυσμα εγγραφών τη φορά, όπου ένα διάνυσμα είναι περίπου 1000 εγγραφές (αρκετά μικρό για να χωρέσει στην L1 μνήμη cache). Κάθε πυρήνας κάνει μια πρώτη διέλευση πάνω από το πρώτο διάνυσμα εγγραφών για να μετρήσει τον αριθμό των εγγραφών που ταιριάζουν στην πρόταση. Το νήμα αυξάνει τον γενικό μετρητή κατά το d για να εκχωρήσει χώρο εξόδου για τα ταιριασμένα αρχεία, και στη συνέχεια κάνει μια δεύτερη διέλευση πάνω από το διάνυσμα για να αντιγράψει τις ταιριασμένες εγγραφές στον πίνακα εξόδου στον εκχωρημένο εύρος της έξοδου. Επειδή η δεύτερη διέλευση διαβάζει δεδομένα από την L1 μνήμη cache, η ανάγνωση είναι ουσιαστικά δωρεάν. Ο γενικός ατομικός μετρητής είναι ένα σημείο πιθανής επιβράδυνσης. Ωστόσο, σημειώστε ότι κάθε νήμα ενημερώνει τον μετρητή μία φορά για κάθε 1000+ εγγραφές και υπάρχουν περίπου 32 νήματα που εκτελούνται παράλληλα σε οποιαδήποτε στιγμή. Συνεπώς ο μετρητής δεν καταλήγει να γίνεται bottleneck και ο συνολικός χρόνος εκτέλεσης είναι περίπου $D/BC + D\sigma/BC$, όπου D είναι το μέγεθος της στήλης και BC είναι η εύρος ζώνης μνήμης στο CPU. Θα μπορούσαμε να εκτελέσουμε το ίδιο σχέδιο στο GPU, διαμορφώντας τα δεδομένα μεταξύ των χιλιάδων νημάτων. Ωστόσο, τα νήματα του GPU έχουν σημαντικά λιγότερους πόρους ανά νήμα. Στο Nvidia V100, κάθε νήμα του GPU μπορεί να αποθηκεύει περίπου 24 4-byte εγγραφές στην κοινόχρηστη μνήμη σε πλήρη απασχόληση, με 5000 νήματα που εκτελούνται παράλληλα. Εδώ, ο γενικός ατομικός μετρητής καταλήγει να γίνεται bottleneck καθώς όλα τα νήματα θα προσπαθήσουν να αυξήσουν τον μετρητή για να βρουν την θέση στον πίνακα εξόδου. Για να αντιμετωπίσουμε αυτό, υπάρχουσες βάσεις δεδομένων που βασίζονται σε GPU θα εκτελούσαν αυτό το ερώτημα σε 3 βήματα όπως φαίνεται στο Σχήμα 6(a).

