ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Μελέτη Τεχνικών Συμπίεσης Αραιών Πινάκων για την Βελτιστοποίηση του Πολλαπλασιασμού Αραιών Πινάκων με Διάνυσμα (SpMxV).

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γκούντουβα Ι. Θεόδωρου

Επιβλέπων: Νεκτάριος Γ. Κοζύρης
Αν. Καθηγητής ΕΜΠ

Αθήνα, Μάιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Μελέτη Τεχνικών Συμπίεσης Αραιών Πινάκων για την Βελτιστοποίηση του Πολλαπλασιασμού Αραιών Πινάκων με Διάνυσμα (SpMxV).

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γκούντουβα Ι. Θεόδωρου

Επιβλέπων: Νεκτάριος Γ. Κοζύρης
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 23 Μαίου 2011

.....	.....	.....
Νεκτάριος Κοζύρης	Τιμολέων Σελλής	Νικόλαος Παπασπύρου
Αν. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2011

.....
Γκούντουβας Ι. Θεόδωρος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Περίληψη

Ο υπολογιστικός πυρήνας του Πολλαπλασιασμού Αραιού Πίνακα με Διάνυσμα επιταχύνεται ελάχιστα σε συστήματα κοινής μνήμης με πολλαπλές υπολογιστικές μονάδες. Προηγούμενη έρευνα έχει δείξει ότι μια αποτελεσματική στρατηγική για να βελτιωθεί η επίδοση του προβλήματος είναι η μείωση του όγκου δεδομένων που συμμετέχουν στους υπολογισμούς. Τα αποθηκευτικά σχήματα για τα μη-μηδενικά στοιχεία των αραιών πινάκων, που χρησιμοποιούνται για να μειώσουν τον αρχικό όγκο δεδομένων του πίνακα, εκμεταλλεύονται κανονικότητες (δομές που συναντιούνται συχνά) μέσα στον πίνακα. Το αποθηκευτικό σχήμα *Compressed Sparse eXtended* (*CSX*) αξιοποιεί αυτές τις δομές εφαρμόζοντας μια επιθετική τεχνική συμπιέσης ώστε να ελαττώσει το μέγεθος δεδομένων του αραιού πίνακα. Ο σκοπός της συγκεκριμένης διπλωματικής εργασίας είναι να ερευνήσει την επίδοση του CSX σε πολυπύρηνες αρχιτεκτονικές και να προτείνει ορισμένες βελτιώσεις στην τεχνική συμπίεσης.

Το κυρίως αντικείμενο της συγκεκριμένης διπλωματικής εργασίας είναι η αξιολόγηση της ευριστικής μεθόδου που χρησιμοποιείται στο CSX, η οποία επιλέγει κανονικότητες (κυρίως μπλοκ) που χρησιμοποιούνται στη συμπίεση, μέσα από εκτενείς μετρήσεις σε μια ποικιλία από πίνακες και πολυπύρηνες αρχιτεκτονικές. Πρώτα, υλοποιήθηκε μια επέκταση στον αρχικό κώδικα που επιτρέπει τον διαχωρισμό των μεγάλων μπλοκ σε μικρότερα, τα οποία εμφανίζονται πιο συχνά σε έναν πίνακα. Η νέα αυτή επέκταση επιφέρει καλύτερη συμπίεση, εφόσον περισσότερα μη-μηδενικά στοιχεία του πίνακα κωδικοποιούνται σε μπλοκ. Στη συνέχεια, ερευνήθηκε εξοντωτικά το πεδίο όλων των διαφορετικών κωδικοποιήσεων, ώστε να προσδιοριστούν οι παράμετροι που επηρεάζουν την επίδοση του SpM×V και να αξιολογηθεί η ευριστική μέθοδος που χρησιμοποιείται στο σχήμα CSX.

Η συνεισφορά αυτής της διπλωματικής είναι διπλή. (α′) Η επέκταση που έγινε στο πεδίο των μπλοκ οδήγησε σε βελτίωση της επίδοσης, αφού επιτυγχάνεται μεγαλύτερος βαθμός συμπίεσης. (β′) Η αξιολόγηση της ευριστικής προσέφερε μια επιπρόσθετη κατανόηση για τα χαρακτηριστικά που επηρεάζουν την επίδοση και θα βοηθήσει στην δημιουργία μιας πιο εξελιγμένης ευριστικής μεθόδου που θα προσαρμόζεται καλύτερα στην εκάστοτε αρχιτεκτονική.

Λέξεις Κλειδιά

Πολλαπλασιασμός Αραιών Πινάκων με Διάνυσμα, SpM×V, CSX, ευριστική μέθοδος, συμπίεση, αξιολόγηση, κανονικότητες, διαχωρισμός μπλοκ

Abstract

The Sparse Matrix Vector Multiplication (SpM×V) kernel scales poorly on shared memory systems with multiple processing units. Previous research has demonstrated that an effective strategy to improve the performance of the kernel is to drastically reduce the data volume involved in the computations. This can be achieved by exploiting substructures (patterns of non-zero elements occurring frequently) within the sparse matrix, in order to reduce the initial data volume of the matrix. The *Compressed Sparse eXtended* storage format (*CSX*) exploits these substructures by applying aggressive compression in the indexing metadata of the sparse matrix. The purpose of this diploma thesis is to study the performance of CSX on multicore architectures and propose some improvements on the compression technique.

The major subject of this diploma thesis is the evaluation of the heuristic method used in CSX, which selects the patterns (basically blocks) that will be used for compression, through extensive measurements on a variety of sparse matrices and multicore architectures. Firstly, an extension to the initial code has been implemented that allows the splitting of larger blocks into smaller ones that occur more frequently in the matrix. This new extension provides better compression results, since more non-zero elements of the matrix can be encoded into patterns. Secondly, we explore the search space of the different matrix encodings exhaustively in order to determine the parameters that affect performance of SpM×V and evaluate the heuristic used in the CSX format.

The contribution of this diploma thesis is two-fold: (α′) Our extension for splitting large blocks leads to performance improvements, since we achieve a higher level of compression, (β′) The evaluation of the CSX heuristic provided us with a better understanding of the performance characteristics of the different matrix encodings, which will help us to implement a more sophisticated heuristic that will better adapt to the underlying architecture.

Keywords

Sparse Matrix Vector Multiplication, SpM×V, CSX, heuristic method, compression, evaluation, patterns, split blocks

Ευχαριστίες

Κατ' αρχάς, ευχαριστώ πολύ τον επιβλέποντα καθηγητή Νεκτάριο Κοζύρη για τη συμπαράσταση του και τη βοήθεια του κατά τη διάρκεια της διπλωματικής εργασίας. Επίσης, οφείλω ένα μεγάλο ευχαριστώ στο Βασίλειο Καρακάση για τη πολύτιμη βοήθεια και τη μεγάλη συνεισφορά του στην εργασία, καθώς, και σε όλους τους συναδέλφους από το $Computer\ Science\ Laboratory$ ($cslab$) που με δέχθηκαν στο εργαστήριο και με αντιμετώπισαν σαν φίλο και μέλος της ομάδας τους. Ακόμα, θέλω να εκφράσω την ευγνωμοσύνη μου στους γονείς μου, Ιωάννη και Μαρίνα Γκούντουβα, στους θείους μου, Παναγιώτη κ Χρυσούλα Μιχοπούλου και Νικόλαο κ Παναγιώτα Φλούδα, στον αδελφό μου, Βασίλειο Γκούντουβα, και στην οικογένεια του, για την διαρκή συμπαράσταση, εμψύχωση και πίστη προς το πρόσωπο μου. Επιπρόσθετα, θα ήθελα να ευχαριστήσω την φίλη και συνάδελφο μου, Ελισάβετ Κοζύρη, για την προσοχή, το χρόνο και την ηθική υποστήριξη κατά τη διάρκεια της διπλωματικής εργασίας. Ξεχωριστές ευχαριστίες οφείλω να δώσω στη συγκάτοικο και αδελφή μου, Μαρία Γκούντουβα, για την υπομονή και τη στήριξη που μου προσέφερε τα τελευταία χρόνια. Τέλος, ευχαριστώ τον συνάδελφο και φίλο μου Ιωάννη Αγριόμαλλο για τη καθοδήγηση και τη βοήθεια του κατά τη διάρκεια των σπουδών μου, καθώς και τους υπόλοιπους στενούς μου φίλους που μου χάρισαν πολύ όμορφα φοιτητικά χρόνια.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Αραιοί Πίνακες και Αναπαράσταση τους

Οι αραιοί πίνακες αποτελούνται κυρίως από μηδενικά και περιέχουν πολύ λίγα μη-μηδενικά στοιχεία σε σχέση με το μέγεθός τους. Η έννοια της αραιότητας ενός πίνακα είναι χρήσιμη στον κλάδο της Συνδυαστικής και σε επιστημονικές περιοχές, όπως η Θεωρία Δικτύων, που έχουν μικρή πυκνότητα σημαντικών δεδομένων. Για παράδειγμα, ένα σύστημα (ή δίκτυο) που κάθε ένα από τα στοιχεία του συνδέεται με το επόμενο, μπορεί να περιγράφει από έναν αραιό πίνακα. Αντιθέτως ένα σύστημα (ή δίκτυο) που κάθε ένα στοιχείο του συνδέεται με όλα τα υπόλοιπα περιγράφεται από έναν πυκνό πίνακα. Οι μεγάλοι αραιοί πίνακες, οι οποίοι μας ενδιαφέρουν στην παρούσα εργασία, συναντιούνται σε εφαρμογές που έχουν σχέση με την επίλυση μερικών διαφορικών εξισώσεων [13]. Πιο συγκεκριμένα, η παρούσα διπλωματική εργασία ασχολείται με εφαρμογές που έχουν να κάνουν με Πολλαπλασιασμό Αραιού Πίνακα με Διάνυσμα (SpM×V). Ο Αλγόριθμος 1 δείχνει πως γίνεται ο πολλαπλασιασμός ενός πίνακα με ένα διάνυσμα.

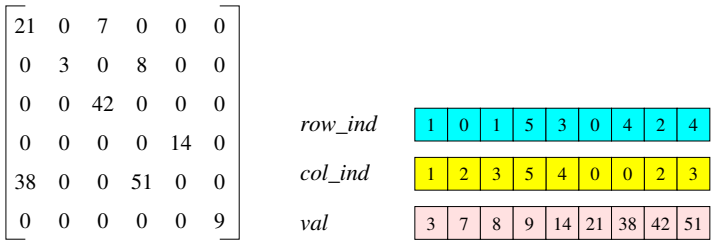
Αλγόριθμος 1: Πολλαπλασιασμός πίνακα με διάνυσμα.

Input: a, sparse array of size $N \times N$
Input: x, vector of size N
Output: y, vector of size N

```
for i ← 0 to N do
    for j ← 0 to N do
        y[i] ← y[i] + a[i][j] × x[j]
```

Για τους αραιούς πίνακες υπάρχουν αναπαραστάσεις που συμπιέζουν τον όγκο των δεδομένων χωρίς να χάνεται καμία σημαντική πληροφορία. Το πιο απλό και χαρακτηριστικό σχήμα αποθήκευσης αραιών πινάκων είναι ίσως το *Coordinate Format* (*COO*). Αυτή η αναπαράσταση του αραιού πίνακα αποτελείται από τρεις πίνακες που έχουν μέγεθος ίσο με τον αριθμό των μη-μηδενικών στοιχείων. Ο πρώτος πίνακας row_ind περιέχει τις γραμμές των μη-μηδενικών στοιχείων, ο δεύτερος col_ind πε-

ριέχει τις στήλες τους και ο τρίτος val την αριθμητική τιμή τους. Αναλυτικότερα, η αναπαράσταση του σχήματος COO φαίνεται στο Σχήμα 1.1. Το συγκεκριμένο σχήμα είναι ελκυστικό γιατί είναι απλό και βολικό στην αναπαράσταση, αλλά όσον αναφορά στην χρησιμοποίηση μνήμης είναι το λιγότερο αποδοτικό καθώς η συμπίεση που επιτυγχάνει είναι μικρότερη σε σχέση με αυτές των υπόλοιπων σχημάτων. Ο Αλγόριθμος 2 είναι η εκτέλεση του SpM×V για έναν αραιό πίνακα σε COO μορφή.



