Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Βελτιστοποίηση του Υπολογιστικού Πυρήνα Πολλαπλασιασμού Αραιού Πίνακα με Πίνακα σε Επεξεργαστές Γραφικών

Διπλωματική εργασία

Ειρήνη Θ. Κουτσανίτη
Μαρία Ν. Σταθοπούλου

Επιβλέπων: Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ελλάδα
Μάρτιος, 2019

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Βελτιστοποίηση του Υπολογιστικού Πυρήνα Πολλαπλασιασμού Αραιού Πίνακα με Πίνακα σε Επεξεργαστές Γραφικών

Διπλωματική εργασία

Ειρήνη Θ. Κουτσανίτη
Μαρία Ν. Σταθοπούλου

Επιβλέπων: Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή επιτροπή την 22 Μαρτίου 2019.

_____	_____	_____
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Νικόλαος Παπασπύρου
Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Ελλάδα
Μάρτιος, 2019

Ειρήνη Κουτσανίτη
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Μαρία Σταθοπούλου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο πολλαπλασιασμός Αραιού Πίνακα με Πίνακα αποτελεί σημαντικό μέρος του συνόλου των υπολογισμών που πραγματοποιούνται σε πολλές επιστημονικές εφαρμογές. Δύο από αυτές είναι τα βαθιά νευρωνικά δίκτυα και η επίλυση αραιών γραμμικών συστημάτων. Αραιός πίνακας ονομάζεται ένας πίνακας που περιέχει μεγάλο πλήθος μηδενικών στοιχείων. Η αποθήκευση ενός αραιού πίνακα, με την κλασική πυκνή αναπαράσταση, δεσμεύει περιττό χώρο στη μνήμη, καθώς τα μηδενικά στοιχεία δεν αποτελούν χρήσιμη πληροφορία. Η επίδοση ενός υπολογιστικού πυρήνα πολλαπλασιασμού Αραιού Πίνακα με Πίνακα είναι άρρηκτα συνδεδεμένη με τον τρόπο αναπαράστασης του αραιού πίνακα στη μνήμη. Για το λόγο αυτό κρίνεται επιτακτική η ανάγκη ανεύρεσης μιας δομής αποθήκευσης αραιών πινάκων, η οποία θα μειώνει τις μεταφορές των δεδομένων από/προς τη μνήμη που είναι απαραίτητα στον υπολογιστικό πυρήνα SpMM. Πολλές δομές αποθήκευσης αραιών πινάκων έχουν ήδη προταθεί, ωστόσο κρίνονται ανεπαρκείς κατά την εφαρμογή τους στον υπολογιστικό πυρήνα πολλαπλασιασμού Αραιού Πίνακα με Πίνακα.
Στόχος της διπλωματικής εργασίας είναι η βελτιστοποίηση του υπολογιστικού πυρήνα πολλαπλασιασμού Αραιού Πίνακα με Πίνακα σε επεξεργαστές γραφικών. Αρχικά, προτείνεται μια νέα δομή αποθήκευσης αραιών πινάκων. Στη συνέχεια, παρουσιάζονται παράλληλες υλοποιήσεις του πυρήνα σε επεξεργαστές γραφικών, κάνοντας χρήση αυτής της δομής και τέλος συγκρίνεται η επίδοσή τους με αυτή της δημοφιλούς βιβλιοθήκης cuSPARSE της Nvidia.

Λέξεις Κλειδιά: πολλαπλασιασμός αραιού πίνακα με πίνακα, GPU, SpMM

Abstract

Sparse Matrix - Matrix Multiplication (SpMM) comprises a significant portion of the overall execution time of many scientific and engineering applications. Deep neural networks and the solution of sparse linear systems are two typical examples. Sparse matrices are finite matrices dominated by zero elements. Storing a sparse matrix using the dense representation could lead to unnecessary memory allocation, as zeros do not consist useful information. The performance of Sparse Matrix-Matrix Multiplication computational kernel is strongly intertwined with the representation of the sparse matrix in memory. As a result, it is imperative to find a compressed format that minimizes memory traffic for the SpMM computation. Many storage formats for sparse matrices have already been proposed, however, they are considered inadequate when applied to SpMM.
This diploma thesis focuses on the optimization of the SpMM kernel on GPUs. Initially, a new storage format for sparse matrices is proposed. Subsequently, many parallel implementations of the SpMM kernel are presented, using this format and finally their performance is compared to the performance of Nvidia's cuSPARSE library.

Keywords: sparse matrix-matrix multiplication, GPU, SpMM, sparse format

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο του προπτυχιακού προγράμματος σπουδών της Σχολής Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου και σηματοδοτεί την ολοκλήρωση των σπουδών μας, ενώ συγχρόνως αποτελεί το ερέθισμα για περαιτέρω έρευνα στο συγκεκριμένο αντικείμενο.

Αρχικά θα θέλαμε να απευθύνουμε τις ευχαριστίες μας στον κ. Γεώργιο Γκούμα, Επ. Καθηγητή Ε.Μ.Π, τον κ. Νεκτάριο Κοζύρη, Καθηγητή Ε.Μ.Π και τον κ. Νικόλαο Παπασπύρου, Αν. Καθηγητή Ε.Μ.Π για τη διδασκαλία τους και τις γνώσεις που μας προσέφεραν όλα αυτά τα χρόνια.

Επίσης οφείλουμε ιδιαίτερες ευχαριστίες στην Υ.Δ. Αθηνά Ελαφρού, για το χρόνο που αφιέρωσε και τη θεμελιώδη συνεισφορά της στην εκπόνηση της συγκεκριμένης εργασίας. Η προθυμία της να μας βοηθήσει μέσω της εμπειρίας και των γνώσεων της σε οποιαδήποτε δυσκολία συναντήσαμε στάθηκαν καθοριστικές και η συνεργασία μας ήταν άκρως ευχάριστη και εποικοδομητική.

Τέλος, θα θέλαμε να ευχαριστήσουμε τις οικογένειές μας και τους κοντινούς μας ανθρώπους, φίλους και συμφοιτητές, για την έμπρακτη και αδιάκοπη υποστήριξή τους σε όλη τη διάρκεια της ακαδημαϊκής μας πορείας.

Ειρήνη Κουτσανίτη και Μαρία Σταθοπούλου
Αθήνα, Μάρτιος 2019

Περιεχόμενα

1

Εισαγωγή

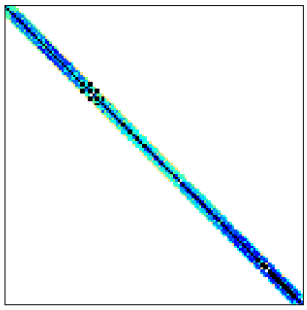
Οι αραιοί πίνακες έχουν απασχολήσει ιδιαίτερα την επιστημονική κοινότητα, καθώς εμφανίζονται σε πολλές εφαρμογές ενός ευρέος φάσματος επιστημονικών πεδίων. Για το λόγο αυτό δημιουργήθηκε η ανάγκη σχεδιασμού αλγορίθμων και τεχνικών που θα αποδίδουν ικανοποιητικά με αυτούς. Στο πλαίσιο αυτής της διπλωματικής εργασίας επικεντρωθήκαμε στην παράλληλη υλοποίηση και βελτιστοποίηση του υπολογιστικού πυρήνα πολλαπλασιασμού Αραιού Πίνακα με Πίνακα (Sparse Matrix-Matrix Multiplication - SpMM) σε επεξεργαστές γραφικών γενικού σκοπού (General Purpose Graphics Processing Units - GPUs), ο οποίος καταλαμβάνει συχνά μεγάλο μέρος του χρόνου εκτέλεσης μιας εφαρμογής.

1.1 Αραιοί Πίνακες

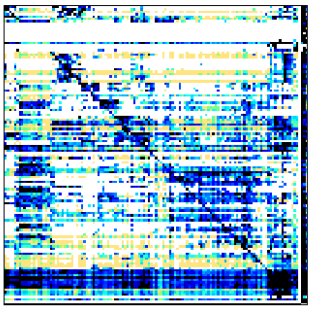
Ο όρος αραιός πίνακας θα χρησιμοποιείται στη συνέχεια αυτής της διπλωματικής για να περιγράψει πίνακες με μεγάλο πλήθος μηδενικών στοιχείων. Ορίζουμε ως αραιότητα (sparsity) ενός πίνακα το λόγο που προκύπτει από το πλήθος των στοιχείων με μηδενική τιμή προς το συνολικό πλήθος των στοιχείων του πίνακα. Για παράδειγμα μια τιμή αραιότητας 0.9 σημαίνει πως το 90% των στοιχείων του πίνακα έχουν μηδενική τιμή. Σε αντιδιαστολή με τον όρο ''αραιός πίνακας'', θα χρησιμοποιήσουμε τον όρο ''πυκνός πίνακας'' για να αναφερθούμε σε πίνακες στους οποίους η τιμή μηδέν (0) αντιμετωπίζεται όπως και οι υπόλοιπες.

Οι αραιοί πίνακες μπορούν να χωριστούν σε δύο βασικές κατηγορίες: τους δομημένους και τους μη-δομημένους. Στην πρώτη

κατηγορία τα μη-μηδενικά στοιχεία ακολουθούν κάποια κατανομή/μορφή η οποία μπορεί εύκολα να αναγνωριστεί, όπως για παράδειγμα στην περίπτωση που όλα τα μη-μηδενικά στοιχεία ανήκουν (ή βρίσκονται κοντά) στην κύρια διαγώνιο. Αντίθετα, στους μη-δομημένους πίνακες δεν δημιουργείται κάποια προφανής κατανομή που θα μπορούσαμε να εκμεταλλευτούμε για εξοικονόμηση χώρου ή υπολογισμών. Η δομή ενός αραιού πίνακα είναι ένας πολύ σημαντικός παράγοντας στην επίδοση των υπολογιστικών πυρήνων που αφορούν αραιούς πίνακες, συμπεριλαμβανομένου και του πυρήνα SpMM, όπως θα δούμε στο κεφάλαιο 4.1. Στις εικόνες 1.1 και 1.2 φαίνονται δύο χαρακτηριστικά παραδείγματα δομημένου και μη-δομημένου πίνακα αντίστοιχα.



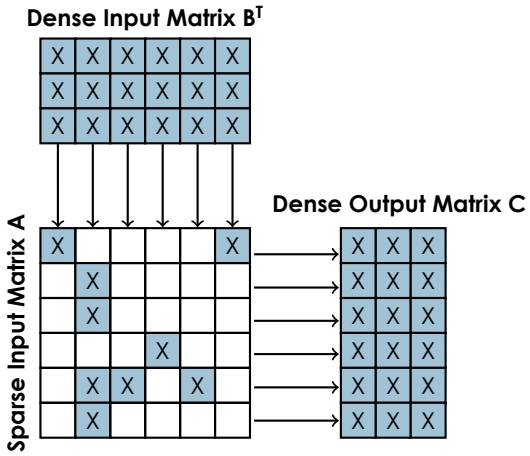
Σχήμα 1.1: Δομημένος Πίνακας



Σχήμα 1.2: Μη-δομημένος πίνακας

1.2 Ο υπολογιστικός πυρήνας SpMM

Ο πυρήνας SpMM υπολογίζει το γινόμενο δύο πινάκων, ένας εκ των οποίων είναι αραιός. Δέχεται ως είσοδο έναν αραιό πίνακα διαστάσεων $M \times K$ και έναν πυκνό πίνακα διαστάσεων $K \times N$ και έχει σαν αποτέλεσμα έναν πυκνό πίνακα $M \times N$, όπως φαίνεται και στο σχήμα 1.3. Ο πυρήνας SpMM συναντάται ευρέως σε πολλές επιστημονικές εφαρμογές. Για παράδειγμα συναντάται στον αλγόριθμο Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG), ο οποίος χρησιμοποιείται για την εύρεση των ιδιοτιμών ενός πίνακα [Anzt et al., 2015], στην υλοποίηση των συνελικτικών επιπέδων των Αραιών Νευρωνικών Δικτύων [Han et al., 2015] και στην εύρεση των κεντρικών κόμβων ενός γράφου [Sariyüce et al., 2015].



Σχήμα 1.3: Sparse Matrix-Matrix Multiplication

Η διαφορά του SpMM με την κλασική ρουτίνα πολλαπλασιασμού πινάκων (General Matrix Multiplication - GEMM) είναι πως στην πρώτη περίπτωση ο αραιός πίνακας αποθηκεύεται κάνοντας χρήση μιας συμπιεσμένης δομής αποθήκευσης, με σκοπό να εξοικονομηθεί χώρος μνήμης και να αποφευχθούν περιττοί υπολογισμοί. Συνεπώς, αν ο αραιός πίνακας έχει *nnz* μη-μηδενικά στοιχεία και δεν υπάρχουν περιοριστικοί παράγοντες (όπως για παράδειγμα περιορισμένο εύρος τιμών), ο αλγόριθμος GEMM έχει πολυπλοκότητα $O(M \cdot K \cdot N)$, ενώ ο πυρήνας SpMM έχει πολυπλοκότητα $O(nnz \cdot N)$. Στις περιπτώσεις, όπου η αραιότητα είναι υψηλή, η συμπίεση του αραιού πίνακα μπορεί να επιτύχει καλύτερα αποτελέσματα.

1.3 GPUs

Τις τελευταίες δεκαετίες, οι μονάδες επεξεργασίας γραφικών έχουν εξελιχθεί από επιταχυντές γραφικών ειδικού σκοπού σε μαζικά παράλληλους συν-επεξεργαστές γενικής χρήσης. Είναι ιδανικές για παράλληλη επεξεργασία μεγάλου όγκου δεδομένων λόγω της υψηλής υπολογιστικής επίδοσης, τόσο από πλευράς χρόνου εκτέλεσης, όσο και κατανάλωσης ενέργειας, για ένα ευρύ φάσμα προβλημάτων που αναδύονται από διάφορους επιστημονικούς κλάδους, όπως μηχανική μάθηση, στατιστική, γραμμική άλγεβρα και ιατρική. Λόγω της τεράστιας υπολογιστικής ιπποδύναμης και του υψηλού εύρους ζώνης μνήμης που διαθέτουν, αποτελούν ένα

σημαντικό μέρος των σύγχρονων υπολογιστικών συστημάτων και υιοθετούνται σε ολοένα και περισσότερους τομείς, όπως κινητές συσκευές και ενσωματωμένα συστήματα. Διάφορα μοντέλα αναπτύχθηκαν με στόχο τον προγραμματισμό των επεξεργαστών γραφικών, όπως CUDA και OpenCL, εισάγοντας την έννοια του ετερογενούς προγραμματισμού, δηλαδή τη δυνατότητα προσδιορισμού τμημάτων κώδικα που θα εκτελεστούν είτε στην κεντρική μονάδα επεξεργασίας, είτε στη GPU.

1.4 Οργάνωση του κειμένου

Το υπόλοιπο αυτής της διπλωματικής εργασίας οργανώνεται σε 4 κεφάλαια όπου παρουσιάζονται τα εμπλεκόμενα θέματα και η συμβολή μας:

Το **Κεφάλαιο 2** ξεκινά με μια επισκόπηση των συμβατικών δομών αποθήκευσης αραιών πινάκων, συνεχίζει με την παρουσίαση ορισμένων τυπικών εφαρμογών στις οποίες κυριαρχεί ο πολλαπλασιασμός Αραιού Πίνακα με Πίνακα και ολοκληρώνεται με την περιγραφή των επεξεργαστών γραφικών γενικού σκοπού.

Το **Κεφάλαιο 3** παρουσιάζει τους αλγορίθμους πολλαπλασιασμού Αραιού Πίνακα με Πίνακα χρησιμοποιώντας τις συμβατικές δομές αποθήκευσης αραιών πινάκων και προχωρά με την ανάλυση της επίδοσης του πυρήνα SpMM σε επεξεργαστές γραφικών γενικού σκοπού. Στη συνέχεια, παρουσιάζεται λεπτομερώς η δομή αποθήκευσης αραιού πίνακα BCSC, καθώς και ορισμένες παράλληλες υλοποιήσεις για τον πυρήνα SpMM χρησιμοποιώντας τη συγκεκριμένη δομή.

Το **Κεφάλαιο 4** περιγράφει την πειραματική διάταξη όπου πραγματοποιήθηκαν οι μετρήσεις, αναλύει την επιλογή των χαρακτηριστικών των δεδομένων και παρουσιάζει τα αποτελέσματα των μετρήσεων. Το κεφάλαιο ολοκληρώνεται με την αξιολόγηση της επίδοσης των διαφορετικών υλοποιήσεων του πυρήνα SpMM χρησιμοποιώντας τη δομή BCSC και τη σύγκριση με αυτή της δημοφιλούς βιβλιοθήκης cuSPARSE της Nvidia.

Τέλος, στο **Κεφάλαιο 5** συγκεντρώνονται τα συμπεράσματα της παρούσας διπλωματικής εργασίας και προτείνονται μελλοντικές επεκτάσεις της.

2

Θεωρητικό Υπόβαθρο

2.1 Συμβατικές Δομές Αποθήκευσης Αραιών Πινάκων

Ένας $M \times N$ πίνακας αποθηκεύεται τυπικά σε δισδιάστατη μορφή και κάθε στοιχείο του είναι προσπελάσιμο σε χρόνο $O(1)$ μέσω δυο δεικτών, γραμμής και στήλης αντίστοιχα. Το αποτύπωμα μνήμης που απαιτείται για την αποθήκευση του πίνακα σε αυτή τη μορφή είναι της τάξης $O(MN)$ και συγκεκριμένα $4 \times M \times N$, για τύπο δεδομένων κινητής υποδιαστολής μονής ακρίβειας, ενώ $8 \times M \times N$ για τύπο δεδομένων κινητής υποδιαστολής διπλής ακρίβειας (χωρίς να ληφθεί υπόψη το γεγονός ότι οι διαστάσεις του πίνακα πρέπει επίσης να αποθηκευτούν).

Ένας πίνακας ονομάζεται αραιός όταν το πλήθος των μη-μηδενικών στοιχείων του είναι αρκετά μικρό συγκριτικά με το μέγεθος του πίνακα. Σε αυτή την περίπτωση, μπορεί να επιτευχθεί σημαντική μείωση του αποτυπώματος μνήμης (memory footprint), αποθηκεύοντας μόνο τα μη-μηδενικά στοιχεία, γεγονός όμως που καθιστά πιο περίπλοκη την πρόσβαση σε αυτά. Ανάλογα με τον αριθμό και την κατανομή των μη-μηδενικών στοιχείων έχουν προταθεί διαφορετικές δομές αποθήκευσης, οι οποίες είναι σε θέση να ανακτήσουν τον αρχικό πίνακα και έχουν στόχο τη μείωση του αποτυπώματος μνήμης, καθώς και των μη-κανονικών αναφορών στη μνήμη για την προσπέλαση των στοιχείων.

2.1.1 Δομή Αποθήκευσης Coordinate (COO)

Ο πιο απλός τρόπος αναπαράστατης ενός αραιού πίνακα, όπως προμηνύει και το όνομα της δομής Coordinate [Tewarson, 1973], είναι με βάση τις συντεταγμένες των μη-μηδενικών στοιχείων

του. Σε μια τυπική υλοποίηση (σχήμα 2.1) χρησιμοποιούνται τρία διανύσματα: ένα διάνυσμα για τις μη-μηδενικές τιμές και δύο βοηθητικά διανύσματα για τη γραμμή και στήλη των μη-μηδενικών στοιχείων αντίστοιχα. Τυπικά, τα δύο βοηθητικά διανύσματα έχουν ακέραιο τύπο δεδομένων (4-byte integer), ενώ το διάνυσμα των μη-μηδενικών τιμών έχει τύπο δεδομένων κινητής υποδιαστολής μονής (4-byte single precision floating point) ή διπλής ακρίβειας (8-byte double precision floating point). Για έναν πίνακα με *nnz* μη-μηδενικά στοιχεία, ο συνολικός απαιτούμενος χώρος μνήμης είναι $4 \cdot nnz + 4 \cdot nnz + 4 \cdot nnz = 12 \cdot nnz$ bytes σε περίπτωση που οι τιμές των στοιχείων έχουν τύπο δεδομένων κινητής υποδιαστολής μονής ακρίβειας, ενώ $4 \cdot nnz + 4 \cdot nnz + 8 \cdot nnz = 16 \cdot nnz$ bytes σε περίπτωση διπλής ακρίβειας.

$$A = \begin{pmatrix} 7.5 & 2.9 & 2.8 & 2.7 & 0 & 0 \\ 6.8 & 5.7 & 3.8 & 0 & 0 & 0 \\ 2.4 & 6.2 & 3.2 & 0 & 0 & 0 \\ 9.7 & 0 & 0 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5.8 & 5.0 \\ 0 & 0 & 0 & 0 & 6.6 & 8.1 \end{pmatrix}$$

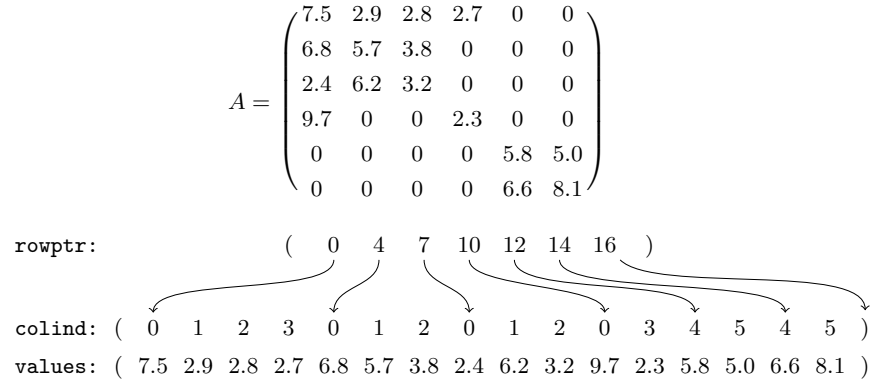
`rowind:`	(	0	0	0	0	1	1	1	2	2	2	3	3	4	4	5	5	)
`colind:`	(	0	1	2	3	0	1	2	0	1	2	0	3	4	5	4	5	)
`values:`	(	7.5	2.9	2.8	2.7	6.8	5.7	3.8	2.4	6.2	3.2	9.7	2.3	5.8	5.0	6.6	8.1	)

Σχήμα 2.1: Coordinate Δομή Αποθήκευσης Αραιού Πίνακα

Στη συγκεκριμένη δομή αποθήκευσης, δεν υπάρχει περιορισμός στη σειρά των εγγραφών, γεγονός που καθιστά εύκολη την εισαγωγή μη-μηδενικών στοιχείων. Ωστόσο κύριο μειονέκτημα αυτής της δομής είναι το μεγάλο αποτύπωμα μνήμης, το οποίο θα πρέπει να υπολογιστεί έτσι ώστε η δομή αυτή να εξοικονομεί χώρο στη μνήμη σε σχέση με την κλασική πυκνή μορφή αποθήκευσης. Πλέον για την αποθήκευση ενός μη-μηδενικού στοιχείου (ας υποθέσουμε κινητής υποδιαστολής διπλής ακρίβειας), απαιτούνται 16 bytes, έναντι 8 bytes. Συνεπώς, θα πρέπει να ισχύει $16 * ((1 - sparsity) * M * N) < 8 * M * N \Rightarrow sparsity > 0.5$, δηλαδή λιγότερα από τα μισά στοιχεία του πίνακα να είναι μη-μηδενικά. Επίσης για την εύρεση ενός στοιχείου απαιτείται, στη χειρότερη περίπτωση, χρόνος $O(nnz)$ καθώς πρέπει να εξετάσουμε όλες τις εγγραφές γραμμικά μέχρι να εντοπίσουμε τη ζητούμενη.

2.1.2 Δομή Αποθήκευσης Compressed Sparse Row (CSR)

Η δομή Compressed Sparse Row (CSR) αποτελεί την πιο διαδεδομένη δομή αποθήκευσης αραιών πινάκων. Διαφοροποιείται από τη δομή αποθήκευσης COO, καθώς αντικαθιστά το διάνυσμα που κρατά ρητά τις γραμμές για κάθε μη-μηδενικό στοιχείο με ένα διάνυσμα που περιέχει δείκτες στο διάνυσμα τιμών από όπου ξεκινά κάθε γραμμή του πίνακα. Η διάσταση του διανύσματος αυτού είναι συνεπώς ίση με το πλήθος των γραμμών του πίνακα συν μια επιπλέον εγγραφή, η οποία περιέχει το πλήθος των μη-μηδενικών στοιχείων. Τα διανύσματα που διατηρούν την τιμή και τη στήλη των μη-μηδενικών στοιχείων του πίνακα παραμένουν ίδια με αυτά της δομής αποθήκευσης COO που περιγράφηκε παραπάνω. Η εικόνα 2.2 δείχνει σχηματικά μια τυπική υλοποίηση της δομής CSR.

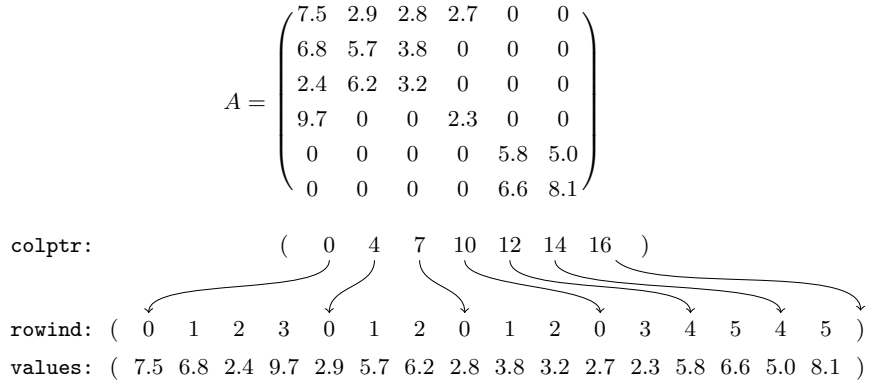


Σχήμα 2.2: Compressed Sparse Row Δομή Αποθήκευσης Αραιού Πίνακα

Για την πλειοψηφία των αραιών πινάκων, η δομή CSR μειώνει σημαντικά το αποτύπωμα μνήμης για την αποθήκευση τους, σε σύγκριση με τη δομή COO, διατηρώντας ταυτόχρονα εύκολη την κατασκευή της. Πιο συγκεκριμένα, για ένα δεδομένο πίνακα με *nnz* μη-μηδενικά στοιχεία, ο συνολικός απαιτούμενος χώρος μνήμης για τη συγκεκριμένη δομή είναι $4 \cdot (M + 1) + 4 \cdot nnz + 4 \cdot nnz = 4 \cdot M + 8 \cdot nnz + 4$ bytes σε περίπτωση τύπου δεδομένων κινητής υποδιαστολής μονής ακρίβειας και $4 \cdot (M + 1) + 4 \cdot nnz + 8 \cdot nnz = 4 \cdot M + 12 \cdot nnz + 4$ bytes σε περίπτωση διπλής ακρίβειας, δεδομένου ότι τα διανύσματα που χρησιμοποιούνται για δεικτοδότηση γραμμής και στήλης έχουν ακέραιο τύπο δεδομένων (4-byte integer).

2.1.3 Δομή Αποθήκευσης Compressed Sparse Column (CSC)

Η δομή αποθήκευσης Compressed Sparse Column (CSC) αποτελεί μια δομή αντίστοιχη του CSR, με τη διαφορά ότι αποθηκεύει τα μη-μηδενικά στοιχεία κατά στήλες, όπως φαίνεται στο σχήμα 2.3. Πιο συγκεκριμένα, διατηρεί ρητά τη γραμμή και την τιμή των μη-μηδενικών στοιχείων και διαθέτει ένα διάνυσμα που περιέχει δείκτες στο διάνυσμα τιμών, στο πρώτο στοιχείο κάθε στήλης του πίνακα.



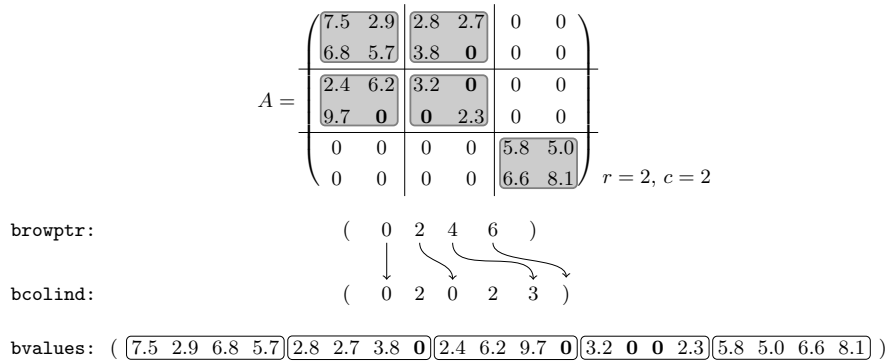
Σχήμα 2.3: Compressed Sparse Column Δομή Αποθήκευσης Αραιού Πίνακα

2.1.4 Δομή Αποθήκευσης Block Compressed Sparse Row (BSR)

Σε πολλές πραγματικές εφαρμογές, οι αραιοί πίνακες που συναντάμε περιέχουν τμήματα που εμφανίζουν κάποιο επίπεδο κανονικότητας. Μια τεχνική για να επωφεληθούμε από τέτοιου είδους τμήματα, γνωστή ως blocking, είναι να ομαδοποιήσουμε γειτονικά μη-μηδενικά στοιχεία ως πυκνά τεμάχια. Δεδομένου ότι το πλήθος των στοιχείων μέσα στο μπλοκ είναι γνωστό εκ των προτέρων θα μπορούσε να διατηρηθεί ένας μοναδικός δείκτης ανά μπλοκ, για τη δεικτοδότηση των υπόλοιπων στοιχείων του. Έχουν προταθεί διάφοροι τύποι blocking, καθένας από τους οποίους εκμεταλλεύεται διαφορετικές κατανομές των μη-μηδενικών στοιχείων (για παράδειγμα, σε ορθογώνια μπλοκ, σε διαγωνίους). Ωστόσο, θα μπορούσαμε να διακρίνουμε δύο μεγάλες κατηγορίες ομαδοποίησης: σταθερού και μεταβλητού μεγέθους.

Το BSR είναι ένα χαρακτηριστικό παράδειγμα blocking σταθερού μεγέθους και είναι πρακτικά μια παραλλαγή του CSR, όπου

αποθηκεύονται σταθερού μεγέθους $r \times c$ μπλοκ στοιχείων αντί για μεμονωμένα μη-μηδενικά στοιχεία. Ένας $M \times N$ αραιός πίνακας A αναπαρίσταται ως ένας τμηματοποιημένος πίνακας A_b με $m_b = \lceil \frac{M+r-1}{r} \rceil$ γραμμές και $n_b = \lceil \frac{N+c-1}{c} \rceil$ στήλες. Αν οι M γραμμές και οι N στήλες δεν είναι ακέραια πολλαπλάσια των r και c αντίστοιχα, τότε ο A_b συμπληρώνεται με μηδενικά. Στο σχήμα 2.4 φαίνεται ένα παράδειγμα αναπαράστασης πίνακα με χρήση της δομής αποθήκευσης BSR.



Σχήμα 2.4: Block Compressed Sparse Row Δομή Αποθήκευσης Αραιού Πίνακα

Για ένα δεδομένο πίνακα σε μορφή BSR, για τον οποίο υποθέτουμε ότι τα μη-μηδενικά στοιχεία μπορούν να οργανωθούν τέλεια σε μπλοκ μεγέθους $r \cdot c$ και δεν απατείται παραγέμισμα με μηδενικά, ο απαιτούμενος χώρος στη μνήμη είναι $4 \cdot (m_b + 1) + 4 \cdot nnzb + 4 \cdot r \cdot c \cdot nnzb$ bytes, στην περίπτωση τύπου δεδομένων κινητής υποδιαστολής μονής ακρίβειας και $4 \cdot (m_b + 1) + 4 \cdot nnzb + 8 \cdot r \cdot c \cdot nnzb$ bytes, σε περίπτωση διπλής ακρίβειας, όπου *nnzb* είναι το πλήθος των μπλοκ με τουλάχιστον ένα μη-μηδενικό στοιχείο. Σε περίπτωση που τα μη-μηδενικά στοιχεία του αρχικού πίνακα είναι ήδη ομαδοποιημένα σε μπλοκ, η αναπαράσταση αυτή έχει σημαντικά οφέλη ως προς τη μνήμη. Ωστόσο, παρά την απλότητα της αναπαράστασής τους και την εύκολη υλοποίησή τους, οι μέθοδοι blocking σταθερού μεγέθους μπορούν σε ορισμένους πίνακες να επιβαρυνθούν από την προσθήκη μηδενικών στοιχείων που απαιτούνται για την κατασκευή πλήρων μπλοκ.