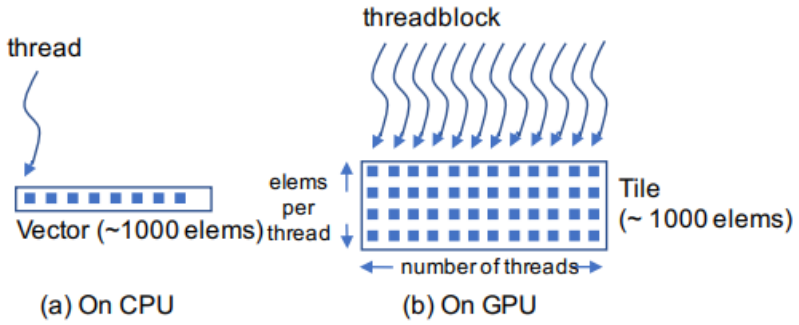


Εικ. 6. Εκτέλεση selection query σε GPU

Ο πρώτος πυρήνας $K1$ θα εκκινούσε σε ένα μεγάλο αριθμό νημάτων. Σε αυτόν, κάθε νήμα θα διάβαζε τις καταχωρήσεις των στηλών με τρόπο διατεταγμένο (διαδοχικά ανά αριθμό νήματος) και θα αξιολογούσε την πρόταση για να μετρήσει τον αριθμό των ταιριασμένων καταχωρήσεων. Αφού επεξεργαστεί όλα τα στοιχεία, ο συνολικός αριθμός των ταιριασμένων καταχωρήσεων ανά νήμα θα καταγραφεί σε έναν πίνακα *count*, όπου *count*$[t]$ είναι ο αριθμός των ταιριασμένων καταχωρήσεων ανα νήμα t. Ο δεύτερος πυρήνας $K2$ θα χρησιμοποιούσε τον πίνακα *count* για να υπολογίσει το prefix sum του αριθμού των καταχωρήσεων και θα το αποθήκευε σε έναν άλλον πίνακα *pf*. Για έναν πίνακα A με k στοιχεία, το prefix sum pA είναι ένας πίνακας με k στοιχεία όπου $pA[j] = \sum_{i=0}^{j-1} A[i]$. Έτσι, η i-οστή είσοδος στο *pf* υποδεικνύει το offset στο οποίο το i-οστό νήμα θα γράψει τα ταιριασμένα αποτελέσματά του στον πίνακα εξόδου o. Οι βάσεις δεδομένων χρησιμοποίησαν μία βελτιστοποιημένη ρουτίνα από μια βιβλιοθήκη CUDA όπως η Thrust για να το εκτελέσουν αποτελεσματικά παράλληλα. Ο τρίτος πυρήνας $K3$ θα διάβαζε ξανά τη στήλη εισόδου. Εδώ το i-οστό νήμα θα σκανάρει ξανά το i-οστό stride της εισόδου, χρησιμοποιώντας το *pf[i]* για να καθορίσει πού θα γράψει τις ικανοποιητικές εγγραφές. Κάθε νήμα διατηρεί επίσης έναν τοπικό μετρητή c_i, αρχικά ρυθμισμένο σε 0. Ειδικά για κάθε ικανοποιητική καταχώρηση, το νήμα i την γράφει στο *pf[i]* + *ci* και στη συνέχεια αυξάνει το ci. Τελικά, $o[pf[t]] \ldots o[pf[t+1]-1]$ θα περιέχει τις ταιριασμένες καταχωρήσεις του νήματος t.

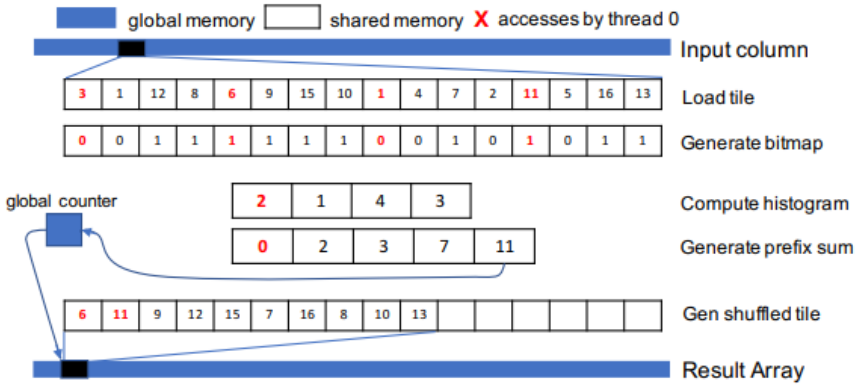
Η παραπάνω προσέγγιση μεταφέρει την εργασία της εύρεσης των offsets στον πίνακα εξόδου σε έναν βελτιστοποιημένο πυρήνα προθέματος άθροισματος (optimized prefix sum kernel), η χρονική διάρκεια του οποίου εξαρτάται από το T (όπου T είναι ο αριθμός των νημάτων, $T << n$), αντί να το βρίσκει εντός γραμμής χρησιμοποιώντας ατομικές ενημερώσεις σε ένα μετρητή. Ως αποτέλεσμα, η προσέγγιση αποδεικνύεται σημαντικά πιο γρήγορη από την απλή μετάφραση της προσέγγισης του CPU στο GPU. Ωστόσο, υπάρχουν αρκετά ζητήματα με αυτήν την προσέγγιση. Πρώτον, διαβάζει τη στήλη εισόδου από την καθολική μνήμη δύο φορές, σε σύγκριση με την ανάγκη να γίνεται μόνο μια φορά με την προσέγγιση του CPU. Διαβάζει/ γράφει επίσης σε ενδιάμεσες δομές *count* και *pf*. Τέλος, κάθε νήμα γράφει σε διαφορετική θέση στον πίνακα εξόδου, με αποτέλεσμα τυχαίες εγγραφές. Για να αντιμετωπίσει αυτά τα προβλήματα, εισάγουμε το μοντέλο εκτέλεσης με βάση το πλακίδιο (Tile based execution model).