(α′) Αραιός πίνακας (β′) Αναπαράσταση πίνακα σε COO μορφή

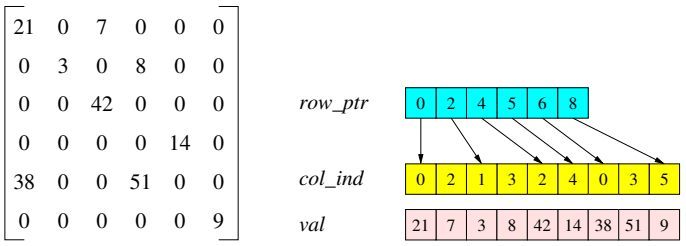
Σχήμα 1.1: Σχήμα αποθήκευσης COO.

Αλγόριθμος 2: SpM×V για σχήμα COO.

Input: (row_ind, col_ind, val), sparse array of size $N \times N$ with NNZ non-zero elements in COO format
Input: x, vector of size N
Output: y, vector of size N

for $i \leftarrow 0$ **to** NNZ **do**
  $y[row\_ind[i]] \leftarrow y[row\_ind[i]] + val[i] \times x[col\_ind[i]]$

Το πιο διαδεδομένο σχήμα αποθήκευσης αραιών πινάκων, που χρησιμοποιείται κατά κόρον, είναι το *Compressed Sparse Row* (*CSR*). Στην συγκεκριμένη αναπαράσταση υπάρχουν δύο πίνακες που έχουν μέγεθος ίσο με τον αριθμό των μη-μηδενικών στοιχείων. Ο πρώτος πίνακας col_ind περιέχει τον αριθμό της στήλης τους και ο δεύτερος val την αριθμητική τιμή τους. Επίσης, υπάρχει ένας πίνακας row_ptr, μεγέθους ίσου με τον αριθμό των γραμμών, που περιέχει δείκτες στον πίνακα col_ind, οι οποίοι σηματοδοτούν την έναρξη της εκάστοτε γραμμής. Αναλυτικότερα, η αναπαράσταση του σχήματος CSR φαίνεται στο Σχήμα 1.2. Παρόμοιο με το προηγούμενο σχήμα είναι και το *Compressed Sparse Column* (*CSC*), όπου ουσιαστικά εναλλάσσεται η αναπαράσταση των γραμμών με αυτήν των στηλών. Παρόλο που το σχήμα αυτό φαίνεται πολυπλοκότερο και απαιτεί την ένταξη των στοιχείων με συγκεκριμένη σειρά, επιτυγχάνει καλύτερη συμπίεση των αραιών πινάκων και επομένως θεωρείται καλύτερο από το COO. Ο Αλγόριθμος 3 είναι η εκτέλεση του SpM×V για έναν αραιό πίνακα σε CSR μορφή.



(α′) Αραιός πίνακας

(β′) Αναπαράσταση πίνακα σε CSR μορφή

Σχήμα 1.2: Σχήμα αποθήκευσης CSR.

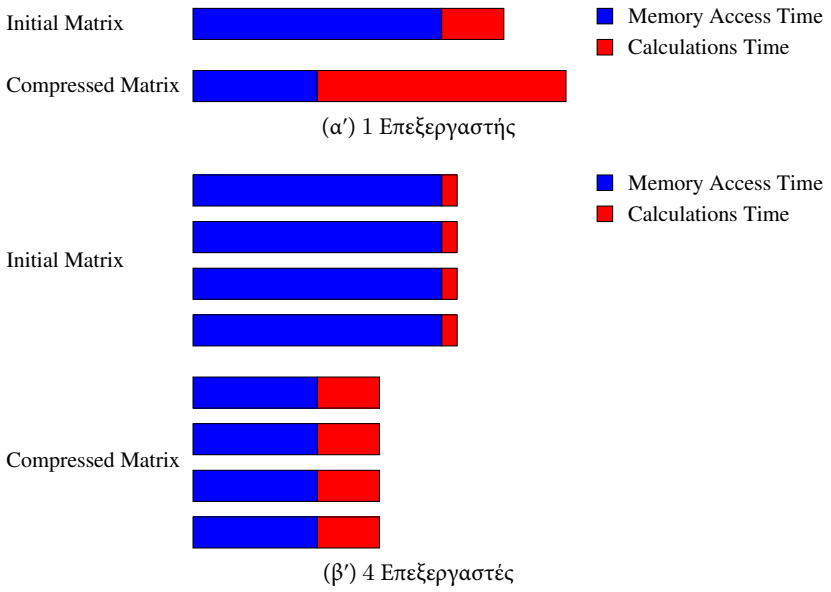
Αλγόριθμος 3: SpM×V για σχήμα CSR.

Input: (row_ptr, col_ind, val), sparse array of size $N \times N$ with NNZ non-zero elements in CSR format
Input: x, vector of size N
Output: y, vector of size N

for $i \leftarrow 0$ **to** N **do**
  for $j \leftarrow row\_ptr[i]$ **to** $row\_ptr[i+1]$ **do**
    $y[i] \leftarrow y[i] + val[j] \times x[col\_ind[j]]$

1.2 Περιγραφή Υπολογιστικού Πυρήνα

Ο Πολλαπλασιασμός Αραιού Πίνακα με Διάνυσμα (SpM×V), είναι ένα πρόβλημα το οποίο επιταχύνεται ελάχιστα σε πολυνηματικές αρχιτεκτονικές μοιραζόμενης μνήμης. Αυτό συμβαίνει διότι γίνονται $O(n^2)$ λειτουργίες σε $O(n^2)$ όγκο δεδομένων, το οποίο σημαίνει ότι το ποσοστό αναφορών στην μνήμη προς τον αριθμό των υπολογισμών (byte:flop ratio) είναι πολύ μεγάλο για το συγκεκριμένο πρόβλημα. Σίγουρα, σε μια πολυνηματική αρχιτεκτονική, οι υπολογισμοί του SpM×V καταμερίζονται σχεδόν ισότιμα μεταξύ των επεξεργαστών, καθώς δεν υπάρχουν εξαρτήσεις μεταξύ των νημάτων, και ο χρόνος των υπολογισμών μειώνεται δραστικά. Ωστόσο, από τη στιγμή που η μνήμη δεν μπορεί να εξυπηρετήσει όλα τα νήματα με τον επιθυμητό ρυθμό, δημιουργείται συμφόρηση η οποία δεν επιτρέπει την αποδοτική παραλληλοποίηση του προβλήματος. Για το SpM×V έχει βρεθεί ότι σημαντικό ρόλο στην βελτίωση της επίδοσης του προβλήματος παίζει η συμπίεση του όγκου των δεδομένων του αραιού πίνακα [3]. Πιο συγκεκριμένα, όσο μικρότερος είναι ο όγκος των δεδομένων τόσο λιγότερος είναι και ο χρόνος προσπέλασης της μνήμης, αφού οι αναφορές σε αυτήν είναι λιγότερες. Όμως, η συμπίεση συνήθως επιτυγχάνεται με πολύπλοκες αναπαραστάσεις του αραιού πίνακα που απαιτούν μεγαλύτερο φόρτο εργασίας για τους επεξεργαστές και επομένως περισσότερους υπολογισμούς. Έτσι παρόλο που μειώνεται ο χρόνος προσπέλασης της μνήμης, αυξάνεται ο χρόνος των υπολογισμών που κάνει ο επεξεργαστής. Ωστόσο σε μια συστοιχία επεξεργαστών, ο χρόνος των υπολογισμών μπορεί να μειωθεί δραστικά σε αναντιστοιχία με τον χρόνο προσπέλασης της μνήμης,



Σχήμα 1.3: Χρόνος εκτέλεσης SpM×V.

ο οποίος δεν παρουσιάζει σημαντικές διαφορές με την σειριακή εκτέλεση. Επομένως, μπορεί μια κωδικοποίηση που συμπιέζει τον αραιό πίνακα να μειώνει τον χρόνο εκτέλεσης του SpM×V όταν τρέχει σε συστοιχία επεξεργαστών, ακόμα και αν αυξάνει κατά πολύ των χρόνο των υπολογισμών, όπως φαίνεται και στο σχήμα 1.3. Στο συγκεκριμένο παράδειγμα, παρατηρείται ότι η σειριακή εκτέλεση του SpM×V, όταν ο αραιός πίνακας δεν έχει υποστεί συμπίεση και χρειάζεται περισσότερος χρόνος για την προσπέλαση μνήμης, είναι ταχύτερη από ότι η αντίστοιχη εκτέλεση του συμπιεσμένου πίνακα, όπου χρειάζονται επιπλέον υπολογισμοί για την αποκωδικοποίηση των συμπιεσμένων δεδομένων. Παρόλα αυτά, για την εκτέλεση σε πολλούς επεξεργαστές οι αριθμητικοί υπολογισμοί παραλληλοποιούνται σχεδόν ιδανικά, ενώ, αντιθέτως, ο χρόνος προσπέλασης της μνήμης παραμένει σταθερός. Αυτό έχει ως αποτέλεσμα στους τέσσερις επεξεργαστές η επίδοση του SpM×V να είναι καλύτερη για τον συμπιεσμένο πίνακα.

1.3 Οργάνωση του Κειμένου

Το κείμενο έχει οργανωθεί έτσι ώστε ο αναγνώστης να έρχεται σε επαφή με το αντικείμενο της εργασίας σταδιακά. Στο Κεφάλαιο 2 παρουσιάζονται μερικά σχήματα αποθήκευσης που συμπιέζουν τον όγκο των δεδομένων του προβλήματος και επιτυγχάνουν καλύτερη επίδοση από το σχήμα CSR. Στο Κεφάλαιο 2 παρουσιάζεται το σχήμα αποθήκευσης *Compressed Sparse eXtended* (*CSX*), το οποίο αποτελεί αντικείμενο της συγκεκριμένης διπλωματικής εργασίας. Στο Κεφάλαιο 4 εξηγείται η επέκταση του CSX, που αφορά στον διαχωρισμό των μπλοκ, και παρουσιάζονται τα αποτελέσματα που δείχνουν την βελτίωση στην επίδοση του SpM×V. Στο Κεφάλαιο 5 φαίνεται αναλυτικά η αναζήτηση κατά εύρος στον χώρο των κωδικοποιήσεων του

CSX που υλοποιήθηκε και παρατίθενται τα συμπεράσματα για την αξιολόγηση της ευριστικής μεθόδου συνοδευόμενα από τις αναγκαίες μετρήσεις. Τέλος, στο Κεφάλαιο 6 παρουσιάζονται οι μελλοντικές προτάσεις για το σχήμα αποθήκευσης CSX ώστε να βελτιωθεί περαιτέρω η επίδοση του.

Κεφάλαιο 2

Σχήματα Αποθήκευσης Αραιών Πινάκων

Σε αυτό το κεφάλαιο παρουσιάζονται σχήματα αποθήκευσης για αραιούς πίνακες, τα οποία είναι επεκτάσεις του σχήματος CSR. Σκοπός των σχημάτων αυτών είναι η περαιτέρω συμπίεση του όγκου δεδομένων και, συνεπώς, η βελτίωση της επίδοσης του υπολογιστικού πυρήνα SpM×V. Στις παρακάτω τεχνικές αποθήκευσης περιέχονται σχήματα που βασίζονται σε κανονικότητες, όπως και το CSX, και μειώνουν τον όγκο των δεδομένων που έχουν να κάνουν με την θέση των μη-μηδενικών στοιχείων του πίνακα. Τα σχήματα αυτά είναι προηγούμενες προσπάθειες για βελτίωση του SpM×V και αποτελούν μια εισαγωγή για το σχήμα αποθήκευσης CSX, το οποίο είναι ακόμα πιο προχωρημένο και περικλείει σχεδόν όλες τις κανονικότητες που καλύπτουν τα προηγούμενα σχήματα. Τέλος, παρουσιάζεται και ένα σχήμα αποθήκευσης που μειώνει τον όγκο των δεδομένων στο πεδίο των τιμών των μη-μηδενικών στοιχείων, για λόγους που θα γίνουν κατανοητοί αργότερα.