2.2 Εφαρμογές του υπολογιστικού πυρήνα SpMM

Ο πυρήνας πολλαπλασιασμού Αραιού Πίνακα με Πίνακα (SpMM) χρησιμοποιείται ευρέως σε πολλές επιστημονικές εφαρμογές και η αποτελεσματική υλοποίηση και βελτιστοποίησή του υπήρξε βασικό μέλημα στην κοινότητα υψηλής υπολογιστικής επίδοσης (HPC). Σε αυτή την ενότητα περιγράφουμε δύο απ' αυτές, στις οποίες ο πυρήνας καταλαμβάνει μεγάλο κομμάτι των υπολογισμών και η βελτίωσή του θα μπορούσε να αυξήσει σημαντικά την επίδοση. Οι εφαρμογές αυτές επιλέχθηκαν για να δείξουμε ότι ο πυρήνας μπορεί να έχει οφέλη για διαφορετικές μορφές αραιών πινάκων, που παρουσιάζουν διαφορετικό βαθμό αραιότητας.

2.2.1 Αραιά Νευρωνικά Δίκτυα

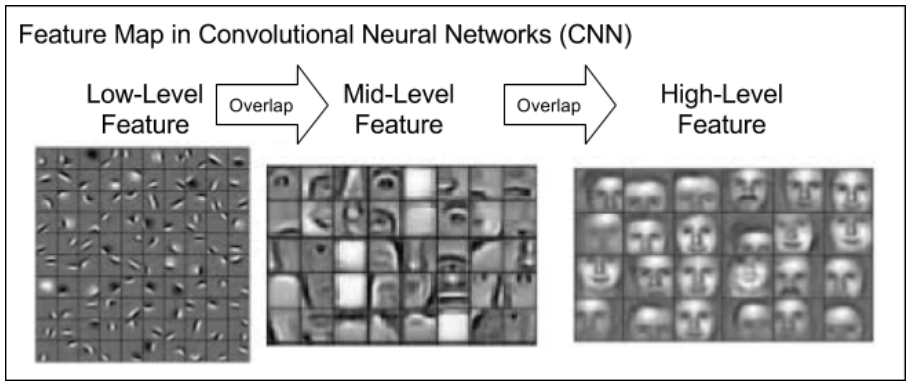
Ένας από τους σημαντικότερους κλάδους της επιστήμης υπολογιστών είναι η μηχανική μάθηση, η οποία διερευνά τη μελέτη και την κατασκευή αλγορίθμων που εκπαιδεύονται από δεδομένα και κάνουν προβλέψεις σχετικά με αυτά. Τέτοιου είδους αλγόριθμοι κατασκευάζουν μοντέλα βασισμένα σε πειραματικά δεδομένα, προκειμένου να δώσουν απάντηση σε προβλήματα αναγνώρισης προτύπων, ταξινόμησης και ελέγχου. Το ανθρώπινο κεντρικό νευρικό σύστημα αποτέλεσε έμπνευση για τη δημιουργία μηχανικών νευρωνικών δικτύων, τα οποία είναι δίκτυα από απλούς υπολογιστικούς κόμβους (νευρώνες), διασυνδεδεμένους μεταξύ τους. Η ευχρηστία τέτοιου είδους δικτύων, καθώς και η επιτυχημένη εφαρμογή τους σε μια πληθώρα κλάδων και προβλημάτων, έχει οδηγήσει στη ραγδαία εξέλιξη τους τα τελευταία χρόνια.

Τα βαθιά νευρωνικά δίκτυα (Deep Neural Networks - DNN) και ειδικά τα βαθιά συνελικτικά νευρωνικά δίκτυα (Deep Convolutional Neural Networks - CNN) έχουν φέρει αξιοσημείωτες επιτυχίες σε προβλήματα όρασης υπολογιστών, αυξάνοντας πολύ την κλίμακα των μοντέλων και αξιοποιώντας έναν τεράστιο όγκο δεδομένων. Τα CNNs βασίζονται σε τρία είδη επιπέδων: τα Συνελικτικά Επίπεδα (Convolutional Layers), τα Συγκεντρωτικά Επίπεδα (Pooling Layers) και τα Πλήρως Συνδεδεμένα Επίπεδα (Fully-Connected Layers). Τα Συνελικτικά Επίπεδα είναι η ουσία των CNNs και σε αυτά πραγματοποιείται το μεγαλύτερο κομμάτι των υπολογισμών, για αυτό το λόγο και τους έδωσαν το όνομά τους.

Στη συνέχεια θα παρουσιαστεί αναλυτικά μόνο η υλοποίηση του συνελικτικού επιπέδου, η οποία σχετίζεται με τον πυρήνα

SpMM.

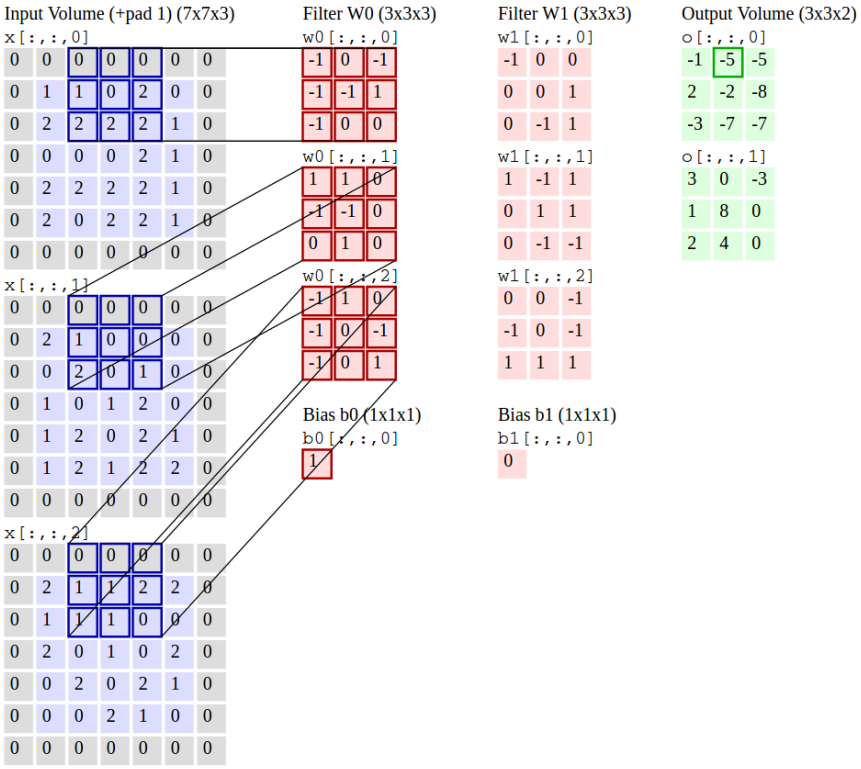
Για να εξηγήσουμε καλύτερα τη χρησιμότητα της συνέλιξης, ας υποθέσουμε ότι η είσοδος ενός δικτύου κατηγοριοποίησης εικόνων, είναι εικόνες που αναπαρίστανται με αριθμητικές τιμές σε μία τρισδιάστατη δομή αποθήκευσης, διαστάσεων *in_height* $\times$ *in_width* $\times$ *in_channels*. Τα *in_height* και *in_width* είναι το ύψος και πλάτος των εικόνων και το *in_channels* ονομάζεται βάθος (depth) ή κανάλια (channels) και σχετίζεται με τη μορφή αναπαράστασης της εικόνας. Για παράδειγμα, στην αναπαράσταση RGB έχουμε *in_channels* $= 3$ και για κάθε pixel διατηρείται μία τιμή από 0 έως 255, που συμβολίζει την ένταση των χρωμάτων κόκκινο, πράσινο και μπλε. Το νευρωνικό δίκτυο θα επιχειρήσει με μία ακολουθία συνελίξεων να εξαγάγει αρχικά απλά χαρακτηριστικά, όπως ευθείες και καμπύλες γραμμές και στη συνέχεια να αναγνωρίσει πιο αφηρημένες ιδέες, όπως για παράδειγμα ένα ζώο ή ένα όχημα, μέχρις ότου καταλήξει σε μία απάντηση, όπως θα έκανε αυτόματα και ο ανθρώπινος εγκέφαλος. Ένα παράδειγμα φαίνεται στην εικόνα 2.5.



Σχήμα 2.5: Εξαγωγή χαρακτηριστικών εικόνας μέσω Συνελικτικών Νευρωνικών Δικτύων.

Η συνέλιξη γίνεται με *out_channels* δομές, που ονομάζονται φίλτρα και έχουν και αυτά τρισδιάσταση μορφή, μεγέθους *k_height* $\times$ *k_width* $\times$ *in_channels*. Αν και θέλουμε οι διαστάσεις *k_height* και *k_width* να είναι αρκετά μικρότερες απο τις αντίστοιχες *in_height* και *in_width* της αρχικής εικόνας, η τρίτη διάσταση είναι ίση με αυτή των εικόνων εισόδου. Προκύπτει λοιπόν ως έξοδος ένας πίνακας ενεργοποίησης (activation map), μεγέθους *out_height* $\times$ *out_width* $\times$ *out_channels*, για κάθε εικόνα εισόδου, όπου όπως έχουμε ήδη πει *out_channels* είναι ο αριθμός των φίλ-

τρων με τα οποία θα πραγματοποιηθεί η συνέλιξη.

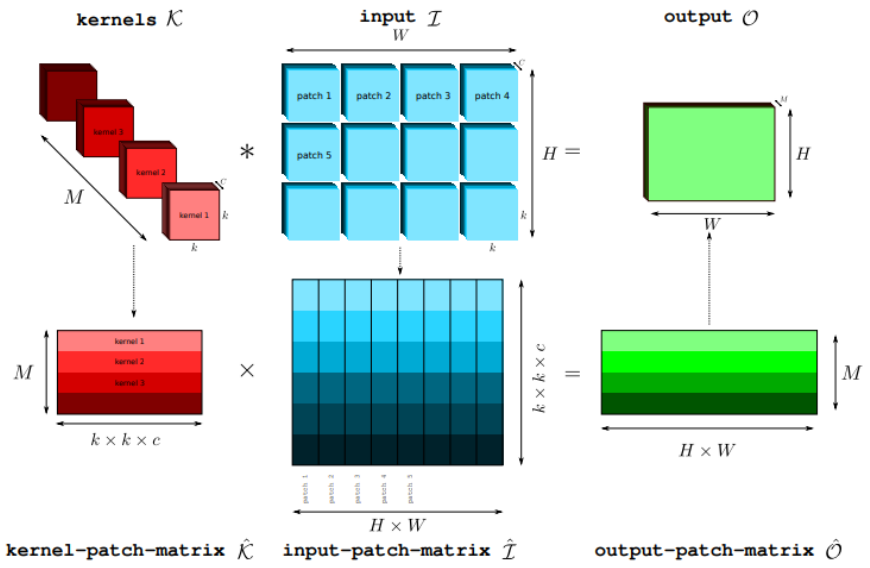


Σχήμα 2.6: Σχηματική αναπαράσταση ενός βήματος της συνέλιξης

Μπορεί κανείς να φανταστεί την πράξη της συνέλιξης σαν την προβολή κάθε φίλτρου πάνω στην εικόνα εισόδου. Το κάθε φίλτρο ξεκινάει από το πάνω αριστερό μέρος της εικόνας και κινείται έτσι ώστε να περάσει σταδιακά από όλη την εικόνα. Σε κάθε βήμα υπολογίζεται ένα εσωτερικό γινόμενο του φίλτρου με το αντίστοιχο κομμάτι της εικόνας και παράγεται τελικά μία τιμή, η οποία είναι το άθροισμα του κάθε επιμέρους πολλαπλασιασμού. Για το παράδειγμα εισόδου που δώσαμε, όπου οι δομές εισόδου και τα φίλτρα έχουν θετικές τιμές βλέπουμε εύκολα πως η τελική τιμή του κάθε εσωτερικού γινομένου μπορεί να χρησιμοποιηθεί σαν μέτρο ομοιότητας, ανάμεσα στο φίλτρο και το κομμάτι της εικόνας. Για να γίνουν τα παραπάνω πιο κατανοητά στην εικόνα 2.6 φαίνεται πώς υπολογίζεται η πρώτη τιμή στον πίνακα ενεργοποίησης. Οι μπλε πίνακες αναπαριστούν μια εικόνα εισόδου, οι κόκκινοι αναπαριστούν τα φίλτρα που χρησιμοποιούμε και οι πράσινοι τον πί-

νακα ενεργοποίησης που προκύπτει.

Οι υλοποιήσεις των συνελικτικών επιπέδων βασίζονται σε ρουτίνες BLAS (Basic Linear Algebra Subprograms), οι οποίες έχουν βελτιστοποιηθεί για την αρχιτεκτονική στην οποία εκτελούνται κάθε φορά. Η πιο διαδεδομένη υλοποίση είναι η im2col [Gu et al., 2016], [Tsai et al., 2016] στην οποία η είσοδος και τα φίλτρα μετασχηματίζονται έτσι ώστε να προκύπτει ο πίνακας ενεργοποίησης της εξόδου με έναν πολλαπλασιασμό πίνακα με πίνακα (GEMM) και χρησιμοποιείται από πολλά εργαλεία μηχανικής μάθησης, όπως τα Caffe, Theano και Torch.



Σχήμα 2.7: Υλοποίηση ενός συνελικτικού επιπέδου με τη μέθοδο im2col.

Για να γίνει πιο κατανοητή η μέθοδος ας υποθέσουμε ότι έχουμε M φίλτρα μεγέθους $k \times k \times c$ και σαν είσοδο εικόνες μεγέθους $H \times W \times c$, όπως φαίνονται στην εικόνα 2.7. Με το κατάλληλο padding μπορεί να προκύψει ως έξοδος ένας πίνακας ενεργοποίησης $H \times W \times M$. Στο πρώτο βήμα του αλγορίθμου δημιουργείται ένας πίνακας, όπου κάθε γραμμή έχει σειριακά τις τιμές των M φίλτρων. Ο πίνακας αυτός έχει M γραμμές και $k \times k \times c$ στήλες και πρέπει να σημειωθεί ότι αυτό το βήμα θα μπορούσε να παραληφθεί αν τα φίλτρα είχαν αποθηκευτεί από την αρχή στη μορφή αυτή, δηλαδή αν οι τιμές του κάθε καναλιού βρίσκονται σειριακά. Στη συνέχεια πρέπει να μετασχηματιστεί η δομή εισόδου. Κάθε τεμάχιο της δομής εισόδου με το οποίο θα πρέπει να πραγματοποιηθεί το

εσωτερικό γινόμενο με το φίλτρο, θα αποθηκεύεται σαν μία στήλη του πίνακα αυτού. Το κάθε τεμάχιο αυτό θα έχει βέβαια μέγεθος $k \times k \times c$ και συνεπώς αυτό θα είναι το πλήθος των γραμμών του μετασχηματισμένου πίνακα και οι γραμμές του θα είναι ίσες με το πλήθος των τεμαχίων, που σε αυτή την περίπτωση θεωρήσαμε πως θα είναι $H \times W$.

Αν και η παραπάνω μέθοδος έχει πολύ ικανοποιητικά αποτελέσματα από άποψη ταχύτητας μπορεί να επιβαρύνει αρκετά το σύστημα από άποψη μνήμης, αφού στον πίνακα μετασχηματισμού της δομής εισόδου επαναλαμβάνονται πολλές φορές οι τιμές της δομής.

Τεχνικές συμπίεσης Βαθιών Νευρωνικών Δικτύων

Τα νευρωνικά δίκτυα, αφού εκπαιδευτούν πάνω σε κάποιο μεγάλο σετ δεδομένων, μπορούν να αξιοποιηθούν από μία συσκευή για να εξάγουν συμπεράσματα για νέα δεδομένα. Λόγω του μεγάλου εύρους εφαρμογών που μπορούν να βελτιώσουν τα νευρωνικά δίκτυα, έχει δημιουργηθεί η ανάγκη χρήσης τους σε ενσωματωμένα συστήματα και κινητές συσκευές.

Η πραγματοποίηση της διαδικασίας εξαγωγής συμπερασμάτων (inference) στην ίδια τη συσκευή, αντί για το cloud, προσφέρει πολλά πλεονεκτήματα, όπως μεγαλύτερη προστασία της ιδιωτικότητας του χρήστη, μικρότερες απαιτήσεις στο εύρος ζώνης δικτύου και κάποιες φορές πιο γρήγορα αποτελέσματα. Ωστόσο αυτό καθίσταται αδύνατο εξαιτίας του μεγάλου αποτυπώματος μνήμης (memory footprint) τέτοιων δικτύων. Η δεύτερη πρόκληση που συναντάται κατά την ενσωμάτωση της μηχανικής μάθησης σε κινητές συσκευές είναι η υψηλή κατανάλωση ισχύος, που προέρχεται από τις υψηλές απαιτήσεις σε εύρος ζώνης μνήμης για ένα μεγάλο μοντέλο καθώς επίσης και το γεγονός ότι πολλές εφαρμογές χρειάζονται απόκριση σε πραγματικό χρόνο.

Μία μέθοδος που έχει μελετηθεί ευρέως για τη συμπίεση ενός νευρωνικού δικτύου είναι ο μηδενισμός βαρών, που δεν θεωρούνται σημαντικά για την ακρίβεια του μοντέλου (pruning) και φαίνεται να μειώνει ικανοποιητικά το μέγεθος κάποιων επιπέδων, χωρίς όμως να μειώνει το ποσοστό ακρίβειας του δικτύου. Η τεχνική αυτή αφαιρεί τον εγγενή πλεονασμό των βαθιών νευρωνικών δικτύων, μηδενίζοντας τις τιμές βαρών που είναι ήδη πολύ χαμηλές και οδηγεί σε πίνακες με μεγάλο πλήθος μηδενικών. Ωστόσο, πρέπει να σημειωθεί πως η βελτίωση της επίδοση δεν είναι πάντα δεδομένη,

αφού πολύ σημαντικό ρόλο σε αυτό παίζει η δομή αποθήκευσης των βαρών, καθώς και ο αλγόριθμος με τον οποίο θα πραγματοποιηθούν οι υπολογισμοί.

2.2.2 Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG)

Σε πολλές περιπτώσεις, η σημαντικότερη πρόκληση των προβλημάτων αριθμητικής γραμμικής άλγεβρας είναι η αποδοτική επίλυση μεγάλων, αραιών γραμμικών συστημάτων και η εύρεση των αντίστοιχων ιδιοτιμών τους. Η επίλυση γραμμικών συστημάτων είναι απαραίτητη σε εφαρμογές προσομοιώσεων, που βασίζονται στη διακριτοποίηση (discretization) μερικών διαφορικών εξισώσεων, ενώ η εύρεση των ιδιοτιμών παίζει κεντρικό ρόλο σε εφαρμογές, όπως η μέθοδος PCA (Principal component analysis - Ανάλυση Κύριων Συνιστωσών). Καθώς το μέγεθος των συστημάτων αυξάνεται και οι πίνακες γίνονται όλο και πιο αραιοί, οι μαθηματικές ρουτίνες για πυκνούς πίνακες, οι οποίες χρησιμοποιούνται σε απευθείας μεθόδους επίλυσης σαν την LU παραγοντοποίηση ή στο πρόβλημα ιδιοτιμών στη μέθοδο αποσύνθεσης Hessenberg [Saad, 2003], γίνονται όλο και πιο ακατάλληλες, καθώς οι απαιτήσεις σε μνήμη και υπολογιστική ισχύ μπορεί να ξεπεράσουν τους διαθέσιμους πόρους. Για το λόγο αυτό, προτιμώνται επαναληπτικές μέθοδοι που παρέχουν προσεγγιστικές λύσεις, αν και η αξιοποίηση των υπολογιστικών πόρων μπορεί να φέρει νέες προκλήσεις.

Έχουν γίνει πολυάριθμες προσπάθειες έτσι ώστε οι επαναληπτικές μέθοδοι πυκνών πινάκων να προσαρμοστούν σε επιταχυντές, όπως οι μέθοδοι υποχώρων Krylov [Anzt et al., 2016]. Εντούτοις, οι επαναληπτικές μέθοδοι αραιών πινάκων δεν έχουν την ίδια απήχηση. Μια πιθανή εξήγηση είναι το γεγονός πως πρέπει να συνδυαστούν μαθηματικές συναρτήσεις που αφορούν τόσο αραιούς όσο και πυκνούς πίνακες. Όταν πρόκειται για συμμετρικούς θετικά ορισμένους πίνακες, δηλαδή πίνακες με μόνο θετικές ιδιοτιμές, η μέθοδος Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) είναι μία από τις πιο αποδοτικές.

Η μέθοδος αυτή είναι σχεδιασμένη να βρίσκει m από τις μικρότερες (ή μεγαλύτερες) ιδιοτιμές λ και τα αντίστοιχα ιδιοδιανύσματα x ενός συμμετρικού θετικά ορισμένου προβλήματος:

$$Ax = \lambda x$$

Όπως και σε άλλες επαναληπτικές μεθόδους αυτό επιτυγχάνεται με τη σταδιακή ελαχιστοποίηση του πηλίκου Rayleigh:

$$\rho(x) = \frac{x^T A x}{x^T x}$$

Στη μέθοδο LOBPCG η ελαχιστοποίηση αυτή γίνεται σε κάθε βήμα τοπικά, στον υποχώρο της τρέχουσας προσέγγισης x_i, της προηγούμενης x_{i-1} και το υπόλοιπο $P(AX_i - \lambda X_i)$, όπου P είναι ο preconditioner του A. Η ελαχιστοποίηση του υποχώρου γίνεται με τη μέθοδο Rayleigh-Ritz. Τα παραπάνω συνοψίζονται στον αλγόριθμο 2.1.

Algorithm 2.1 Ο Αλγόριθμος της μεθόδου LOBPCG

1: **for** i = 0 **to** *niter* **do**:
2: $R = P(AX_i - \lambda X_i)$
3: check convergence criteria
4: $[X_i, \lambda]$ = Rayleigh-Ritz on span $\{X_i, X_{i1}, R\}$

Υπάρχουν τρία σημαντικά πλεονεκτήματα της μεθόδου αυτής, σε σχέση με τις κλασικές μεθόδους υποχώρων Krylov. Αρχικά η μέθοδος LOBPCG μπορεί να αξιοποιήσει έναν καλό preconditioner, όταν είναι διαθέσιμος και η χρήση αυτού μπορεί να μειώσει δραματικά τον αριθμό των απαιτούμενων επαναλήψεων και συνεπώς και το χρόνο υπολογισμού. Ακόμα, χρησιμοποιώντας τα ίδια δεδομένα διατηρεί τις απαιτήσεις σε μνήμη σχετικά χαμηλές, με αποτέλεσμα να κάνει δυνατή την αντιμετώπιση προβλημάτων πολύ μεγαλύτερης κλίμακας. Και τέλος, η blocked μορφή του αλγορίθμου δίνει τη δυνατότητα μιας αποδοτικής παράλληλης υλοποίησης σε μοντέρνα υπολογιστικά συστήματα. Σχεδόν κάθε κομμάτι του αλγορίθμου μπορεί να διατυπωθεί σε level-3 BLAS (Basic Linear Algebra Subprograms) ρουτίνες για αραιούς ή πυκνούς πίνακες, οι οποίες είναι ήδη υλοποιημένες σε αρκετές βιβλιοθήκες για αποδοτική παράλληλη κλιμάκωση, όπως CML, Cray LibSci, Intel MKL, cuSPARSE και πολλές άλλες.

Ο πυρήνας του πολλαπλασιασμού Αραιού Πίνακα με Πίνακα (SpMM) είναι βασικός για τους υπολογισμούς της μεθόδου και καταλαμβάνει σημαντικό ποσοστό του χρόνου εκτέλεσης.

2.3 Επεξεργαστές Γραφικών Γενικού Σκοπού (GPGPUs)

Μια μονάδα επεξεργασίας γραφικών (GPU) είναι ένα εξειδικευμένο ηλεκτρονικό κύκλωμα, σχεδιασμένο αρχικά για να εκτελεί αποδοτικά μαθηματικούς υπολογισμούς που χρησιμοποιούνται στην απεικόνιση εικόνων (image rendering). Στα πρώιμα χρόνια της επιστήμης των υπολογιστών, η κεντρική μονάδα επεξεργασίας (CPU) εκτελούσε αυτούς τους υπολογισμούς, όμως καθώς αυξήθηκαν οι απαιτήσεις υπολογιστικής ισχύος των γραφικών, η GPU ήρθε ως λύση για να την αποφορτώσει από αυτά τα καθήκοντα, απελευθερώνοντας την επεξεργαστική της δύναμη. Με την πάροδο των χρόνων επήλθε μια σημαντική αύξηση της επίδοσης και των δυνατοτήτων της GPU, γεγονός που την κατέστησε δημοφιλή επιλογή ακόμα και για εφαρμογές που δεν σχετίζονται με γραφικά.

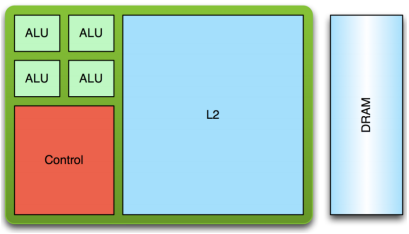
Με την κυκλοφορία του προγραμματιστικού περιβάλλοντος CUDA (Compute Unified Device Architecture) από την Nvidia το 2006, έγινε δυνατή η έννοια του προγραμματισμού καρτών γραφικών γενικού σκοπού (GPGPUs). Η CUDA επικεντρώθηκε στο να εκμεταλλευτεί στο μέγιστο τις δυνατότητες των GPUs της Nvidia για μια πληθώρα διαφορετικών εφαρμογών και σχεδιάστηκε για να δώσει στον προγραμματιστή μερική εποπτεία του work flow της GPU, χωρίς όμως να αποκαλύπτει κάθε λεπτομέρεια της αρχιτεκτονικής της.

Δύο χρόνια αργότερα κυκλοφόρησε η OpenCL και υποστηρίχθηκε από ένα ευρύ φάσμα τεχνολογιών, επιτρέποντας την ανάπτυξη λογισμικού για GPUs και CPUs δίνοντας έμφαση στη φορητότητα. Τα επόμενα χρόνια η OpenCL υποστήριξε και άλλους επιταχυντές, εκτός από GPU.

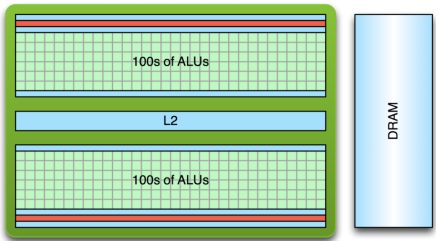
Στις μέρες μας, οι GPUs χρησιμοποιούνται ευρέως σε ενσωματωμένα συστήματα, κινητά τηλέφωνα, προσωπικούς υπολογιστές και κονσόλες παιχνιδιών και έχουν εισέλθει σε μια πληθώρα πεδίων, όπως μηχανική μάθηση, έρευνα μεγάλου όγκου δεδομένων (big data), τεχνητή νοημοσύνη, επεξεργασία εικόνας, στατιστική, γραμμική άλγεβρα και ιατρική έρευνα. Η αρχιτεκτονική της GPU παρουσιάζει υψηλό παραλληλισμό, γεγονός που την καθιστά πιο αποδοτική από μια CPU γενικού σκοπού σε αλγορίθμους που επεξεργάζονται παράλληλα μεγάλα μπλοκ δεδομένων και εμφανίζουν υψηλή ένταση λειτουργιών (operational intensity).

2.3.1 Σύγκριση GPU με CPU

Η κεντρική μονάδα επεξεργασίας (CPU) του συστήματος, είναι υπεύθυνη για την ερμηνεία και την εκτέλεση εντολών του λογισμικού και του υλικού και θεωρείται ο εγκέφαλος του υπολογιστή. Είναι βελτιστοποιημένη για την επίδοση ενός νήματος (single-thread performance), καθώς διαθέτει μικρό αριθμό πυρήνων (cores). Οι πυρήνες αυτοί έχουν υψηλή συχνότητα ρολογιού (2-4 GHz), μικρό αρχείο καταχωρητών (register file) και λίγες, αλλά ισχυρές μονάδες αριθμητικής λογικής (ALUs), όπως φαίνεται στο σχήμα 2.8. Η λογική ελέγχου (control logic) μιας CPU είναι σύνθετη, με τεχνικές πρόβλεψης διακλάδωσης (branch prediction) και εκτέλεσης εκτός σειράς των εντολών (out-of-order execution). Το υποσύστημα μνήμης της CPU περιέχει 3 με 4 επίπεδα κρυφής μνήμης μεγάλης χωρητικότητας (μερικά MB) και μέτριο εύρος ζώνης κύριας μνήμης. Προτερήματα της CPU αποτελούν η ευελιξία, η πολυλειτουργικότητα και η ευκολία προγραμματισμού. Ωστόσο, το context-switching, η εναλλαγή δηλαδή μεταξύ των διεργασιών, πραγματοποιείται μέσω λογισμικού και είναι χρονοβόρο για το λειτουργικό σύστημα, γεγονός που την καθιστά βελτιστοποιημένη για σειριακή επεξεργασία με περιορισμένο παραλληλισμό.



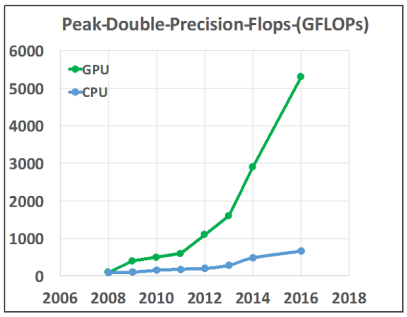
Σχήμα 2.8: CPU chip



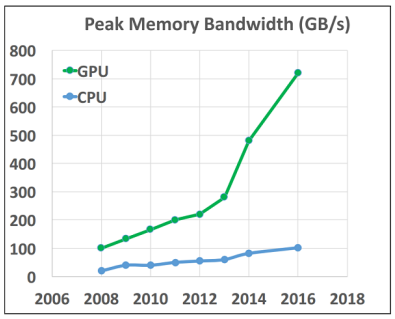
Σχήμα 2.9: GPU chip

Αντίθετα, η GPU βελτιστοποιεί την επίδοση πολλών νημάτων (multi-thread performance) σε εφαρμογές με παράλληλη επεξεργασία δεδομένων (data-parallel workloads), λόγω της εξαιρετικά παράλληλης δομής της. Διαθέτει περισσότερους πυρήνες, με μικρότερη συχνότητα ρολογιού (0.6-1.6 GHz), μεγάλο register file και απλούστερες ALUs, χωρίς το σύνθετο control logic της CPU, όπως φαίνεται στο σχήμα 2.9. Το υποσύστημα μνήμης της διαθέτει μόνο 1 με 2 επίπεδα κρυφής μνήμης, μικρής χωρητικότητας (μερικά KB) και υψηλό εύρος ζώνης κύριας μνήμης. Αυτό που κάνει τη GPU ιδανική για ταυτόχρονη εκτέλεση πολλών νημάτων είναι ότι η εναλλαγή τους (context-switching) πραγματοποιείται μέσω του

υλικού και έχει μηδαμινό κόστος. Ωστόσο, είναι ακατάλληλη για ορισμένους αλγορίθμους, οι οποίοι πρέπει να αναδιατυπωθούν για να εκμεταλλευτούν τον παραλληλισμό που προσφέρει, ενώ τέλος χαρακτηρίζεται από υψηλή κατανάλωση ενέργειας. Στα σχήματα 2.10 και 2.11 αντίστοιχα φαίνεται η υπεροχή της GPU σε υπολογιστική ιπποδύναμη και εύρος ζώνης μνήμης έναντι της CPU.