Η επεξεργασία με βάση το πλακίδιο επεκτείνει την επεξεργασία με βάση το διάνυσμα στον CPU, όπου κάθε νήμα επεξεργάζεται ένα διάνυσμα ανά φορά στο GPU. Η Εικόνα 7 επεξηγεί το μοντέλο.



Εικ. 7. Vector-based to Tile-based execution models.

Τα νήματα στο GPU ομαδοποιούνται σε τμήματα νημάτων. Τα νήματα εντός ενός τμήματος νημάτων μπορούν να επικοινωνούν μέσω κοινόχρηστης μνήμης και να συγχρονίζονται μέσω φραγμών. Επομένως, ακόμα και αν ένα μόνο νήμα στο GPU σε πλήρη απασχόληση μπορεί να κρατήσει μόνο έως και 24 ακέραιους στην κοινόχρηστη μνήμη, ένα μόνο τμήμα νημάτων μπορεί να κρατήσει ένα σημαντικά μεγαλύτερο σύνολο στοιχείων συλλογικά μεταξύ τους στην κοινόχρηστη μνήμη. Αυτή τη μονάδα την ονομάζουμε ”Πλακίδιο”. Στο μοντέλο εκτέλεσης με βάση το Πλακίδιο (tile), αντί να θεωρούμε κάθε νήμα ως ανεξάρτητη μονάδα εκτέλεσης, θεωρούμε ένα τμήμα νημάτων ως τη βασική μονάδα εκτέλεσης, με κάθε τμήμα νημάτων επεξεργαζόμενο ένα πλακίδιο καταχωρήσεων τη φορά. Ένα πλεονέκτημα αυτής της προσέγγισης είναι ότι αφού ένα πλακίδιο φορτώθηκε στην κοινόχρηστη μνήμη, οι επόμενες διαδικασίες πάνω στο πλακίδιο θα διαβαστούν απευθείας από την κοινόχρηστη μνήμη και όχι από την καθολική μνήμη.