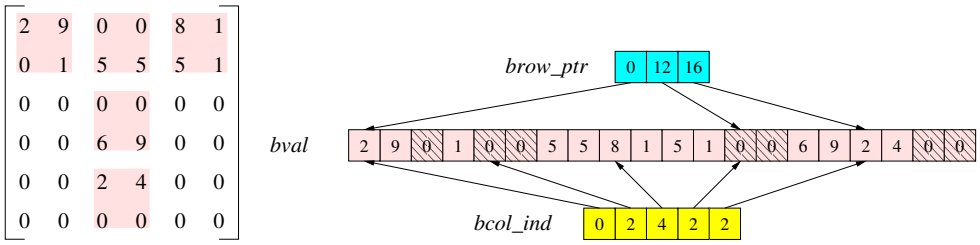
2.1 Σχήματα Αποθήκευσης που Χρησιμοποιούν Συμπλήρωση με Μηδενικά (*Padding*)

Τα σχήματα αποθήκευσης που ανήκουν στην συγκεκριμένη κατηγορία είναι επεκτάσεις του CSR και μειώνουν το μέγεθος του πεδίου col_ind ομαδοποιώντας τα στοιχεία του πίνακα. Έτσι χρειάζεται ένας δείκτης ανά ομάδα μη μηδενικών στοιχειών παρά ένας δείκτης ανά μη μηδενικό στοιχείο, όπως συμβαίνει στην περίπτωση του σχήματος CSR. Παρόλα αυτά, επειδή το μέγεθος κάθε ομάδας είναι σταθερό, όταν υπάρχουν μηδενικά σε ομάδες στοιχείων γίνεται συμπλήρωση με μηδενικά (*padding*), πράγμα που οδηγεί σε μια αύξηση του όγκου δεδομένων του πεδίου val.

2.1.1 Σχήμα Αποθήκευσης *BCSR*

Το *Blocked Compressed Sparse Row* (*BCSR*) είναι το πιο διαδεδομένο σχήμα αποθήκευσης που βασίζεται σε κανονικότητες και χρησιμοποιεί συμπλήρωση με μηδενικά. Το συγκεκριμένο σχήμα αποτελεί μια επέκταση του CSR, όπου αντί να απο-

θηκεύονται δείκτες σε μη-μηδενικά στοιχεία του πίνακα, αποθηκεύονται δείκτες σε μπλοκ $r \times c$ που περιέχουν τουλάχιστον μια μη-μηδενική τιμή. Τα μπλοκ στον BCSR είναι σταθερού μεγέθους και με αυστηρές προϋποθέσεις ευθυγράμμισης, με αποτέλεσμα να απαιτείται συμπλήρωση με μηδενικά για να σχηματιστούν πλήρη μπλοκ. Πιο συγκεκριμένα, για την αναπαράσταση του αραιού πίνακα χρειάζονται τρεις μονοδιάστατοι πίνακες, ο $bval$, ο $bcol\_ind$ και ο $brow\_ptr$. Ο $bval$ περιέχει τις τιμές των στοιχείων όλων των μπλοκ, ο $bcol\_ind$ περιέχει την στήλη του πρώτου στοιχείου κάθε μπλοκ και ο $brow\_ptr$ αποτελείται από δείκτες στον $bval$ που σηματοδοτούν την έναρξη της εκάστοτε γραμμής από μπλοκ. Το μεγαλύτερο πρόβλημα αυτής της υλοποίησης είναι ότι τα μπλοκ έχουν αυστηρές απαιτήσεις ευθυγράμμισης, (π.χ., η αρχή κάθε μπλοκ πρέπει να βρίσκεται σε ακέραια πολλαπλάσια των r και c) με αποτέλεσμα η συμπλήρωση που γίνεται με τα μηδενικά στον πίνακα των τιμών να μην είναι η καλύτερη δυνατή. Όπως φαίνεται και στο Σχήμα 2.1 (α′), ενώ τα στοιχεία που βρίσκονται στις θέσεις $(3,2)$,$(3,3)$,$(4,2)$ και $(4,3)$ σχηματίζουν ένα πλήρες μπλοκ 2×2, στην αναπαράσταση αυτή δεν μπορεί να ξεκινήσει μπλοκ από την θέση $(3,2)$ και τα συγκεκριμένα στοιχεία αναπαριστώνται με δυο μπλοκ συμπληρωμένα συνολικά με τέσσερα μηδενικά. Συνεπώς, εξαιτίας των πολλών επιπρόσθετων μηδενικών, το πεδίο των τιμών αυξάνεται σημαντικά και αρκετές φορές δεν επιτυγχάνεται ικανοποιητική συμπίεση του αραιού πίνακα. Παρόλα αυτά αποτελεί ένα σχήμα που εκμεταλλεύεται την ύπαρξη των μπλοκ στον αραιό πίνακα και επιτρέπει την χρήση υπολογιστικών βελτιστοποιήσεων (loop unrolling, vectorization) για την βελτίωση του χρόνου των αριθμητικών υπολογισμών [6]. Ο Αλγόριθμος 4 είναι η εκτέλεση του SpM×V για έναν αραιό πίνακα σε BCSR μορφή, χρησιμοποιώντας loop unrolling.



(α′) Αραιός πίνακας με μπλοκ μεγέθους 2×2

(β′) Αναπαράσταση πίνακα σε BCSR

Σχήμα 2.1: Σχήμα αποθήκευσης BCSR.

2.1.2 Σχήμα Αποθήκευσης *BCSD*

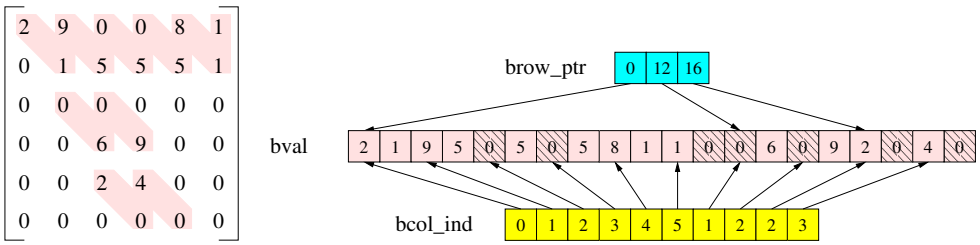
Ένα παρεμφερές σχήμα αποθήκευσης με το BCSR είναι το *Blocked Compressed Sparse Diagonal* (*BCSD*). Η διαφορά είναι ότι το συγκεκριμένο σχήμα εκμεταλλεύεται διαγώνιες κανονονικότητες μεγέθους d και όχι μπλοκ όπως το BCSR. Πιο συγκεκριμένα, για την αναπαράσταση του πίνακα χρησιμοποιούνται και πάλι τρεις πίνακες, ο $bval$, ο $bcol\_ind$ και ο $brow\_ptr$. Ο $bval$ περιέχει τις τιμές όλων των στοιχείων των διαγωνίων, ο $bcol\_ind$ περιέχει τις στήλες των πρώτων στοιχείων των διαγωνίων και ο $brow\_ptr$ αποτελείται από δείκτες στον $bval$ που σηματοδοτούν την έναρξη

Αλγόριθμος 4: SpM×V για σχήμα BCSR με μπλοκς μεγέθους $r \times c$.

Input: $(brow\_ptr, bcol\_ind, bval)$, sparse array of size $N \times N$ with NNZ non-zero elements in BCSR format
Input: x, vector of size N
Output: y, vector of size N

for $i \leftarrow 0$ **to** N/r **do**
  for $j \leftarrow brow\_ptr[i]/(r \times c)$ **to** $brow\_ptr[i+1]/(r \times c)$ **do**
    for $rr \leftarrow 0$ **to** r **do**
      for $cc \leftarrow 0$ **to** c **do**
        $y[i+rr] \leftarrow y[i+rr] + val[j \times r \times c + rr \times c + cc] \times x[bcol\_ind[j]+cc]$

της εκάστοτε γραμμής. Όπως και το BCSR, το BCSD χρησιμοποιεί συμπλήρωση με μηδενικά στις διαγώνιους που έχουν μηδενικά στοιχεία για λόγους συμβατότητας. Οπότε στις περισσότερες περιπτώσεις δεν επιτυγχάνεται ικανοποιητική συμπίεση του αραιού πίνακα. Επιπρόσθετα, δεν μπορούν να χρησιμοποιηθούν οι βελτιστοποιήσεις που αφορούν μπλοκ, όπως στην περίπτωση του BCSR, αλλά μπορεί να εφαρμοστεί vectorization για τις διαγώνιες κανονικότητες που προκύπτουν. Ο Αλγόριθμος 5 είναι η εκτέλεση του SpM×V για έναν αραιό πίνακα σε BCSD μορφή.



(α′) Αραιός πίνακας με διαγώνιες μεγέθους 2

(β′) Αναπαράσταση πίνακα σε BCSD

Σχήμα 2.2: Σχήμα αποθήκευσης BCSD.

2.1.3 Σχήματα Αποθήκευσης *BCSR-DE* και *BCSD-DE*

Το *BCSR-Decomposed* (*BCSR-DEC*) και το *BCSD-Decomposed* (*BCSD-DEC*) αποτελούν εναλλακτικές υλοποιήσεις για το BCSR και το BCSD αντίστοιχα που διορθώνουν το προβλήματα με τα επιπρόσθετα μηδενικά. Η ιδέα είναι ότι χρησιμοποιούνται k αναπαραστάσεις για τον πίνακα αντί για μία. Π.χ. στο σχήμα αποθήκευσης BCSR-DEC για $k = 2$ έχουμε μια αναπαράσταση για τα πλήρη μπλοκ 2×2 (*no-padding*) και τα υπόλοιπα μη-μηδενικά στοιχεία που δεν ανήκουν σε κάποιο πλήρες μπλοκ 2×2 αποθηκεύονται σε CSR μορφή. Με αυτό τον τρόπο επιτυγχάνεται καλύτερη συμπίεση του όγκου των δεδομένων του πίνακα καθώς δεν αυξάνεται το πεδίο των τιμών με επιπρόσθετα στοιχεία. Όμως η συνολική αναπαράσταση γίνεται πιο περίπλοκη εφόσον

Αλγόριθμος 5: SpM×V για σχήμα BCSD με διαγώνιες μεγέθους d.

Input: $(brow\_ptr, bcol\_ind, bval)$, sparse array of size $N \times N$ in BCSD format
Input: x, vector of size N
Output: y, vector of size N

```
for i ← 0 to N/d do
    for j ← brow_ptr[i]/d to brow_ptr[i+1]/d do
        for dc ← 0 to d do
            y[i+dc] ← y[i+dc] + bval[j×d+dc] * x[bcol_ind[j]+dc]
```

έχουμε παραπάνω από μια επιμέρους αναπαραστάσεις. Αναλυτικότερα, δεν υπάρχει ούτε χρονική ούτε χωρική τοπικότητα δεδομένων μεταξύ των k διαφορετικών αναπαραστάσεων και χρειάζονται περισσότεροι υπολογισμοί για την εξαγωγή του τελικού αποτελέσματος από τα επιμέρους (πρόσθεση των επιμέρους αποτελεσμάτων).

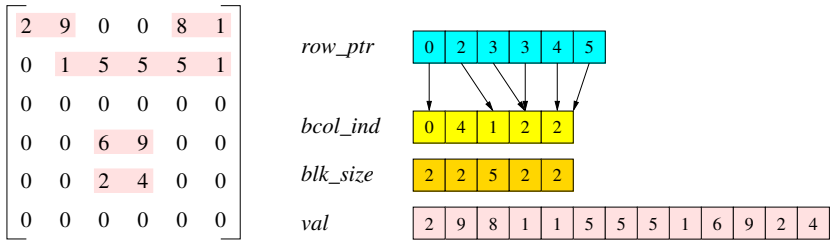
2.2 Σχήματα Αποθήκευσης με Επιπλέον Πεδία

Τα σχήματα αποθήκευσης που ανήκουν στην συγκεκριμένη κατηγορία είναι επεκτάσεις του CSR και μειώνουν το μέγεθος του πεδίου col_ind ομαδοποιώντας όσα περισσότερα στοιχεία μπορούν αποφεύγοντας την συμπλήρωση του πίνακα με μηδενικά (padding). Παρόλ'αυτά χρησιμοποιούν επιπλέον πεδία για την περιγραφή του τύπου της κάθε ομάδας και συνεπώς, υπάρχει ένας επιπλέον όγκος δεδομένων για την αναπαράσταση του αραιού πίνακα.