Σχήμα 2.10: Πράξεις κινητής υποδιαστολής ανά δευτερόλεπτο για CPU και GPU

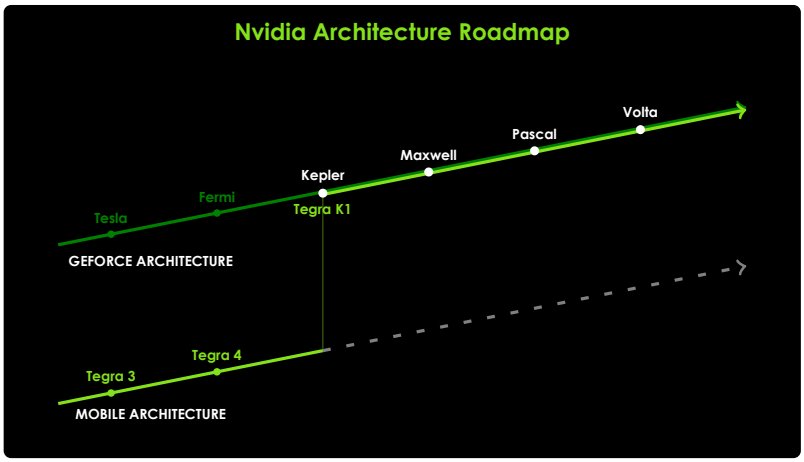


Σχήμα 2.11: Εύρος ζώνης μνήμης για CPU και GPU

2.3.2 Γενική αρχιτεκτονική μιας Nvidia GPU

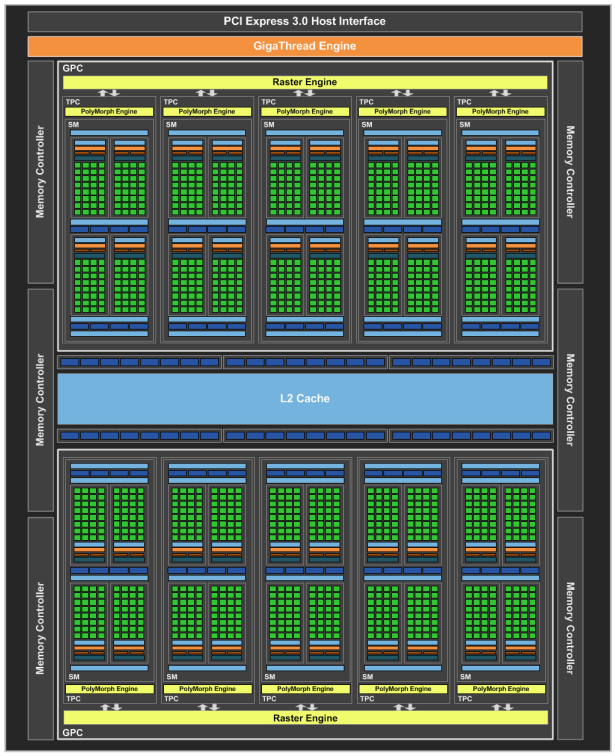
Η NVIDIA ανέδειξε την GPU σε μια κορυφαία μηχανή παράλληλης επεξεργασίας για εφαρμογές που απαιτούν υψηλή υπολογιστική ισχύ, αναπτύσσοντας με το πέρασμα των χρόνων όλο και πιο εξελιγμένες αρχιτεκτονικές (Σχήμα 2.12). Κάθε GPU διαθέτει έναν αριθμό που προσδιορίζει την υπολογιστική της ικανότητα (compute capability) και καθορίζει την αρχιτεκτονική, τις γενικές προδιαγραφές και τα διαθέσιμα χαρακτηριστικά της.

Σύμφωνα με τη γενική αρχιτεκτονική μιας NVIDIA GPU, κάθε συσκευή αποτελείται από συστοιχίες πολυεπεξεργαστών ροών (Streaming Multiprocessors – SMs). Κάθε πολυεπεξεργαστής ροών αποτελείται από επεξεργαστές ροών (Streaming Processors – SPs ή αλλιώς CUDA cores), ένα πολύ μεγάλο αρχείο καταχωρητών (8K - 32K) και διάφορα είδη μνήμης (Shared, Constant και Texture). Κάθε CUDA core διαθέτει δύο υπολογιστικούς επεξεργαστές, έναν για πράξεις ακεραίων και έναν για πράξεις κινητής υποδιαστολής. Στο σχήμα 2.13 φαίνεται ένα παράδειγμα της αρχιτεκτονικής Pascal.



Σχήμα 2.12: Εξέλιξη της Αρχιτεκτονικής των Nvidia GPUs

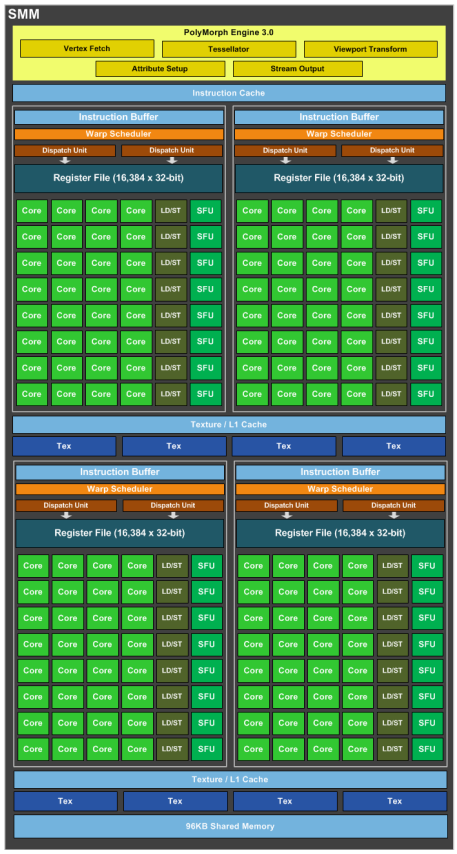
Ο πολυεπεξεργαστής ροών έχει σχεδιαστεί για να εκτελεί εκατοντάδες νήματα ταυτόχρονα. Για να διαχειριστεί ένα τόσο μεγάλο πλήθος από νήματα, υλοποιεί την SIMT (Single-Instruction, Multiple-Thread) αρχιτεκτονική. Σε αυτή την αρχιτεκτονική κάθε πολυεπεξεργαστής ροών δημιουργεί, διαχειρίζεται, χρονοδρομολογεί και εκτελεί νήματα σε ομάδες 32 παράλληλων νημάτων, που ονομάζονται warps και κάθε νήμα εκτελείται σε έναν CUDA core. Κάθε πολυεπεξεργαστής διαθέτει τουλάχιστον έναν χρονοδρομολογητή των warps (warp scheduler) που συντονίζει την εναλλαγή των warps, επιλέγοντας σε κάθε στιγμή έκδοσης εντολής (instruction issue time) ένα warp του οποίου τα νήματα είναι έτοιμα να εκτελέσουν την επόμενη εντολή. Συγκεκριμένα, για Compute Capability ≥ 3 ένας πολυεπεξεργαστής διαθέτει 4 warp schedulers, που σημαίνει πως μπορούν να τρέχουν ταυτόχρονα τέσσερα warps. Όταν σε έναν πολυεπεξεργαστή ανατεθεί ένα πλήθος από warps, τα κατανέμει στους 4 χρονοδρομολογητές που διαθέτει, καθένας από τους οποίους με τη σειρά του, σε κάθε στιγμή έκδοσης εντολής εκδίδει δύο ανεξάρτητες εντολές (dual-issue) για ένα από τα warps που έχει αναλάβει και είναι έτοιμο να εκτελεστεί. Σε αυτό το σημείο αξίζει να σημειωθεί, πως η δυνατότητα έκδοσης δύο ανεξάρτητων εντολών στον ίδιο κύκλο ρολογιού είναι ιδιαίτερα χρήσιμη στην περίπτωση που μια εντολή αφορά πρόσβαση στη μνήμη (load/store), ενέργεια που είναι ιδιαίτερα χρονοβόρα, καθώς άλλες υπολογιστικές εντολές, που δεν εξαρτώνται από αυτή τη μεταφορά δεδομένων, μπορούν να εκτελεστούν χωρίς να περιμένουν την ολοκλήρωσή της. Στο σχήμα



Σχήμα 2.13: Nvidia GeForce GTX 1060 Block Diagram

2.14 φαίνεται η δομή ενός πολυεπεξεργαστή της αρχιτεκτονικής Pascal.

Στις πρώτες αρχιτεκτονικές της Nvidia όλα τα νήματα ενός warp μοιράζονται τον ίδιο program counter και σε συνδυασμό με μια μάσκα ενεργών νημάτων μπορούν να εκτελούν τις διαφορετικές διακλαδώσεις σειριακά. Επομένως κάθε warp εκτελεί μια κοινή εντολή κάθε στιγμή, οπότε βέλτιστη απόδοση επιτυγχάνεται όταν όλα τα νήματα ενός warp ακολουθούν το ίδιο μονοπάτι εκτέλεσης. Ξεκινώντας από την Volta αρχιτεκτονική ($\geq$ 7.0) τα νήματα του warp έχουν δικούς τους program counters και σωρό κλήσεων (call stack) έτσι ώστε να εκτελούνται ανεξάρτητα. Τα νήματα ενός warp ξεκινάνε ταυτόχρονα στην ίδια διεύθυνση (program address), αλλά διατηρούν το δικό τους program counter και τους δικούς τους καταχωρητές και συνεπώς είναι ελεύθερα να ακολουθήσουν διαφορετικές διακλαδώσεις και να εκτελεστούν ανεξάρτητα. Βελτιστοποιήσεις στο χρονοδρομολογητή των νημάτων στο warp ήταν απαραίτητες έτσι ώστε να διατηρηθεί τόσο η ευελιξία των νημάτων, όσο και η υψηλή διακίνηση (throughput) εντολών.

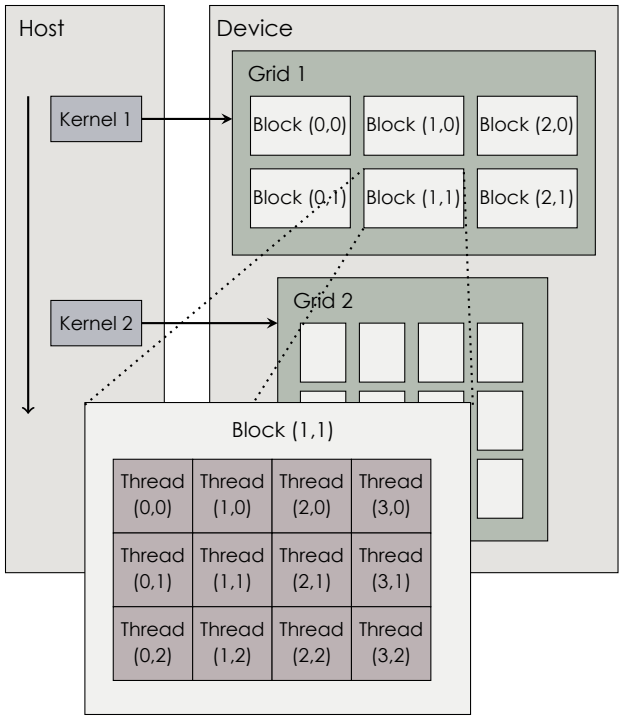


Σχήμα 2.14: Πολυεπεξεργαστής στην Αρχιτεκτονική Pascal

2.3.3 Μοντέλο προγραμματισμού CUDA

Η έλευση των πολυπύρηνων GPUs σήμανε πως τα chips των επεξεργαστών συντελούν πλέον παράλληλα συστήματα, των οποίων ο παραλληλισμός συνεχίζει να κλιμακώνεται με το Νόμο του Moore. Για το λόγο αυτό, δημιουργήθηκε η πρόκληση για την ανάπτυξη ενός λογισμικού που κλιμακώνει με διαφάνεια τον παραλληλισμό του ώστε να μοχλεύσει τον αυξανόμενο αριθμό επεξεργαστικών πυρήνων. Το μοντέλο προγραμματισμού CUDA πρόκειται για μια αρχιτεκτονική υλικού και λογισμικού που επιτρέπει στις NVIDIA GPUs να εκτελούν προγράμματα γραμμένα σε C, C++, Fortran, OpenCL και άλλες γλώσσες και σχεδιάστηκε για να ξεπεράσει αυτή την πρόκληση.

Πιο συγκεκριμένα, ένα πρόγραμμα CUDA στην CPU καλεί πυρήνες (kernels), δηλαδή ρουτίνες που θα εκτελεστούν στην GPU και κάθε πυρήνας εκτελείται παράλληλα από ένα σύνολο πα-

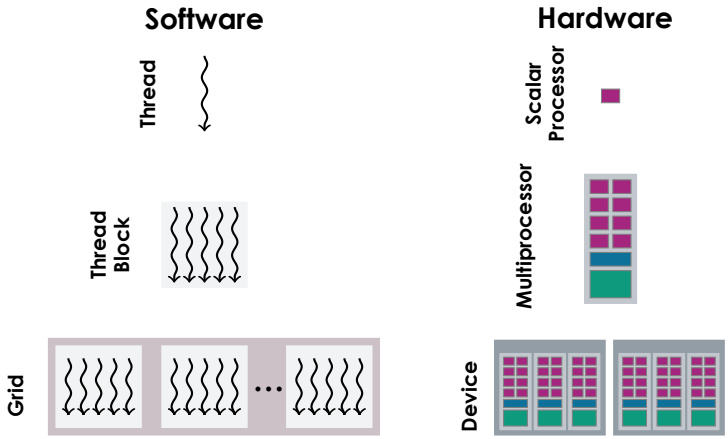


Σχήμα 2.15: Εκτέλεση ενός πυρήνα στη GPU

ράλληλων νημάτων. Ο προγραμματιστής οργανώνει αυτά τα νήματα (threads) σε ομάδες νημάτων (thread blocks) και πλέγματα από thread blocks (grids). Κάθε νήμα εντός ενός thread block εκτελεί ένα στιγμιότυπο του πυρήνα, έχει ένα διακριτικό (thread ID) μέσα στο thread block, όπου ανήκει και διαθέτει προσωπικούς καταχωρητές και ιδιωτική μνήμη. Κάθε thread block είναι μια ομάδα παράλληλα εκτελέσιμων νημάτων που έχουν αντιστοίχως ένα διακριτικό (block ID) μέσα στο grid, όπου ανήκουν και μπορούν να συνεργάζονται μεταξύ τους μέσω συγχρονισμού (barrier synchronization), ατομικών λειτουργιών (atomic operations) και κοινής μνήμης. Ένα πλέγμα (grid) περιλαμβάνει ένα σύνολο από thread blocks που εκτελούν τον ίδιο πυρήνα, διαβάζουν την είσοδο και γράφουν τα αποτελέσματα στην κύρια μνήμη (global memory). Νήματα που ανήκουν σε διαφορερικά thread blocks μπορούν να επικοινωνήσουν, μόνο μέσω της κύριας μνήμης. Στο σχήμα 2.15 παρουσιάζεται η ιεραρχική δομή των οντοτήτων που περιγράψαμε κατά την κλήση ενός πυρήνα.

Σε επίπεδο υλικού, η ιεραρχία των νημάτων χαρτογραφείται σε ιεραρχία επεξεργαστών στην GPU. Πιο συγκεκριμένα, ένα ή

περισσότερα πλέγματα πυρήνων (kernel grids) ανατίθενται στην GPU, τα thread blocks ενός πλέγματος απαριθμούνται και ανατίθενται στους πολυεπεξεργαστές με διαθέσιμη υπολογιστική χωρητικότητα και τέλος τα μεμονωμένα νήματα εκτελούνται παράλληλα στους CUDA cores, όπως φαίνεται στο σχήμα 2.16. Μετά την ολοκλήρωση της εκτέλεσης ενός thread block, νέα thread blocks ανατίθενται στους ελεύθερους πολυεπεξεργαστές.

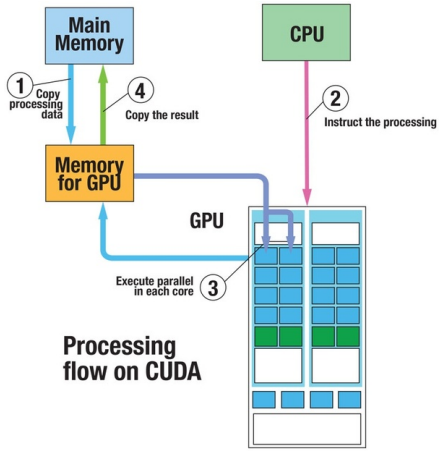


Σχήμα 2.16: CUDA Software - Hardware Matching

Συνοψίζοντας, το μοντέλο προγραμματισμού CUDA υποθέτει ότι τα CUDA νήματα εκτελούνται σε μια ξεχωριστή φυσική συσκευή (device) που λειτουργεί ως συν-επεξεργαστής στη CPU (host) που εκτελεί το πρόγραμμα. Επίσης υποθέτει ότι διατηρούν τους δικούς τους ξεχωριστούς χώρους μνήμης στη DRAM, που αναφέρονται ως host memory και device memory αντίστοιχα. Τη δέσμευση, την αποδέσμευση, καθώς και τη μεταφορά δεδομένων μεταξύ αυτών των μνημών αναλαμβάνει η CPU. Δεδομένης λοιπόν αυτής της ετερογενής φύσης του μοντέλου προγραμματισμού CUDA, μια τυπική ακολουθία λειτουργιών για ένα πρόγραμμα CUDA, φαίνεται στο σχήμα 2.17 και είναι η παρακάτω:

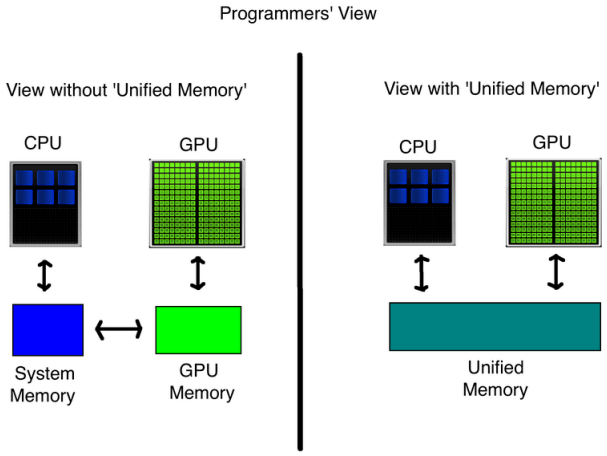
- Δήλωση και δέσμευση μνήμης στον host και στη συσκευή
- Αρχικοποίηση των δεδομένων στον host
- Μεταφορά των δεδομένων από τον host στη συσκευή
- Εκτέλεση ενός ή περισσότερων πυρήνων
- Μεταφορά των αποτελεσμάτων από τη συσκευή στον host

Με την έλευση της αρχιτεκτονικής Pascal πραγματοποιήθηκε η δυνατότητα της ενοποιημένης μνήμης (Unified Memory), που



Σχήμα 2.17: Μοντέλο Εκτέλεσης Στη GPU

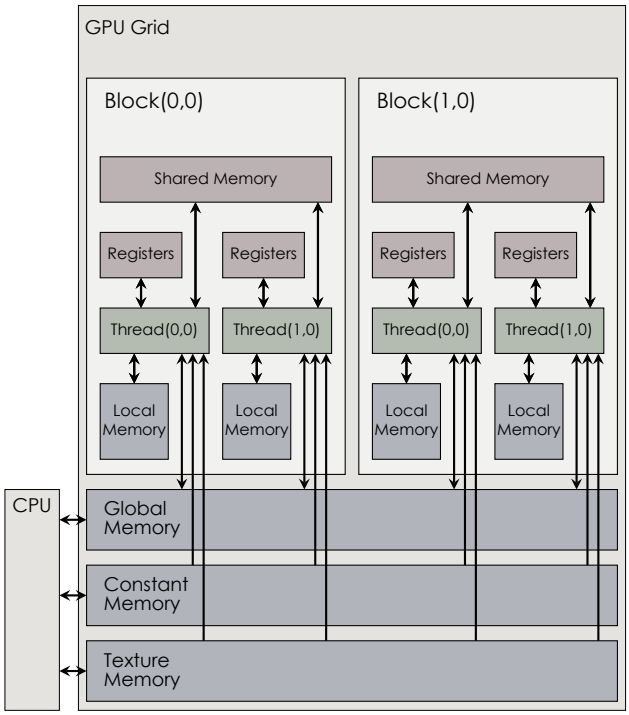
αποτελεί διαχειριζόμενη μνήμη, προσβάσιμη από όλες τις CPUs και GPUs του συστήματος ως μια ενιαία, συνεκτική μνήμη με κοινή διευθυνσιοδότηση με σκοπό τη γεφύρωση των χώρων μνήμης της CPU και των GPUs έτσι ώστε να εξαλειφθεί η ανάγκη να μεταφέρονται ρητά τα δεδομένα μεταξύ τους. Στο σχήμα 2.18 φαίνεται η σκοπιά του προγραμματιστή με και χωρίς τη δυνατότητα της ενοποιημένης μνήμης.



Σχήμα 2.18: Unified Memory

2.3.4 Μοντέλο Μνήμης της GPU

Σε επίπεδο μνήμης, κάθε νήμα διαθέτει ιδιωτικούς καταχωρητές και ιδιωτικό τοπικό χώρο μνήμης που χρησιμοποιείται για διαρροή καταχωρητών (register spilling) και κλήση συναρτήσεων. Κάθε thread block διαθέτει κοινή μνήμη (shared memory), ορατή σε όλα τα νήματα του μπλοκ και με την ίδια διάρκεια ζωής με το μπλοκ, με σκοπό την επικοινωνία και την ανταλλαγή δεδομένων και αποτελεσμάτων μεταξύ των νημάτων που περιέχει. Όλα τα νήματα που ανήκουν στο grid, έχουν πρόσβαση στην κύρια μνήμη (global memory). Επίσης υπάρχουν δύο ακόμα πρόσθετοι χώροι μνήμης μόνο για ανάγνωση, που είναι προσβάσιμοι από όλα τα νήματα και είναι βελτιστοποιημένοι για διαφορετικές χρήσεις, η Constant και η Texture Memory. Τα δεδομένα των Global, Constant και Texture μνημών διατηρούνται καθ' όλη τη διάρκεια ζωής της εφαρμογής. Στο σχήμα 2.19 φαίνεται η ιεραρχία των μνημών μιας GPU.



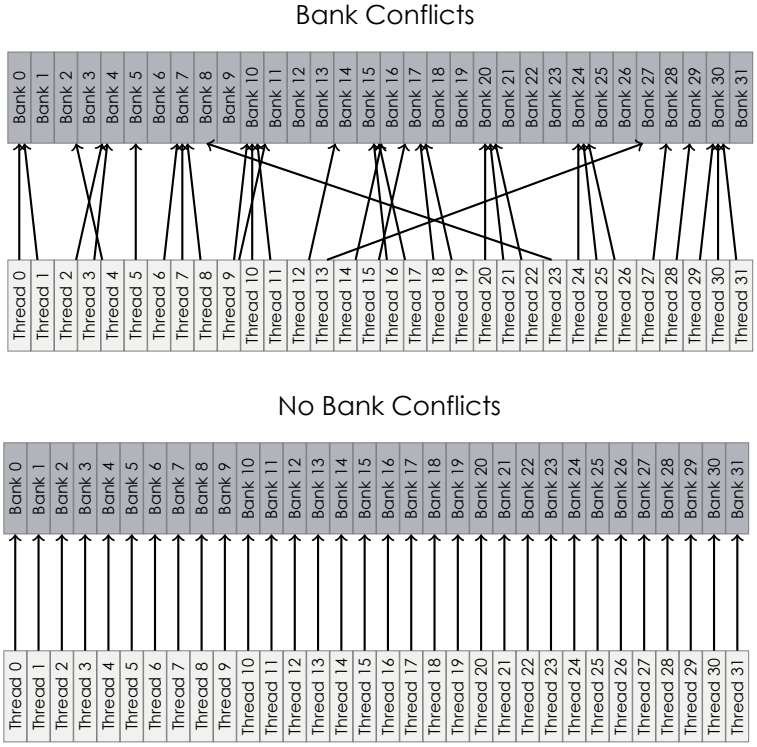
Σχήμα 2.19: Μοντέλο Μνήμης της GPU

Πιο συγκεκριμένα, κάθε πολυεπεξεργαστής διαθέτει αρχείο καταχωρητών, μεγέθους μερικών KB (64-128KB), που διαμοιράζεται

μεταξύ των ενεργών νημάτων. Η προσπέλαση καταχωρητών είναι ταχύτατη καθώς υλοποιείται μέσω του υλικού. Όταν δεν υπάρχουν επαρκείς καταχωρητές για μια δεδομένη εργασία, τα δεδομένα μεταφέρονται στην τοπική μνήμη, η οποία δεν αποτελεί ένα φυσικό τύπο μνήμης αλλά μια αφαίρεση της κύριας μνήμης (global memory), γεγονός που καθιστά ακριβή την προσπέλασής της αφού βρίσκεται off-chip.

Στο σημείο αυτό αξίζει να σημειωθεί πως η έλευση της αρχιτεκτονικής Kepler συνέστησε μια σημαντική δυνατότητα, την ανταλλαγή δεδομένων μεταξύ των νημάτων που ανήκουν στο ίδιο thread block μέσω καταχωρητών. Σε παλιότερες αρχιτεκτονικές η ανταλλαγή δεδομένων σε επίπεδο thread block ήταν δυνατή μόνο μέσω της κοινής μνήμης, που συνοπτικά περιλάμβανε την εγγραφή δεδομένων στην κοινή μνήμη, συγχρονισμό και στη συνέχεια ανάγνωση από την κοινή μνήμη. Η εντολή shuffle της Kepler αρχιτεκτονικής επιτρέπει σε ένα νήμα να διαβάζει απευθείας έναν καταχωρητή ενός άλλου νήματος που ανήκει στο ίδιο warp, γεγονός που επιτρέπει σε νήματα ενός warp να ανταλλάσσουν ή να μεταδίδουν (broadcast) συλλογικά δεδομένα.

Κάθε πολυεπεξεργαστής διαθέτει 64KB μνήμης on-chip που μπορεί να χωριστεί μεταξύ της L1 μνήμης και της κοινής μνήμης και συνήθως η δεύτερη καταλαμβάνει 48KB. Καθώς λοιπόν η κοινή μνήμη ενός thread block βρίσκεται on-chip, είναι πολύ πιο γρήγορη από την τοπική και κύρια μνήμη. Στην πραγματικότητα, η καθυστέρηση για την προσπέλαση αυτής της μνήμης είναι περίπου 100φορές μικρότερη σε σχέση με την προσπέλαση μιας uncached θέσης στην κύρια μνήμη, δεδομένου ότι δεν πραγματοποιούνται bank conflicts. Με τον όρο bank conflicts αναφερόμαστε στην περίπτωση όπου πολλά νήματα αιτούνται ταυτόχρονα πρόσβαση σε δεδομένα που ανήκουν στο ίδιο bank στην κοινή μνήμη, όπως φαίνεται στο σχήμα 2.20. Σε μια τέτοια περίπτωση, οι προσπελάσεις εκτελούνται σειριακά, καθώς μπορεί να εξυπηρετηθεί μόνο ένα αίτημα ανά κύκλο ρολογιού, γεγονός που μειώνει την απόδοση της εφαρμογής. Για να μετριαστεί αυτή η πιθανή συμφόρηση, η κοινή μνήμη είναι χωρισμένη σε 32 λογικά μέρη (banks) έτσι ώστε διαδοχικές 32-bit ή 64-bit words να ανατίθενται σε διαδοχικά banks και κάθε bank έχει εύρος ζώνης 32bits ή 64bits αντίστοιχα ανά κύκλο ρολογιού. Η επιλογή αυτή δεν είναι τυχαία, καθώς υπάρχουν 32 νήματα σε ένα warp και ακριβώς 32 banks. Συνεπώς, για να επιτευχθεί το μέγιστο εύρος ζώνης μνήμης θα πρέπει κάθε νήμα να πραγματοποιεί φόρτωση (ή αποθήκευση)



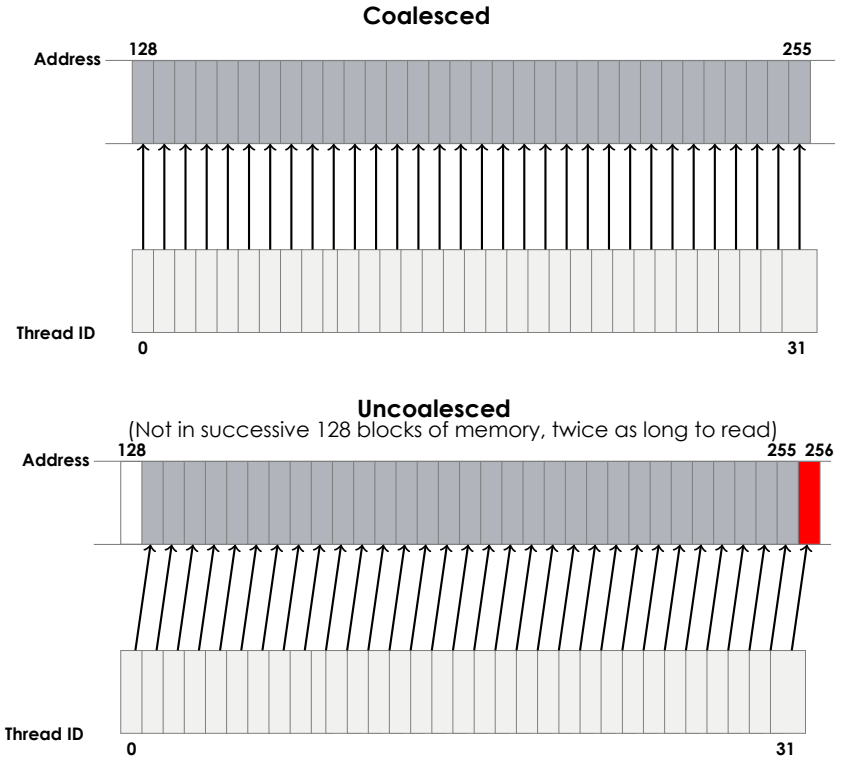
Σχήμα 2.20: Bank Conflicts

μνήμης από ξεχωριστό bank.

Σε αυτό το σημείο αξίζει να σημειωθεί ότι το πλήθος των καταχωρητών και το μέγεθος της κοινής μνήμης που δεσμεύει ένα thread block καθορίζει και το πλήθος των thread block που μπορούν να είναι ενεργά, καθώς οι πόροι αυτοί είναι περιορισμένοι ανά πολυεπεξεργαστή. Συνεπώς, είναι απαραίτητος ο πειραματισμός σε αυτούς τους περιοριστικούς παράγοντες ώστε να επιτευχθεί η μέγιστη πληρότητα (occupancy) στους πολυεπεξεργαστές, όπου ο όρος πληρότητα συμβολίζει το λόγο των ενεργών warps σε έναν πολυεπεξεργαστή προς το μέγιστο πλήθος των ενεργών warps που υποστηρίζονται σε ένα SM.

Τέλος, η κύρια μνήμη (global memory) βρίσκεται off-chip (σε DRAM), είναι αργή, αλλά cached και έχει διάρκεια ζωής ίση με αυτή της εφαρμογής. Η πρόσβαση σε αυτή τη μνήμη πραγματοποιείται μέσω συναλλαγών μνήμης μεγέθους 32, 64 ή 128-bytes συνεχόμενων και ευθυγραμμισμένων θέσεων μνήμης. Όταν ένα warp εκτελέσει μια εντολή πρόσβασης στην κύρια μνήμη, τότε συγχωνεύει τις μεμονωμένες προσβάσεις των νημάτων, που ανή-

κουν σε αυτό, σε μια ή περισσότερες τέτοιες συναλλαγές μνήμης, ανάλογα με το μέγεθος της λέξης που προσπελαύνει κάθε νήμα και την κατανομή των προσπελαυνουσών θέσεων μνήμης μεταξύ των νημάτων. Συνεπώς, όσο περισσότερες συναλλαγές πραγματοποιηθούν, τόσο περισσότερες αχρησιμοποίητες λέξεις θα μεταφερθούν μαζί με τις λέξεις που προσπέλασε κάθε νήμα, με αποτέλεσμα να μειώνεται ανάλογα η διεκπεραιωτικότητα εντολών (instruction throughput). Η τεχνική κατά την οποία τα δεδο-



Σχήμα 2.21: Συγχώνευση Συναλλαγών της Κύριας Μνήμης

μένα που βρίσκονται στην κύρια μνήμη διαβάζονται ή εγγράφονται με όσο το δυνατόν λιγότερες συναλλαγές, συγχωνεύοντας αιτήματα πρόσβασης μνήμης σε μια μόνο συναλλαγή, αναφέρεται ως coalescing. Στο σχήμα 2.21 φαίνεται μια επιτυχής συγχώνευση των συναλλαγών μνήμης από τα 32 νήματα ενός warp σε μια συναλλαγή, καθώς και μια ανεπιτυχής συγχώνευσή τους σε μια συναλλαγή, λόγω του ότι ενώ οι προσβάσεις στη μνήμη, συνολικού μεγέθους 128bytes, είναι ευθυγραμμισμένες, η πρώτη διεύθυνση δεν είναι πολλαπλάσιο του μεγέθους τους.

3

Σχεδιασμός και Υλοποίηση του Πυρήνα Πολλαπλασιασμού Αραιού Πίνακα με Πίνακα

Σε αυτό το κεφάλαιο θα μελετήσουμε τον πυρήνα πολλαπλασιασμού Αραιού Πίνακα με Πίνακα (SpMM) εφαρμόζοντας αρχικά το μοντέλο Roofline και εξετάζοντας τους αλγόριθμους που χρησιμοποιούνται για τις συμβατικές δομές αποθήκευσης αραιών πινάκων. Στη συνέχεια προτείνουμε μία νέα δομή αποθήκευσης η οποία επιχειρεί να αντιμετωπίσει τα μειονεκτήματα των συμβατικών δομών κατά την εφαρμογή τους στον πυρήνα SpMM και στη συνέχεια περιγράφουμε πώς μπορεί να υλοποιηθεί αποδοτικά σε επεξεργαστές γραφικών γενικού σκοπού.