Εικ. 8. Query Q0 Kernel running y > 5 with tile size 16 and thread block size 4

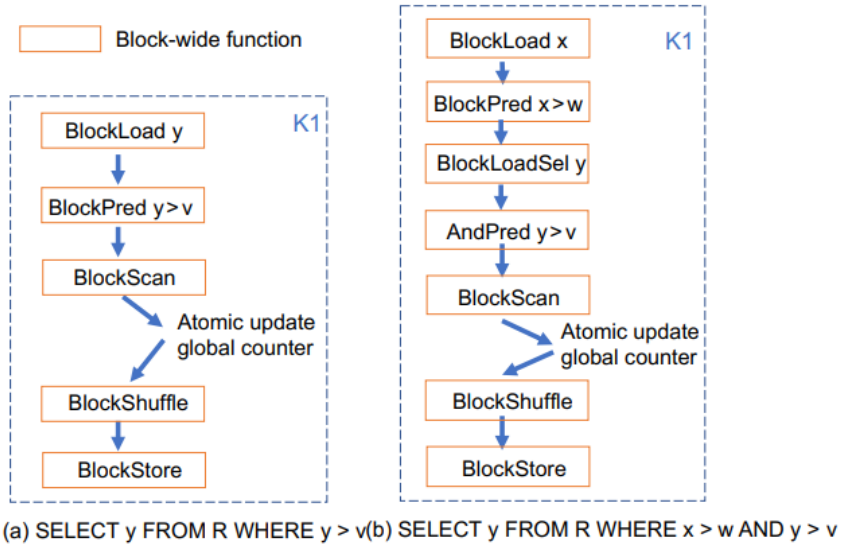
Η Εικόνα 6(b) δείχνει πώς υλοποιείται η επιλογή χρησιμοποιώντας το μοντέλο με βάση το πλακίδιο. Ολόκληρο το ερώτημα υλοποιείται ως ένας μόνο πυρήνας αντί για τρεις. Η Εικόνα 8 δείχνει μια δειγματική εκτέλεση με ένα πλακίδιο μεγέθους 16 και ένα τμήμα νημάτων με 4 νήματα για την πρόταση y > 5. Σημειώστε ότι αυτό είναι μόνο για εικονικοποίηση, καθώς οι περισσότερες σύγχρονες κάρτες γραφικών θα χρησιμοποιούσαν ένα μέγεθος τμήματος νημάτων που είναι πολλαπλάσιο του 32 (το μέγεθος του ομίλου) και ο αριθμός των στοιχείων που φορτώνονται θα ήταν 4-16 φορές το μέγεθος του τμήματος νημάτων. Ξεκινούμε από την αρχικοποίηση του γενικού μετρητή στο 0. Ο πυρήνας φορτώνει ένα πλακίδιο αντικειμένων από την καθολική μνήμη στην κοινόχρηστη μνήμη. Τα νήματα εφαρμόζουν στη συνέχεια την πρόταση σε όλα τα στοιχεία παράλληλα για να παράγουν ένα bitmap. Για παράδειγμα, το νήμα 0 αξιολογεί την πρόταση για τα στοιχεία 0,4,8,12 (εμφανίζονται με κόκκινο). Κάθε νήμα μετράει στοιχεία που ταιριάζουν ανά νήμα για να παράγει ένα ιστόγραμμα. Το τμήμα νημάτων συνεργάζεται για να υπολογίσει το πρόθεμα άθροισμα πάνω στο ιστόγραμμα για να βρει τη θέση στην οποία κάθε νήμα θα γράψει στην κοινόχρηστη μνήμη. Στο παράδειγμα, τα νήματα 0,1,2,3 ταιριάζουν αντίστοιχα με 2,1,4,3 εγγραφές. Τα προθέματα 0,2,3,7 μας λένε ότι το νήμα 0 θα πρέπει να γράψει τις αντίστοιχες εγγραφές του στην έξοδο στο δείκτη 0, το νήμα 1 θα πρέπει να γράψει ξεκινώντας από τον δείκτη 2, κ.λπ. Αυξάνουμε έναν καθολικό μετρητή ατομικά κατά τον συνολικό αριθμό των ταιριασμένων εγγραφών για να βρούμε τον δείκτη στον οποίο το τμήμα νημάτων θα πρέπει να γράψει στον πίνακα εξόδου. Το βήμα του shuffle χρησιμοποιεί το bitmap και το πρόθεμα άθροισμα για να δημιουργήσει ένα συνεχές πίνακα ταιριασμένων εγγραφών στην κοινόχρηστη μνήμη. Το τελικό βήμα εγγραφής αντιγράφει τις συνεχείς εγγραφές από την κοινόχρηστη μνήμη στην καθολική μνήμη στον σωστό δείκτη. Με τη μεταχείριση του τμήματος νημάτων ως μονάδας εκτέλεσης, μειώνουμε τον αριθμό των ατομικών ενημερώσεων του καθολικού μετρητή κατά έναν παράγοντα του μεγέθους του πλακιδίου T. Ο πυρήνας κάνει επίσης μια μόνο διέλευση πάνω από την κολώνα εισόδου με το Gen Shuffled Tile εξασφαλίζοντας ότι η τελική εγγραφή στον πίνακα εξόδου είναι συνεκτική, επιλύο-

ντας τα δύο προβλήματα που σχετίζονται με την προσέγγιση που χρησιμοποιήθηκε στις προηγούμενες βάσεις δεδομένων του GPU.

Η γενική έννοια του μοντέλου εκτέλεσης βασισμένου σε πλακίδια, δηλαδή, τη διαίρεση των δεδομένων σε πλακίδια και την αντιστοίχιση των τμημάτων νημάτων σε πλακίδια, έχει χρησιμοποιηθεί σε άλλους τομείς όπως η επεξεργασία εικόνων και στον υπολογιστικό υψηλής απόδοσης. Ωστόσο είναι από τις πρώτες φορές που χρησιμοποιήθηκε για λειτουργίες βάσης δεδομένων. Στο επόμενο κεφάλαιο, θα παρουσιάσουμε την Crystal, μια βιβλιοθήκη πρωτογενών επεξεργασιών δεδομένων που μπορεί να υλοποιήσει ερωτήματα SQL στη GPU.