2.2.1 Σχήμα Αποθήκευσης *1D-VBL*

Το 1-dimensional *Variable Block Length* (*1D-VBL*) είναι ένα σχήμα αποθήκευσης το οποίο εκμεταλλεύεται τις οριζόντιες κανονικότητες [10]. Το σχήμα αυτό βρίσκει τα διαδοχικά μη-μηδενικά στοιχεία του αραιού πίνακα και τα αποθηκεύει ως πλήρη μπλοκ. Πιο συγκεκριμένα, για την αναπαράσταση χρησιμοποιούνται τέσσερις πίνακες, ο val, ο row_ptr, ο $bcol\_ind$ και ο blk_size. Ο val περιέχει τις τιμές των μη-μηδενικών στοιχείων του αραιού πίνακα, ο row_ptr αποτελείται από δείκτες στον $bcol\_ind$ που σηματοδοτούν την έναρξη της εκάστοτε γραμμής, ο $bcol\_ind$ που περιέχει τους αριθμούς των στηλών των μπλοκ και ο blk_size που περιέχει τα μεγέθη των μπλοκ. Παρόλο που δεν υπάρχει αύξηση του πεδίου τιμών όπως στην περίπτωση των BCSR και BCSD, υπάρχει το επιπλέον πεδίο blk_size που αυξάνει τον συνολικό όγκο δεδομένων. Ωστόσο, το σχήμα αυτό φαίνεται να πετυχαίνει ικανοποιητική συμπίεση του αραιού πίνακα σε σχέση με τα προηγούμενα σχήματα αποθήκευσης. Όμως, για την εύρεση της διεύθυνσης ενός μη-μηδενικού στοιχείου χρειάζονται περισσότεροι αριθμητικοί υπολογισμοί, όπως φαίνεται και στον Αλγόριθμο 6, και ο χρόνος υπολογισμών του SpM×V αυξάνεται σημαντικά. Αυτό έχει ως αποτέλεσμα να μειώνεται η επίδοση του SpM×V στην σειριακή εκτέλεση όπου οι αριθμητικοί υπολογισμοί παίζουν σημαντικό ρόλο, αλλά να αυξάνεται κατά πολύ η επίδοση της παράλληλης

εκτέλεσης του, όπου ο όγκος δεδομένων του αραιού πίνακα είναι καταλυτικός παράγοντας [5].



(α′) Αραιός πίνακας με οριζόντιες κανονικότητες

(β′) Αναπαράσταση πίνακα σε 1D-VBL

Σχήμα 2.3: Σχήμα αποθήκευσης 1D-VBL.

Αλγόριθμος 6: SpM×V για σχήμα 1D-VBL.

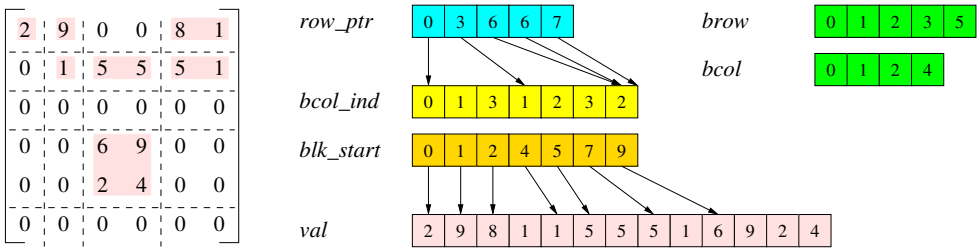
Input: $(row\_ptr, bcol\_ind, blc\_size, val)$, sparse array of size $N \times N$ in 1D-VBL format
Input: x, vector of size N
Output: y, vector of size N

$vc \leftarrow 0$
for $i \leftarrow 0$ **to** N **do**
  for $j \leftarrow row\_ptr[i]$ **to** $row\_ptr[i+1]$ **do**
    for $k \leftarrow 0$ **to** blc_size **do**
      $y[i] \leftarrow y[i] + val[vc] * x[bcol\_ind[j]+k]$
      $vc \leftarrow vc + 1$

2.2.2 Σχήμα Αποθήκευσης *VBR*

Το *Variable Block Row* (*VBR*) εκμεταλλεύεται τα μπλοκ που εμφανίζονται σε έναν αραιό πίνακα [11]. Το συγκεκριμένο σχήμα αποθήκευσης δεν αυξάνει το πεδίο τιμών και χωρίζει τον πίνακα σε μπλοκ έτσι ώστε τα μη-μηδενικά στοιχεία να ανήκουν σε ένα πλήρες συμπληρωμένο μπλοκ. Πιο συγκεκριμένα, για την αναπαράσταση του αραιού πίνακα σε VBR μορφή χρησιμοποιούνται δύο επιπλέον πίνακες ο $brow$ και ο $bcol$ που αναπαριστούν τις γραμμές-μπλοκ και στήλες-μπλοκ αντίστοιχα που έχουν διαμορφωθεί μετά τον χωρισμό του πίνακα σε μπλοκ. Ακόμα χρησιμοποιούνται οι τέσσερις πίνακες, ο val, ο $brow\_ptr$, ο $bcol\_ind$ και ο blk_start. Ο val περιέχει τις τιμές των μη-μηδενικών στοιχείων του αραιού πίνακα, ο $brow\_ptr$ αποτελείται από δείκτες στον $bcol\_ind$ που σηματοδοτούν την έναρξη της εκάστοτε γραμμής-μπλοκ, ο $bcol\_ind$ που περιέχει τους αριθμούς των στηλών-μπλοκ και ο blk_start που περιέχει δείκτες προς τον val οι οποίοι δείχνουν στο πρώτο στοιχείο των μπλοκ. Η μέθοδος αυτή, παρόλο που σε ορισμένους πίνακες εκμεταλλεύεται σημαντικά την ύπαρξη

δισδιάστατων μπλοκ, έχει τρία επιπρόσθετα πεδία, όπως φαίνεται και στο Σχήμα 2.4 (β′), σε σχέση με την αναπαράσταση του CSR τα οποία αυξάνουν αρκετά τον όγκο δεδομένων του αραιού πίνακα. Επίσης, χρειάζονται ακόμα περισσότεροι αριθμητικοί υπολογισμοί, όπως φαίνεται και στον Αλγόριθμο 7, από το 1D-VBL για την εύρεση της διεύθυνσης των μη-μηδενικών στοιχείων του αραιού πίνακα που έχουν ως αποτέλεσμα την περαιτέρω μείωση της επίδοσης του SpM×V για το σχήμα αποθήκευσης VBR.



(α′) Αραιός πίνακας χωρισμένος σε πλήρη μπλοκ

(β′) Αναπαράσταση πίνακα σε VBR

Σχήμα 2.4: Σχήμα αποθήκευσης VBR.

Αλγόριθμος 7: SpM×V για σχήμα VBR.

Input: $(brow, bcol, row\_ptr, bcol\_ind, blc\_start, val)$, sparse array of size $N \times N$ in VBR format
Input: x, vector of size N
Output: y, vector of size N

```
for bi ← 0 to BN do
    for bj ← row_ptr[bi] to row_ptr[bi+1] − 1 do
        vc ← 0
        for i ← brow[bi] to brow[bi+1] do
            for j ← brow[bj] to brow[bj+1] do
                y[i] ← y[i] + val[blc_start[bj]+vc] ∗ x[j]
                vc ← vc + 1
```

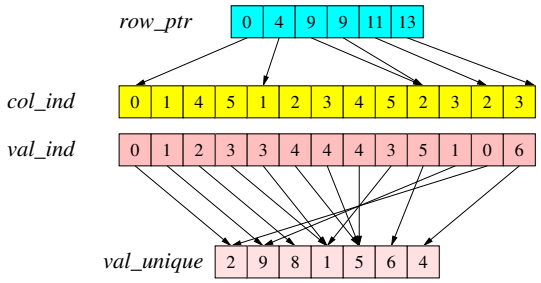
2.3 Σχήματα Αποθήκευσης που Μειώνουν τον Όγκο Δεδομένων του Πεδίου Τιμών

Τα σχήματα αποθήκευσης αυτής της κατηγορία είναι επεκτάσεις του CSR, που αφήνουν αναλλοίωτα τα πεδία row_ptr και col_ind και προσπαθούν να ομαδοποιήσουν τις αριθμητικές τιμές (val) των μη-μηδενικών στοιχείων. Ένα τέτοιο εγχείρημα

είναι σημαντικό γιατί συνήθως οι τιμές αυτού του πεδίου καταλαμβάνουν τον μεγαλύτερο όγκο δεδομένων και πιθανή συμπίεση τους θα επιφέρει πολύ σημαντική βελτίωση στην επίδοση του SpM×V. Όμως, μια τέτοια συμπίεση είναι δύσκολο να γίνει καθώς οι αριθμητικές τιμές των μη-μηδενικών στοιχείων είναι πολύ πιο δύσκολο να ομαδοποιηθούν σε σχέση με τις διευθύνσεις.

2.3.1 Σχήμα Αποθήκευσης *CSR-VI*

Το *Compressed Sparse Row - Value Indexed* (*CSR-VI*) είναι ένα σχήμα αποθήκευσης το οποίο συμπιέζει το πεδίο τιμών των μη-μηδενικών στοιχείων του αραιού πίνακα αφήνοντας αναλλοίωτα τα υπόλοιπα πεδία [8]. Το σχήμα αποθηκεύει τις διάφορες τιμές που συναντιούνται στον πίνακα μόνο μια φόρα κερδίζοντας επιπλέον χώρο κάθε φορά που μια τιμή επαναλαμβάνεται. Πιο συγκεκριμένα, χρησιμοποιούνται οι row_ptr και col_ind, όπως ακριβώς στο σχήμα αποθήκευσης CSR, και αντικαθιστάται ο πίνακας val από δύο καινούργιους, τον val_unique και τον val_ind. Ο πρώτος πίνακας αποθηκεύει τις διαφορετικές τιμές των μη-μηδενικών στοιχείων και ο val_ind ουσιαστικά κρατάει δείκτες προς τον val_unique που σηματοδοτούν την τιμή των εκάστοτε στοιχείων(δες Σχήμα 2.5). Στην συγκεκριμένη αναπαράσταση του αραιού πίνακα γίνεται σημαντική συμπίεση υπό την προϋπόθεση ότι ο λόγος των πραγματικών προς των μοναδικών τιμών θα είναι αρκετά μεγάλος. Ωστόσο είναι ξεχωριστή αναπαράσταση διότι κατορθώνει και μειώνει τον όγκο του πεδίου τιμών και μπορεί να συνδυαστεί με τα σχήματα αποθήκευσης που μειώνουν μόνο τον όγκο των δεδομένων που αφορούν την θέση των μη-μηδενικών στοιχείων στον αραιό πίνακα. Τέλος, δεν σημειώνεται μεγάλη αύξηση του χρόνου υπολογισμών αφού η μόνη διαφορά με το CSR είναι ότι υπάρχει μια παραπάνω διευθυνσιοδότηση, όπως φαίνεται και στον Αλγόριθμο 8.



Σχήμα 2.5: Σχήμα αποθήκευσης CSR-VI.

Αλγόριθμος 8: SpM×V για σχήμα CSR-VI.

Input: (row_ptr, col_ind, val_ind, val_unique), sparse array of size $N \times N$ in CSR-VI format
Input: x, vector of size N
Output: y, vector of size N

for $i \leftarrow 0$ **to** N **do**
  for $j \leftarrow row\_ptr[i]$ **to** $row\_ptr[i+1]$ **do**
    $y[i] \leftarrow y[i] + val\_unique[val\_ind[j]] * x[col\_ind[j]]$

Κεφάλαιο 3

Σχήματα Αποθήκευσης Βασισμένα σε Τεχνικές Συμπίεσης Δεδομένων

Στο κεφάλαιο αυτό, αρχικά παρουσιάζεται το σχήμα αποθήκευσης *Compressed Sparse Row with Delta Units* (*CSR-DU*), το οποίο εφαρμόζει μια τεχνική συμπίεσης που δεν βασίζεται στην αξιοποίηση κανονικοτήτων, αλλά στην μείωση του μεγέθους των δεικτών του πεδίου διευθυνσιοδότησης col_ind. Έπειτα παρουσιάζεται το σχήμα *Compressed Sparse eXtended* (*CSX*), το οποίο αποτελεί επέκταση του CSR-DU. Ο στόχος του συγκεκριμένου σχήματος είναι να εφαρμόσει μια επιθετική τεχνική συμπίεσης βασισμένη σε κανονικότητες και να επιφέρει σημαντική μείωση του όγκου δεδομένων του πίνακα. Το αποτέλεσμα της συμπίεσης αυτής είναι να αυξηθεί η επίδοση του υπολογιστικού προβλήματος SpM×V. Πιο συγκεκριμένα, οι κανονικότητες που αξιοποιεί το CSX είναι μονοδιάστατες (οριζόντιες, κατακόρυφες, διαγώνιες, αντι-διαγώνιες), αλλά και δισδιάστατες (μπλοκ). Η τεχνική συμπίεσης του CSX βασίζεται σε τέσσερα βήματα, (α′) στον διαχωρισμό του SpM×V σε νήματα, (β′) στην εύρεση στατιστικών στοιχείων για τις κανονικότητες του σχήματος CSX και στην επιλογή των κανονικοτήτων που θα επιλεχθούν ως βάση για την συμπίεση του αραιού πίνακα μέσω της ευριστικής μεθόδου του CSX, (γ′) στην συμπλήρωση των πεδίων του σχήματος αποθήκευσης CSX και (δ′) στην δημιουργία κώδικα με στόχο την εκτέλεση του SpM×V. Τα παραπάνω βήματα παρουσιάζονται αναλυτικά στις παρακάτω παραγράφους.