3.1 Μοντέλο Roofline

Το μοντέλο Roofline [Williams et al., 2009] παρέχει μια μέθοδο εκτίμησης της επίδοσης συναρτήσει του υπολογιστικού πυρήνα και των χαρακτηριστικών του υλικού και βασίζεται στην υπόθεση ότι περιοριστικό παράγοντα της επίδοσης αποτελεί είτε η ρυθμαπόδοση των υπολογισμών, είτε το εύρος ζώνης μνήμης του επεξεργαστή. Ο προσδιορισμός της κάθε περίπτωσης επιτυγχάνεται μέσω του υπολογισμού της λειτουργικής έντασης (operational intensity) της εφαρμογής, δηλαδή μιας μετρικής που υπολογίζει το λόγο των υπολογισμών και ειδικότερα των λειτουργιών κινητής υποδιαστολής, προς το συνολικό μέγεθος των δεδομένων που μεταφέρονται από/προς τη μνήμη και εκτιμάται σε μονάδες *FLOP/Byte*. Το μοντέλο Roofline είναι σε μεγάλο βαθμό αφαιρετικό

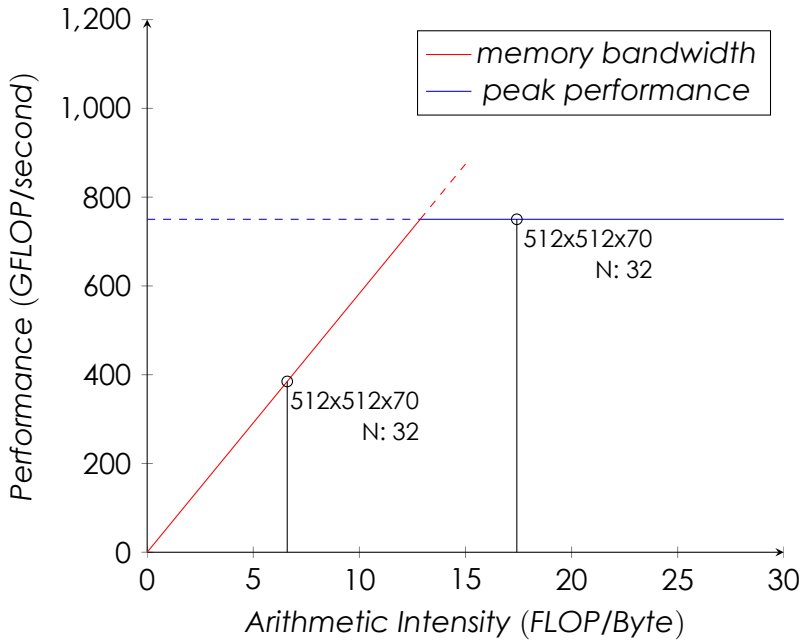
και προσφέρει μόνο μια εκτίμηση άνω φράγματος της επίδοσης του πυρήνα, όμως μπορεί να φανεί πολύ χρήσιμο για να καταλάβουμε αν ένας υπολογιστικός πυρήνας είναι compute-bound, δηλαδή αν περιορίζεται από την ταχύτητα της υπολογιστικής μονάδας του υλικού, ή memory-bound, δηλαδή αν περιορίζεται από το εύρος ζώνης της μνήμης.

Το απλό μοντέλο που προτάθηκε συνδέει την επίδοση κινητής υποδιαστολής, την αριθμητική ένταση και το εύρος ζώνης της μνήμης σε ένα δισδιάστατο γράφημα. Η μέγιστη επίδοση κινητής υποδιαστολής εξαρτάται από τις προδιαγραφές του υλικού, έχει μονάδες μέτρησης *FLOPS/second*, δηλαδή τις πράξεις κινητής υποδιαστολής ανά δευτερόλεπτο και είναι ένα όριο για τη μέγιστη επίδοση που μπορεί να επιτύχει ένας πυρήνας σε ένα δεδομένο υπολογιστή. Σε αυτό το όριο έρχεται να προστεθεί ένα δεύτερο που προέρχεται από τις απαιτήσεις από του συστήματος μνήμης και το εύρος ζώνης του, δηλαδή πόσο γρήγορα μπορεί να μεταφέρει τα απαραίτητα δεδομένα στον πυρήνα, ώστε να γίνουν οι απαραίτητες πράξεις. Το μέγιστο εύρος ζώνης μνήμης μπορεί να μετρηθεί μέσω προγραμμάτων.

Για την κατασκευή του μοντέλου σχεδιάζουμε ένα γράφημα με δύο άξονες, όπου ο άξονας y αναπαριστά τη μέγιστη τιμή *FLOP/second* που μπορεί να φτάσει ένας πυρήνας και ο άξονας x αναπαριστά την αριθμητική ένταση για ένα δεδομένο πυρήνα με μονάδες *FLOP/Byte*. Σχεδιάζουμε αρχικά μία οριζόντια γραμμή που αντιπροσωπεύει τη μέγιστη επίδοση κινητής υποδιαστολής του υλικού μας και είναι προφανές ότι κανένας πυρήνας δεν θα μπορούσε να την ξεπεράσει. Στη συνέχεια, σχεδιάζουμε μία ευθεία γραμμή για το εύρος ζώνης της μνήμης, η οποία ξεκινάει από το 0 και τέμνει το όριο της μέγιστης επίδοσης κινητής υποδιαστολής του υλικού. Παράδειγμα του μοντέλου Roofline παρουσιάζεται στα σχήματα 3.1 και 3.2.

Σε αυτό το διάγραμμα μπορούμε να σχεδιάσουμε τον πυρήνα που μας ενδιαφέρει ως μία κάθετη γραμμή και το όριο της επίδοσης του πυρήνα βρίσκεται στο σημείο όπου θα συναντήσει πρώτα ένα από τα δύο όρια που θέσαμε. Επομένως τα μέγιστα εφικτά *FLOP/sec* που θα μπορούσε να επιτύχει ένας πυρήνας είναι η τιμή $\max\{B \cdot I, MAX\_FLOP\}$, όπου B το μέγιστο εύρος ζώνης της μνήμης, I η αριθμητική ένταση του πυρήνα και MAX_FLOP η μέγιστη επίδοση κινητής υποδιαστολής.

Οι δύο ευθείες που έχουμε θέσει σαν όρια τέμνονται σε ένα σημείο και η τιμή της αριθμητικής έντασης στο σημείο τομής ονο-

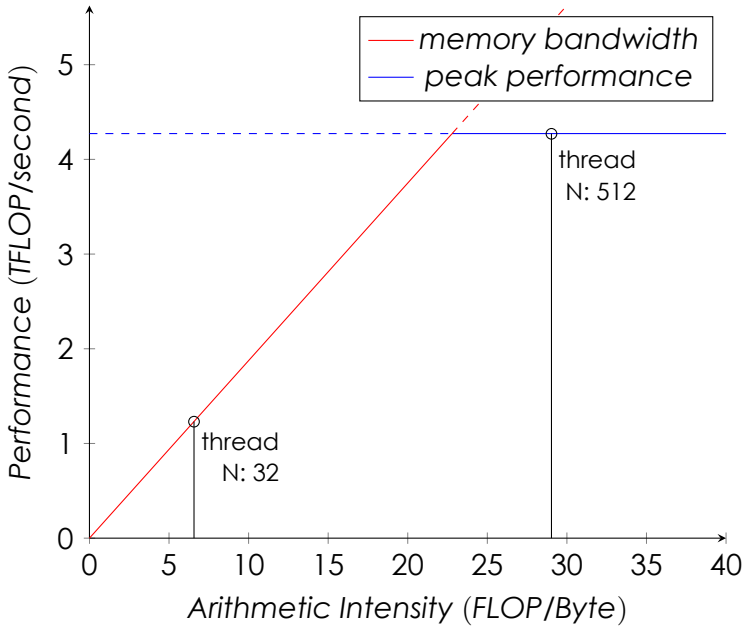


Σχήμα 3.1: Μοντέλο Roofline για Tegra X2

μάζεται γόνατο του μοντέλου Roofline. Οι υπολογιστικοί πυρήνες με αριθμητική ένταση μικρότερη από αυτή ονομάζονται memory-bound, αφού περιορίζονται από την ταχύτητα της μνήμης του συστήματος, ενώ όσοι βρίσκονται δεξιά της ονομάζονται compute-bound και περιορίζονται από την ταχύτητα των υπολογισμών.

Στη δική μας ανάλυση, για τη διευκόλυνσή μας, θα χρησιμοποιήσουμε τη μετρική της αριθμητικής έντασης, στην οποία θεωρούμε ότι τα δεδομένα που είναι απαραίτητα για τον πυρήνα θα έρθουν μία μόνο φορά από την κύρια μνήμη. Αυτό θα έχει ως αποτέλεσμα να κάνουμε μία πιο αισιόδοξη εκτίμηση της μέγιστης θεωρητικής επίδοσης του πυρήνα μας.

Αρχικά θα περιγράψουμε το μοντέλο Roofline για τον πυρήνα πολλαπλασιασμού Αραιού Πίνακα με Διάνυσμα (SpMV), ο οποίος είναι γνωστό ότι είναι memory-bound. Θα θεωρήσουμε έναν αραιό πίνακα A με nnz μη-μηδενικά στοιχεία, M γραμμές και K στήλες και διανύσματα x και y, διαστάσεων $K \times 1$ και $M \times 1$ αντίστοιχα. Οι απαραίτητες λειτουργίες κινητής υποδιαστολής του πυρήνα SpMV που δέχεται ως είσοδο τους πίνακες A και x και παράγει ως έξοδο το διάνυσμα y είναι $2 \cdot nnz$ και το συνολικό αποτύπωμα μνήμης είναι $MF = MF_A + MF_y + MF_x$. Αν κάνουμε την πα-



Σχήμα 3.2: Μοντέλο Roofline για GTX 1060

ραδοχή ότι $nnz \gg M$ και ότι ο A είναι ένας τετραγωνικός πίνακας έτσι ώστε $M = K$, τότε η αριθμητική ένταση εξαρτάται κυρίως από το αποτύπωμα μνήμης του αραιού πίνακα και για την πιο διαδεδομένη δομή αποθήκευσης αραιών πινάκων που είναι το CSR, η αριθμητική ένταση παίρνει την τιμή 0.167, τιμή πολύ πιο χαμηλή από το γόνατο των επεξεργαστών γραφικών γενικού σκοπού. Ο πυρήνας αυτός είναι σχεδόν σε κάθε αρχιτεκτονική memory-bound.

Ας θεωρήσουμε πάλι αραιό πίνακα A με nnz μη-μηδενικά στοιχεία, M γραμμές και K στήλες και αυτή τη φορά, αντί για διανύσματα, τους πυκνούς πίνακες B και C, διαστάσεων $K \times N$ και $N \times M$ αντίστοιχα. Οι απαραίτητες λειτουργίες κινητής υποδιαστολής του πυρήνα SpMM που δέχεται ως είσοδο τους πίνακες A και B και παράγει ως έξοδο τον πίνακα C είναι $2 \cdot nnz \cdot N$. Το συνολικό αποτύπωμα μνήμης του υπολογισμού είναι $MF = MF_A + MF_B + MF_C$ και για αραιούς πίνακες όπου $nnz \gg N$, $nnz \gg M$ και $nnz \gg K$ μπορούμε να θεωρήσουμε ότι το αποτύπωμα μνήμης του αραιού πίνακα είναι $M_A = a \cdot nnz$, με a μία μικρή σχετικά σταθερά. Επομένως αν ο πίνακας A είναι αρκετά αραιός και αποθηκευμένος σε μία δομή αποθήκευσης αραιών πινάκων θα έχουμε $MF = a \cdot nnz + 4 \cdot M \cdot N + 4 \cdot K \cdot N$. Η αριθμητική ένταση του πυρήνα SpMM είναι επομένως

$I = \frac{2 \cdot nnz \cdot N}{MF} = \frac{2 \cdot nnz \cdot N}{a \cdot nnz + 4 \cdot M \cdot N + 4 \cdot K \cdot N}$. Θεωρώντας τετραγωνικό πίνακα A, δηλαδή $M = K$ προκύπτει $I = \frac{2 \cdot nnz \cdot N}{a \cdot nnz + 8 \cdot M \cdot N} = \frac{1}{\frac{a}{2 \cdot N} + 4\frac{M}{nnz}}$.

Για μεγάλες τιμές του N, για τον πρώτο όρο ισχύει $\frac{a}{2 \cdot N} \approx 0$ και συνεπώς προκύπτει $I = \frac{nnz}{4M}$. Σε αυτές τις περιπτώσεις, έχοντας θεωρήσει ότι $nnz \gg M$, ο πυρήνας τείνει συνήθως στην κατηγορία compute-bound. Αντίθετα, για μικρά N, ο όρος $\frac{a}{2 \cdot N}$ είναι σημαντικός και δεν μπορούμε να τον παραλείψουμε στους υπολογισμούς μας, καθώς θα είναι αρκετά μεγαλύτερος από το $4\frac{M}{nnz}$ και συνεπώς προκύπτει $I = \frac{2N}{a}$. Σε αυτές τις περιπτώσεις ο πυρήνας SpMM είναι πιο πιθανό να είναι memory-bound, συμπέρασμα που προκύπτει και διαισθητικά, διότι μοιάζει με τον πυρήνα SpMV.

3.2 Αλγόριθμοι Πολλαπλασιασμού Αραιού Πίνακα με Πίνακα για τις Συμβατικές Δομές Αποθήκευσης Αραιών Πινάκων

Ο αλγόριθμος πολλαπλασιασμού πυκνού πίνακα με πίνακα πραγματοποιεί για τον υπολογισμό κάθε στοιχείου του πίνακα εξόδου $C[i, j]$, το εσωτερικό γινόμενο της $i-$οστής γραμμής του πίνακα A με την $j-$ιοστή στήλη του πίνακα B. Ο αλγόριθμος έχει πολυπλοκότητα $O(MNK)$ και παρουσιάζεται παρακάτω με ψευδοκώδικα (Αλγόριθμος 3.1).

Algorithm 3.1 Ψευδοκώδικας για τον πυρήνα GEMM

```
    A: input M × K in dense format
    B: input K × N dense matrix
    C: output M × N dense matrix
1:  for i = 0 to M do:
2:      for j = 0 to N do:
3:          for k = 0 to K do:
4:              C[i, j] = C[i, j] + A[i, k] ∗ B[k, j];
```

Το συνολικό αποτύπωμα μνήμης του πυρήνα, για τετραγωνικούς πίνακες με τιμές κινητής υποδιαστολής μονής ακρίβειας, είναι $4MN + 4MK + 4KN = 4N^2$. Απαιτούνται $2MNK = 2N^3$ πράξεις, επομένως στην ιδανική περίπτωση που τα δεδομένα που απαιτούνται από τον πυρήνα θα έρθουν μόνο μία φορά από την κύρια μνήμη, η υπολογιστική ένταση είναι $\frac{2N^3}{4N^2} = \frac{1}{2}N$. Ο πυρήνας πολλαπλασιασμού ανάμεσα σε πυκνούς πίνακες έχει μεγάλη αριθμητική ένταση

και, συνεπώς, για πίνακες μεγάλων διαστάσεων περιορίζεται μόνο από τη μέγιστη επίδοση κινητής υποδιαστολής της επεξεργαστικής μονάδας.

Algorithm 3.2 Ψευδοκώδικας για τον πυρήνα SpMM χρησιμοποιώντας τη δομή αποθήκευσης COO

```
   A: input M × K in COO format
   B: input K × N dense matrix
   C: output M × N dense matrix
1: for i = 0 to nnz do:
2:     row = A.rowind[i];
3:     col = A.colind[i];
4:     for j = 0 to N do:
5:         C[row, j] += A.values[i] * B[col, j];
6:     end for
7: end for
```

Algorithm 3.3 Ψευδοκώδικας για τον πυρήνα SpMM χρησιμοποιώντας τη δομή αποθήκευσης CSR

```
   A: input M × K in CSR format
   B: input K × N dense matrix
   C: output M × N dense matrix
1: for i = 0 to M do:
2:     for k = A.rowptr[i] to A.rowptr[i + 1] do:
3:         col = A.colind[k];
4:         for j = 0 to N do:
5:             C[i, j] += A.values[k] * B[col, j];
6:         end for
7:     end for
8: end for
```

Σε αντίθεση με τον πολλαπλασιασμό Πυκνού Πίνακα με Πίνακα, όπου κάθε στοιχείο (μηδενικό ή μη-μηδενικό) λαμβάνει μέρος στον υπολογισμό, ο πυρήνας πολλαπλασιασμού Αραιού Πίνακα με Πίνακα υπολογίζει τον πίνακα εξόδου, λαμβάνοντας υπόψη μόνο τα μη-μηδενικά στοιχεία του αραιού πίνακα εισόδου A. Στους αλγόριθμους 3.2, 3.3 και 3.4 παρουσιάζεται ψευδοκώδικας για τον πυρήνα SpMM, χρησιμοποιώντας τις δομές αποθήκευσης αραιών πινάκων που έχουν παρουσιαστεί στην ενότητα 2.1. Στους 3.2 και 3.3 που αφορούν τις δομές COO και CSR αντίστοιχα, ο αλγόριθμος διατρέχει όλα τα μη-μηδενικά στοιχεία του αραιού πίνακα A και

Algorithm 3.4 Ψευδοκώδικας για τον πυρήνα SpMM χρησιμοποιώντας τη δομή αποθήκευσης BSR

```
    A: input M × K in BSR format
    B: input K × N dense matrix
    C: output M × N dense matrix
    r, c: block dimensions
 1: i_r = 0
 2: for i = 0 to M step by r do:
 3:     for j = A.browptr[i_r] to A.browptr[i_r + 1] step by r · c do:
 4:         j_b = j / (r · c)
 5:         x_0 = A.bcolind[j_b]
 6:         for x = 0 to r do:
 7:             for y = 0 to c do:
 8:                 for z = 0 to N do:
 9:                     Y[i + x, z] += A.bvalues[j + x * c + y] · B[x_0 + y, z];
10:                 end for
11:             end for
12:         end for
13:     end for
14:     i_r = i_r + 1
15: end for
```

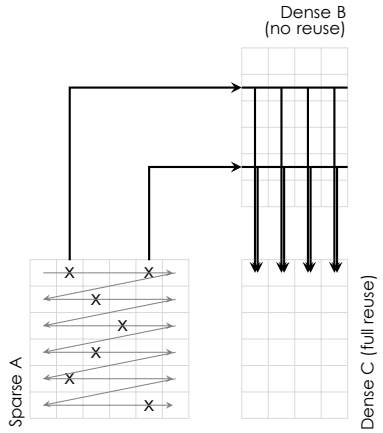
για το εκάστοτε μη-μηδενικό στοιχείο που βρίσκεται στη γραμμή i και στη στήλη j, πολλαπλασιάζει την τιμή του με τη γραμμή j του πίνακα εισόδου B και συσσωρεύει το αποτέλεσμα στη γραμμή i του πίνακα εξόδου C. Στην ίδια λογική, ο αλγόριθμος 3.4 διατρέχει το κάθε μη-μηδενικό block και μέσα σε αυτό πραγματοποιεί έναν πυρήνα GEMM ανάμεσα στο πυκνό block και τον πυκνό πίνακα B. Ο αλγόριθμος SpMM έχει πολυπλοκότητα $O(nnz \cdot N)$, γεγονός που τον κάνει πολύ πιο αποδοτικό από τον πυρήνα GEMM για υψηλές τιμές αραιότητας, όμως κάθε δομή αποθήκευσης οδηγεί σε διαφορετική υλοποίηση με διαφορετική κίνηση στο δίαυλο της μνήμης (memory traffic).

3.3 Δομή Αποθήκευσης Block Compressed Sparse Column (BCSC)

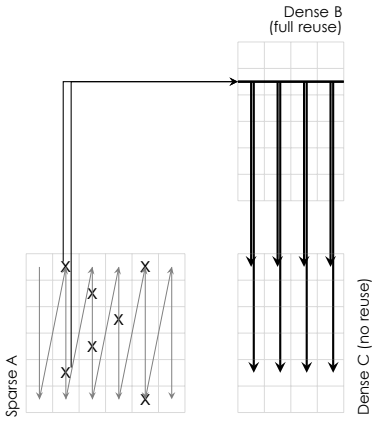
Η υλοποίηση του πυρήνα πολλαπλασιασμού Αραιού Πίνακα με Πίνακα σε κάρτες γραφικών γενικού σκοπού (GPGPUs) συνοδεύεται από δύο σημαντικές προκλήσεις, τον ισορροπημένο δια-

μοιρασμό εργασίας-φορτίου στα νήματα και την επαναχρησιμοποίηση των δεδομένων των πινάκων.

Οι αραιοί πίνακες χαρακτηρίζονται από ποικίλα μοτίβα αραιότητας, δομημένα ή μη-δομημένα και μπορεί να έχουν μη-μηδενικά στοιχεία συγκεντρωμένα μόνο σε λίγες γραμμές ή στήλες, γεγονός που μπορεί να οδηγήσει σε ανισορροπία κατά την ανάθεση γραμμών ή στηλών στα νήματα.



Σχήμα 3.3: Επαναχρησιμοποίηση δεδομένων για CSR SpMM

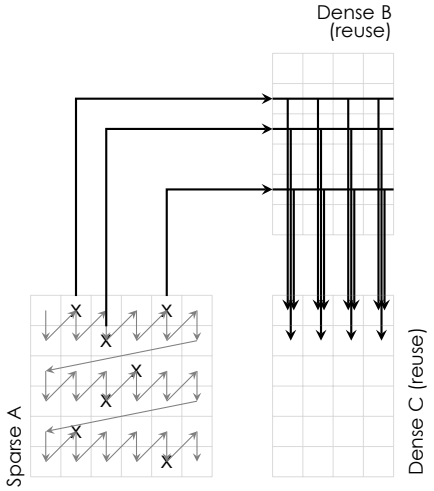


Σχήμα 3.4: Επαναχρησιμοποίηση δεδομένων για CSC SpMM

Επίσης σημαντική πρόκληση αποτελεί η επαναχρησιμοποίηση δεδομένων του πυκνού πίνακα εισόδου B και του πίνακα εξόδου C, δηλαδή η εκμετάλλευση της τοπικότητας (temporal locality) των γραμμών τους που ήδη έχουν φορτωθεί στη cache. Ο πολλαπλασιασμός αραιού πίνακα με πίνακα χρησιμοποιώντας τις υπάρχουσες δομές αποθήκευσης αραιών πινάκων CSR και CSC που παρουσιάστηκαν στο κεφάλαιο 2.1, δεν επιτρέπει την επαναχρησιμοποίηση δεδομένων στον πίνακα εισόδου B και στον πίνακα εξόδου C ταυτόχρονα.

Πιο συγκεκριμένα στην περίπτωση του CSR, επιτυγχάνουμε μέγιστη επαναχρησιμοποίηση των δεδομένων του πίνακα εξόδου C, καθώς τον διατρέχουμε κατά γραμμές, όπως φαίνεται στο σχήμα 3.3. Ωστόσο δεν είναι πάντα δυνατή η επαναχρησιμοποίηση των δεδομένων του πίνακα εισόδου B, καθώς μια γραμμή που βρίσκεται στην cache μπορεί να εξαχθεί από αυτή πριν προλάβει να επαναχρησιμοποιηθεί, σε περίπτωση που προσπελαστούν πολλές άλλες γραμμές προτού γίνει ξανά αναφορά σε αυτή.

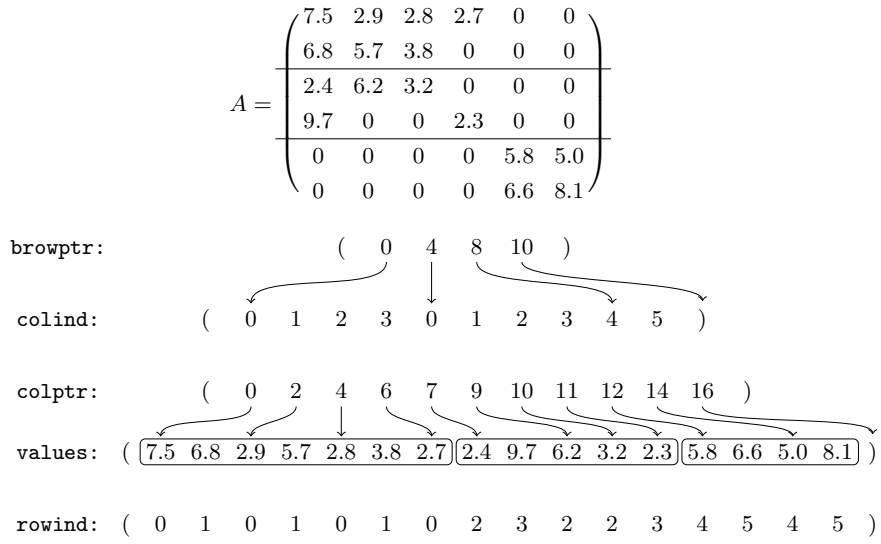
Αντίστοιχα στην περίπτωση του CSC, όπου διατρέχουμε τα μη-μηδενικά στοιχεία κατά στήλη, διαδοχικά στοιχεία χρειάζονται την ίδια γραμμή του πίνακα B και επομένως επιτυγχάνουμε επαναχρησιμοποίηση του πυκνού πίνακα εισόδου B, ωστόσο προκύπτουν ακανόνιστες αναφορές στον πίνακα εξόδου C και μπορεί να φορτώσουμε πολλές γραμμές του, με αποτέλεσμα να διώξουμε κάποιες οι οποίες θα ξαναχρειαστούν αργότερα, όταν προχωρήσουμε στην επόμενη στήλη του αραιού πίνακα A, όπως φαίνεται στο σχήμα 3.4. Από την άλλη πλευρά, το BSR εκμεταλλεύεται την τοπικότητα των πυκνών πινάκων, όμως λόγω του παραγεμίσματος με μηδενικά για την κατασκευή πλήρων μπλοκ, δεν μπορεί να κατασκευάσει μεγάλα μπλοκς, με αποτέλεσμα να μην την εκμεταλλεύεται σε ικανοποιητικό βαθμό. Παρατηρούμε λοιπόν πως είναι εφικτό να εκμεταλλευτούμε την τοπικότητα των δεδομένων τόσο του πυκνού πίνακα εισόδου B, όσο και του πίνακα εξόδου C, αν χωρίσουμε τον αραιό πίνακα εισόδου A σε ομάδες γραμμών και διατρέχουμε τα μη-μηδενικά στοιχεία κατά στήλη σε κάθε ομάδα γραμμών (Σχήμα 3.5).



Σχήμα 3.5: Επαναχρησιμοποίηση δεδομένων για BCSC SpMM

Στη διπλωματική αυτή προτείνουμε μία νέα δομή αποθήκευσης για αραιούς πίνακες που συνδυάζει την τεχνική blocking με τη γνωστή δομή δεδομένων CSC. Οι διαδοχικές γραμμές του αρχικού πίνακα οργανώνονται σε ομάδες σταθερού μεγέθους *m_block* (block) και τα μη-μηδενικά στοιχεία που περιέχουν αποθηκεύονται χρησιμοποιώντας μια παραλλαγή της δομής CSC. Η

δομή BCSC συγκροτείται από πέντε διανύσματα: browptr, colind, values, colptr, rowind, όπου τα τρία τελευταία συνάδουν με τη λογική της δομής CSC. Τα διανύσματα values και rowind περιέχουν την τιμή και το index της γραμμής στην πυκνή αναπαράσταση του πίνακα για κάθε μη-μηδενικό στοιχείο, τα οποία είναι αποθηκευμένα κατά στήλη στο επίπεδο του block που ανήκουν. Το διάνυσμα colptr περιέχει δείκτες στο πρώτο στοιχείο κάθε στήλης (που περιέχει τουλάχιστον ένα μη-μηδενικό στοιχείο) που υπάρχει στο εκάστοτε block, ενώ το διάνυσμα colind διατηρεί το index αυτών των στηλών στην πυκνή αναπαράσταση του πίνακα. Τέλος, το διάνυσμα browptr περιέχει δείκτες στο διάνυσμα colptr, από όπου ξεκινά κάθε block γραμμών. Όλα τα παραπάνω φαίνονται καλύτερα με ένα παράδειγμα στο σχήμα 3.6.



Σχήμα 3.6: BCSC Δομή Αποθήκευσης Αραιού Πίνακα

Για έναν πίνακα σε αυτή την αναπαράσταση, με τιμές κινητής υποδιαστολής μονής ακρίβειας, ο απαιτούμενος χώρος στη μνήμη είναι $4 \cdot nnz + 4 \cdot nnz + 4 \cdot nnzc + 4 \cdot (nnzc + 1) + 4 \cdot (nnzb + 1) = 8 \cdot nnz + 8 \cdot nnzc + 4 \cdot nnzb + 8$ bytes, όπου nnz είναι το πλήθος των μη-μηδενικών στοιχείων, $nnzc$ το πλήθος των μη-μηδενικών στηλών σε όλα τα block που δημιουργούνται και $nnzb = \lceil \frac{m}{m\_block} \rceil$ το πλήθος των block. Αντίστοιχα προκύπτει ότι για τιμές κινητής υποδιαστολής διπλής ακρίβειας το αποτύπωμα μνήμης του πίνακα είναι $12 \cdot nnz + 8 \cdot nnzc + 4 \cdot nnzb + 8$ bytes. Αυτή η δομή αποθήκευ-

σης είναι ιδανική για πίνακες με μεγάλο πλήθος μηδενικών στηλών ανά μπλοκ γραμμών, ωστόσο το αποτύπωμα μνήμης δεν εξαρτάται μόνο από το πλήθος των μη-μηδενικών στοιχείων, αλλά και από την κατανομή τους.

Στον αλγόριθμο 3.5 φαίνεται σε ψευδοκώδικα η σειριακή έκδοση του πυρήνα SpMM για τη δομή δεδομένων BCSC.

Algorithm 3.5 Ψευδοκώδικας για τον πυρήνα SpMM χρησιμοποιώντας τη δομή αποθήκευσης BCSC

```
   A: input M × K in BCSC format
   B: input K × N dense matrix
   C: output M × N dense matrix
 1: for mblock = 0 to A.browptr.size():
 2:     for i = A.browptr[mblock] to A.browptr[mblock + 1]:
 3:         col = A.colind[i]
 4:         for k = A.colptr[i] to A.colptr[i + 1]:
 5:             row = A.rowind[k]
 6:             for j = 0 to N:
 7:                 C[row, j] += A.values[k] ·B[col, j]
 8:             end for
 9:         end for
10:     end for
11: end for
```

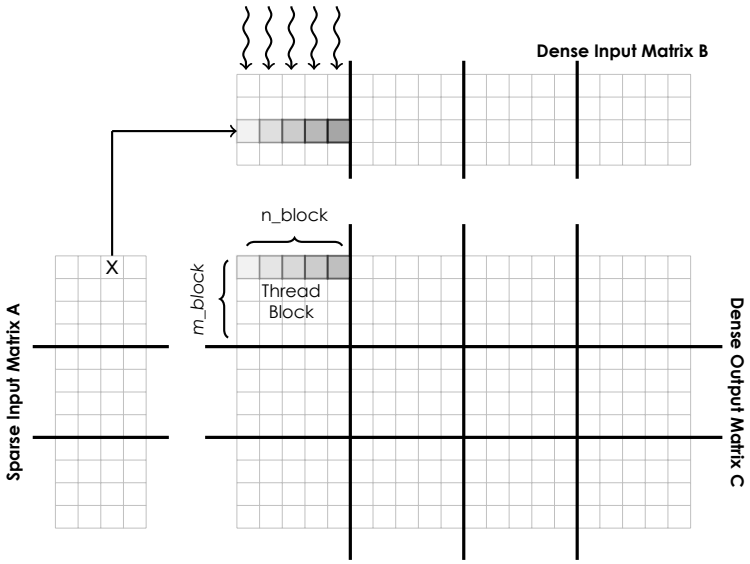
3.4 Παράλληλες Υλοποιήσεις του Πυρήνα Πολλαπλασιασμού Αραιού Πίνακα με Πίνακα με τη Δομή BCSC σε GPUs

Για την αναλυτική περιγραφή των υλοποιήσεων που θα παρουσιαστούν στη συνέχεια, θεωρούμε ότι ο πίνακας εισόδου A, μεγέθους M × K, είναι αποθηκευμένος σε μορφή BCSC, ο πίνακας εισόδου B, μεγέθους K × N, είναι αποθηκευμένος σε πυκνή δομή και ο πίνακας εξόδου C, μεγέθους M × N που προκύπτει, αποθηκεύεται σε πυκνή δομή κατά γραμμές. Και στις τρεις υλοποιήσεις που θα παρουσιαστούν στη συνέχεια, εφαρμόσαμε τη μέθοδο tiling στον πίνακα εξόδου C, δηλαδή αναθέσαμε σε κάθε thread block τον υπολογισμό ενός ορθογώνιου τμήματος *M_Tile* × *N_Tile* του πίνακα.