5.2 Η βιβλιοθήκη Crystal

Το Crystal2 είναι μια βιβλιοθήκη προτύπων συσκευής CUDA που υλοποιεί το πλήρες σύνολο των πρωτογενών επεξεργασιών που απαιτούνται για την εκτέλεση τυπικών αναλυτικών ερωτημάτων SQL SPJA. Η εικόνα 9(a) δείχνει ένα σχεδιάγραμμα του απλού ερωτήματος επιλογής που υλοποιείται χρησιμοποιώντας λειτουργίες πλαισίου. Η εικόνα 11 δείχνει τον πυρήνα ερωτήματος του ίδιου ερωτήματος που υλοποιείται με το Crystal. Χρησιμοποιούμε αυτό το παράδειγμα για να επισημάνουμε τα κύρια χαρακτηριστικά του Crystal.



Εικ. 9. Υλοποίηση queries χρησιμοποιώντας την crystal

Primitive	Description
`BlockLoad`	Copies a tile of items from global memory to shared memory. Uses vector instructions to load full tiles.
`BlockLoadSel`	Selectively load a tile of items from global memory to shared memory based on a bitmap.
`BlockStore`	Copies a tile of items in shared memory to device memory.
`BlockPred`	Applies a predicate to a tile of items and stores the result in a bitmap array.
`BlockScan`	Co-operatively computes prefix sum across the block. Also returns sum of all entries.
`BlockShuffle`	Uses the thread offsets along with a bitmap to locally rearrange a tile to create a contiguous array of matched entries.
`BlockLookup`	Returns matching entries from a hash table for a tile of keys.
`BlockAggregate`	Uses hierarchical reduction to compute local aggregate for a tile of items.

Εικ. 10. Εντολές και λειτουργίες τους

```
// Implements SELECT y FROM R WHERE y > v
// NT => NUM_THREADS
// IPT => ITEMS_PER_THREAD
template<int NT, int IPT>
__global__ void Q(int* y, int* out, int v, int* counter) {
int tile_size = get_tile_size();
int offset = get_tile_offset();
__shared__ struct buffer {
 int col[NT * IPT];
 int out[NT * IPT];
};
int items[IPT];
int bitmap[IPT];
int indices[IPT];

BlockLoadInt<NT, IPT>(col+offset,items,buffer.col,tile_size);
BlockPredIntGT<NT, IPT>(items,buffer.col,cutoff,bitmap);
BlockScan<NT, IPT>(bitmap,indices,buffer.col,
 num_selections,tile_size);

if(threadIdx.x == 0)
 o_off = atomic_update(counter,num_selections);

BlockShuffleInt<NT, IPT>(items,indices,buffer.out);
BlockStoreInt<NT, IPT>(buffer.out,out + o_off,num_selections);
}
```

Εικ. 11. Υλοποίηση του Q0 με την crystal

5.3 Σύγκριση απόδοσης

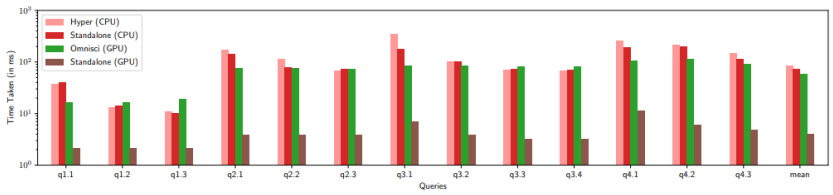
Σε αυτή την ενότητα, συγκρίνουμε τους χρόνους εκτέλεσης των ερωτημάτων του benchmark που υλοποιούνται χρησιμοποιώντας λειτουργίες πλαισίου στη GPU (Αυτόνομη GPU) με μία ισοδύναμη αποδοτική υλοποίηση του ερωτήματος στη CPU (Αυτόνομη CPU). Επίσης, συγκρίνουμε με το Hyper (Hyper), ένα προηγμένο DBMS OLAP και το Omnisci (Omnisci), ένα εμπορικό DBMS OLAP βασισμένο σε GPU. Προκειμένου να διασφαλίσουμε μια δίκαιη σύγκριση ανάμεσα στα συστήματα, κωδικοποιούμε τις στήλες συμβολοσειρών σε ακεραίους πριν από τη φόρτωση των δεδομένων και επαναγράφουμε χειροκίνητα τα ερωτήματα για να αναφέρονται απευθείας στην κωδικοποιημένη με λεξικό τιμή.