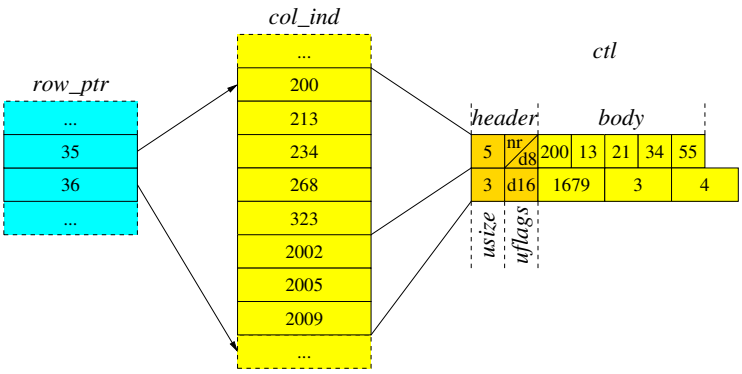
3.1 Αποθηκευτικό Σχήμα *CSR με Delta Units* (*CSR-DU*)

Για να εξηγηθεί το σχήμα αποθήκευσης CSX πρέπει πρώτα να γίνει μια αναφορά στο σχήμα CSR-DU πάνω στον οποίο βασίζεται. Το CSR-DU έχει μια τελείως διαφορετική λογική από τα προηγούμενα σχήματα αποθήκευσης που παρουσιάστηκαν στο Κεφάλαιο 2. Ο στόχος του είναι να μειώσει τον όγκο δεδομένων του πεδίου col_ind μειώνοντας τον όγκο των δεδομένων που κρατιέται για κάθε στοιχείο και όχι τον αριθμό των στοιχείων του, όπως κάνουν πολλά σχήματα αποθήκευσης. Πιο συγκεκριμένα, αντί να αποθηκεύεται σε κάθε θέση του col_ind ο αριθμός της στήλης του εκάστοτε μη-μηδενικού στοιχείου του αραιού πίνακα, αποθηκεύεται μόνο η απόσταση

από το προηγούμενο μη-μηδενικό στοιχείο της γραμμής ή αν το στοιχείο είναι το πρώτο στη γραμμή αποθηκεύεται η απόσταση από την αρχή της γραμμής (απόσταση-Δ). Με αυτό τον τρόπο χρειάζονται πιο λίγα bit για την αναπαράσταση ενός στοιχείου, αφού η απόσταση-Δ είναι κατά κανόνα αρκετά μικρότερη από τον αριθμό της στήλης.

Στη συνέχεια, διαδοχικά στοιχεία σχηματίζουν ομάδες ($d8, d16, d32$ και $d64$) ανάλογα με το μέγεθος της αναπαράστασης τους στο δυαδικό σύστημα ($8, 16, 32$ και 64 bit αντίστοιχα). Για παράδειγμα, αν n διαδοχικά στοιχεία έχουν τιμές $d_i < 2^8$, τότε σχηματίζουν ομάδα $d8$ μεγέθους n της οποίας το κάθε στοιχείο καταλαμβάνει μέγεθος 8 bit. Αντιθέτως αν ισχύει ότι $2^{16} \leq d_i < 2^{32}$, τότε σχηματίζουν ομάδα $d32$ μεγέθους n της οποίας το κάθε στοιχείο καταλαμβάνει μέγεθος 32 bit. Με αυτό τον τρόπο επιτυγχάνεται μια σημαντική συμπίεση σε σχέση με το σχήμα CSR, εφόσον μειώνεται σημαντικά ο όγκος δεδομένων του πεδίου col_ind.

Αναλυτικότερα, η υλοποίηση του σχήματος αποθήκευσης CSR-DU [7, 8] αντικαθιστά το col_ind με έναν μονοδιάστατο πίνακα ctl για κάθε ομάδα που σχηματίζεται. Ο πίνακας ctl αποτελείται από δύο μέρη, την επικεφαλίδα ($header$) και το σώμα ($body$). Η επικεφαλίδα αποτελείται από 2 πεδία, το $usize$ και το $uflags$. Το $usize$ είναι ένα πεδίο με μέγεθος ένα byte που περιέχει τον αριθμό των στοιχείων μιας ομάδας, το οποίο αυτόματα σημαίνει ότι τα στοιχεία μια ομάδας δεν μπορούν να ξεπερνούν τα $2^8 = 256$. Το $uflags$ έχει μέγεθος ένα byte και περιέχει τον τύπο της ομάδας ($d8, d16$, $d32$ ή $d64$) καθώς και μια σημαία (bit) που ενεργοποιείται στην περίπτωση νέας σειράς. Το σώμα έχει μεταβλητό μέγεθος ανάλογα με το μέγεθος και τον τύπο της ομάδας και περιέχει τις τιμές των αποστάσεων-Δ των στοιχείων της ομάδας. Για βελτιστοποίηση του σχήματος αρκετές φορές χρησιμοποιείται και ένα πεδίο $ujmp$ μεταβλητού μεγέθους στο τέλος της ομάδας το οποίο είναι η απόσταση-Δ της επόμενης ομάδας από αυτήν που εξετάζεται.



Σχήμα 3.1: Σχήμα αποθήκευσης CSR-DU, όπου μια σειρά χωρίζεται σε δύο ομάδες.

Στο Σχήμα 3.1 φαίνεται ένα παράδειγμα λειτουργίας του σχήματος αποθήκευσης CSR-DU. Αναλυτικότερα, φαίνεται στο σχήμα ότι τα πρώτα πέντε στοιχεία σχηματίζουν μια ομάδα $d8$, όπου ενεργοποιείται και η σημαία για νέα σειρά και τα επόμενα τρία σχηματίζουν ομάδα $d16$. Αν υποθέσουμε ότι το κάθε στοιχείο του πεδίου col_ind με την αναπαράσταση CSR έχει μέγεθος 32 bit, τότε το συνολικό μέγεθος του πεδίου είναι $8 \times 32 = 256bit$, ενώ με την αναπαράσταση CSR-DU το αντίστοιχο μέγεθος

είναι $16+5\times 8+16+3\times 16=120bit$, οδηγώντας σε μεγαλύτερη του 50% μείωσης του.

3.2 Κανονικότητες του CSX

Το σχήμα αποθήκευσης CSX βασίζεται πάνω σε συγκεκριμένες κανονικότητες που συναντιούνται συχνά σε αραιούς πίνακες. Έτσι, καταφέρνει και δημιουργεί ομάδες στοιχείων που ανήκουν σε κάποια κανονικότητα συμπιέζοντας περαιτέρω το πεδίο col_ind, μειώνοντας αυτή τη φορά το πλήθος των στοιχείων του και όχι μόνο το μέγεθος τους όπως με την χρήση των αποστάσεων-Δ. Στην συνέχεια παρουσιάζονται αναλυτικά όλες οι κανονικότητες που ανιχνεύει το CSX σε αραιούς πίνακες.

3.2.1 Οριζόντιες Κανονικότητες

Οι οριζόντιες κανονικότητες, που ανιχνεύονται από το CSX είναι παρόμοιες με εκείνες του σχήματος αποθήκευσης 1D-VBL (Κεφάλαιο 2.2.1 σελ.25). Αρχικά, γίνεται η αντικατάσταση στο πεδίο col_ind του CSR με τις αποστάσεις-Δ έναντι των αριθμών των στηλών. Έτσι οποιαδήποτε n διαδοχικά στοιχεία του col_ind έχουν ίδια απόσταση-Δ αποτελούν οριζόντια κανονικότητα και μπορούν να σχηματίσουν ομάδα με στόχο την περαιτέρω συμπίεση του πίνακα. Σε αντίθεση με το σχήμα αποθήκευσης 1D-VBL, που εκμεταλλεύεται ακολουθίες στοιχείων του col_ind με την μορφή $(a,\ a+1,\ a+2,\ ...)$, το CSX γενικεύει ακόμα περισσότερο την μορφή αυτή σε $(a,\ a+d,\ a+2d,\ \ldots)$ με αποτέλεσμα να εκμεταλλεύεται όχι μόνο διαδοχικά μη-μηδενικά στοιχεία αλλά και στοιχεία που έχουν μεγαλύτερη απόσταση μεταξύ τους.

indices	1	10	11	12	13	14	21	41	61	81	101	107
deltas	1	9	1	1	1	1	7	20	20	20	20	6

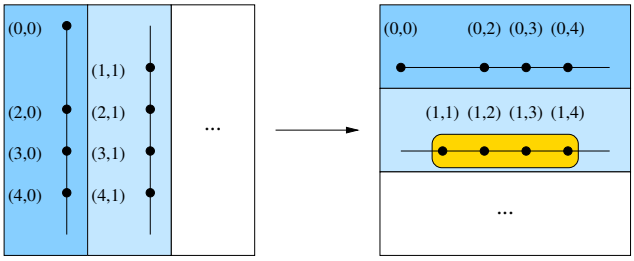
Σχήμα 3.2: Ανίχνευση οριζόντιων κανονικοτήτων.

Στο Σχήμα 3.2 φαίνεται ένα παράδειγμα λειτουργίας της ανίχνευσης των οριζόντιων κανονικοτήτων του CSX. Ενδιαφέρον έχει το γεγονός ότι και οι δύο οριζόντιες κανονικότητες στο συγκεκριμένο σχήμα έχουν τέσσερα στοιχεία. Κάτω από τον συγκεκριμένο αριθμό η κωδικοποίηση σε ομάδα δεν έχει πρακτικό νόημα αφού δεν επιτυγχάνεται σημαντική συμπίεση του όγκου δεδομένων του πίνακα. Επίσης, πολύ σημαντικό είναι το γεγονός ότι παρόλο που έχουμε ακολουθία $(1, 21, 41, 61, 81, 101)$ με αποστάσεις-Δ 20, δεν επιτυγχάνεται η κωδικοποίησή της γιατί αλληλοεπικαλύπτεται με τις ακολουθίες που φαίνονται στο Σχήμα 3.2.

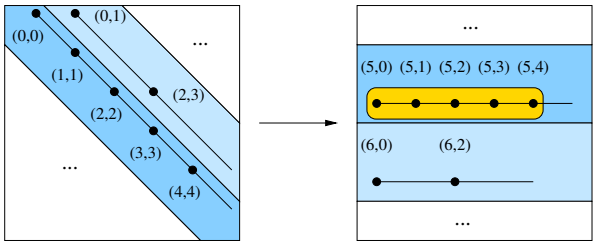
3.2.2 Κατακόρυφες και Διαγώνιες Κανονικότητες

Η λογική των κατακόρυφων και των διαγωνίων κανονικοτήτων είναι απλή. Εφαρμόζεται, αρχικά, ένας μετασχηματισμός των συντεταγμένων των στοιχείων του πίνακα, ώστε συνεχόμενα στοιχεία σε μια διεύθυνση (κατακόρυφη ή διαγώνια) να μετασχηματιστούν σε συνεχόμενα στοιχεία ως προς την οριζόντια διεύθυνση. Τέλος, επα-

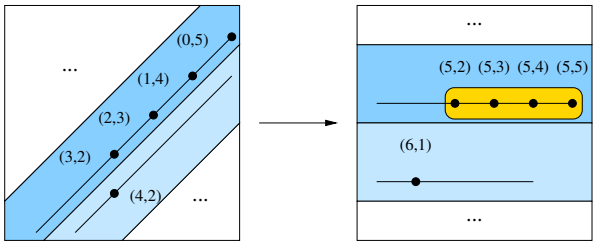
ναλαμβάνεται η ίδια διαδικασία με τις οριζόντιες κανονικότητες για την ανίχνευση κανονικοτήτων και εφαρμόζεται αντίστροφος μετασχηματισμός για να επανέλθουν τα στοιχεία του πίνακα στην αρχική τους κατάσταση. Στο Σχήμα 3.3 φαίνεται ο μετασχηματισμός του αραιού πίνακα για (α′) κατακόρυφες, (β′) διαγώνιες και (γ′) αντί-διαγώνιες κανονικότητες.



(α′) Μετασχηματισμός για ανίχνευση κατακόρυφων κανονικοτήτων



(β′) Μετασχηματισμός για ανίχνευση διαγωνίων κανονικοτήτων



(γ′) Μετασχηματισμός για ανίχνευση αντι-διαγωνίων κανονικοτήτων

Σχήμα 3.3: Ανίχνευση κανονικοτήτων μιας διάστασης για πίνακα μεγέθους 6x6.