3.4.1 Naive Υλοποίηση

Η πρώτη μας προσέγγιση για την υλοποίηση του πυρήνα SpMM χρησιμοποιώντας τη δομή BCSC, βασίζεται στη σειριακή προσπέλαση των μη-μηδενικών στοιχείων του αραιού πίνακα A, τον παράλληλο πολλαπλασιασμό καθενός με τα αντίστοιχα στοιχεία του πίνακα εισόδου B και την παράλληλη ενημέρωση των κατάλληλων εγγραφών στον πίνακα εξόδου C.

Αναφορικά με το προγραμματιστικό μοντέλο CUDA, ο πίνακας εξόδου C χωρίζεται σε ορθογώνια τμήματα, μεγέθους *M_Tile* $\times$ *N_Tile* και ο υπολογισμός του καθενός ανατίθεται σε μια συγκεκριμένη ομάδα νημάτων (thread block), μεγέθους *N_Tile*, όπως φαίνεται στο σχήμα 3.7. Συνεπώς, το πλέγμα του πυρήνα συγκροτείται από $\lceil \frac{M}{M\_Tile} \rceil \times \lceil \frac{N}{N\_Tile} \rceil$ thread blocks και το συνολικό πλήθος νημάτων που εκκινούνται είναι $\lceil \frac{M}{M\_Tile} \rceil \cdot \lceil \frac{N}{N\_Tile} \rceil \cdot N\_Tile$.



Σχήμα 3.7: Υλοποίηση Naive του πυρήνα SpMM

Πιο συγκεκριμένα κάθε νήμα αναλαμβάνει τον υπολογισμό μιας στήλης στο τμήμα του πίνακα C που έχει αναλάβει το thread block όπου ανήκει. Διατρέχει σειριακά όλα τα μη-μηδενικά στοιχεία του αντίστοιχου block του A και τα πολλαπλασιάζει με τις αντίστοιχες τιμές του B, οι οποίες βρίσκονται όλες στη στήλη που έχει αναλάβει. Για παράδειγμα, ας εξετάσουμε την περίπτωση ενός νήματος με διακριτικό 3 που ανήκει στο thread block (0,0), δηλαδή στο πάνω αριστερά ορθογώνιο τμήμα του πίνακα εξόδου C. Αυτό

το μεμονωμένο νήμα, διατρέχει όλα τα μη-μηδενικά στοιχεία που βρίσκονται στις *M_Tile* πρώτες γραμμές του πίνακα A και πολλαπλασιάζει το εκάστοτε στοιχείο με το αντίστοιχο στοιχείο του πίνακα B που ανήκει στη στήλη 3, ενημερώνοντας την αντίστοιχη εγγραφή στη στήλη 3 του πίνακα C. Τα παραπάνω φαίνονται καλύτερα στον αλγόριθμο 3.6.

Το προγραμματιστικό μοντέλο της CUDA διαθέτει ειδικούς καταχωρητές για την ταυτοποίηση των thread blocks και των νημάτων μέσα σε αυτά. Κατά σύμβαση χρησιμοποιούμε το blockIdx για να οριοθετήσουμε το ορθογώνιο τμήμα του πίνακα C και το threadIdx για να αντιστοιχίσουμε το κάθε νήμα με τη στήλη του πίνακα C που θα αναλάβει.

Algorithm 3.6 Ψευδοκώδικας της υλοποίησης Naive για τον πυρήνα SpMM

A: input $M \times K$ sparse matrix in BCSC format
B: input $K \times N$ dense matrix
C: output $M \times N$ dense matrix

1: **for** $mb = 0$ **to** $\frac{M}{M\_Tile}$:
2: **for** $nb = 0$ **to** $\frac{N}{N\_Tile}$:
3: **for** $i = A.browptr[mb]$ **to** $A.browptr[mb+1]$:
4: $col = A.colind[i]$
5: **for** $j = A.colptr[i]$ **to** $A.colptr[i+1]$:
6: $row = A.rowind[j]$
7: **for** $tid = 0$ **to** N_Tile:
8: $x = nb \cdot N\_Tile$
9: $C[row, x + tid] += A.values[j] \cdot B[col, x + tid]$
10: **end for** — by each CUDA thread
11: **end for**
12: **end for**
13: **end for** — by each CUDA thread block
14: **end for**

Όπως αναφέραμε στο προγραμματιστικό μοντέλο CUDA, για να επιτύχουμε μέγιστη εκμετάλλευση του εύρους ζώνης της κύριας μνήμης, θα πρέπει όλα τα νήματα εντός ενός thread block όταν εκτελούν μια εντολή μεταφοράς δεδομένων (load/store) να κάνουν πρόσβαση σε διαδοχικές θέσεις μνήμης, διότι αυτές οι μεμονωμένες προσβάσεις μνήμης θα συνενωθούν σε μια ενοποιημένη πρόσβαση. Το γεγονός αυτό μας οδήγησε στην παρακάτω επιλογή. Κάθε νήμα υπολογίζει μία στήλη στο τμήμα του πίνακα

C (αντί για παράδειγμα να υπολογίζει μια γραμμή) έτσι ώστε διπλανά νήματα ενός warp να διαβάζουν διπλανές θέσεις μνήμης στον B και στη συνέχεια να γράφουν σε διπλανές θέσεις του C. Επιπλέον, για να αποφύγουμε την πρόσβαση στην κύρια μνήμη της συσκευής σε κάθε άθροισμα ενός γινομένου, κάθε νήμα διατηρεί έναν buffer με *M_Tile* θέσεις, όπου συσσωρεύει το γινόμενο που υπολογίζει σε κάθε βήμα. Στο τέλος του πυρήνα αντιγράφεται το περιεχόμενο του buffer στην κύρια μνήμη, πάλι με coalesced τρόπο.

3.4.2 Warp - Centric Υλοποίηση

Βασικό μειονέκτημα της παραπάνω υλοποίησης είναι το σειριακό διάβασμα όλων των μη-μηδενικών στοιχείων του εκάστοτε μπλοκ γραμμών του αραιού πίνακα A από όλα τα νήματα του thread block. Επομένως στη δεύτερη υλοποίηση του πυρήνα SpMM, για να μειώσουμε τις προσβάσεις στην κύρια μνήμη, αναθέτουμε σε κάθε warp του thread block τους υπολογισμούς που αφορούν μια ξεχωριστή μη-μηδενική στήλη του μπλοκ γραμμών του αραιού πίνακα A. Το ορθογώνιο τμήμα του πίνακα εξόδου C προκύπτει λοιπόν από το σύνολο των μερικών αθροισμάτων των διαφορετικών warps, όπου τα νήματα εντός του εκάστοτε warp λειτουργούν με τον τρόπο που παρουσιάστηκε στην πρώτη προσέγγιση. Επειδή όμως νήματα από διαφορετικά warps μπορεί να γράφουν ταυτόχρονα σε ίδιες θέσεις μνήμης, δημιουργούνται συνθήκες ανταγωνισμού (race conditions) και είναι απαραίτητη η χρήση ατομικών λειτουργιών (atomic operations) του προγραμματιστικού μοντέλου CUDA.

Οι ατομικές λειτουργίες είναι λειτουργίες που εμποδίζουν την παρέμβαση άλλων νημάτων μέχρι την ολοκλήρωσή τους και επομένως ένα νήμα είναι ικανό να διαβάσει, να τροποποιήσει και να αποθηκεύσει μία τιμή στη μνήμη χωρίς την παρεμβολή εντολών από άλλα νήματα, γεγονός που εγγυάται την αποφυγή συνθηκών ανταγωνισμού. Πιο συγκεκριμένα στην υλοποίηση αυτή χρησιμοποιείται η ατομική λειτουργία *atomicAdd(address, val)* η οποία διαβάζει μία τιμή από τη θέση μνήμης *address*, προσθέτει σε αυτή την τιμή *val* και την αποθηκεύει ξανά στη θέση μνήμης *address*. Στις περιπτώσεις, όπου δύο νήματα πραγματοποιούν μία ατομική λειτουργία στην ίδια θέση μνήμης, οι δύο λειτουργίες πραγματοποιούνται σειριακά και αυτό μπορεί να έχει ως αποτέλεσμα την επιβάρυνση του χρόνου εκτέλεσης, όταν συμβαίνει συχνά.

Πιο αναλυτικά, όσον αφορά το επίπεδο του warp, κάθε warp αποτελείται από 32 λωρίδες (lanes) και κάθε λωρίδα αντιστοιχίζεται σε ένα νήμα που ανήκει σε αυτό. Συνεπώς κάθε νήμα, μέσω του αναγνωριστικού threadId, υπολογίζει το warp στο οποίο ανήκει, διαβάζει σειριακά ένα ένα τα μη-μηδενικά στοιχεία της στήλης αυτής και με βάση τη λωρίδα στην οποία ανήκει ενημερώνει την κατάλληλη στήλη. Τα παραπάνω μπορούν εύκολα να γενικευτούν όταν ένα warp αναλαμβάνει πολλές στήλες του A και κάθε νήμα αυτού ενημερώνει πολλές στήλες του πίνακα εξόδου C.

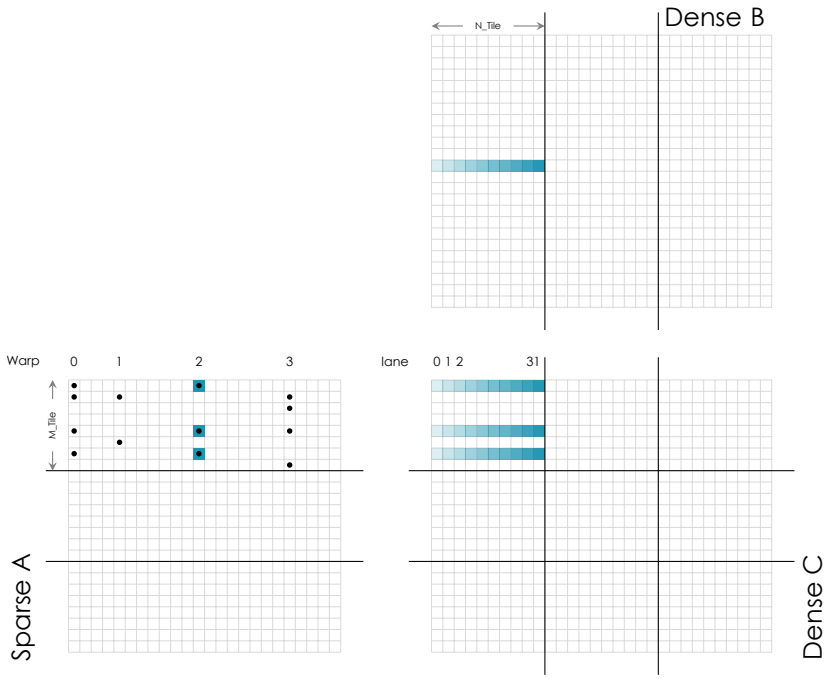
Στον αλγόριθμο 3.7 φαίνονται πιο αναλυτικά, σε ψευδοκώδικα, οι λειτουργίες που αναλαμβάνει κάθε νήμα και στο σχήμα 3.8 παρουσιάζεται ο τρόπος με τον οποίο ανατίθενται οι μη-μηδενικές στήλες του αραιού πίνακα A στα warps και τα στοιχεία που αναλαμβάνει να ενημερώσει το κάθε νήμα με βάση τη λωρίδα στην οποία ανήκει.

Algorithm 3.7 Ψευδοκώδικας της υλοποίησης Warp-Centric για τον πυρήνα SpMM

A: input $M \times K$ in BCSC format
B: input $K \times N$ dense matrix
C: output $M \times N$ dense matrix

```
1:  warp_id = threadIdx.x / 32                    ▶ find thread's warp
2:  lane = threadIdx.x % 32                       ▶ find thread's lane
3:  for mb = 0 to M/M_Tile:
4:      for nb = 0 to N/N_Tile:
5:          x = nb · N_Tile
6:          mb_start = browptr[mb]
7:          mb_end = browptr[mb + 1]
8:          for w = 0 to mb_end − mb_start
9:              col = colind[mb_start + w]
10:             for j = A.colptr[mb_start+w] to A.colptr[mb_start+w+1]:
11:                 row = rowind[j]
12:                 for l = 0 to N_Tile:
13:                     atomicAdd(&C[row, x + l], values[j] · B[col, x + l]
14:                 end for
15:             end for
16:         end for                                   by each CUDA warp
17:     end for                                       by each CUDA thread block
18: end for
```

Ένας πολύ σημαντικός παράγοντας για την επίδοση αυτής της



Σχήμα 3.8: Υλοποίηση Warp-Centric για τον πυρήνα SpMM

υλοποίησης είναι η σωστή χρήση των ατομικών λειτουργιών. Η εφαρμογή των ατομικών λειτουργιών σε θέσεις μνήμης που βρίσκονται στην κύρια μνήμη της συσκευής αποτελεί μια πολύ χρονοβόρα διαδικασία, ενώ όταν οι ατομικές λειτουργίες επηρεάζουν θέσεις της κοινής μνήμης ολοκληρώνονται πολύ πιο σύντομα. Για το λόγο αυτό, κάθε thread block δεσμεύει έναν προσωρινό πίνακα μεγέθους *M_Tile* $\times$ *N_Tile* στην κοινή μνήμη, όπου συσσωρεύονται προσωρινά οι υπολογισμοί του τμήματος του πίνακα εξόδου C που έχει αναλάβει και στο τέλος του πυρήνα αντιγράφει τα αποτελέσματα στην κύρια μνήμη. Επιπλέον, πρέπει να σημειωθεί πως η αραιότητα του πίνακα A μπορεί να έχει θετικά αποτελέσματα στην επίδοση του πυρήνα , διότι δεν παρουσιάζονται τόσο συχνά συνθήκες ανταγωνισμού, κατά τις οποίες οι ατομικές λειτουργίες πραγματοποιούνται σειριακά, σε σύγκριση με μια υλοποίηση του πολλαπλασιασμού πυκνών πινάκων, καθώς τα νήματα διαφορετικών warps δεν γράφουν συχνά ταυτόχρονα στην ίδια γραμμή του πίνακα C.

Το διάβασμα των πινάκων εισόδου πραγματοποιείται με ενοποιημένο (coalesced) τρόπο, ωστόσο λόγω της αραιότητας του πίνακα A κάποια νήματα μένουν ανενεργά κατά την ανάγνωσή

του. Επίσης η μεταφορά του τμήματος του πίνακα εξόδου C από την κοινή μνήμη στην κύρια μνήμη της συσκευής πραγματοποιείται πάλι με ενοποιημένο τρόπο και ταυτόχρονα φροντίζουμε να αποφεύγονται τα bank conflicts στην κοινή μνήμη.

Algorithm 3.8 Ψευδοκώδικας για τη χρήση της shuffle εντολής της CUDA

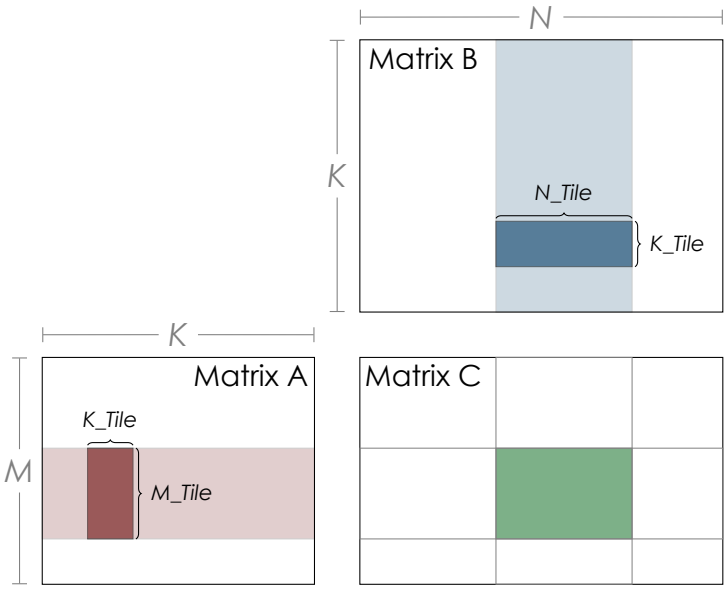
```
1: for j = colptr[mb_start + w] to colptr[mb_start + w + 1] step 32:
2:     my_val = values[j + lane]
3:     my_row = rowind[j + lane]
4:     for shfl_ind = 0 to 32:
5:         val = __shfl_sync(0xffffffff, my_val, shfl_ind, 32)
6:         row = __shfl_sync(0xffffffff, my_row, shfl_ind, 32)
7:         for l = lane to k < N_Tile step 32:
8:             atomicAdd(&C[row, x + l], val ·B[col, x + l])
9:         end for
10:    end for
11: end for
```

Η τελευταία βελτιστοποίηση αυτού του πυρήνα έχει στόχο να μειώσει περαιτέρω τις προσβάσεις στην κύρια μνήμη κατά το διάβασμα του αραιού πίνακα A. Το προγραμματιστικό μοντέλο της CUDA προσφέρει τη δυνατότητα άμεσης ανταλλαγής δεδομένων ανάμεσα σε νήματα του ίδιου warp, μέσω των shuffle εντολών. Οι εντολές αυτές δίνουν τη δυνατότητα ανταλλαγής δεδομένων ανάμεσα σε καταχωρητές των νημάτων και συνεπώς είναι πιο γρήγορες από μια πρόσβαση στην κοινή μνήμη, ενώ παραλείπουν ταυτόχρονα την ανάγκη για συγχρονισμό. Πιο συγκεκριμένα αντί όλα τα νήματα να διαβάζουν σειριακά όλα τα μη-μηδενικά στοιχεία της στήλης που έχει αναλάβει το warp όπου ανήκουν, κάθε νήμα διαβάζει ένα ξεχωριστό στοιχείο (δύο προσβάσεις στην κύρια μνήμη: μία για την τιμή από το διάνυσμα *values* και μία για τη γραμμή από το διάνυσμα *rowind*), διαδικασία που πραγματοποιείται με ενοποιημένο (coalesced) τρόπο. Στη συνέχεια, σε κάθε επανάληψη του βρόγχου το υπεύθυνο νήμα κάνει αναμετάδοση (broadcast) στα υπόλοιπα την απαραίτητη πληροφορία για το στοιχείο που διάβασε. Τα παραπάνω επιτυγχάνονται με την εντολή *__shfl_sync(mask, var, srcLane, width=warpSize)* στην οποία η τιμή *mask* καθορίζει ποια νήματα συμμετέχουν στην ανταλλαγή, *var* είναι ο καταχωρητής από τον οποίο θα γίνει η αναμετάδοση, *scrLane* η λωρίδα του νήματος που θα κάνει την αναμετάδοση και

width το εύρος του warp. Στον αλγόριθμο 3.8 φαίνεται η αλλαγή που πρέπει να γίνει στο βρόγχο *j* και για να είναι πιο κατανοητό έχουν παραληφθεί οι έλεγχοι για την περίπτωση που η στήλη έχει αριθμό μη-μηδενικών στοιχείων που δεν είναι πολλαπλάσιο του 32.

3.4.3 Tiling Υλοποίηση

Η μέθοδος tiling αποτελεί μια βασική τεχνική για την αποτελεσματική εκμετάλλευση της επαναχρησιμοποίησης δεδομένων μέσα στο thread block και χρησιμοποιείται σε όλες τις εφαρμογές γραμμικής άλγεβρας πυκνών υπολογισμών, συνελικτικών νευρωνικών δικτύων και stencil υπολογισμών. Η μέθοδος αυτή εφαρμόζεται κυρίως όταν οι διαστάσεις των πινάκων είναι μεγάλης τάξης μεγέθους και οι πίνακες δεν χωράνε στη μνήμη της GPU ολόκληροι, οπότε πρέπει να μεταφέρονται σε κομμάτια.



Σχήμα 3.9: Υλοποίηση Tiling για τον πυρήνα SpMM - Επίπεδο Thread Block

Η ιεραρχία της μνήμης των GPUs αποτελείται από τρία επίπεδα: την κύρια μνήμη, την κοινή μνήμη και τους καταχωρητές και το εύρος ζώνης αυξάνεται, ενώ το latency μειώνεται καθώς πηγαίνουμε από το πρώτο επίπεδο προς το τελευταίο. Επομένως η μέθοδος tiling εφαρμόζεται στους πίνακες εισόδου σε δύο στάδια,

τα δεδομένα που αφορούν ένα thread block έρχονται σταδιακά από την κύρια μνήμη στην κοινή μνήμη και στη συνέχεια τα δεδομένα που αφορούν το κάθε νήμα έρχονται κι αυτά κατά ομάδες από την κοινή μνήμη στο αρχείο καταχωρητών.

Σε αυτή την προσέγγιση, ο πίνακα εξόδου C χωρίζεται σε ορθογώνια τμήματα, μεγέθους *M_Tile* $\times$ *N_Tile* και ο υπολογισμός κάθε τμήματος ανατίθεται σε ένα thread block, όπως προηγουμένως, με τη διαφορά ότι κάθε thread block πλέον διαθέτει *ThreadsY* $\times$ *ThreadsX* νήματα. Κάθε νήμα αναλαμβάνει τον υπολογισμό *ThreadItemsY* $\times$ *ThreadItemsX* στοιχείων του πίνακα C, όπου *ThreadItemsY* $= \lceil \frac{M\_Tile}{ThreadsY} \rceil$ και *ThreadItemsX* $= \lceil \frac{N\_Tile}{ThreadsX} \rceil$, με σκοπό τον πιο λεπτομερή (fine-grained) παραλληλισμό υπολογισμών και δεδομένων.

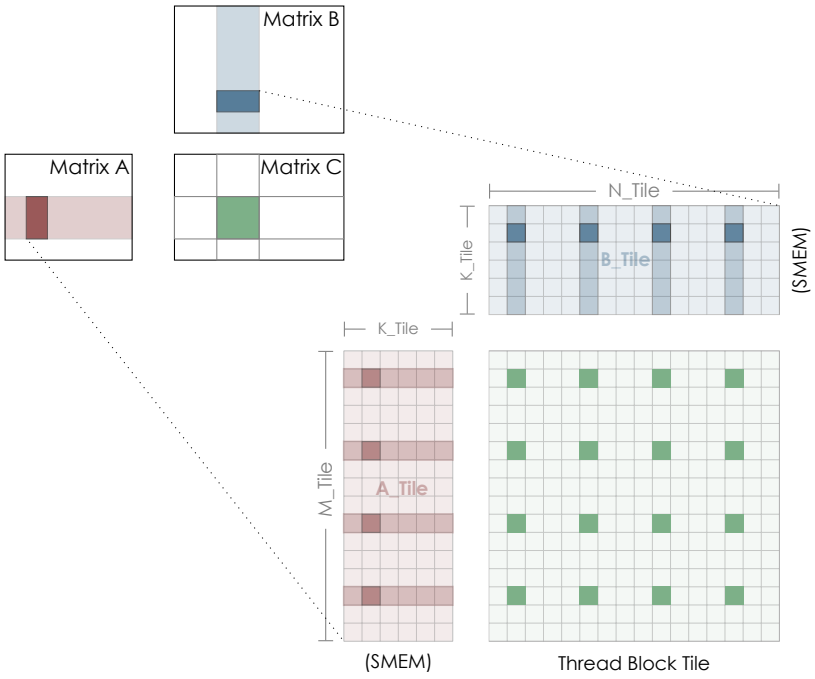
Πιο συγκεκριμένα όσον άφορά το πρώτο στάδιο, κάθε thread block για τον υπολογισμό του ορθογώνιου τμήματος *M_Tile* $\times$ *N_Tile* του πίνακα εξόδου C που έχει αναλάβει, χρειάζεται ένα τμήμα του πίνακα εισόδου A, μεγέθους *M_Tile* $\times$ *K* και ένα τμήμα του πίνακα εισόδου B, μεγέθους *K* $\times$ *N_Tile*. Ένα στοιχείο $C[i, j]$ του πίνακα εξόδου C προκύπτει από τον υπολογισμό του εσωτερικού γινομένου της γραμμής *i* του πίνακα A με τη στήλη *j* του πίνακα B. Συνεπώς για τον υπολογισμό όλων των στοιχείων $C[i, :]$ που βρίσκονται στη ίδια γραμμή *i* του πίνακα εξόδου, απαιτείται μόνο η συγκεκριμένη γραμμή *i* του πίνακα A και αντίστοιχα για τον υπολογισμό των στοιχείων $C[:, j]$ η στήλη *j* του πίνακα B. Αυτό σημαίνει πως τα νήματα που ανήκουν στην ίδια γραμμή του thread block και έχουν αναλάβει τον υπολογισμό στοιχείων που βρίσκονται στην ίδια γραμμή του πίνακα C, χρειάζονται την ίδια γραμμή του πίνακα A. Ενώ αντίστοιχα, τα νήματα που ανήκουν στην ίδια στήλη του thread block και έχουν αναλάβει τον υπολογισμό στοιχείων στην ίδια στήλη του πίνακα C, χρειάζονται την ίδια στήλη του πίνακα B.

Παρατηρούμε λοιπόν πως τα νήματα ενός thread block έχουν κοινά δεδομένα μεταξύ τους, τα οποία αντί να φορτώνονται πολλές φορές, ξεχωριστά από το κάθε νήμα, θα μπορούσαν να φορτωθούν μια φορά στην κοινή μνήμη του thread block και να είναι διαθέσιμα σε όλα τα ενδιαφερόμενα νήματα. Ωστόσο, επειδή το μέγεθος της κοινής μνήμης είναι περιορισμένο και δεν είναι δυνατή η αποθήκευση ολόκληρων των γραμμών του πίνακα A και των στηλών του πίνακα B, επιλέξαμε τη μέθοδο tiling, κατά την οποία φορτώνονται διαδοχικά κομμάτια των πινάκων εισόδου και συγκεκριμένα φέρνουμε *K_Tile* μη-μηδενικές στήλες του A και τις

Algorithm 3.9 Ψευδοκώδικας τον πυρήνα SpMM με τη τεχνική tiling

```
    A: input M × K in BCSC format
    B: input K × N dense matrix
    C: output M × N dense matrix
 1: for mb = 0 to A.browptr.size():
 2:     for nb = 0 to N step N_Tile:
 3:         for kb = A.colptr[mb] to A.colptr[mb + 1] step K_Tile:
 4:         {
 5:             ... compute M_Tile by N_Tile by K_Tile GEMM
 6:         }
 7:         end for
 8:     end for                          by each CUDA thread block
 9: end for
```

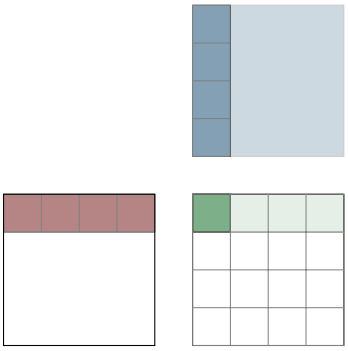
αντίστοιχες *K_Tile* γραμμές του B σε κάθε επανάληψη του βασικού loop του αλγορίθμου 3.9, όπως φαίνεται και στο σχήμα 3.9.



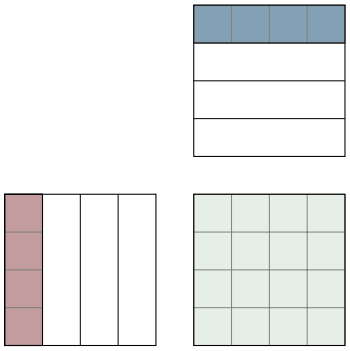
Σχήμα 3.10: Thread Block Tile Structure

Τα δεδομένα αποθηκεύονται στην κοινή μνήμη σε πυκνή δομή έτσι ώστε κάθε νήμα να προσπελαύνει άμεσα, σε χρόνο O(1), το στοιχείο που το ενδιαφέρει και να μετριαστούν τα bank conflicts.

Επομένως το thread block υλοποιεί τον αλγόριθμο πολλαπλασιασμού πυκνού πίνακα με πίνακα, μεγέθους *M_Tile* × *K_Tile* και *K_Tile* × *N_Tile* αντίστοιχα, όπως φαίνεται στο σχήμα 3.10.



Σχήμα 3.11: SpMM MNK



Σχήμα 3.12: SpMM KMN

Σε αυτό το σημείο πρέπει να σημειωθεί πως ο πολλαπλασιασμός πυκνού πίνακα με πίνακα που παρουσιάστηκε στον αλγόριθμο 3.1, για τον υπολογισμό ενός στοιχείου του πίνακα εξόδου *C*[*i*][*j*], πραγματοποιεί το εσωτερικό γινόμενο της γραμμής *i* του πίνακα A με τη στήλη *j* του πίνακα B, επαναχρησιμοποιώντας κάθε στοιχείο των πινάκων εισόδου K φορές, σχήμα 3.11. Αυτή η προσέγγιση για να είναι αποδοτική απαιτεί τη διατήρηση των πινάκων εισόδου σε γρήγορες μνήμες και στη συγκεκριμένη περίπτωση σε καταχωρητές, γεγονός που είναι αδύνατο για μεγάλες τάξεις μεγέθους των *M_Tile*, *N_Tile* και *K_Tile*.

Algorithm 3.10 Ψευδοκώδικας για κάθε thread block της παράλληλης υλοποίησης του πυρήνα SpMM

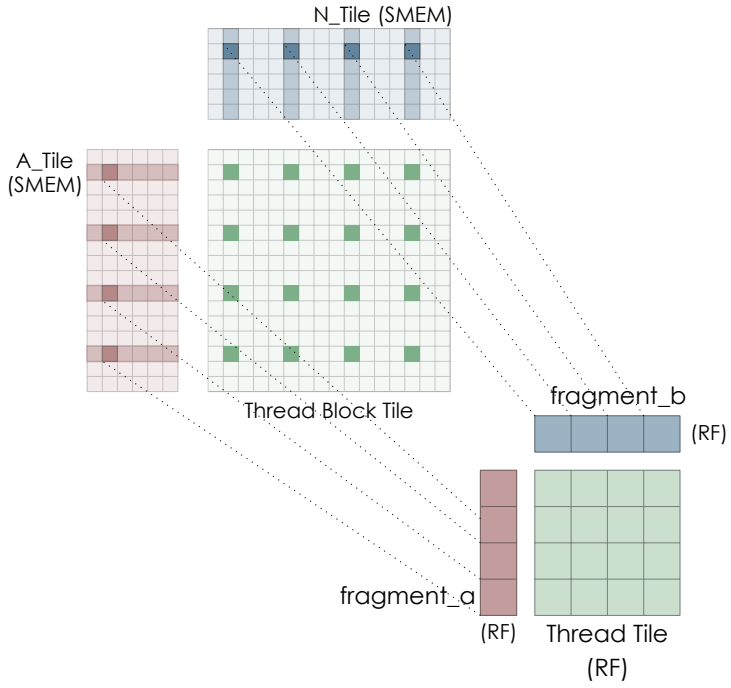
```
1: for kb = A.colptr[mb] to A.colptr[mb + 1] step K_Tile:
2: {
3:   ... load A and B tiles to shared memory
4:   for k = 0 to K_Tile:
5:   ... compute ThreadItemsY by ThreadItemsX by K_Tile GEMM
6: }                                          by each CUDA thread
7: end for
```

Μεταθέτοντας το βρόγχο επανάληψης του *K_Tile* στο εξωτερικό επίπεδο, ο αλγόριθμος πλέον φορτώνει μια στήλη του πίνακα A και μια γραμμή του πίνακα B, από την κοινή μνήμη σε καταχωρητές, υπολογίζει το εξωτερικό τους γινόμενο και συσσωρεύει το αποτέλεσμα στον πίνακα C, όπως φαίνεται στο σχήμα 3.12. Στη

συνέχεια αυτή η στήλη του A και η γραμμή του B δεν χρησιμοποιούνται ποτέ ξανά.