Platform	CPU	GPU
Model	Intel i7-6900	Nvidia V100
Cores	8 (16 with SMT)	5000
Memory Capacity	64 GB	32 GB
L1 Size	32KB/Core	16KB/SM
L2 Size	256KB/Core	6MB (Total)
L3 Size	20MB (Total)	-
Read Bandwidth	53GBps	880GBps
Write Bandwidth	55GBps	880GBps
L1 Bandwidth	-	10.7TBps
L2 Bandwidth	-	2.2TBps
L3 Bandwidth	157GBps	-

Εικ. 12. Hardware specifications

Για παράδειγμα, ένα ερώτημα με πρόταση $s\_region =$ ΑΣΙΑ επαναγράφεται με την πρόταση $s\_region = 2$ όπου το 2 είναι η κωδικοποιημένη τιμή του 'ΑΣΙΑ'. Ορισμένες στήλες έχουν έναν μικρό αριθμό διακριτών τιμών και μπορούν να αναπαρασταθούν/κωδικοποιηθούν με τιμές 1-2 byte. Ωστόσο, στο benchmark μας βεβαιωνόμαστε ότι όλες οι εγγραφές των στηλών είναι τιμές 4-byte για να εξασφαλίσουμε την ευκολία σύγκρισης με άλλα συστήματα και να αποφύγουμε τις τεχνολογικές ανακλήσεις υλοποίησης. Ο στόχος μας είναι να κατανοήσουμε τη φύση των κερδών απόδοσης ισοδύναμων υλοποιήσεων σε GPU και CPU, και όχι να επιτύχουμε την καλύτερη διάταξη αποθήκευσης. Αποθηκεύουμε τα δεδομένα σε στήλη με κάθε στήλη αναπαριστώμενη ως ένας πίνακας τιμών 4-byte. Στη GPU, χρησιμοποιούμε μέγεθος τεμαχίου (tile) 2056 (= 8×256) με αριθμό νημάτων ανά τεμάχιο 256, προκαλώντας 8 εγγραφές ανά νήμα ανά

τεμάχιο. Η Εικόνα 13 δείχνει τα αποτελέσματα. Συγκρίνοντας την Αυτόνομη CPU με το Hyper δείχνει ότι η πρώτη κάνει κατά μέσο όρο 1,17 φορές καλύτερα από το δεύτερο. Πιστεύουμε ότι το Hyper χάνει ευκαιρίες διανυσματοποίησης και χρησιμοποιεί μια διαφορετική υλοποίηση πινάκων κατακερματισμού. Η σύγκριση δείχνει ότι η δική μας υλοποίηση είναι μια δίκαιη σύγκριση και είναι αρκετά ανταγωνιστική σε σχέση με ένα OLAP DBMS κορυφαίας τεχνολογίας. Συγκρίναμε επίσης με το MonetDB, ένα δημοφιλές σημείο αναφοράς για πολλές από τις παρελθούσες εργασίες σχετικά με βάσεις δεδομένων που βασίζονται σε GPU. Βρήκαμε ότι η Αυτόνομη CPU είναι κατά μέσο όρο 2,5 φορές ταχύτερη από το MonetDB. Δεν το συμπεριλάβαμε στο σχήμα καθώς έκανε το γράφημα δυσανάγνωστο. Προσπαθήσαμε επίσης να συγκρίνουμε με το Pelaton με χαλαρή συγχώνευση τελεστών. Βρήκαμε ότι το σύστημα δεν μπορούσε να φορτώσει το σύνολο δεδομένων κλίμακας 20. Καταλήγοντας στην κλίμακα παράγοντα 10, τα ερωτήματά του ήταν σημαντικά πιο αργά (>5×) από το Hyper ή την προσέγγισή μας. Συγκρίνοντας την Αυτόνομη GPU με το Omnisci, βλέπουμε ότι η υλοποίηση της GPU μας το κάνει σημαντικά καλύτερα από το Omnisci με μια μέση βελτίωση περίπου 16×. Και οι δύο μέθοδοι λειτουργούν με όλη την εργαστηριακή σειρά αποθηκευμένη στη GPU. Το Omnisci αντιμετωπίζει κάθε νήμα GPU ως ανεξάρτητη μονάδα. Ως αποτέλεσμα, δεν αντιλαμβάνεται τα οφέλη της φόρτωσης τεμαχισμού και καλύτερης εκμετάλλευσης της GPU από τη χρήση του μοντέλου με βάση το tile. Η σύγκριση της Αυτόνομης GPU έναντι του Omnisci και της Αυτόνομης CPU έναντι του Hyper λειτουργούν ως έλεγχος λογικής και δείχνουν ότι οι υλοποιήσεις των ερωτημάτων μας είναι αρκετά ανταγωνιστικές.