Γενικά θα μπορούσαν να βρεθούν και οι μετασχηματισμοί για όλες τις κατευθύνσεις, αλλά οι υπόλοιποι μετασχηματισμοί δεν είναι τόσο απλοί και η συχνότητα με την οποία εμφανίζονται οι αντίστοιχες κανονικότητες δεν είναι ικανοποιητική. Οπότε για μονοδιάστατες κανονικότητες το CSX ανιχνεύει μόνο τις οριζόντιες, κατακόρυφες, διαγώνιες και αντί-διαγώνιες, όπως φαίνεται και στον Πίνακα 3.1.

Διεύθυνση		Συντεταγμένες	
		y	x
Οριζόντια	$\rightarrow$	y_0	$x_0 + i\delta$
Κατακόρυφη	$\downarrow$	$y_0 + i\delta$	x_0
Διαγώνια	$\searrow$	$y_0 + i\delta$	$x_0 + i\delta$
Αντι-Διαγώνια	$\swarrow$	$y_0 + i\delta$	$x_0 - i\delta$

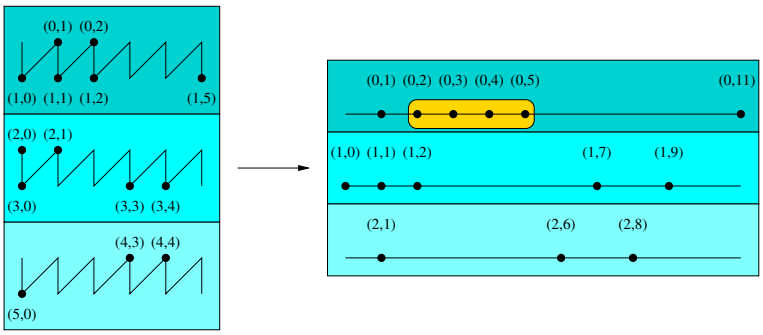
Πίνακας 3.1: Κατευθύνσεις για ανίχνευση μονοδιάστατων κανονικοτήτων που χρησιμοποιεί το σχήμα αποθήκευσης CSX.

3.2.3 Δισδιάστατες Κανονικότητες (Μπλοκ)

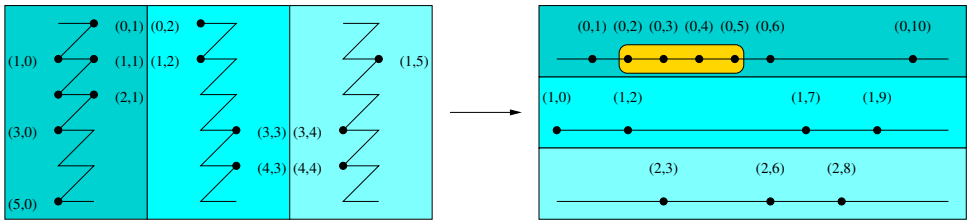
Οι δισδιάστατες κανονικότητες του CSX χωρίζονται σε δύο κατηγορίες, στα ευθυγραμμισμένα κατά γραμμές μπλοκ και στα ευθυγραμμισμένα κατά στήλες μπλοκ. Για την ανίχνευση των ευθυγραμμισμένων κατά γραμμές μπλοκ που έχουν μέγεθος k χωρίζεται ο πίνακας σε N/k ισοδύναμα κομμάτια, όπου N η διάσταση του αραιού πίνακα. Το κάθε κομμάτι περιέχει k γραμμές και N στήλες και εφαρμόζεται σε αυτό μετασχηματισμός όπου η νέα γραμμή είναι ο αριθμός του κομματιού και η νέα στήλη είναι ο αριθμός του στοιχείου (μηδενικού και μη-μηδενικού) όταν το κομμάτι διατρέχεται από την αρχή προς το τέλος κατά στήλες. Έπειτα, ανιχνεύονται οι κανονικότητες που εμφανίζονται στον αραιό πίνακα, όπως ακριβώς και στις οριζόντιες κανονικότητες. Για την ανίχνευση των ευθυγραμμισμένων κατά στήλες μπλοκ γίνεται ακριβώς η ίδια διαδικασία με μόνες διαφορές ότι ο πίνακας χωρίζεται κατά στήλες και όχι κατά γραμμές και ότι διατρέχονται τα στοιχεία κατά γραμμή σε κάθε επιμέρους κομμάτι και όχι κατά στήλη.

Στο Σχήμα 3.4 φαίνονται παραδείγματα για ανίχνευση (α′) ευθυγραμμισμένων κατά γραμμές και (β′) ευθυγραμμισμένων κατά στήλες μπλοκ σε έναν αραιό πίνακα. Παρατηρούμε ότι στα δύο μπλοκ που ανιχνεύονται υπάρχουν παραπάνω από τέσσερα διαδοχικά στοιχεία (που είναι τονισμένα), αλλά δεν συμμετέχουν όλα στις κανονικότητες. Αυτό γίνεται επειδή υπάρχουν στοιχεία που δεν αντιστοιχούν σε στοιχεία ενός πλήρους μπλοκ. Επομένως τα στοιχεία που δεν σχηματίζουν μια πλήρη στήλη ή γραμμή (ανάλογα με το αν ανιχνεύουμε μπλοκ ευθυγραμμισμένα κατά γραμμή ή κατά στήλη αντίστοιχα) δεν πρέπει να ανιχνεύονται ως μέλη της κανονικότητας και πρέπει να αποκόπτονται.

Παρόλα αυτά, οι συγκεκριμένοι μετασχηματισμοί έχουν και ορισμένα προβλήματα. Όπως φαίνεται και στα διαγράμματα, υπάρχει μπλοκ 2x2 το οποίο οι δύο μετασχηματισμοί αποτυγχάνουν να ανιχνεύσουν, επειδή το μπλοκ δεν υπάρχει ολόκληρο σε κανένα από τα κομμάτια που χωρίζεται ο πίνακας σε κάποια από τις δύο περιπτώσεις. Στην περίπτωση που ανιχνεύονται μπλοκ μεγέθους 2 μόνο στην περίπτωση που τα μπλοκ ξεκινάνε από σημεία $(2 \times r + 1, 2 \times c + 1)$ είναι μη-ανιχνεύσιμα (25% των περιπτώσεων), αλλά όσο μεγαλώνει το μέγεθος των μπλοκ τόσο μεγαλύτερο είναι το ποσοστό των μπλοκ που δεν ανιχνεύεται με καμία από τις δύο τεχνικές. Επίσης, το CSX δεν ανιχνεύει μπλοκ με κενά ανάμεσα τους όπως κάνει με τις μονοδιάστατες κανονικότητες, αλλά ανιχνεύει μόνο πυκνά μπλοκ που στο μετασχηματισμένο πίνακα είναι διαδοχικά στοιχεία.



(α′) Μετασχηματισμός για ανίχνευση ευθυγραμμισμένων κατά γραμμές μπλοκ μεγέθους 2



(β′) Μετασχηματισμός για ανίχνευση ευθυγραμμισμένων κατά στήλες μπλοκ μεγέθους 2

Σχήμα 3.4: Ανίχνευση κανονικοτήτων δύο διαστάσεων για πίνακα μεγέθους 6x6.

Οι δισδιάστατες κανονικότητες συναντιούνται πάρα πολύ σε αραιούς πίνακες, ειδικά σε αυτούς που προκύπτουν από προβλήματα δισδιάστατης ή τρισδιάστατης γεωμετρίας (2D/3D geometry) [1, 4, 5]. Επίσης, εκτός του ότι μειώνουν σημαντικά τον όγκο δεδομένων είναι πιο αξιοποιήσιμες από τις υπόλοιπες κανονικότητες καθώς μπορούν να εφαρμοστούν πάνω τους διάφορες τεχνικές βελτίωσης, που μειώνουν αισθητά τους αριθμητικούς υπολογισμούς [5, 12], όπως το *register blocking*, το *vectorization* και το *loop unroll*.

Καταληκτικά, το CSX ανιχνεύει μπλοκ από 2 έως 8 μέγεθος και για τις δύο περιπτώσεις. Τα μεγέθους ένα ευθυγραμμισμένα κατά γραμμές μπλοκ είναι ισοδύναμα με τις οριζόντιες κανονικότητες, ενώ τα μεγέθους ένα ευθυγραμμισμένα κατά στήλες μπλοκ είναι ισοδύναμα με τις κατακόρυφες κανονικότητες. Επιπρόσθετα, τα μπλοκ μεγέθους μεγαλύτερου του 8 αποτελούν πολύ μεγάλες πυκνές κανονικότητες που δύσκολα συναντιούνται σε αραιούς πίνακες και επιπλέον πολύ δύσκολα ανιχνεύονται από τις τεχνικές εύρεσης δισδιάστατων κανονικοτήτων λόγω του προβλήματος που αντιμετωπίζουν το οποίο περιγράφηκε στην προηγούμενη παράγραφο.

Στον Πίνακα 3.2 φαίνονται οι μετασχηματισμοί που χρησιμοποιούνται για την εύρεση όλων των κανονικοτήτων του σχήματος CSX. Αξιοσημείωτο είναι ότι για κάθε μετασχηματισμό υπάρχει και ένας αντίστροφος μετασχηματισμός που επαναφέρει τον αραιό πίνακα στην αρχική του μορφή. Οι μετασχηματισμοί αυτοί αποτελούν ένα χρήσιμο και σημαντικό εργαλείο, καθώς προσφέρουν ευελιξία στον τρόπο που επεξεργαζόμαστε έναν αραιό πίνακα, και για αυτό το λόγο χρησιμοποιούνται ευρέως κατά την διάρκεια της μετατροπής ενός πίνακα σε CSX μορφή.

Κανονικότητες	**Μετασχηματισμός**
Οριζόντιες	$(i', j') = (i, j)$
Κάτακόρυφες	$(i', j') = (j, i)$
Διαγώνιες	$(i', j') = (nrows - 1 + j - i, min(i, j))$
Αντι-διαγώνιες	$(i', j') = \begin{cases} (i+j, j), & i+j < nrows \\ (i+j, nrows-1-i), & i+j \geq nrows \end{cases}$
Μπλοκ ευθυγραμμισμένα κατά γραμμές	$(i', j') = (\lfloor \frac{i}{r} \rfloor, r \times j + mod(i, r))$
Μπλοκ ευθυγραμμισμένα κατά στήλες	$(i', j') = (\lfloor \frac{j}{c} \rfloor, c \times i + mod(j, c))$

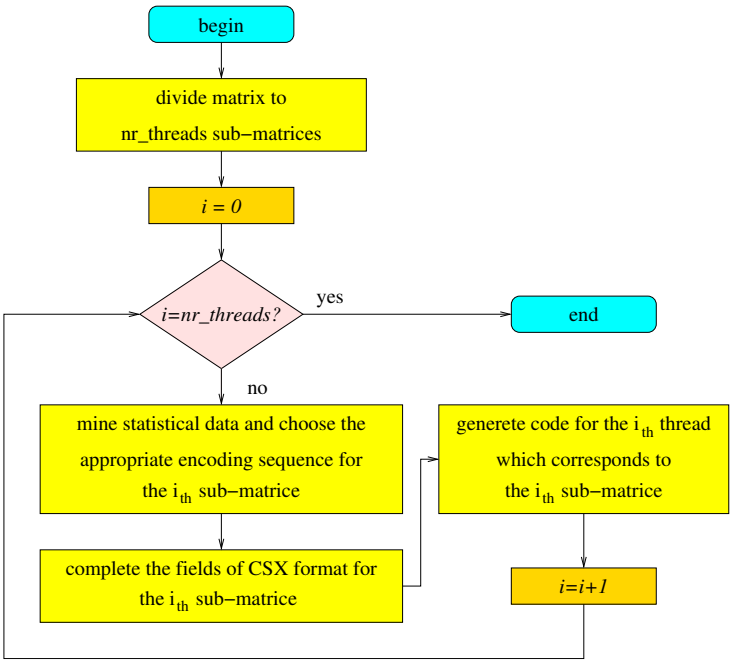
Πίνακας 3.2: Μετασχηματισμοί για όλες τις Κανονικότητες που υποστηρήζει το σχήμα αποθήκευσης CSX.