Algorithm 3.11 Ψευδοκώδικας για κάθε νήμα της παράλληλης υλοποίησης του πυρήνα SpMM

```
1: for k = 0 to K_Tile:
2:     … load A Tile from SMEM into registers
3:     … load B Tile from SMEM into registers
4:     for i = 0 to ThreadItemsY:
5:         for j = 0 to ThreadItemsX:
6:             accum[i, j] += fragment_a[i] ·fragment_b[j];
7:         end for
8:     end for
9: end for
```



Σχήμα 3.13: Thread Tile Structure

Στο δεύτερο στάδιο, κάθε νήμα υπολογίζει μια ακολουθία μερικών αθροισμάτων σε ένα εσωτερικό loop (Αλγόριθμος 3.10), όπου φέρνει σε καταχωρητές μία μία τις στήλες του A και τις γραμμές του B, στα διανύσματα *fragment_a* και *fragment_b*, όπως

φαίνεται αναλυτικά στον αλγόριθμο 3.11 και στο σχήμα 3.13. Τα διανύσματα αυτά έχουν μέγεθος *ThreadItemsY* και *ThreadItemsX* αντίστοιχα και κάθε νήμα πραγματοποιεί ένα εξωτερικό γινόμενο σε αυτά, το οποίο προστίθεται στον τοπικό πίνακα συσσώρευσης (accumulator) που διατηρεί το κάθε νήμα. Αυτός ο πίνακας ενημερώνεται πολύ συχνά και συνεπώς πρέπει να βρίσκεται στη γρηγορότερη μνήμη κάθε SM, σε καταχωρητές.

Το μέγεθος των διανυσμάτων *fragment_a* και *fragment_b* είναι συνήθως μικρό σε σχέση με τη διάσταση K για να μεγιστοποιηθεί η υπολογιστική ένταση (compute intensity) σε σχέση με το πλήθος των δεδομένων που φορτώνονται στην κοινή μνήμη και να αποφευχθεί η συμφόρηση του εύρους ζώνης της κοινής μνήμης.

Τεχνικές Βελτιστοποίησης για την υλοποίηση σε GPU

Για να επιτύχουμε μέγιστη εκμετάλλευση του εύρους ζώνης της κύριας μνήμης, όπως στις προηγούμενες προσεγγίσεις, οι προσβάσεις στη μνήμη είναι διαδοχικές σε επίπεδο warp έτσι ώστε να είναι δυνατή η ενοποίησή (coalescing) τους σε όσο το δυνατόν λιγότερες μεταφορές. Πιο συγκεκριμένα επειδή τα νήματα ενός thread block συνήθως είναι λιγότερα από το πλήθος των στοιχείων των πινάκων A και B αντίστοιχα που θέλουμε να φορτώσουμε από την κύρια μνήμη στην κοινή μνήμη, κάθε νήμα αναλαμβάνει τη φόρτωση στοιχείων περισσότερων του ενός, έστω πλήθους x. Αν αναθέταμε σε κάθε νήμα να φορτώσει διαδοχικά στη μνήμη στοιχεία, τότε σε κάθε επανάληψη οι προσβάσεις στη μνήμη σε επίπεδο warp θα είχαν απόσταση x και δεν θα ήταν δυνατή η μέγιστη ενοποίηση των συναλλαγών μνήμης (memory transactions). Αντίθετα, αναθέτοντας σε κάθε νήμα να φορτώσει στοιχεία σε απόσταση ίση με *ThreadsY*, τότε επιτυγχάνουμε τη βέλτιστη ενοποίηση των μεταφορών δεδομένων, καθώς σε κάθε επανάληψη οι προσβάσεις στη μνήμη σε επίπεδο warp είναι διαδοχικές.

Αντίστοιχα σε επίπεδο thread block, θέλουμε να αποφύγουμε τα bank conflicts στις προσβάσεις της κοινής μνήμης. Σε compute capability 6.x η συσκευή έχει 32 banks μεγέθους 4 bytes και, δεδομένου ότι τα ThreadItemsX και ThreadItemsY είναι πολλαπλάσια του 32, θα δημιουργούνταν πολλές συγκρούσεις κατά τη μεταφορά των δεδομένων από την κοινή μνήμη στους καταχωρητές αν τα νήματα διάβαζαν με απόσταση ThreadItemsX και ThreadItemsY. Γι' αυτό το λόγο τα στοιχεία *ThreadItemsY* $\times$

ThreadItemsX που αναλαμβάνει κάθε νήμα και φορτώνει στους καταχωρητές του *fragment_a* και *fragment_b*, δεν βρίσκονται σε διαδοχικές θέσεις, αλλά σε απόσταση *ThreadsY* και *ThreadsX*, αντίστοιχα.

Τέλος, εκτός από το μοτίβο των προσβάσεων στη μνήμη θα πρέπει να επιλεχθεί προσεκτικά το μέγεθος της κοινής μνήμης και το πλήθος των καταχωρητών που απαιτείται από κάθε thread block. Θεωρητικά, όσο μεγαλύτερη είναι η πληρότητα των πολυεπεξεργαστών, δηλαδή όσο περισσότερα thread blocks είναι ενεργά σε ένα SM, τόσο μεγαλύτερη επίδοση μπορεί να επιτύχει η GPU, καθώς με την εναλλαγή των warps κρύβεται το latency. Καθώς το μέγεθος του αρχείου καταχωρητών και της κοινής μνήμης είναι περιορισμένο ανά SM, μόνο ένα περιορισμένο σύνολο από warps μπορεί να εκτελεστεί ταυτόχρονα σε έναν πολυεπεξεργαστή και συγκεκριμένα $(Threads/SM) * (Registers/Thread) \leqslant (Restisters/SM)$. Αν το πλήθος των καταχωρητών ανά νήμα υπερβεί αυτό το όριο, τότε πραγματοποιείται διαρροή των καταχωρητών στην τοπική μνήμη και καθώς ο πυρήνας είναι bandwidth-limited το γεγονός αυτό μπορεί να μειώσει την επίδοση.

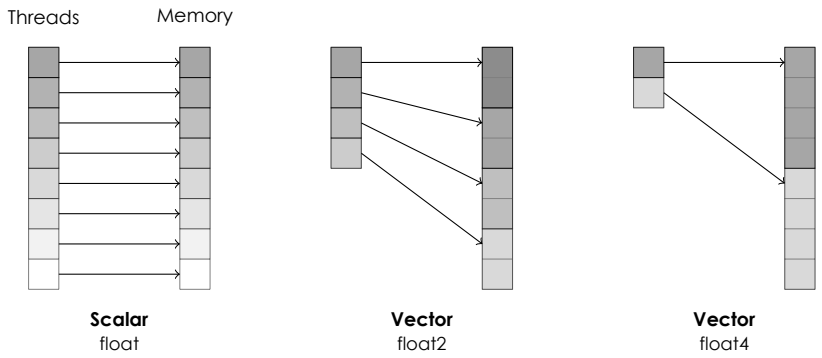
Στη συνέχεια, εξετάζουμε κάποιες ακόμα γνωστές τεχνικές βελτιστοποίησης, όπως η εκτύλιξη βρόγχου (loop unrolling), οι διανυσματικές προσβάσεις μνήμης (vectorized memory access) και η προφόρτωση δεδομένων (prefetching).

Εκτύλιξη Βρόγχου

Η τεχνική της εκτύλιξης βρόγχου πρόκειται για μια τεχνική μετασχηματισμού βρόγχου που συμβάλλει στη βελτιστοποίηση του χρόνου εκτέλεσης ενός προγράμματος και πραγματοποιείται από το μεταγλωττιστή, εισάγοντας τη μακροεντολή #pragma unroll πάνω από το βρόγχο που θέλουμε να μετασχηματίσουμε. Πιο συγκεκριμένα αναπαράγει το σώμα ενός βρόγχου τόσες φορές όσες ο παράγοντας εκτύλιξης που έχουμε ορίσει, μειώνοντας ή καταργώντας τις επαναλήψεις, τους ελέγχους συνθήκης τερματισμού και τις ενημερώσεις του μετρητή επαναλήψεων. Η τεχνική αυτή όμως μπορεί να οδηγήσει σε αύξηση του αρχείου καταχωρητών λόγω της δημιουργίας προσωρινών μεταβλητών, γεγονός που μπορεί να επιφέρει μείωση της επίδοσης, καθώς επίσης προϋποθέτει ότι ο παράγοντας εκτύλιξης είναι γνωστός κατά τη διάρκεια της μεταγλώττισης. Η τεχνική αυτή εφαρμόστηκε στους βρόγχους επανάληψης που έχουν σταθερό εύρος, γνωστό κατά τη μεταγλώττιση.

Διανυσματικές προσβάσεις μνήμης

Για να επιτευχθεί καλύτερη επίδοση του πυρήνα, είναι σημαντικό να ελαχιστοποιηθεί το ποσοστό βοηθητικών εντολών, όπου με τον όρο βοηθητικές εντολές εννοούμε μη μαθηματικές εντολές, όπως εντολές φόρτωσης/αποθήκευσης μνήμης. Οι μεταφορές στη μνήμη που αφορούν θέσεις μνήμης στη σειρά μπορούν να βελτιστοποιηθούν με την τεχνική της διανυσματικής πρόσβασης μνήμης, στην οποία η μεταφορά πραγματοποιείται με διανύσματα από βασικούς τύπους δεδομένων αντί για τον αρχικό βασικό τύπο, όπως φαίνεται στο σχήμα 3.14.



Σχήμα 3.14: Διανυσματικές Προσβάσεις Μνήμης

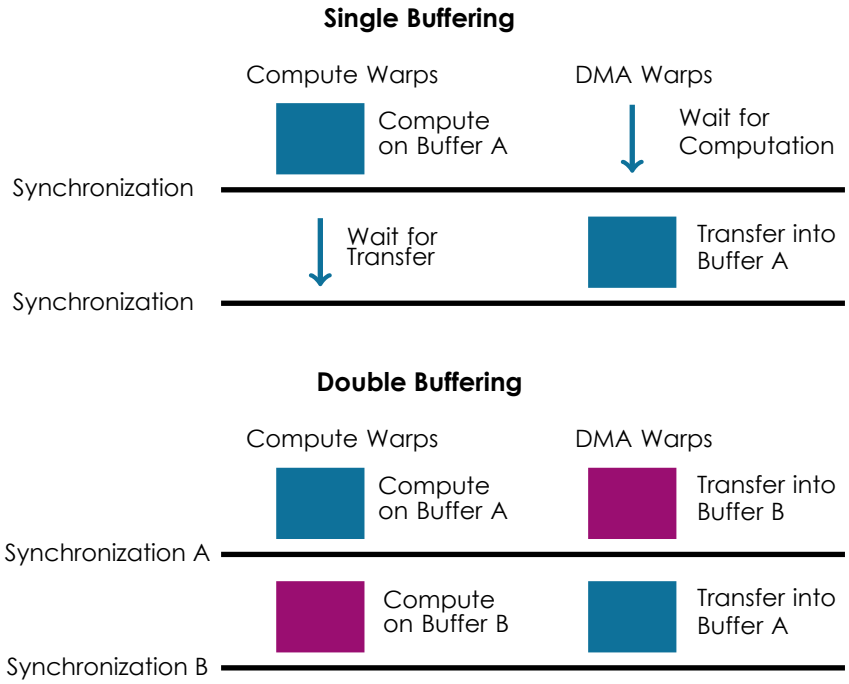
Για παράδειγμα μία μεταφορά 2*n* δεδομένων μεγέθους 4 byte μπορεί να μετασχηματιστεί στη μεταφορά *n* δεδομένων μεγέθους 8 byte. Το προγραμματιστικό μοντέλο της CUDA υποστηρίζει τέτοιου είδους μεταφορές με τύπους δεδομένων, όπως int4, float2 ή float4 που είναι ήδη ορισμένοι στα αρχεία επικεφαλίδων της CUDA και αρκεί να αλλάξουμε τον τύπο του δείκτη που δείχνει σε αυτά τα δεδομένα με εντολές, όπως reinterpret_cast ή τον τελεστή casting της C. Σχεδόν πάντα η τεχνική των διανυσματικών προσβάσεων μνήμης είναι προτιμότερη αφού επιτυγχάνει αυξημένη χρησιμοποίηση του εύρους ζώνης, μειώνοντας ταυτόχρονα τον αριθμό των εκτελεσθέντων εντολών και το latency, όμως πρέπει να σημειωθεί ότι μπορεί να αυξήσει την πίεση στους καταχωρητές. Στον πυρήνα μας η τεχνική έδειξε να έχει θετικά αποτελέσματα κατά τη μεταφορά των τμημάτων του πίνακα B από την κύρια μνήμη στην κοινή μνήμη, μετατρέποντας τη βαθμωτή (scalar) μεταφορά δεδομένων τύπου float σε διανυσματική (vectorized) δεδομένων τύπου float4.

Προφόρτωση δεδομένων

Η τεχνική της προφόρτωσης δεδομένων πρόκειται για τη διαδικασία φόρτωσης δεδομένων στη μνήμη προτού ζητηθούν. Καθώς η υπολογιστική ισχύς των GPUs συνεχίζει να κλιμακώνεται, σύμφωνα με το νόμο του Moore, ένας αυξανόμενος αριθμός εφαρμογών περιορίζεται από το εύρος ζώνης μνήμης, καθώς τυχόν υπολογισμοί που εμπεριέχουν δεδομένα που απαιτούνται να φορτωθούν από την κύρια μνήμη δεν μπορούν να πραγματοποιηθούν μέχρις ότου να ολοκληρωθεί η μεταφορά. Οι επεξεργαστές γραφικών γενικού σκοπού κρύβουν την επιβάρυνση της επίδοσης λόγω χρονοβόρων ενεργειών, όπως είναι η μεταφορά δεδομένων από την κύρια μνήμη, εναλλάσσοντας τα ενεργά warps. Πιο συγκεκριμένα όταν ένα warp εκδώσει μια εντολή μεγάλης καθυστέρησης τότε ο χρονοδρομολογητής των warps (warp-scheduler) το εναλλάσσει με κάποιο άλλο warp που είναι έτοιμο έτσι ώστε να ελαχιστοποιηθούν οι χαμένοι κύκλοι της μεταφοράς.

Αυτή τη δυνατότητα των GPUs επιχειρούμε να εκμεταλλευτούμε με την οργάνωση των warps σε δύο ομάδες και την εξειδίκευσή τους σε ξεχωριστές λειτουργίες [Bauer et al., 2011]. Η πρώτη ομάδα είναι υπεύθυνη για την εκτέλεση του υπολογιστικού μέρους του αλγορίθμου καθώς και τη μεταφορά των αποτελεσμάτων από τους καταχωρητές στην κύρια μνήμη, ενώ η δεύτερη ομάδα είναι υπεύθυνη για τη μεταφορά δεδομένων από την κύρια μνήμη στην κοινή μνήμη. Το πλήθος των νημάτων που ανατίθενται σε κάθε ομάδα είναι πολλαπλάσιο του μεγέθους του warp, δηλαδή 32 έτσι ώστε να μην υπάρχει διαφοροποίηση της εκτέλεσης εντός του warp και η επικοινωνία μεταξύ των δύο ομάδων επιτυγχάνεται μέσω του μηχανισμού συγχρονισμού παραγωγού-καταναλωτή (producer-consumer), χρησιμοποιώντας τις assembly εντολές που διαθέτει η βιβλιοθήκη της CUDA. Πιο συγκεκριμένα η υπολογιστική ομάδα δίνει εντολή για την έναρξη της μεταφοράς δεδομένων, περιμένει την ολοκλήρωσή της, καταναλώνει τα δεδομένα επαναληπτικά και στο τέλος μεταφέρει τα αποτελέσματα στην κύρια μνήμη. Αντίστοιχα επαναληπτικά, η ομάδα που είναι υπεύθυνη για τη μεταφορά δεδομένων περιμένει το σήμα για να ξεκινήσει τη μεταφορά από την κύρια στην κοινή μνήμη και όταν την ολοκληρώσει ενημερώνει την υπολογιστική ομάδα πως η ενδιαφερόμενη κοινή μνήμη είναι έτοιμη για διάβασμα.

Στη συνέχεια, παρουσιάζουμε δύο τεχνικές προφόρτωσης δεδομένων που χρησιμοποιούν έναν ή δύο ενταμιευτές αντίστοιχα και φαίνονται στο σχήμα 3.15.



Σχήμα 3.15: Τεχνικές Προφόρτωσης Δεδομένων

Η τεχνική του single buffering, χρησιμοποιεί έναν ενταμιευτή και η ομάδα που είναι υπεύθυνη για τη μεταφορά των δεδομένων από την κύρια στην κοινή μνήμη γεμίζει αυτόν τον buffer με δεδομένα που απαιτούνται για τη συγκεκριμένη επανάληψη του βασικού loop του αλγορίθμου (Αλγόριθμος 3.12).

Αντίθετα, η τεχνική double buffering χρησιμοποιεί δύο ενταμιευτές, δεσμεύοντας διπλάσιο χώρο στην κοινή μνήμη και όσο η υπολογιστική ομάδα πραγματοποιεί τους υπολογισμούς στον έναν buffer η δεύτερη ομάδα φέρνει στην κοινή μνήμη, στον άλλο buffer, τα δεδομένα που απαιτούνται για την επόμενη επανάληψη του βασικού loop του αλγορίθμου (Αλγόριθμος 3.13).

Η εφαρμογή αυτής της τεχνικής παρουσίασε τελικά μείωση της επίδοσης του πυρήνα και εκτιμήθηκε πως αυτό συνέβη διότι απαιτείται η δέσμευση διπλάσιου χώρου στην κοινή μνήμη με αποτέλεσμα την υποχρησιμοποίηση των SMs. Ακόμα, εφόσον στον απλό πυρήνα είναι ενεργά αρκετά thread blocks ανά SM, η καθυστέρηση των χρονοβόρων εντολών κρύβεται ικανοποιητικά μέσω της εναλλαγής των warps από διαφορετικά thread blocks.

Algorithm 3.12 Ψευδοκώδικας τον πυρήνα SpMM με Single Buffering

```
    A: input M × K in BCSC format
    B: input K × N dense matrix
    C: output M × N dense matrix
 1: for mb = 0 to A.browptr.size():
 2:     for nb = 0 to N step N_Tile:
 3:         if thread ∈ Compute Threads:
 4:         {
 5:             __start_transfer()
 6:             for kb = A.colptr[mb] to A.colptr[mb + 1] step K_Tile:
 7:             {
 8:                 __wait_for_transfer()
 9:                 … Load Tiles from SMEM into registers
10:                 __start_transfer()
11:                 … compute M_Tile by N_Tile by K_Tile GEMM
12:             }
13:         }
14:         if thread ∈ DMA Threads:
15:         {
16:             for kb = A.colptr[mb] to A.colptr[mb + 1] step K_Tile:
17:             {
18:                 __wait_for_computation()
19:                 … Load Tiles from Global to Shared Memory
20:                  __start_computation()
21:             }
22:             __wait_for_computation()
23:         }
24:         end if
```

Algorithm 3.13 Ψευδοκώδικας τον πυρήνα SpMM με Double Buffering

```
   A: input M × K in BCSC format
   B: input K × N dense matrix
   C: output M × N dense matrix
 1: for mb = 0 to A.browptr.size():
 2:     for nb = 0 to N step N_Tile:
 3:         if thread ∈ Compute Threads:
 4:         {
 5:             __start_transfer_in_shmem1()
 6:             __start_transfer_in_shmem2()
 7:             for kb = A.colptr[mb] to A.colptr[mb+1] step 2 ·K_Tile:
 8:             {
 9:                 __wait_for_transfer_in_shmem1()
10:                 … Load Tiles from SMEM1 into registers
11:                 __start_transfer_in_shmem1()
12:                 … compute M_Tile by N_Tile by K_Tile GEMM
13:                 __wait_for_transfer_in_shmem2()
14:                 … Load Tiles from SMEM2 into registers
15:                 __start_transfer_in_shmem2()
16:                 … compute M_Tile by N_Tile by K_Tile GEMM
17:             }
18:         }
19:         if thread ∈ DMA Threads:
20:         {
21:             for kb = A.colptr[mb] to A.colptr[mb+1] step 2 ·K_Tile:
22:             {
23:                 __wait_for_computation_in_shmem1()
24:                 … Load Tiles from Global Memory to SHMEM1
25:                  __start_computation_in_shmem1()
26:                 __wait_for_computation_in_shmem2()
27:                 … Load Tiles from Global Memory to SHMEM2
28:                  __start_computation_in_shmem2()
29:             }
30:             __wait_for_computation_in_shmem1()
31:             __wait_for_computation_in_shmem2()
32:         }
33:         end if
```

4

Αξιολόγηση Αποτελεσμάτων

Στο κεφάλαιο αυτό, παρουσιάζονται τα πειραματικά αποτελέσματα από την εκτέλεση των πυρήνων SpMM σε δύο διαφορετικές αρχιτεκτονικές υλικού, για διαφορετικά σύνολα δεδομένων και εξετάζονται τα πιθανά οφέλη από την αποθήκευση των αραιών πινάκων στη δομή BCSC. Αρχικά, αναλύονται οι παράμετροι του κάθε πυρήνα και στη συνέχεια παρουσιάζεται η καλύτερη παραμετροποίηση του καθενός που προέκυψε μέσω benchmarking. Στη συνέχεια, συγκρίνονται οι διαφορετικοί πυρήνες μεταξύ τους και τέλος παρουσιάζεται η επίδοση του καλύτερου πυρήνα σε σύγκριση με αυτή της δημοφιλούς βιβλιοθήκης cuSPARSE της Nvidia.

4.1 Τα Χαρακτηριστικά των Δύο Πειραμάτων

4.1.1 Σύνολα Δεδομένων

Εξετάζουμε δύο διαφορετικά σύνολα δεδομένων αραιών πινάκων, που αντιπροσωπεύουν τους πίνακες που συναντώνται σε δύο διαφορετικές εφαρμογές του πυρήνα SpMM.

Η πρώτη εφαρμογή που μελετάμε είναι τα αραιά νευρωνικά δίκτυα και συγκεκριμένα οι πίνακες που προκύπτουν από τα βάρη των συνελικτικών επιπέδων ενός αραιού νευρωνικού δικτύου μετά την εφαρμογή της τεχνικής pruning σε αυτά, όπως αναλύεται και στην ενότητα 2.2.1. Το σύνολο αυτό αποτελείται από τετραγωνικούς πίνακες που δημιουργήθηκαν συνθετικά έτσι ώστε να αντιπροσωπεύουν πίνακες αραιών νευρωνικών μοντέλων. Οι μημηδενικές τιμές των πινάκων αυτών είναι ομοιόμορφα κατανεμημένες σε ολόκληρο τον πίνακα και δεν ακολουθούν κάποιο συγκεκριμένο μοτίβο. Το μέγεθός τους ξεκινάει από πίνακες διαστάσεων

256×256 στην πυκνή αναπαράστασή τους και φτάνει μέχρι και 2048×2048, ενώ η αραιότητά τους κυμαίνεται από 0.6 έως 0.9. Οι τιμές αυτές βασίζονται σε προηγούμενες ερευνητικές εργασίες που δείχνουν πόσο μπορεί να μειωθεί το μέγεθος ενός νευρωνικού δικτύου προτού ξεκινήσει να μειώνεται το ποσοστό επιτυχίας στις προβλέψεις του. Τέλος, το μέγεθος του πυκνού πίνακα για αυτές τις εφαρμογές μπορεί να πάρει τιμές από 32 μέχρι 2048 και αντιπροσωπεύει τον πίνακα εισόδου του συνελικτικού επιπέδου.

Το δεύτερο σύνολο αποτελείται από πίνακες με αραιότητα μεγαλύτερη από 0.99 και όλοι οι πίνακες προέρχονται από τη συλλογή αραιών πινάκων SuiteSparse Matrix Collection. Οι πίνακες της συλλογής προέρχονται από πραγματικές εφαρμογές και χρησιμοποιούνται ευρέως για την αξιολόγηση αλγορίθμων που αφορούν αραιούς πίνακες. Για το λόγο αυτό, είναι ιδανικοί για τη δεύτερη εφαρμογή, η οποία αναλύεται στην ενότητα 2.2.2. Στις επαναληπτικές μεθόδους επίλυσης συστημάτων ή εύρεσης ιδιοτιμών, το πλήθος των γραμμών και των στηλών του αραιού πίνακα είναι μερικές χιλιάδες, ενώ οι μη-μηδενικές τιμές ακολουθούν συχνά δομημένη κατανομή. Ακόμα, σε αυτή την εφαρμογή, το πλήθος των στηλών του πυκνού πίνακα εισόδου είναι μόνο μερικές εκατοντάδες και συνεπώς τα πειράματά μας πραγματοποιήθηκαν για πλήθος στηλών από 8 έως 512.

4.1.2 Hardware

Οι μετρήσεις που θα μελετήσουμε στη συνέχεια προέρχονται από δύο επεξεργαστές γραφικών, την Nvidia GeForce GTX 1060 και την NVIDIA Tegra X2, που έχουν σχεδιαστεί σύμφωνα με την αρχιτεκτονική Pascal.

Ο επεξεργαστής γραφικών Nvidia GeForce GTX 1060 διαθέτει 10 πολυεπεξεργαστές, 6GB GDDR5 μνήμη εύρους ζώνης 192GB/s και μέγιστη συχνότητα ρολογιού 1.81GHz. Κάθε πολυεπεξεργαστής περιλαμβάνει 128 CUDA πυρήνες, 256KB αρχείο καταχωρητών, 96KB κοινή μνήμη και L1 cache μνήμη συνολικού μεγέθους 48KB. Όπως φαίνεται και στο διάγραμμα 2.13 η GTX 1060 διαθέτει έξι 32-bit ελεγκτές μνήμης, που προσφέρουν συνολική διασύνδεση διαύλου 192-bit. Επίσης, κάθε πολυεπεξεργαστής διαθέτει τέσσερις χρονοδρομολογητές των warps και σε κάθε SM μπορούν να είναι ενεργά το πολύ 2048 threads τα οποία οργανώνονται σε warps μεγέθους 32.

Το NVIDIA Tegra X2 πρόκειται για ένα σύστημα on chip (SoC)

που αναπτύχθηκε από την Nvidia για κινητές συσκευές και ενσωματώνει μια κεντρική μονάδα επεξεργασίας (CPU), αρχιτεκτονικής ARM και μια μονάδα επεξεργασίας γραφικών (GPU) σε ένα πακέτο. Τα πρώιμα Tegra SoCs σχεδιάστηκαν ως αποδοτικοί επεξεργαστές πολυμέσων, ενώ πιο πρόσφατα μοντέλα δίνουν έμφαση στην απόδοση για εφαρμογές παιχνιδιών και μηχανικής μάθησης, χωρίς να θυσιάζουν την αποδοτικότητα της ισχύος. Η GPU του Tegra X2 διαθέτει 2 πολυεπεξεργαστές, 8GB LPDDR4 μνήμη εύρους ζώνης 58.3GB/s με μέγιστη ταχύτητα ρολογιού 1.3GHz. Κάθε πολυεπεξεργαστής περιλαμβάνει 128 CUDA πυρήνες, 128KB αρχείο καταχωρητών, L1 cache μνήμη συνολικού μεγέθους 64KB και μπορεί να υποστηρίξει 2048 ενεργά νήματα τα οποία οργανώνονται σε warps μεγέθους 32.

	Gold	**Board**
Μοντέλο GPU	GeForce GTX 1060	Nvidia Tegra X2
Υπολογιστική Ικανότητα	6.1	6.2
Έκδοση nvcc	9.2	9.0
Μέγεθος Κύριας Μνήμης	6078 MBytes	7854 MBytes
Αριθμός Πολυεπεξεργαστών	10	2
Συνολικός Αριθμός πυρήνων	1280	256
Μέγιστη Συχνότητα Ρολογιού GPU	1.81 GHz	1.30 GHz
Συχνότητα Ρολογιού Μνήμης	4004 Mhz	1600 Mhz
Μέγεθος Shared Memory	48 KB	48 KB
Linux kernel	4.15.18	4.4.38

Πίνακας 4.1: Χαρακτηριστικά αρχιτεκτονικών

4.2 Πειράματα σε πίνακες που προκύπτουν από Βαθιά Νευρωνικά Δίκτυα

Θα ξεκινήσουμε παρουσιάζοντας τα αποτελέσματα που εξήχθησαν από το πρώτο σύνολο δεδομένων.

4.2.1 Επιλογή Παραμέτρων για τον κάθε Πυρήνα BCSC

Naive

Στην πρώτη υλοποίηση του πυρήνα BCSC υπάρχουν μόνο δύο παράμετροι που πρέπει να λάβουμε υπόψη μας, το μέγεθος του *M_Tile* κατά την αποθήκευση του αραιού πίνακα με χρήση της δομής BCSC, καθώς και ο αριθμός των νημάτων σε κάθε thread block (*number_of_threads*).

Εξετάσαμε τον πυρήνα για τιμές $M\_Tile \in \{8, 16, 32, 64\}$ σε όλους τους πίνακες του συνόλου και τα καλύτερα αποτελέσματα προέρχονται πάντα από μέγεθος $M\_Tile = 16$. Καθώς το μέγεθος των πινάκων αυξάνεται η τιμή του *M_Tile* που επιτυγχάνει την καλύτερη επίδοση να παραμένει ίδια, αφού οι πίνακες έχουν την ίδια περίπου κατανομή στα μη-μηδενικά στοιχεία, όμως αυτή δεν αλλάζει ούτε για διαφορετικές τιμές αραιότητας, οι οποίες κυμαίνονται από 0.6 έως και 0.9.

Αντίθετα, το πλήθος των νημάτων δεν παραμένει ίδιο όσο αυξάνεται το πλήθος στηλών του πυκνού πίνακα εισόδου *B*, δηλαδή το *N*. Κάθε νήμα του thread block αναλαμβάνει μία στήλη του πίνακα εξόδου *C* και έτσι για τις πρώτες τιμές του *N* η καλύτερη επίδοση προκύπτει όταν $number\_of\_threads = N$. Για $N \geq 256$ καλύτερη επίδοση πετυχαίνουμε με 256 νήματα ανά thread block. Αυτό είναι φυσικό, αφού και η κάρτα γραφικών που χρησιμοποιούμε για αυτά τα πειράματα έχει συνολικά 256 πυρήνες CUDA και δεν θα μπορούσαμε να εκμεταλλευτούμε περαιτέρω την παραλληλία του αλγορίθμου SpMM.

Warp-Centric

Σε αυτή την υλοποίηση εξερευνήσαμε την επίδραση τριών παραμέτρων του πυρήνα: το μέγεθος του *M_Tile* στην αποθήκευση του αραιού πίνακα, το μέγεθος του κάθε warp (*warp_size*) και το πλήθος των warps ανά thread block (*number_of_warps*). Φυσικά το μέγεθος του warp στην πραγματικότητα είναι 32, όμως είναι πολλές φορές συμφέρον να ομαδοποιήσουμε λογικά τα

	number_of_threads	**M_Tile**
Πιθανές τιμές	8-16-32-64-128-256	8-16-32-64
Καλύτερες Παράμετροι για κάθε N		
N = 32	32	16
N = 64	64	16
N = 128	128	16
N = 256	256	16
N = 512	256	16
N = 1024	256	16
N = 2048	256	16

Πίνακας 4.2: Παράμετροι που εξετάστηκαν για τη Naive Υλοποίηση

νήματα του thread block ανά 16. Ο σημαντικότερος περιορισμός αυτού του πυρήνα είναι το μέγεθος της κοινής μνήμης ανά thread block, το οποίο για την κάρτα γραφικών μας είναι 48KB. Εφόσον διατηρούμε στην κοινή μνήμη το κομμάτι του πίνακα εξόδου το οποίο έχει αναλάβει το κάθε thread block θα πρέπει $4 \cdot M\_Tile \cdot warp\_size < 48KB$.

Το μέγεθος του warp παίρνει τιμές $warp\_size \in \{16, 32\}$ έτσι ώστε να μπορούμε να χρησιμοποιήσουμε τις εντολές shuffle του προγραμματιστικού μοντέλου CUDA. Από το benchmarking προέκυψε ότι το $warp\_size = 16$ επιτυγχάνει την καλύτερη επίδοση.

Η παράμετρος *number_of_warps* λαμβάνει τιμές ανάλογα με το *warp_size* έτσι ώστε κάθε thread block να έχει 128 ή 256 νήματα. Σε αυτόν τον πυρήνα, που εκμεταλλεύεται καλύτερα την παραλληλία του αλγορίθμου σε σχέση με τον naive, είναι πάντα καλύτερη επιλογή τα 256 νήματα ανά thread block. Επομένως, το πλήθος των warps ανά block είναι $number\_of\_warps = 16$.

Η παράμετρος *M_Tile* παίρνει τιμές $\{4, 8, 16, 32, 64, 128, 256\}$. Και σε αυτή την περίπτωση τα καλύτερα αποτελέσματα προέρ-

χονται σταθερά από την τιμή *M_Tile* = 128 για όλους τους πίνακες, για διαφορετικά μεγέθη και για διαφορετικές τιμές αραιότητας. Στους αραιούς πίνακες οι στήλες του μπλοκ δεν έχουν συνήθως πολλά μη-μηδενικά στοιχεία και συνεπώς πρέπει να αυξήσουμε αρκετά το *M_Tile* έτσι ώστε να έχουν περισσότερη εργασία τα νήματα του κάθε thread block.