Εικ. 13. Final Results

```
SELECT SUM(lo_revenue) AS revenue, d_year, p_brand
FROM lineorder, date, part, supplier
WHERE lo_orderdate = d_datekey
AND lo_partkey = p_partkey AND lo_suppkey = s_suppkey
AND p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY d_year, p_brand
```

Εικ. 14. SSB Query 2.1

Συγκρίνοντας την Αυτόνομη GPU με την Αυτόνομη CPU, βλέπουμε ότι η Αυτόνομη GPU είναι μέσως 25 φορές ταχύτερη από την υλοποίηση της CPU. Αυτό είναι υψηλότερο από τον λόγο εύρους ζώνης GPU προς CPU που είναι ίσος με 16.2. Ο κύριος λόγος για το γεγονός ότι το κέρδος απόδοσης είναι υψηλότερο από τον λόγο εύρους ζώνης είναι η καλύτερη ικανότητα απόκρυψης καθυστέρησης των GPU.

6 Conclusion

Σε αυτό το άρθρο, επιδείξαμε γιατί το μοντέλο της GPU ως συνεπεξεργαστή με διασύνδεση PCIe δεν είναι βέλτιστο και εκτιμήσαμε επίσης

τεχνικές φόρτωσης δεδομένων με τη μέθοδο της αργής φόρτωσης και κοινής χρήσης μεταφορών δεδομένων. Επιπλέον, αποδείξαμε ότι η εκτέλεση ολόκληρου ενός ερωτήματος SQL σε μια GPU παρέχει καλύτερη απόδοση από τη χρήση της GPU ως επιταχυντή. Για να διευκολύνουμε την υλοποίηση υψηλής απόδοσης ερωτημάτων SQL σε GPUs, χρησιμοποιήσαμε τη Crystal, μια βιβλιοθήκη που υποστηρίζει ένα μοντέλο εκτέλεσης βασισμένο σε πλακίδια. Η ανάλυσή μας στο SSB, ένα δημοφιλές benchmark, δείχνει ότι οι σύγχρονες GPUs είναι 25 φορές ταχύτερες και 4 φορές πιο οικονομικές από τις CPUs. Αυτό δημιουργεί ισχυρή βάση για τη χρήση GPUs ως τον κύριο μηχανισμό εκτέλεσης όταν το σύνολο δεδομένων χωράει στη μνήμη τους.

Αναφορές

6.1 Βασικές έρευνες που μελετήθηκαν

1. https://infoscience.epfl.ch/record/277001
2. https://dl.acm.org/doi/pdf/10.1145/3318464.3380595
3. https://www.mdpi.com/2227-9709/4/3/24

6.2 Έρευνες που αναφέρονται στο άρθρο

4. CUDA C Programming Guide. http://docs.nvidia.com/cuda/cuda-cprogramming-guide/index.html.

5. Opencl. https://www.khronos.org/opencl/
6. H. Funke, S. Breß, S. Noll, V. Markl, and J. Teubner. Pipelined query processing in coprocessor environments. In Proceedings of the 2018 International Conference on Management of Data, pages 1603–1618. ACM, 2018
7. https://community.fs.com/article/an-overview-of-nvidia-nvlink.html
8. T. Neumann. Efficiently compiling efficient query plans for modern hardware. Proceedings of the VLDB Endowment, 4(9):539–550, 2011.