3.3 Μετατροπή Αραιού Πίνακα σε *CSX* Μορφή Αποθήκευσης

Στην προηγούμενη παράγραφο περιγράφηκαν οι κανονικότητες που εκμεταλλεύεται το σχήμα CSX. Παρακάτω, φαίνεται ολόκληρη η διαδικασία μετατροπής του αρχικού αραιού πίνακα σε μορφή CSX. Στο Σχήμα 3.5 παρουσιάζεται η διαδικασία που ακολουθείται για την συγκεκριμένη μετατροπή. Αρχικά, πρέπει (α′) ο πίνακας να χωριστεί σε υπο-πίνακες, ώστε να μοιραστεί στα νήματα εκτέλεσης. Έπειτα, ακολουθεί (β′) η εξόρυξη στατιστικών στοιχείων, από τα οποία προκύπτει η επιλογή της ακολουθίας κανονικοτήτων που αποτελεί την βάση για την συμπίεση του πίνακα. Αφού εφαρμοστούν οι κατάλληλες κωδικοποιήσεις στον πίνακα και επιτευχθεί η επιθυμητή συμπίεση, γίνεται (γ′) η συμπλήρωση των πεδίων του σχήματος αποθήκευσης CSX και, τέλος, (δ′) παράγεται ο κώδικας για το SpM×V υπολογιστικό πρόβλημα. Στα πλαίσια αυτής της εργασίας, παραλληλοποιήθηκαν τα βήματα (β΄), (γ΄) και (δ΄) με αποτέλεσμα να μειώνεται αισθητά ο χρόνος μετατροπής του πίνακα σε CSX μορφή για περισσότερα του ενός νήματα.

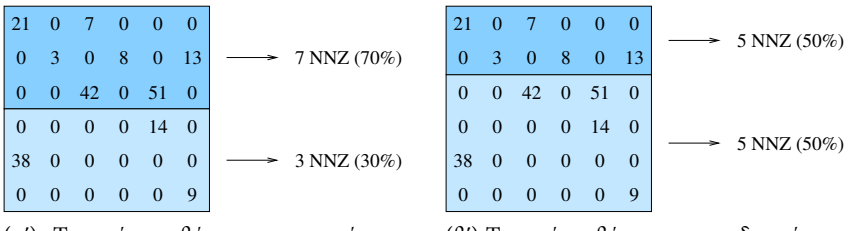
3.3.1 Διαχωρισμός SpM×V σε Νήματα

Το πρώτο βήμα που γίνεται στο SpM×V, όταν αυτό τρέχει σε πολυνηματικές εφαρμογές, είναι ο διαχωρισμός του πυρήνα σε νήματα. Στο SpM×V υπάρχουν μόνο αναγνώσεις με εξαίρεση το διάνυσμα εξόδου y το οποίο διαβάζεται και γράφεται συνέχεια από τα νήματα. Οποιοδήποτε από τα στοιχεία $y[i]$, αλλάζει τιμή από το πρόγραμμα όταν γίνεται ο πολλαπλασιασμός των στοιχείων της γραμμής i με τα αντίστοιχα στοιχεία του διανύσματος εισόδου x. Οπότε, το λογικό είναι να χωριστεί ο αραιός πίνακας κατά γραμμές, ώστε να μην υπάρχουν εξαρτήσεις μεταξύ των νημάτων. Ο πιο απλός τρόπος είναι τα n νήματα που είναι διαθέσιμα να πάρουν N/n γραμμές του αραιού πίνακα, όπου N ο συνολικός αριθμός των γραμμών του πίνακα. Ωστόσο, με αυτήν την τεχνική διαχωρισμού του πίνακα ανάμεσα στα νήματα δεν επιτυγχάνεται ιδανικός διαχωρισμός του φόρτου εργασίας, αφού ενδέχεται σε έναν υπο-πίνακα τα μη-μηδενικά στοιχεία και, επομένως, και το μέγεθος της μνήμης που χρειάζεται αλλά



Σχήμα 3.5: Διάγραμμα ροής μετατροπής αραιού πίνακα σε CSX μορφή.

και ο αριθμός των αριθμητικών υπολογισμών να είναι πολύ μεγαλύτερος από ότι σε ένα άλλο. Γι' αυτό το λόγο γίνεται ο χωρισμός του πίνακα σε n υπο-πίνακες κατά γραμμές φροντίζοντας ο αριθμός των μη-μηδενικών στοιχείων του κάθε υπο-πίνακα να διαφέρει ελάχιστα από το NNZ/n, όπου NNZ είναι ο συνολικός αριθμός των μη-μηδενικών στοιχείων του πίνακα. Στον Αλγόριθμο 9 φαίνεται αναλυτικότερα πως λειτουργεί η συγκεκριμένη τεχνική. Στο Σχήμα 3.6 φαίνεται ένα παράδειγμα που φαίνεται η υπεροχή της (β′) τεχνικής που βασίζεται στον αριθμό των μη-μηδενικών στοιχείων του πίνακα έναντι της (α′) τεχνικής που βασίζεται στον αριθμό των γραμμών.



(α′) Τεχνική με βάση τις γραμμές του αραιού πίνακα

(β′) Τεχνική με βάση τα μη-μηδενικά στοιχεία του αραιού πίνακα

Σχήμα 3.6: Διαχωρισμός αραιού πίνακα σε υπο-πίνακες για SpM×V όταν τρέχει σε 2 Νήματα.

Αλγόριθμος 9: Διαχωρισμός αραιού πίνακα με βάση τα μη-μηδενικά στοιχεία.

Input: N, size of sparse matrix
Input: n, number of threads
Input: NNZ, number of non-zero elements

$start\_line \leftarrow 0$
$current\_line \leftarrow 0$
for $i \leftarrow 0$ **to** $n-1$ **do**
  $count \leftarrow 0$
  while $count < NNZ/n$ **do**
    $count \leftarrow count + CountElems(current\_line)$
    $current\_line \leftarrow current\_line + 1$
  $AssignLines(i, start\_line, current\_line-1)$
  $start\_line \leftarrow current\_line$
$AssignLines(n-1, start\_line, N-1)$

3.3.2 Εξόρυξη Στατιστικών Στοιχείων και Επιλογή Ακολουθίας Κωδικοποιήσεων μέσω της Ευριστικής Μεθόδου

Αφού χωριστεί ο αραιός πίνακας με το τρόπο που περιγράφηκε στην προηγούμενη παράγραφο, κάθε νήμα επεξεργάζεται τον αντίστοιχο υπο-πίνακα. Αρχικά, κρατείται μια λίστα με όλους τους επιθυμητούς τύπους κανονικοτήτων. Στον Πίνακα 3.3 και στον Πίνακα 3.4 φαίνονται όλοι οι δυνατοί μονοδιάστατοι και δισδιάστατοι τύποι κανονικοτήτων, αντίστοιχα. Έπειτα, για κάθε τύπο γίνεται ο αντίστοιχος μετασχηματισμός, ανιχνεύονται οι κανονικότητες των υπο-τύπων του, που προκύπτουν από τις διαφορετικές αποστάσεις-Δ στις μονοδιάστατες κανονικότητες και τα διάφορα μεγέθη των δεύτερων διαστάσεων στα μπλοκ, και κρατείται ο αριθμός των κανονικοτήτων καθώς και ο αριθμός των μη-μηδενικών στοιχείων που καλύπτουν (Κεφάλαιο 3.2 σελ. 32). Αξιοσημείωτο στο συγκεκριμένο σημείο είναι το γεγονός ότι έχει υλοποιηθεί μια νέα τεχνική συλλογή στατιστικών στοιχείων από μικρά παραθυρά του υπο-πίνακα και όχι από όλη την επιφάνεια του [9]. Η νέα αυτή τεχνική έχει παρόμοια αποτελέσματα και ελάχιστες φορές αποκλίνουν τα αποτελέσματα της με τα πραγματικά με το επιπλέον πλεονέκτημα ότι μειώνει τον χρόνο της μετατροπής του αραιού πίνακα σε CSX μορφή πάρα πολύ. Αφού συλλεχθούν τα αποτελέσματα περνούν από φίλτρο και οι υπο-τύποι που τα συνολικά μη-μηδενικά στοιχεία όλων των κανονικοτήτων τους στον πίνακα δεν ξεπερνούν κάποιο ποσοστό min_perc επί των συνολικών, αφαιρούνται για λόγους που θα γίνουν ξεκάθαροι αργότερα (Κεφάλαιο 3.3.3 σελ. 39 και Κεφάλαιο 3.3.4 σελ. 41). Οι τύποι κανονικοτήτων που κανένας υπο-τύπος τους δεν παραμένει στα αποτελέσματα αφαιρούνται από την λίστα και δεν λαμβάνονται υπόψιν στην περαιτέρω επεξεργασία του πίνακα.

Στο σημείο αυτό επιλέγεται ο καταλληλότερος τύπος κανονικοτήτων κάτα την ευριστική μέθοδο και κωδικοποιούνται όλα τα μη-μηδενικά στοιχεία που καλύπτουν οι κανονικότητες του σε στοιχεία του συγκεκριμένου τύπου. Επειδή η συμπίεση που επιτυγχάνεται οφείλεται στην μείωση του όγκου δεδομένων του πεδίου col_ind, η ευ-

Κανονικότητες	**Αποστάσεις-Δ**
Οριζόντιες	$1-255$
Κατακόρυφες	$1-255$
Διαγώνιες	$1-255$
Αντι-Διαγώνιες	$1-255$

Πίνακας 3.3: Μονοδιάστατοι τύποι κανονικοτήτων με όλες τις δυνατές αποστάσεις-Δ.

Κανονικότητες	**Δεύτερη Διάσταση** (c)	**Κανονικότητες**	**Δεύτερη Διάσταση** (r)
Μπλοκ $2 \times c$	$2-127$	Μπλοκ $r \times 2$	$2-127$
Μπλοκ $3 \times c$	$2-85$	Μπλοκ $r \times 3$	$2-85$
Μπλοκ $4 \times c$	$2-63$	Μπλοκ $r \times 4$	$2-63$
Μπλοκ $5 \times c$	$2-51$	Μπλοκ $r \times 5$	$2-51$
Μπλοκ $6 \times c$	$2-42$	Μπλοκ $r \times 6$	$2-42$
Μπλοκ $7 \times c$	$2-36$	Μπλοκ $r \times 7$	$2-36$
Μπλοκ $8 \times c$	$2-31$	Μπλοκ $r \times 8$	$2-31$

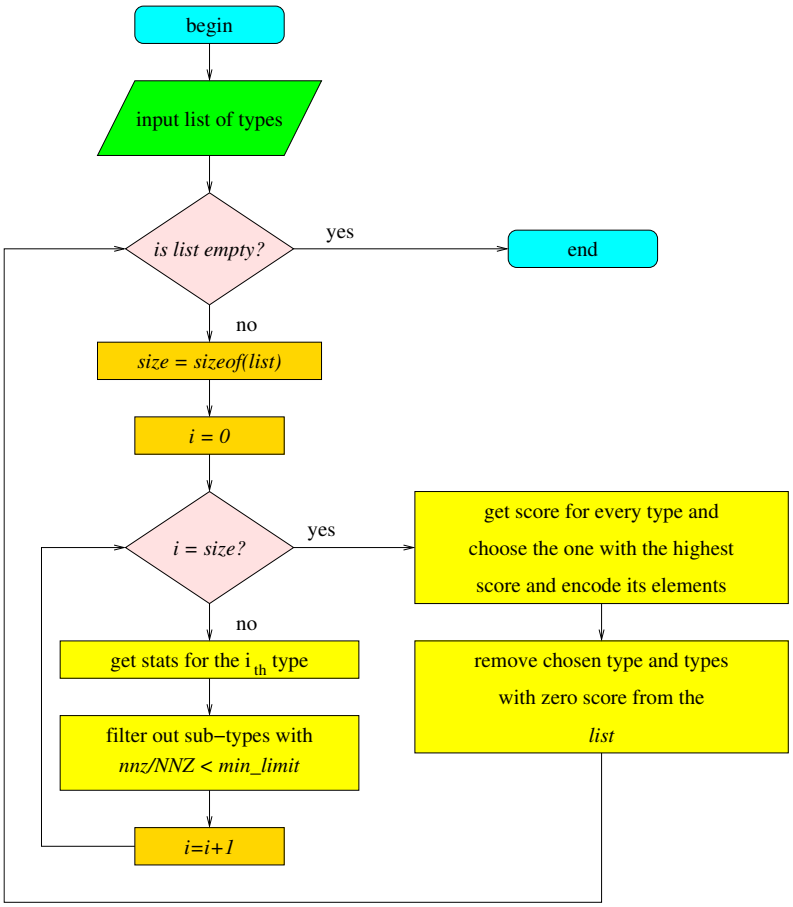
Πίνακας 3.4: Δισδιάστατοι τύποι κανονικοτήτων με όλες τις δυνατές διαστάσεις.