	M_Tile	**number_of_warps**	**warp_size**
Πιθανές τιμές	4-8-16-32-64-128-256	4-8-16	8-16-32
Καλύτερες Παράμετροι για κάθε N			
N = 32	64	16	16
N = 64	64	16	16
	128	16	16
N = 128	128	16	16
N = 256	128	16	16
N = 512	128	16	16
N = 1024	128	16	16
N = 2048	128	16	16

Πίνακας 4.3: Παράμετροι που εξετάστηκαν για τη Warp-Centric Υλοποίηση

Tiling

Σε αυτή την προσέγγιση έχουμε μεγάλο πλήθος παραμέτρων και πρέπει να επιλεχθούν προσεκτικά για να πετύχουμε υψηλή επίδοση. Οι παράμετροι που εξετάζουμε είναι το μέγεθος του thread block, το οποίο έχει διάστασεις *ThreadsY* × *ThreadsX*, το μέγεθος *M_Tile* = *ThreadsY* ·*ThreadItemsY*, το μέγεθος *N_Tile* = *ThreadsX* ·*ThreadItemsX* και ο παράγοντας του tiling *K_Tile*.

Και σε αυτό τον πυρήνα υπάρχει ο περιορισμός της κοινής μνήμης, αφού μάλιστα κρατάμε δύο πίνακες σε αυτή. Χρειαζό-

μαστε 4 ·*M_Tile* ·*K_Tile* bytes για τον αραιό πίνακα εισόδου A και 4 ·N_*Tile* ·*K_Tile* bytes για τον πυκνό πίνακα εισόδου B. Για να καταλήξουμε στις καλύτερες παραμέτρους εξετάσαμε εξαντλητικά ένα μεγάλο πλήθος συνδυασμών για όλους τους πίνακες, ελέγχοντας φυσικά αν υπάρχει αρκετός διαθέσιμος χώρος στην κοινή μνήμη για τα κομμάτια των A και B που χρειάζεται το κάθε thread block και παρουσιάζονται αναλυτικά στον πίνακα 4.4.

	ThreadsX	**ThreadsY**	**ThreadItemsX**	**ThreadItemsY**	**K_Tile**
Πιθανές τιμές	8-16-32	8-16-32	1-4-8-16	1-4-8-16	8-16-32
Καλύτερες Παράμετροι για κάθε N					
N = 32	8	32	4	4	32
N = 64	16	16	4	8	16
N = 128	16	16	4	8	16
N = 256	16	16	4	8	16
N = 512	16	16	4	8	16
N = 1024	16	16	4	8	16
N = 2048	16	16	4	8	16

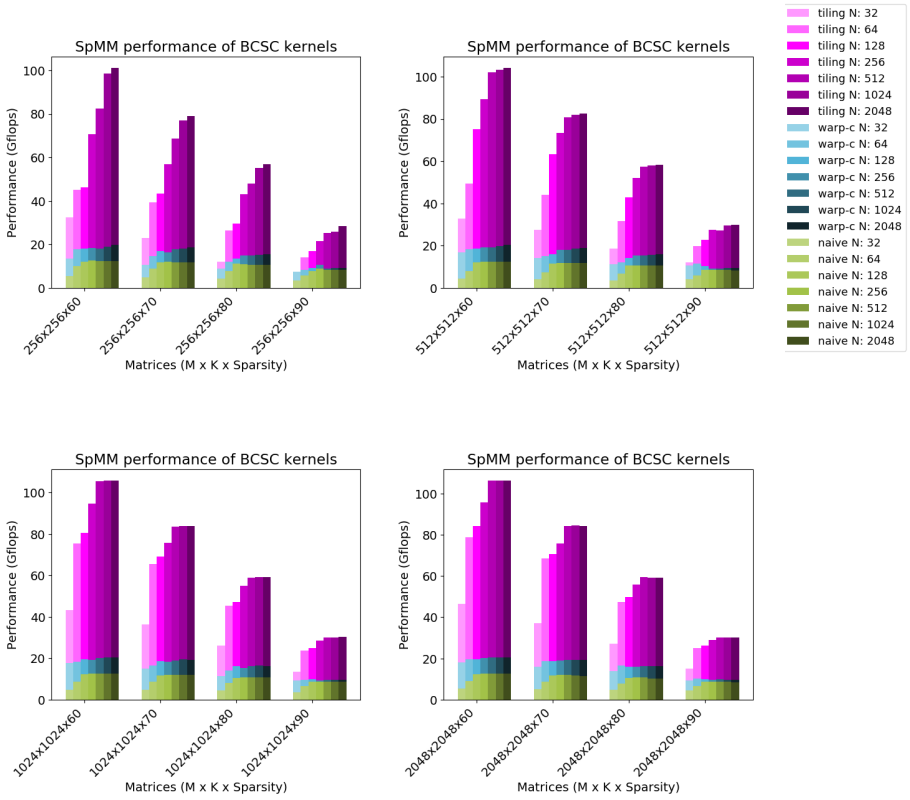
Πίνακας 4.4: Παράμετροι που εξετάστηκαν για την Tiling Υλοποίηση

Αρχικά εξετάζουμε την επίδραση του μεγέθους *N_Tile* στην επίδοση του πυρήνα και παρατηρούμε ότι καλύτερη επίδοση επιτυγχάνεται με $N\_Tile = 64$, εκτός βέβαια από την περίπτωση όπου $N = 32$, στην οποία $N\_Tile = 32$. Αντίστοιχα, σε αυτόν τον πυρήνα, όπως και στον warp-centric, η τιμή $M\_Tile = 128$ επιτυγχάνει μέγιστη επίδοση. Παρατηρούμε ότι η τιμή *M_Tile* δεν διαφέρει για τους πίνακες του συνόλου, το οποίο πιθανότατα συμβαίνει επειδή οι μη-μηδενικές τιμές σε όλο το σύνολο αυτών των πινάκων έχουν παρόμοια κατανομή στα μη-μηδενικά τους στοιχεία.

Και σε αυτόν τον πυρήνα αξιοποιείται καλύτερα η κάρτα γραφικών δημιουργώντας thread blocks με 256 νήματα και οι συνδυασμοί παραμέτρων που δίνουν τις καλύτερες μετρήσεις προκύπτουν όταν $ThreadsX \cdot ThreadsY = 256$, όπου $ThreadsY \cdot ThreadItemsY = 128$ και $ThreadsX \cdot ThreadItemsX = 64$.

4.2.2 Επιλογή της Καλύτερης Υλοποίησης από τους Πυρήνες SpMM της δομής BCSC

Αφού επιλέξαμε τις καλύτερες παραμέτρους για την κάθε υλοποίηση της δομής BCSC, στη συνέχεια συγκρίνουμε την επίδοσή τους στους πίνακες που έχουμε επιλέξει για την πρώτη εφαρμογή. Λόγω της ομοιομορφίας στην κατανομή των μη-μηδενικών στοιχείων των συνθετικών πινάκων, μπορούμε να παρουσιάσουμε τα αποτελέσματα των μετρήσεων τμηματικά έτσι ώστε να εξερευνήσουμε το πώς επηρεάζεται η επίδοση τόσο από την αραιότητα του αραιού πίνακα εισόδου, όσο και από το μέγεθός του.



Σχήμα 4.1: Η επίδοση των υλοποιήσεων BCSC για διαφορετικά μεγέθη του τετραγωνικού αραιού πίνακα

Αρχικά στο σχήμα 4.1 παρουσιάζονται οι πίνακες σε τέσσερα διαγράμματα, όπου στον άξονα *y* φαίνεται η επίδοση του πυρήνα και στον άξονα x οι πίνακες. Σε καθένα από αυτά, οι διαστάσεις

του αραιού πίνακα παραμένουν σταθερές και αντιστοιχούμε τα χρώματα πράσινο, μπλε και μωβ στις υλοποιήσεις naive, warp-centric και tiling αντίστοιχα. Τέλος με διαφορετικές αποχρώσεις του ίδιου χρώματος βλέπουμε διαφορετικές τιμές του N, δηλαδή του πλήθους των στηλών του πυκνού πίνακα εισόδου.

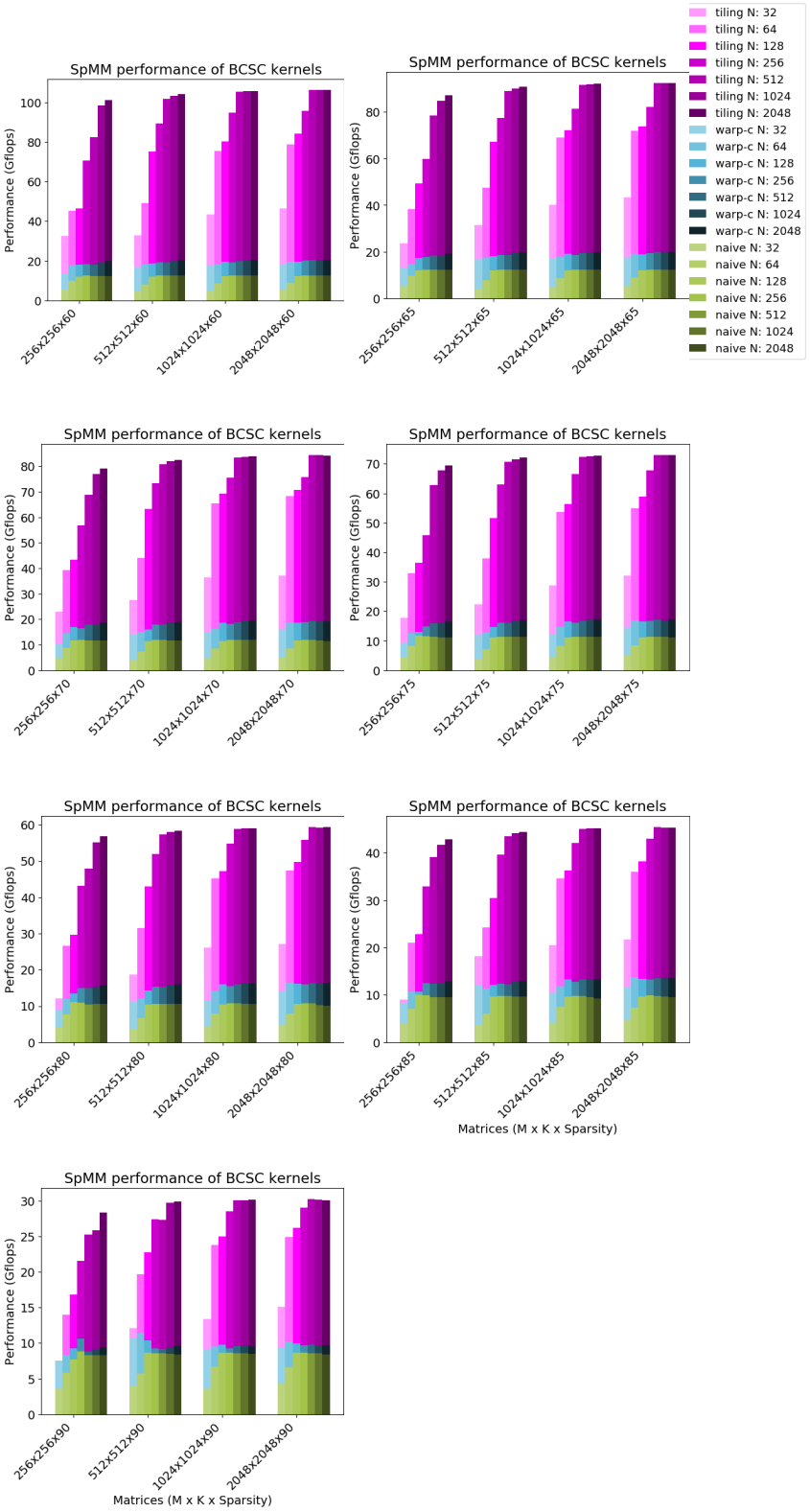
Η υπεροχή της υλοποίησης tiling σε όλες τις περιπτώσεις είναι προφανής, ενώ στη συνέχεια ακολουθεί η υλοποίηση warp-centric και τελευταία η naive.

Αξίζει να παρατηρήσουμε ότι η επίδοση της υλοποίησης tiling αυξάνεται καθώς οι στήλες του πυκνού πίνακα εισόδου αυξάνονται και φτάνει το μέγιστο για $N \geq 1024$ σε μικρούς πίνακες διαστάσεων 256×256, ενώ για τους μεγαλύτερους πίνακες σταματάει να κλιμακώνεται όταν $N \geq 512$ για κάθε τιμή αραιότητας. Αντίθετα οι υλοποιήσεις naive και warp-centric σταματούν να αυξάνουν την επίδοσή τους πολύ νωρίτερα, για $N \geq 128$.

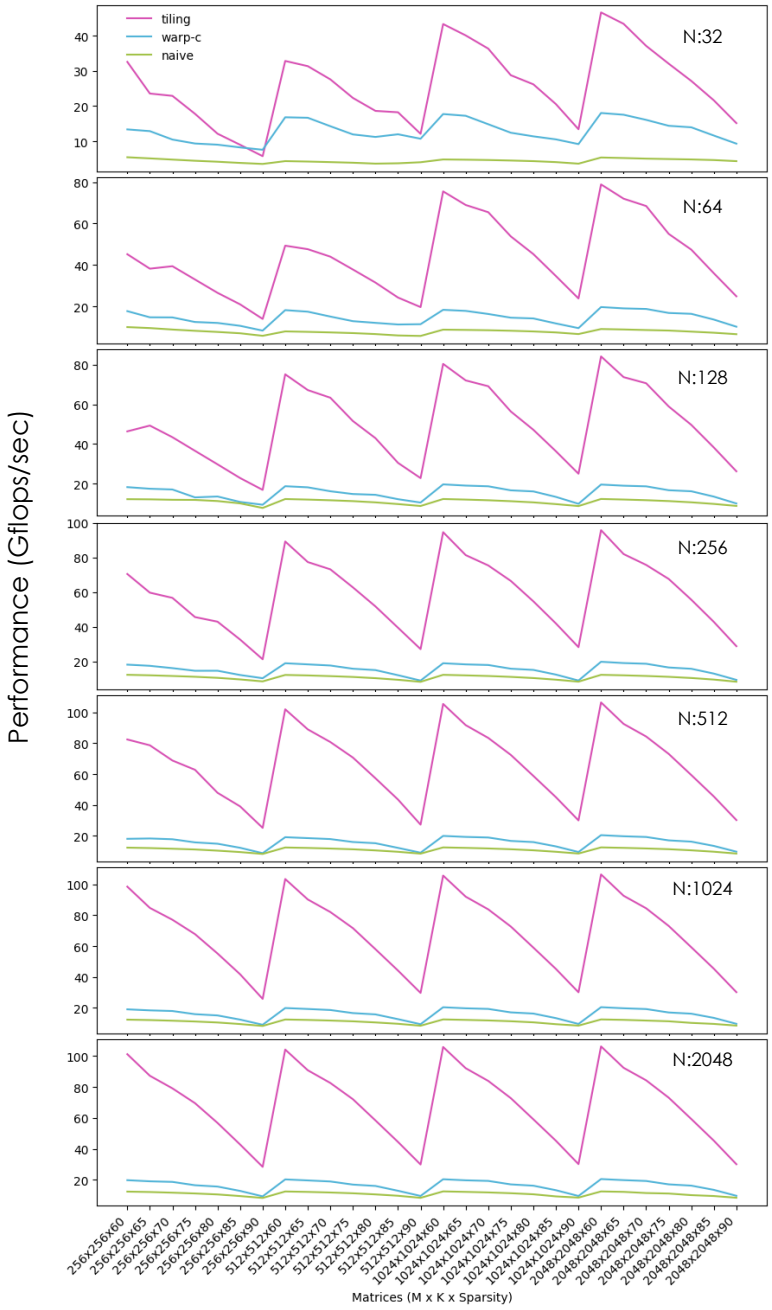
Στη συνέχεια παρουσιάζονται όλοι οι πίνακες, όμως αυτή τη φορά κρατώντας σταθερή την αραιότητα σε κάθε γράφημα. Στον άξονα x φαίνονται οι πίνακες του διαγράμματος και στον άξονα y η επίδοση της κάθε υλοποίησης. Οι τρεις υλοποιήσεις tiling, warp-centric και naive έχουν και πάλι τα χρώματα μωβ, μπλε και πράσινο αντίστοιχα και με διαφορετικές αποχρώσεις αποδίδονται οι διαφορετικές τιμές του N.

Οι παρατηρήσεις μας σε αυτά τα διαγράμματα είναι παρόμοιες με τις προηγούμενες, καθώς για διαφορετικές τιμές αραιότητας η κατάταξη των πυρήνων, από την καλύτερη προς τη χειρότερη, είναι σταθερά tiling, warp-centric, naive με μόνη εξαίρεση έναν πίνακα για $N = 32$, όπου η warp-centric είναι καλύτερη. Ακόμα, η υλοποίηση tiling κλιμακώνεται καλύτερα από τις άλλες δύο. Παρατηρούμε, τέλος, πως η επίδοση των υλοποιήσεων tiling και naive δεν εξαρτώνται από το μέγεθος του αραιού πίνακα A, αλλά κυρίως από την αραιότητά του.

Στο σχήμα 4.3 φαίνονται συγκεντρωτικά τα παραπάνω αποτελέσματα με τους πίνακες στον άξονα x και η επίδοση της κάθε υλοποίησης στον άξονα y.



Σχήμα 4.2: Η επίδοση των υλοποιήσεων BCSC για διαφορετική αραιότητα

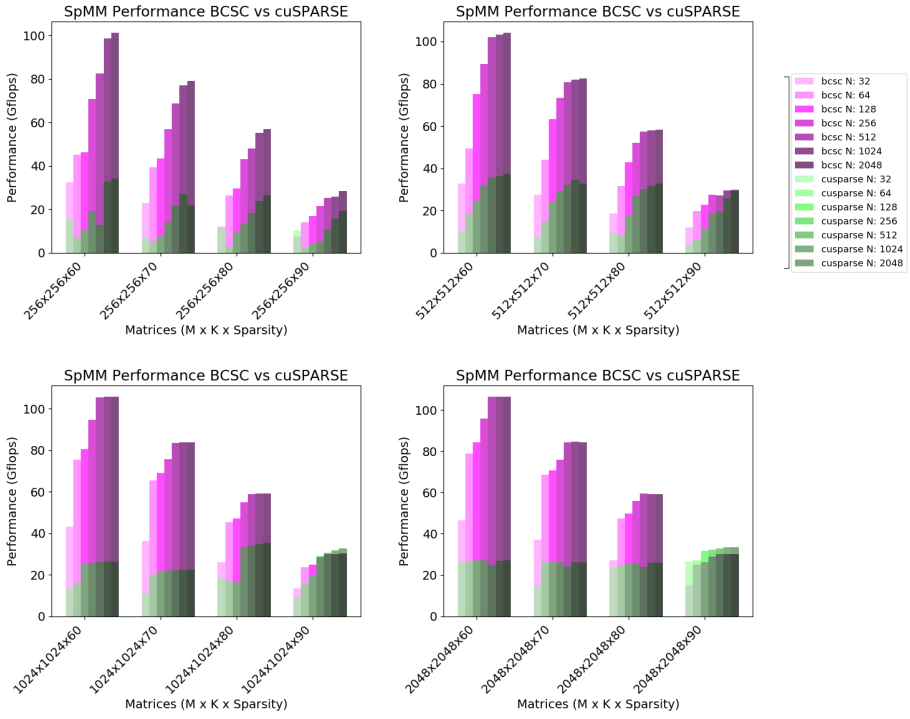


Σχήμα 4.3: Συγκεντρωτικά αποτελέσματα για την επίδοση των υλοποιήσεων BCSC

4.2.3 Σύγκριση με τη Βιβλιοθήκη cuSPRARSE της Nvidia

Για να αξιολογήσουμε την επίδοση των υλοποιήσεών μας συγκρίναμε τα αποτελέσματά μας με την πιο διαδεδομένη βιβλιοθήκη για αραιούς πίνακες σε GPU, τη βιβλιοθήκη cuSPARSE. Η βιβλιοθήκη αυτή έχει αναπτυχθεί από την ίδια την NVIDIA και παρέχει υλοποιήσεις του υπολογιστικού πυρήνα SpMM για τις δομές αποθήκευσης CSR και BSR, οι οποίες έχουν αναλυθεί στην ενότητα 2.1. Η υλοποίηση του πυρήνα με τη δομή CSR δίνει πάντα καλύτερα αποτελέσματα από άποψη επίδοσης και για το λόγο αυτό, στα διαγράμματα στη συνέχεια συγκρίνουμε την επίδοσή μας με αυτή.

Σε αυτή την ενότητα, όπως και στην προηγούμενη, θα παρουσιάσουμε τα πειράματα που πραγματοποιήθηκαν ομαδοποιημένα, ώστε να βγάλουμε συμπεράσματα για την επίδραση της αραιότητας και του μεγέθους των πινάκων στην επίδοση.



Σχήμα 4.4: Η επίδοση της καλύτερης υλοποίησης BCSC και της βιβλιοθήκης cuSPARSE για διαφορετικά μεγέθη του τετραγωνικού πίνακα

Στο σχήμα 4.4 παρουσιάζονται οι πίνακες έτσι ώστε σε κάθε

διάγραμμα να είναι σταθερές οι διαστάσεις του αραιού πίνακα εισόδου. Στον άξονα *x* φαίνονται οι πίνακες του διαγράμματος και στον άξονα *y* η επίδοση των υλοποιήσεων σε Gflops, για διάφορες τιμές του N.

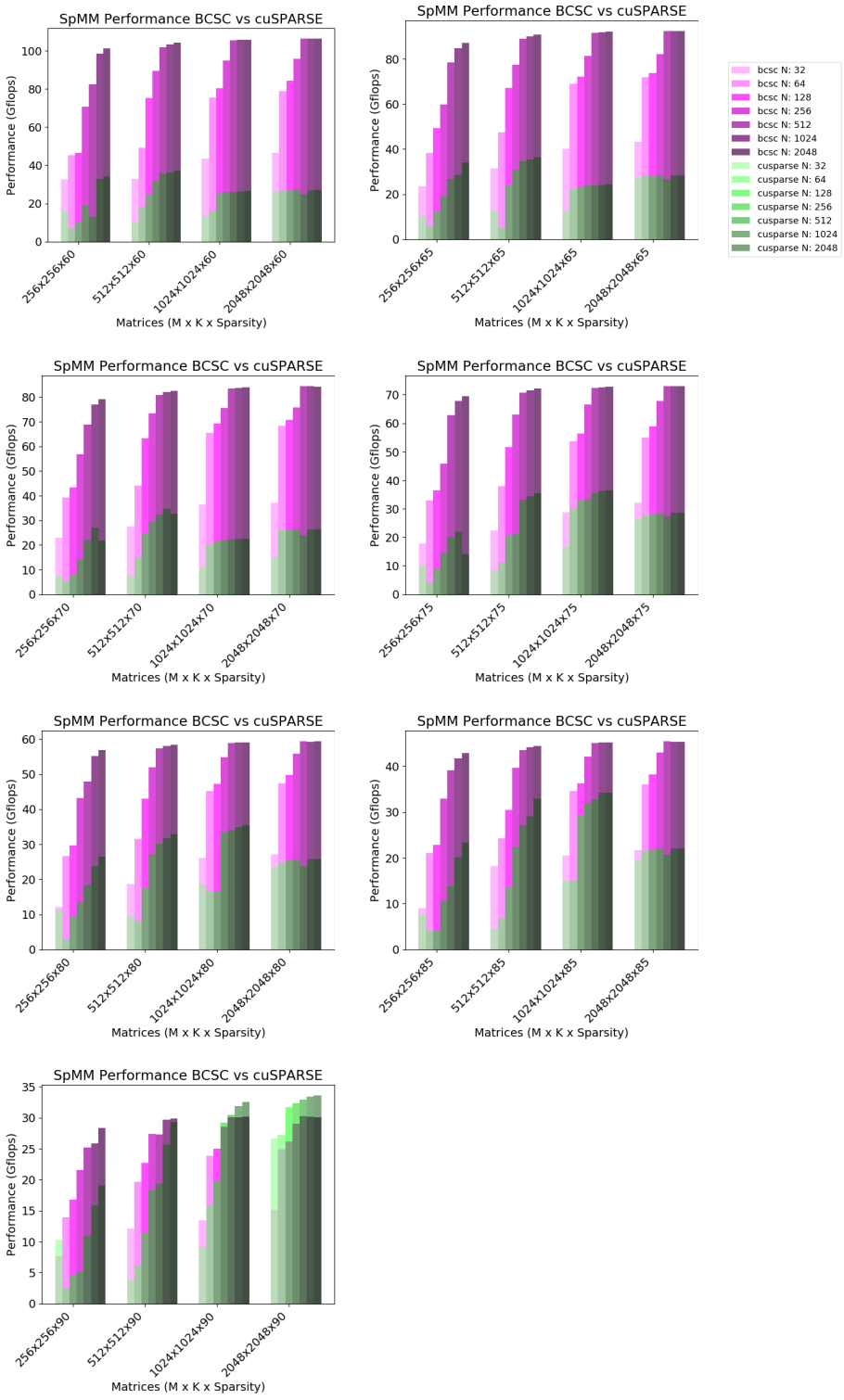
Ανεξάρτητα από το μέγεθος του αραιού πίνακα οι υλοποιήσεις παρουσιάζουν παρόμοια συμπεριφορά. Για μικρές τιμές αραιότητας, όπου ο αραιός πίνακας έχει περισσότερα μη-μηδενικά στοιχεία, η δομή αποθήκευσης BCSC έχει καλύτερη επίδοση, που ξεπερνά έως και τρεις φορές την επίδοση της βιβλιοθήκης cuSPARSE. Για μεγάλες τιμές αραιότητας η επίδοση των δύο δομών αποθήκευσης είναι πιο κοντά και για αραιότητα 0.9 η δομή CSR έχει καλύτερη επίδοση σε κάποιες περιπτώσεις.

Παρατηρούμε ακόμα ότι η υλοποίηση του SpMM για τη δομή BCSC κλιμακώνεται καλύτερα με την αύξηση του πλήθους των στηλών του πυκνού πίνακα εισόδου. Μάλιστα, όταν ο αραιός πίνακας είναι αρκετά μεγάλος, διαστάσεων 2048×2048, η cuSPARSE φαίνεται να έχει σταθερή επίδοση, ανεξάρτητα από το N, ενώ η επίδοση του BCSC συνεχίζει να αυξάνεται μέχρι $N = 512$.

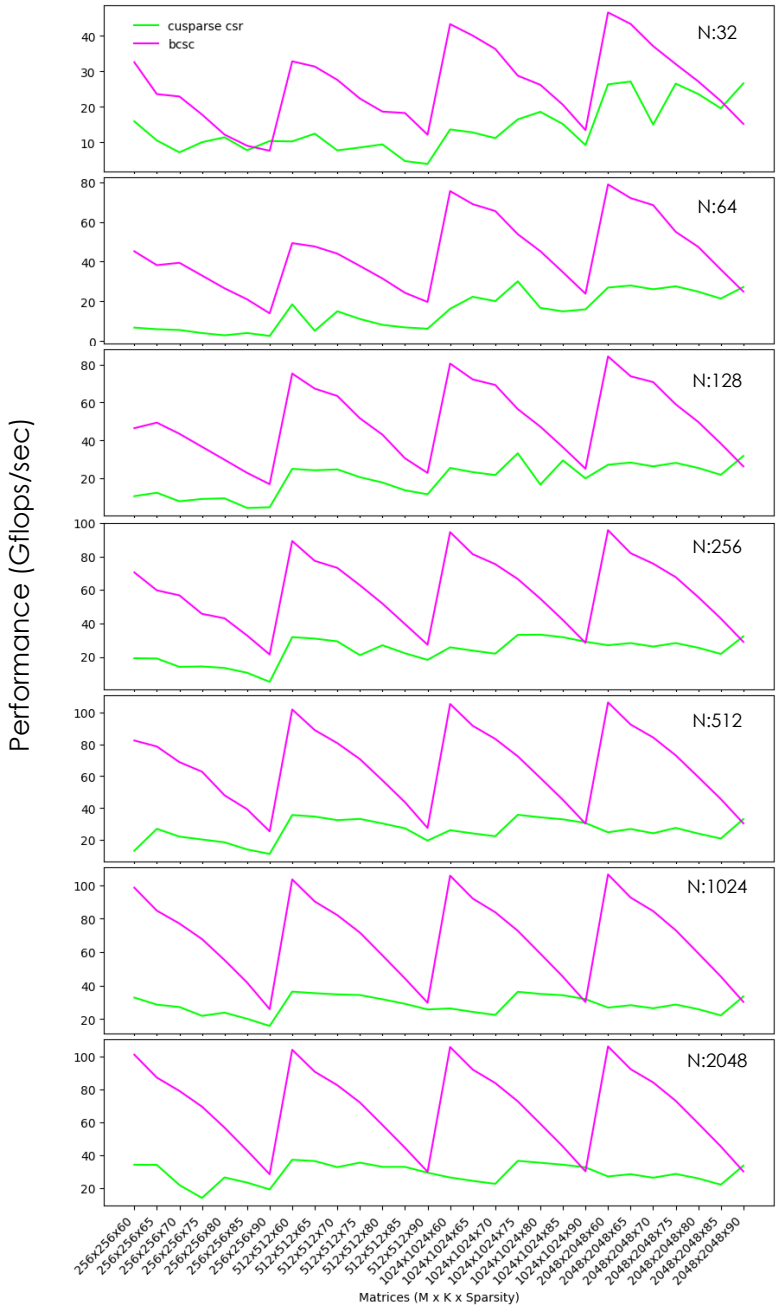
Στο σχήμα 4.5 φαίνονται όλοι οι πίνακες, σε διαφορετικά διαγράμματα για κάθε τιμή αραιότητας που εξετάσαμε. Τα αποτελέσματα αυτά παρουσιάζουν μεγαλύτερο ενδιαφέρον, αφού διαφοροποιούνται για διαφορετικές τιμές αραιότητας.

Όσο μικρότερη είναι η αραιότητα του αραιού πίνακα, τόσο μεγαλύτερη είναι η επίδοση της δομής BCSC και τόσο μεγαλύτερη είναι και η διαφορά της από αυτή της βιβλιοθήκης cuSPARSE. Η διαφορά αυτή μειώνεται σταδιακά μέχρι να φτάσει η αραιότητα την τιμή 0.85 και για πολύ αραιούς πίνακες η βιβλιοθήκη cuSPARSE ξεπερνάει την επίδοση του BCSC.

Για πληρότητα στο σχήμα 4.6 φαίνονται συγκεντρωτικά οι επιδόσεις όλων των πινάκων.



Σχήμα 4.5: Η επίδοση της καλύτερης υλοποίησης BCSC και της βιβλιοθήκης cuSPARSE για διαφορετική αραιότητα



Σχήμα 4.6: Συγκεντρωτικά αποτελέσματα για την επίδοση της καλύτερης υλοποίησης BCSC και της βιβλιοθήκης cuSPARSE

4.3 Πειράματα σε Πίνακες από Επαναληπτικές Μεθόδους Επίλυσης Γραμμικών Συστημάτων

Σε αυτή την ενότητα παρουσιάζονται τα αποτελέσματα των πειραμάτων σε πίνακες από το δεύτερο σύνολο δεδομένων. Οι πίνακες αυτοί είναι πολύ πιο αραιοί, με μεγαλύτερες διαστάσεις και η κατανομή των μη-μηδενικών στοιχείων διαφέρει από πίνακα σε πίνακα.

4.3.1 Επιλογή Παραμέτρων για τον κάθε Πυρήνα BCSC

Naive

Οι παράμετροι που επηρεάζουν την επίδοση σε αυτή την υλοποίηση είναι το μέγεθος του *M_Tile* στην αποθήκευση του αραιού πίνακα, καθώς και ο αριθμός των νημάτων σε κάθε thread block (*number_of_threads*).

Σε αυτό το σύνολο δεδομένων υπάρχει μεγάλη ανομοιογένεια ανάμεσα στους αραιούς πίνακες εισόδου και γι' αυτό δοκιμάσαμε περισσότερες τιμές για *M_Tile*, όπως φαίνεται και στο πίνακα 4.5. Σε όλες τις περιπτώσεις καλύτερη επίδοση επιτυγχάνεται για τιμή $M\_Tile = 8$.

Ο αριθμός των νημάτων ακολουθεί το πλήθος των στηλών του πυκνού πίνακα εισόδου μέχρι να φτάσει τα 256. Για $N = 512$ το 256 παραμένει η καλύτερη παράμετρος.

Warp-centric

Όπως είπαμε και στην προηγούμενη ενότητα, σε αυτόν τον πυρήνα οι παράμετροι που μελετήσαμε είναι το μέγεθος *M_Tile* στην αποθήκευση του αραιού πίνακα, το μέγεθος του κάθε warp (*warp_size*) και το πλήθος των warps ανά thread block (*number_of_warps*).

Και σε αυτή την υλοποίηση υπάρχει διαφοροποίηση των αποτελεσμάτων σε σχέση με το πρώτο σύνολο δεδομένων που περιείχε λιγότερο αραιούς πίνακες. Τα 256 νήματα για κάθε thread block φαίνονται η καλύτερη επιλογή, όμως μεγαλύτερη επίδοση πετυχαίνουμε με *warp_size* μεγέθους 32, για κάθε $N \geq 32$. Για $N = 8$ θέτουμε βέβαια $warp\_size = 8$, αφού δεν έχει νόημα να εκκινήσουμε περισσότερα νήματα τα οποία θα παραμείνουν ανενεργά και αντίστοιχα για $N = 32$ η καλύτερη παράμετρος είναι $warp\_size = 32$.

	number_of_threads	**M_Tile**
Πιθανές τιμές	8-16-32-64-128-256	8-16-32-64-128-256
Καλύτερες Παράμετροι για κάθε N		
N = 8	8	8
N = 16	16	8
N = 32	32	8
N = 64	64	8
N = 128	128	8
N = 256	256	8
N = 512	256	8

Πίνακας 4.5: Παράμετροι που εξετάστηκαν για τη Naive Υλοποίηση

Η τιμή της παραμέτρου *M_Tile* για την πλειοψηφία των πινάκων προκύπτει 64, όταν $N \geq 64$. Όμως, για μικρότερες τιμές δεν είναι σταθερή και για το λόγο αυτό στον πίνακα 4.6 φαίνονται οι 2 ή 3 καλύτερες υλοποιήσεις που καλύπτουν την πλειοψηφία των πινάκων.

Tiling

Σε αυτή την προσέγγιση ελέγξαμε εξαντλητικά ένα μεγάλο πλήθος παραμέτρων, ώστε να εξετάσουμε την επίδρασή τους και να επιλέξουμε την παραμετροποίηση που δίνει τη μεγαλύτερη επίδοση. Οι παράμετροι που εξετάζουμε είναι το μέγεθος του thread block, το οποίο έχει διάστασεις *ThreadsY* ·*ThreadsX*, το μέγεθος *M_Tile* = *ThreadsY* ·*ThreadItemsY*, το μέγεθος *N_Tile* = *ThreadsX* ·*ThreadItemsX* και ο παράγοντας του tiling *K_Tile*, καθώς υπάρχει και ο περιορισμός της κοινής μνήμης που αναφέραμε στην ενότητα 4.2.1. Στον πίνακα 4.7 παρουσιάζονται οι τιμές των παραμέτρων που εξετάστηκαν.

	M_Tile	**number_of_warps**	**warp_size**
Πιθανές τιμές	4-8-16-32-64-128-256-512	4-8-16-32	8-16-32
Καλύτερες Παράμετροι για κάθε N			
N = 8	16	16	8
	32	32	8
	64	32	8
N = 16	32	16	16
	64	16	16
N = 32	32	8	32
	64	8	32
N = 64	64	8	32
N = 128	64	8	32
N = 256	64	8	32
N = 512	64	8	32

Πίνακας 4.6: Παράμετροι που εξετάστηκαν για τη Warp-Centric Υλοποίηση

Οι παράμετροι που δίνουν την καλύτερη επίδοση είναι διαφορετικές από αυτές του πρώτου συνόλου πειραμάτων.

Για αρκετά μικρά $N \in \{8, 16, 32, 64, 128\}$ πετυχαίνουμε την καλύτερη επίδοση όταν το πλήθος των νημάτων μέσα στο thread block είναι 128, ενώ για $N \in \{256, 512\}$ πετυχαίνουμε την καλύτερη επίδοση για 256 νήματα. Επίσης, παρατηρούμε ότι το *M_Tile* παίρνει τιμές 16 και 32.

Το *N_Tile* είναι ίσο με το *N* μέχρι το πλήθος των στηλών του πυκνού πίνακα B να φτάσει την τιμή 256 και στη συνέχεια παραμένει σταθερό, ενώ το *K_Tile* είναι 32 για μικρά *N* και 16 για μεγαλύτερα. Αυτό συμβαίνει λόγω του περιορισμού στην κοινή μνήμη που έχουμε έτσι ώστε να μπορούν να είναι ενεργά ταυτόχρονα περισσότερα από ένα thread blocks σε κάθε SM.

	ThreadsX	**ThreadsY**	**ThreadItemsX**	**ThreadItemsY**	**K_Tile**
Πιθανές τιμές	8-16-32	8-16-32	1-2-4-8	1-2-4-8	8-16-32
Καλύτερες Παράμετροι για κάθε N					
N = 8	8	16	1	1	32
N = 16	8	16	2	1	32
N = 32	8	16	4	1	16
N = 64	16	8	4	1	16
	16	8	4	4	16
N = 128	16	8	8	4	16
N = 256	32	8	8	4	16
N = 512	32	8	8	4	16

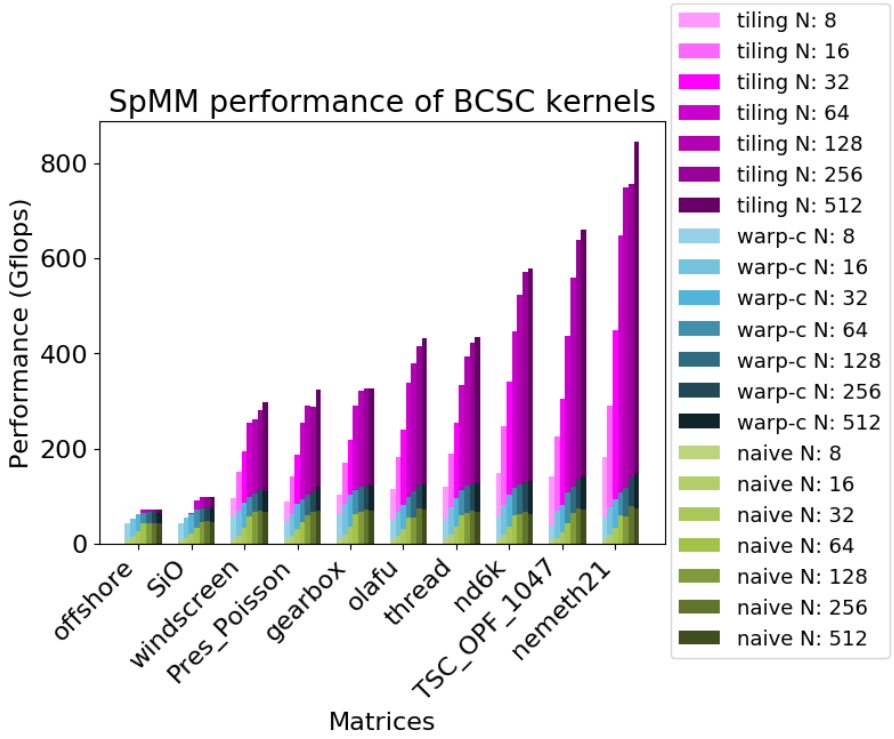
Πίνακας 4.7: Παράμετροι που εξετάστηκαν για την Tiling Υλοποίηση

4.3.2 Επιλογή της Καλύτερης Υλοποίησης από τους Πυρήνες SpMM της δομής BCSC

Σε αυτή την ενότητα εξετάζουμε την επίδοση κάθε υλοποίησης για τη δομή BCSC στους πίνακες που έχουν προέλθει από τη συλλογή SuiteSparse. Οι πίνακες αυτοί παρουσιάζουν ταυτόχρονα πολύ μεγάλες τιμές αραιότητας (οι περισσότεροι > 0.99) και διαφορετική κατανομή στα μη-μηδενικά τους στοιχεία.

Στο τέλος της ενότητας, στο σχήμα 4.8, παρουσιάζονται συγκεντρωτικά τα αποτελέσματα για όλους τους πίνακες που εξετάσαμε, όμως θα αναλύσουμε τα αποτελέσματα ενός υποσυνόλου που επιλέχθηκαν έτσι ώστε να αντιπροσωπεύουν όσο γίνεται ολόκληρο το σύνολο δεδομένων. Αυτοί οι πίνακες και η επίδοση της κάθε υλοποίησης παρουσιάζονται στο σχήμα 4.7. Στον άξονα *x* βλέπουμε τους πίνακες που έχουμε επιλέξει και στον άξονα *y* την επίδοση των υλοποιήσεων. Οι διαφορετικές μπάρες αντιπροσωπεύουν διαφορετικές τιμές του πλήθους των στηλών του πυκνού πίνακα B (*N*) και κάθε υλοποίηση αντιστοιχίζεται σε ένα διαφορετικό χρώμα.

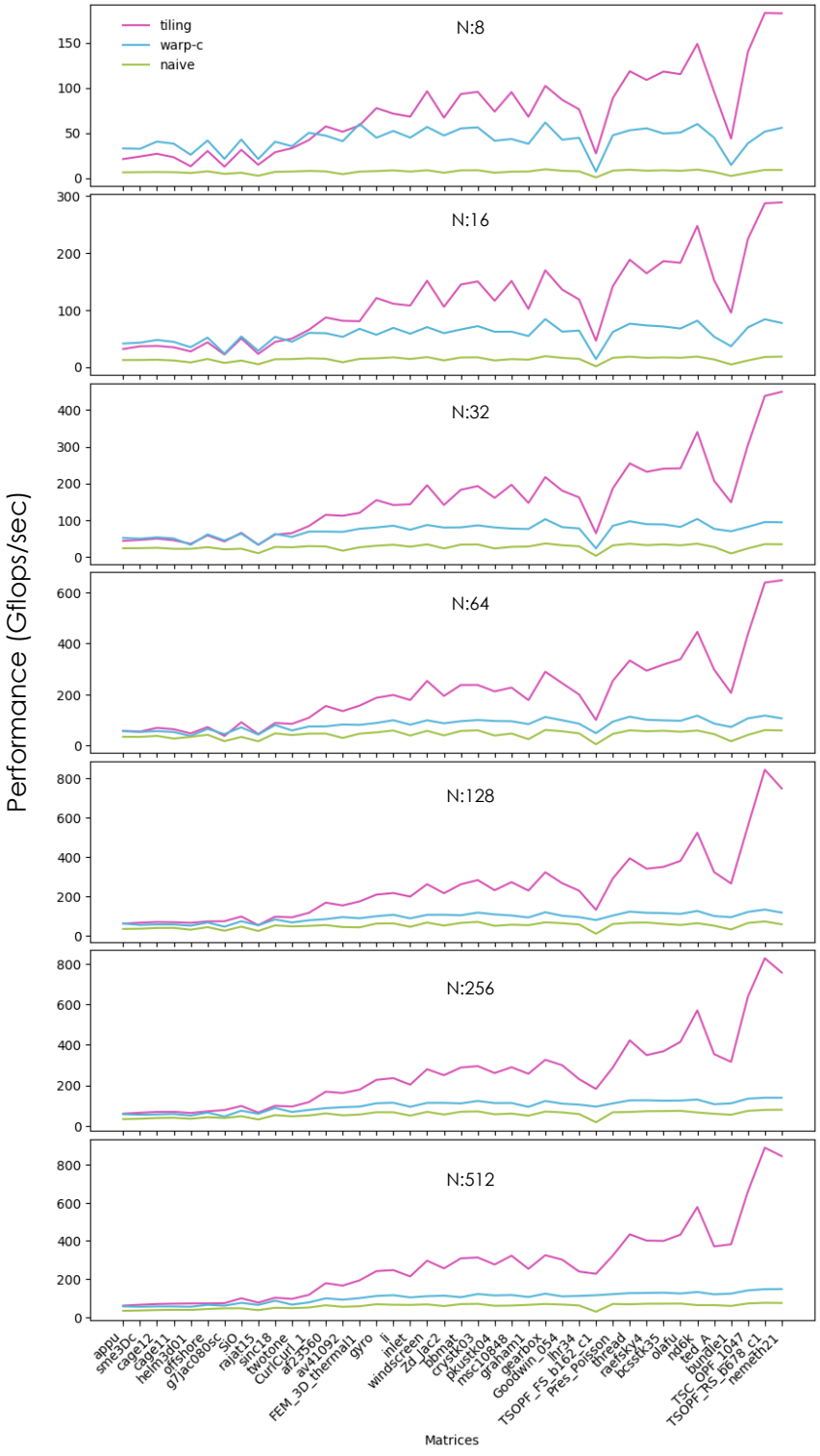
Παρατηρούμε ότι στις περισσότερες περιπτώσεις η υλοποίηση tiling είναι με μεγάλη διαφορά η καλύτερη και πάντα η naive υλο-



Σχήμα 4.7: Η επίδοση των υλοποιήσεων BCSC για ένα υποσύνολο των πινάκων

ποίηση είναι χειρότερη από τις άλλες δύο. Αν εξετάσουμε τις περιπτώσεις στις οποίες η warp-centric υλοποίηση είναι καλύτερη από την tiling, αυτό συμβαίνει μόνο για μικρές τιμές του N και είναι αριθμητικά πολύ κοντά στην επίδοση της υλοποίησης tiling. Ο πιο πιθανός λόγος που συμβαίνει αυτό είναι πως η υλοποίηση tiling, η οποία τμηματικά αποθηκεύει στην κοινή μνήμη τον αραιό πίνακα σε πυκνή δομή και εφαρμόζει GEMM, κάνει πολλούς περιττούς υπολογισμούς που πλήττουν την επίδοση. Ακόμα, παρατηρούμε πως δεν πετυχαίνουμε καλή επίδοση σε αυτούς του πίνακες, γεγονός που οδηγεί στο συμπέρασμα ότι η δομή αποθήκευσης BCSC δεν τους συμπιέζει αποδοτικά.

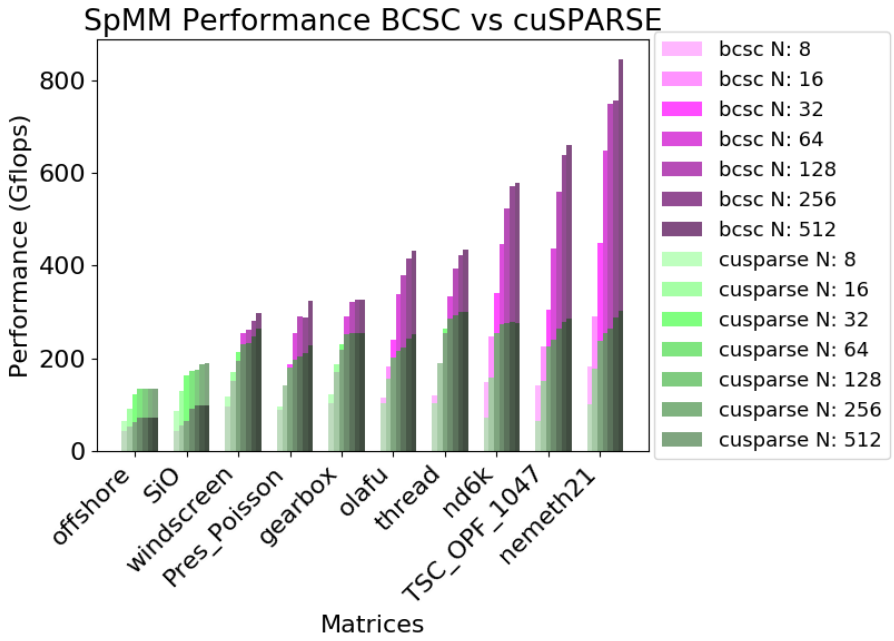
Παρά το γεγονός ότι η υλοποίηση tiling δεν πετυχαίνει πάντα τα καλύτερα αποτελέσματα, αν έπρεπε να επιλεχθεί μία υλοποίηση θα ήταν σίγουρα αυτή. Στις περιπτώσεις που δεν πετυχαίνει την καλύτερη επίδοση, δεν είναι σημαντικά χειρότερη από την warp-centric, ενώ όταν είναι η πρώτη στην κατάταξη μπορεί να δώσει μέχρι και 5.5 φορές την επίδοση της warp-centric υλοποίησης.



Σχήμα 4.8: Συγκεντρωτικά αποτελέσματα για την επίδοση των υλοποιήσεων BCSC

4.3.3 Σύγκριση με τη Βιβλιοθήκη cuSPARSE της Nvidia

Αν και η σύγκριση της επίδοσης πραγματοποιήθηκε με όλες τις υλοποιήσεις του πυρήνα SpMM της βιβλιοθήκης cuSPARSE, στα επόμενα διαγράμματα φαίνεται μόνο η καλύτερη, η οποία προέρχεται από τη δομή αποθήκευσης CSR και μάλιστα θεωρεί τον πυκνό πίνακα B αντεστραμμένο. Στις επιδόσεις που παρουσιάζονται έχει προστεθεί και ο χρόνος για την αντιστροφή του ενός πυκνού πίνακα.

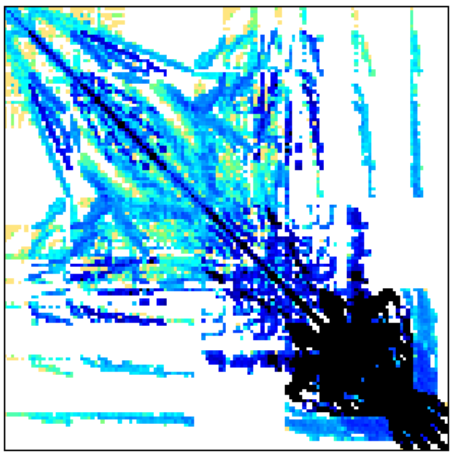


Σχήμα 4.9: Η επίδοση της καλύτερης υλοποίησης BCSC και της βιβλιοθήκης cuSPARSE

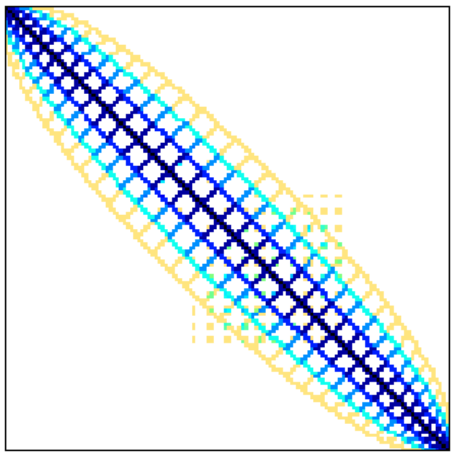
Στο σχήμα 4.14 φαίνονται συγκεντρωτικά τα αποτελέσματα, όμως και σε αυτή την ενότητα θα αναλύσουμε μόνο τους πίνακες του σχήματος 4.9, οι οποίοι είναι ίδιοι με αυτούς στο σχήμα 4.7.

Μπορούμε να χωρίσουμε τα αποτελέσματα σε τρεις κατηγορίες. Στην πρώτη είναι οι πίνακες στους οποίους το BCSC δίνει χειρότερη επίδοση για κάθε τιμή του *N* σε σύγκριση με τη βιβλιοθήκη cuSPARSE, στους οποίους ανήκουν οι πίνακες *offshore* (σχήμα 4.12) και *SiO*(σχήμα 4.13) και αποτελούν το ένα τέταρτο του συνόλου δεδομένων. Στη δεύτερη κατηγορία ανήκουν οι πίνακες που η βιβλιοθήκη cuSPARSE είναι καλύτερη για μικρές μόνο τιμές του N,

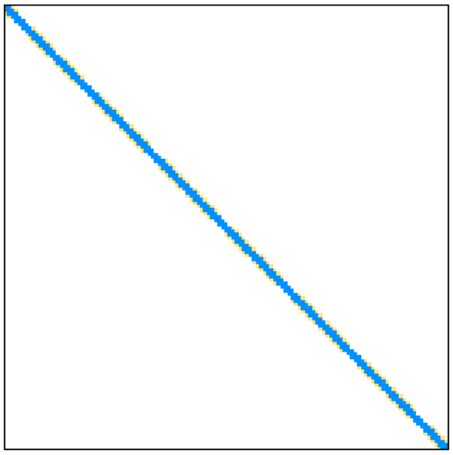
όπου $N \leq 64$ και τέλος στην τρίτη κατηγορία ανήκουν οι πίνακες στους οποίους η δομή BCSC είναι πάντα καλύτερη.



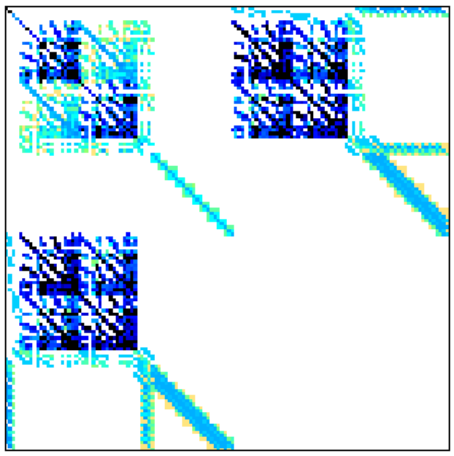
Σχήμα 4.10: Πίνακας *offshore*



Σχήμα 4.11: Πίνακας *SiO*



Σχήμα 4.12: Πίνακας *nemeth21*

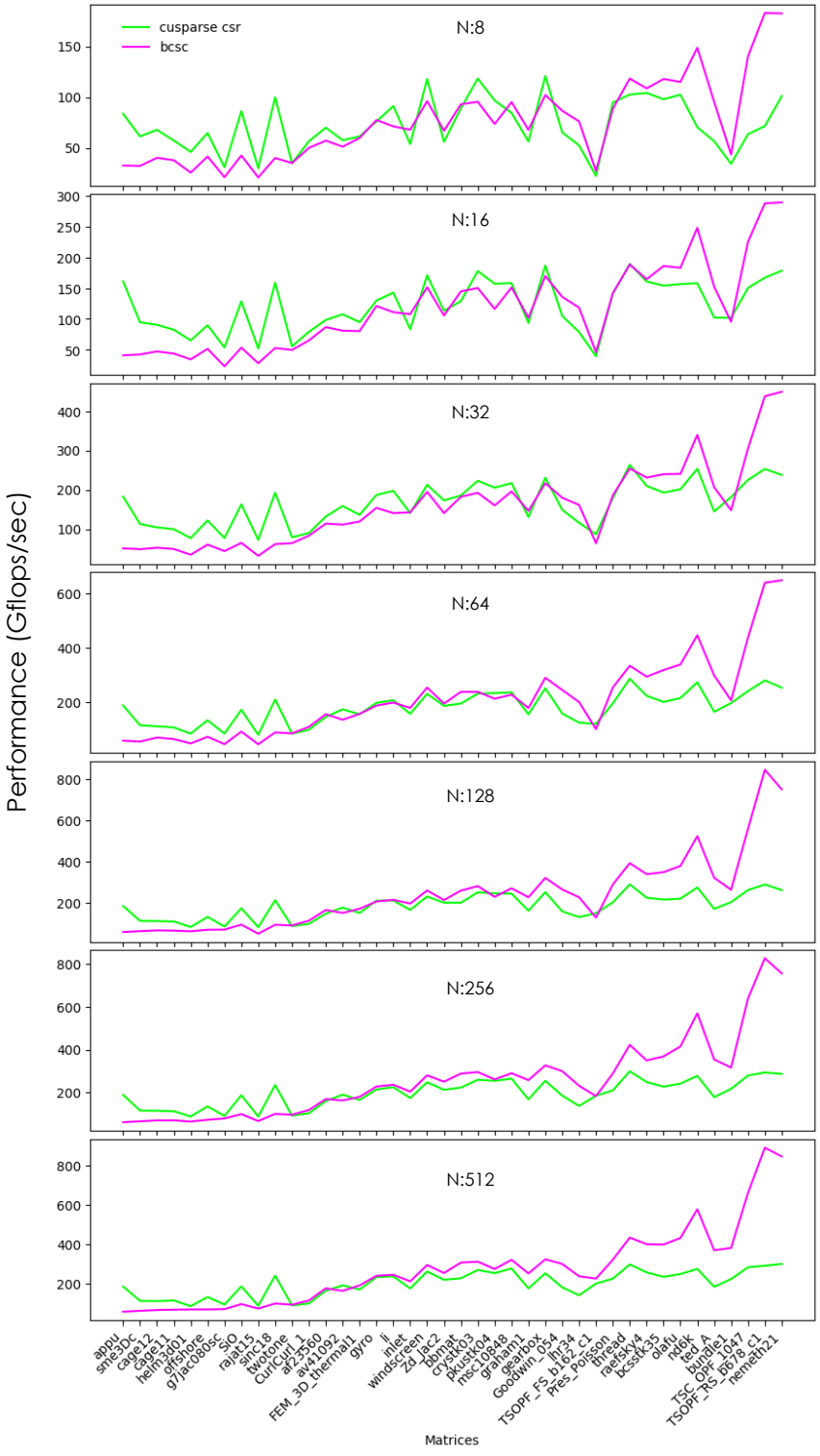


Σχήμα 4.13: Πίνακας *TSC_OPF_1047*

Με μια προσεκτική ματιά των πινάκων φαίνεται πως αυτοί στους οποίους η αναπαράσταση CSR της cuSPARSE δίνει καλύτερη επίδοση έχουν κάποια κοινά χαρακτηριστικά. Είναι πίνακες με πολύ μεγάλη τιμή αραιότητας, χαμηλή τιμή αριθμητικής έντασης και ο πυρήνας SpMM με είσοδο αυτούς βρίσκεται στην κατηγορία memory-bound ακόμα και για $N = 512$. Παρατηρούμε ότι και ο πυρήνας της βιβλιοθήκης cuSPARSE δεν καταφέρνει να φτάσει πολύ υψηλή επίδοση και για $N \geq 128$ η απόδοση μένει σχεδόν σταθερή. Σε αυτούς τους πίνακες το αποτύπωμα μνήμης της δομής BCSC

είναι σημαντικά μεγαλύτερο από αυτό του CSR, έως και 1.5 φορά. Αυτό μας δείχνει πως πιθανώς για αυτούς τους πίνακες δεν είναι καλή επιλογή η δομή αποθήκευσης BCSC.

Αντίθετα, οι πίνακες, όπου η δομή BCSC έχει πολύ καλή επίδοση, έχουν σχετικά μικρό αποτύπωμα μνήμης και για μεγάλα *N* ο πυρήνας SpMM με είσοδο αυτούς, είναι compute-bound. Επιπλέον, για τον πυρήνα tiling της δομής BCSC είναι συμφέρουσα η περίπτωση στην οποία τα μπλοκ που δημιουργούνται έχουν περίπου τον ίδιο αριθμό από μη-μηδενικά στοιχεία, καθώς τα thread blocks αναλαμβάνουν να πραγματοποιήσουν την ίδια ποσότητα υπολογισμών.



Σχήμα 4.14: Συγκεντρωτικά αποτελέσματα για την καλύτερη υλοποίηση BCSC και τη βιβλιοθήκη cuSPARSE

5

Συμπεράσματα και Μελλοντικές Επεκτάσεις

Στη διπλωματική αυτή πραγματοποιήσαμε ένα πρώτο βήμα προς την αναγνώριση των δυνατοτήτων μίας νέας δομής δεδομένων, τη δομή Blocked Compressed Sparse Row (BCSC), τόσο όσον αφορά την εξοικονόμηση χώρου κατά την αποθήκευση αραιών πινάκων, όσο και την επίδοση του υπολογιστικού πυρήνα Sparse Matrix - Matrix Multiplication (SpMM), χρησιμοποιώντας αυτή την αναπαράσταση. Η μέθοδος tiling που παρουσιάστηκε αποτελεί βασική τεχνική για την εκμετάλλευση της τοπικότητας των δεδομένων (data locality) και χρησιμοποιείται ευρέως σε εφαρμογές υψηλής απόδοσης. Η παράλληλη υλοποίηση, όπου εφαρμόστηκε αυτή η τεχνική, παρουσιάζει υψηλή επίδοση όταν τα μπλοκ γραμμών που σχηματίζονται περιέχουν πυκνές στήλες, ειδάλλως πραγματοποιούνται πολλοί περιττοί υπολογισμοί με μηδενικά στοιχεία. Αντίστοιχα η δεύτερη προσέγγιση που παρουσιάστηκε επιτυγχάνει μεγαλύτερη επίδοση όταν οι πίνακες είναι αρκετά αραιοί και υπάρχει μεγαλύτερη πιθανότητα να αποφευχθούν οι ατομικές λειτουργίες στην κοινή μνήμη.

Επομένως, γεννήθηκε η ιδέα για μια συνδυαστική προσέγγιση, στην οποία οι στήλες στο εκάστοτε μπλοκ γραμμών θα αναδιατάσσονται έτσι ώστε να σχηματιστούν δύο ομάδες, όπου η πρώτη θα περιέχει τις πιο πυκνές στήλες, ενώ η δεύτερη τις πιο αραιές. Αυτές οι διαχωρισμένες ομάδες στηλών μπορούν να δεχθούν διαφορετικό χειρισμό και επομένως να εφαρμοστεί στην κάθε μία η υλοποίηση που αποδείχτηκε πιο αποδοτική. Σημαντική παράμετρο αποτελεί η επιλογή του κατάλληλου κατωφλιού για την κατάταξη της εκάστοτε στήλης σε μια από τις δύο ομάδες. Ανα-

γκαιότητα αποτελεί ο πειραματισμός και η εξαγωγή κρίσιμων χαρακτηριστικών του αραιού πίνακα για την ανάπτυξη κατάλληλων μετρικών που θα οδηγούν στην κατάλληλη επιλογή κατωφλιού. Ακόμα, είναι δυνατή η επέκταση της συγκεκριμένης δομής δεδομένων για την αποθήκευση ομάδων γραμμών μεταβλητού μεγέθους, με σκοπό την περαιτέρω μείωση του αποτυπώματος μνήμης. Ταυτόχρονα, μια τέτοια παραλλαγή της αναπαράστασης επιτρέπει πιο ισορροπημένο διαμοιρασμό φορτίου, αποφυγή δημιουργίας ανενεργών νημάτων, καθώς και αποφυγή άσκοπης δέσμευσης των ήδη περιορισμένων πόρων.

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κατάλογος Αλγορίθμων

Βιβλιογραφία

H. Anzt, J. Dongarra, M. Kreutzer, G. Wellein και M. Köhler. Efficiency of general krylov methods on gpus – an experimental study. Στο *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 683–691, May 2016. doi: 10.1109/IPDPSW.2016.45.

Hartwig Anzt, Stanimire Tomov και Jack Dongarra. Accelerating the lobpcg method on gpus using a blocked sparse matrix vector product. Στο *Proceedings of the Symposium on High Performance Computing*, HPC '15, pages 75–82, San Diego, CA, USA, 2015. Society for Computer Simulation International. ISBN 978-1-5108-0101-1. URL http://dl.acm.org/citation.cfm?id=2872599.2872609.

Michael Bauer, Henry Cook και Brucek Khailany. Cudadma: Optimizing gpu memory bandwidth via warp specialization. page 12, 11 2011. doi: 10.1145/2063384.2063400.

Junli Gu, Yibing Liu, Yuan Gao και Maohua Zhu. Opencl caffe: Accelerating and enabling a cross platform machine learning framework. Στο *Proceedings of the 4th International Workshop on OpenCL*, IWOCL '16, pages 8:1–8:5, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4338-1. doi: 10.1145/2909437.2909443. URL http://doi.acm.org/10.1145/2909437.2909443.

Song Han, Huizi Mao και William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015. URL http://arxiv.org/abs/1510.00149.

Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003. ISBN 0898715342.

Ahmet Erdem Sariyüce, Erik Saule, Kamer Kaya και Ümit V. Çatalyürek. Regularizing graph centrality computations. *J. Parallel Distrib. Comput.*, 76:106–119, 2015.

R. P. Tewarson. Sparse Matrices. Academic Press, 1973. ISBN 0-124-11098-3.

Yaohung M. Tsai, Piotr Luszczek, Jakub Kurzak και Jack Dongarra. Performance-portable autotuning of opencl kernels for convolutional layers of deep neural networks. Στο *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*, MLHPC '16, pages 9–18, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-5090-3882-4. doi: 10.1109/MLHPC.2016.5. URL https://doi.org/10.1109/MLHPC.2016.5.

Samuel Williams, Andrew Waterman και David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, Απρίλιος 2009. ISSN 0001-0782. doi: 10.1145/1498765.1498785. URL http://doi.acm.org/10.1145/1498765.1498785.