ριστική μέθοδος υπολογίζει σαν σκορ τη μείωση του πλήθους των στοιχείων του συγκεκριμένου πεδίου. Πιο συγκεκριμένα το σκορ υπολογίζεται ως $nnz-(npatterns+nnz-nnz_{enc}) = nnz_{enc}-npatterns$, όπου nnz το αρχικό πλήθος στοιχείων του col_ind, $npatterns$ ο αριθμός των κανονικοτήτων που ανιχνεύτηκαν για τον εκάστοτε τύπο και nnz_{enc} το πλήθος των στοιχείων που καλύπτονται από αυτές τις κανονικότητες (ώστε το $nnz-nnz_{enc}$ να είναι το πλήθος των στοιχείων που δεν καλύπτονται από κανονικότητες). Κάθε φορά επιλέγεται να κωδικοποιηθεί ο τύπος κανονικοτήτων που επιτυγχάνει την καλύτερη συμπίεση, δηλαδή αυτος με το μεγαλύτερο σκορ.

Αφού, γίνει η κωδικοποίηση του τύπου κανονικοτήτων που επιλέγεται, αφαιρείται ο συγκεκριμένος τύπος από την λίστα και επαναλαμβάνεται η ίδια διαδικασία μέχρι όλοι οι τύποι να αφαιρεθούν από την λίστα. Είναι βέβαιο ότι επιλέγεται ο τύπος που στο επόμενο βήμα θα έχει συμπιέσει πιο πολύ τον πίνακα σε σχέση με τους υπόλοιπους (άπληστος αλγόριθμος). Παρόλ'αυτά, δεν είναι καθόλου σίγουρο ότι η ακολουθία κωδικοποιήσεων που επιλέγεται από την συγκεκριμένη ευριστική θα έχει την βέλτιστη συμπίεση και επομένως ένα από τα καλύτερα δυνατά αποτελέσματα στην επίδοση του SpM×V. Αναλυτικά, όλη η διαδικασία επιλογής της ακολουθίας κωδικοποιήσεων που εφαρμόζεται στον αραιό πίνακα φαίνεται στον Σχήμα 3.7.

3.3.3 Συμπλήρωση των Πεδίων του Σχήματος Αποθήκευσης CSX

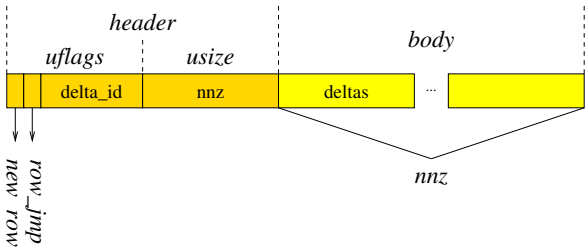
Αφού κωδικοποιηθούν οι κανονικότητες που συναντιούνται στον αραιό πίνακα σε στοιχεία, συμπληρώνονται τα πεδία του σχήματος αποθήκευσης CSX. Τα πεδία που συμπληρώνονται είναι το *uflags* και το *usize* τα οποία αποτελούν την επικεφαλίδα (*header*) και το κυρίως σώμα (*body*) το οποίο περιέχει τις αποστάσεις-Δ των στοιχείων όπως ακριβώς και στο σχήμα αποθήκευσης *CSR-DU* (Κεφάλαιο 3.1 σελ.30). Το *uflags* έχει μέγεθος 8 bit, από τα οποία το πιο σημαντικό bit είναι μια σημαία που ενεργο-



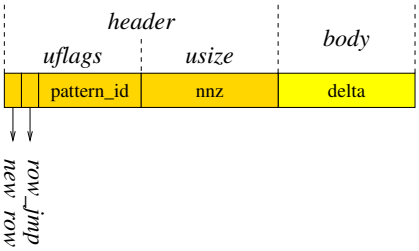
Σχήμα 3.7: Εξόρυξη στατιστικών στοιχείων και επιλογή ακολουθίας κωδικοποιήσης αραιού πίνακα.

ποιείται στην περίπτωση νέας σειράς (new_row), το δεύτερο πιο σημαντικό bit είναι μια σημαία που ενεργοποιείται όταν υπάρχουν κενές γραμμές (row_jmp) και τα υπόλοιπα έξι bit χρησιμοποιούνται για την περιγραφή του υπο-τύπου της κανονικότητας που συναντιέται. Σε περίπτωση που το row_jmpbit είναι ενεργοποιημένο υπάρχουν επιπλέον πεδία των 8 bit που δηλώνουν το μέγεθος των κενών γραμμών. Στο συγκεκριμένο σημείο πρέπει να τονιστεί ότι μπορούν να υπάρχουν μόνο 2^6 διαφορετικοί υπο-τύποι κανονικοτήτων και, επομένως, οι επιλογές των υπο-τύπων που γίνονται πρέπει να είναι περιορισμένες σε αριθμό. Σε αυτό βοηθάει αρκετά τό γεγονός ότι υπάρχει κατώφλι (min_perc) για την ένταξη τους στην κωδικοποίηση του πίνακα. Το *usize* έχει μέγεθος 8 bit και περιέχει τον αριθμό των μη-μηδενικών στοιχείων που ανήκουν σε μια μονάδα CSX, οπότε το μέγεθος των ομάδων που δημιουργούνται δεν μπορεί να ξεπερνά το 2^8. Επίσης, στο σώμα περιέχονται οι αποστάσεις ή η απόσταση, ανάλογα με τον αν το *CSX*-στοιχείο περιγράφει κάποια κανονικότητα ή όχι, από το προηγούμενο στοιχείο (απόσταση-Δ).

Στο Σχήμα 3.8 φαίνεται αναλυτικά η απεικόνιση του σχήματος αποθήκευσης CSX.



(α′) Απεικόνιση στοιχείων που δεν περιγράφουν κανονικότητες



(β′) Απεικόνιση στοιχείων που περιγράφουν κανονικότητες

Σχήμα 3.8: Σχήμα αποθήκευσης CSX.

Αξίζει να ειπωθεί ότι στην περίπτωση των κανονικοτήτων, παρόλο που μπορεί να περιέχουν πολλά μη-μηδενικά στοιχεία, χρειάζεται μόνο η θέση του πρώτου στοιχείου στον πίνακα για να βρεθούν και τα υπόλοιπα.

3.3.4 Παραγωγή Τελικού Κώδικα

Μετά την συμπλήρωση των πεδίων γίνεται η παραγωγή κώδικα για το SpM×V για το συγκεκριμένο πίνακα που ερευνάται. Αξιοσημείωτο είναι το γεγονός ότι γίνεται just-in-time compilation με την χρήση του $llvminfastucture$. Ο Αλγόριθμος 10 δείχνει πως γίνεται η εκτέλεση του SpM×V για έναν πίνακα σε μορφή CSX και οι Αλγόριθμοι από 11 έως 17 δείχνουν τους υπολογισμούς που γίνονται για κάθε κατηγορία κανονικοτήτων. Θετικό είναι το γεγονός ότι παράγεται κώδικας μόνο για τους υπο-τύπους κανονικοτήτων που συναντιούνται στον πίνακα και όχι για όλους με αποτέλεσμα να μειώνεται ο χρόνος των υπολογισμών δραστικά. Αυτό, άλλωστε, αποτελεί και τον σημαντικότερο λόγο που γίνεται το φιλτράρισμα των υπο-τύπων κατά την διάρκεια της κωδικοποίησης του πίνακα. Με την εισαγωγή κατώτερου κατωφλίου στους υπο-τύπους ο αριθμός τους μειώνεται σημαντικά.

Αλγόριθμος 10: SpM×V για σχήμα CSX.

Input: $elems$, CSX elements
Input: val, Values of non-zero elements
Input: x, Input Vector
Output: y, Output Vector

$cur\_row \leftarrow -1$
$cur\_col \leftarrow 0$
$counter \leftarrow 0$
forall $elem \in elems$ **do**
  $new\_row \leftarrow GetNewRow(elem)$
  if $new\_row = 1$ **then**
    $row\_jmp \leftarrow GetRowJmp(elem)$
    if $row\_jmp = 1$ **then**
      $jmp \leftarrow GetJmp(elem)$
      $cur\_row \leftarrow cur\_row + jmp$
    $cur\_row \leftarrow cur\_row + 1$
    $cur\_col \leftarrow 0$
  $type \leftarrow GetType(elem)$
  $size \leftarrow GetSize(elem)$
  switch $type$ **do**
    case *none*
      $DeltaCase$
    case *horizontal*
      $HorizontalCase$
    case *vertical*
      $VerticalCase$
    case *diagonal*
      $DiagonalCase$
    case *reverse diagonal*
      $ReverseDiagonalCase$
    case *block row*
      $BlockRowCase$
    case *block col*
      $BlockColCase$

Αλγόριθμος 11: DeltaCase

for $i = 0$ **to** $size$ **do**
  $cur\_col \leftarrow cur\_col + GetDelta(elem, i)$
  $y[cur\_row] \leftarrow y[cur\_row] + val[counter] \times x[cur\_col]$
  $counter \leftarrow counter + 1$

Αλγόριθμος 12: HorizontalCase

$cur\_col \leftarrow cur\_col + GetDelta(elem)$
$pdelta \leftarrow GetPatternDelta(elem)$
$temp\_col \leftarrow cur\_col$
for $i = 0$ **to** $size$ **do**
  $y[cur\_row] \leftarrow y[cur\_row] + val[counter] \times x[temp\_col]$
  $temp\_col \leftarrow temp\_col + pdelta$
  $counter \leftarrow counter + 1$

Αλγόριθμος 13: VerticalCase

$cur\_col \leftarrow cur\_col + GetDelta(elem)$
$pdelta \leftarrow GetPatternDelta(elem)$
$temp\_row \leftarrow cur\_row$
for $i = 0$ **to** $size$ **do**
  $y[temp\_row] \leftarrow y[temp\_row] + val[counter] \times x[cur\_col]$
  $temp\_row \leftarrow temp\_row + pdelta$
  $counter \leftarrow counter + 1$

Αλγόριθμος 14: DiagonalCase

$cur\_col \leftarrow cur\_col + GetDelta(elem)$
$pdelta \leftarrow GetPatternDelta(elem)$
$temp\_row \leftarrow cur\_row$
$temp\_col \leftarrow cur\_col$
for $i = 0$ **to** $size$ **do**
  $y[temp\_row] \leftarrow y[temp\_row] + val[counter] \times x[temp\_col]$
  $temp\_row \leftarrow temp\_row + pdelta$
  $temp\_col \leftarrow temp\_col + pdelta$
  $counter \leftarrow counter + 1$

Αλγόριθμος 15: ReverseDiagonalCase

$cur\_col \leftarrow cur\_col + GetDelta(elem)$
$pdelta \leftarrow GetPatternDelta(elem)$
$temp\_row \leftarrow cur\_row$
$temp\_col \leftarrow cur\_col$
for $i = 0$ **to** $size$ **do**
  $y[temp\_row] \leftarrow y[temp\_row] + val[counter] \times x[temp\_col]$
  $temp\_row \leftarrow temp\_row + pdelta$
  $temp\_col \leftarrow temp\_col - pdelta$
  $counter \leftarrow counter + 1$

Αλγόριθμος 16: RowBlockCase

```
cur_col ← cur_col + GetDelta(elem)
col_size ← GetOtherDimension(elem)
row_size ← size/col_size
for row = 0 to row_size do
    for col = 0 to col_size do
        y[cur_row+row] ← y[cur_row+row]+val[counter]×x[cur_col+col]
        counter ← counter + 1
```

Αλγόριθμος 17: ColBlockCase

```
cur_col ← cur_col + GetDelta(elem)
row_size ← GetOtherDimension(elem)
col_size ← size/row_size
for col = 0 to col_size do
    for row = 0 to row_size do
        y[cur_row+row] ← y[cur_row+row]+val[counter]×x[cur_col+col]
        counter ← counter + 1
```

Κεφάλαιο 4

Επέκταση Διαχωρισμού Μπλοκ (*CSX-split*)

Στο συγκεκριμένο Κεφάλαιο παρουσιάζονται, αρχικά, οι αραιοί πίνακες που χρησιμοποιούνται για τις μετρήσεις (στο Κεφάλαιο 4 και στο Κεφάλαιο 5), οι οποίοι προέρχονται από την σουίτα *University of Florida Sparse Matrix Collection*. Έπειτα, η πειραματική πλατφόρμα πάνω στις οποίες έγιναν οι μετρήσεις και ειδικότερα το κομμάτι των αρχιτεκτονικών που έχει να κάνει με την ιεραρχία μνήμης (*caches*). Τέλος, αναλύεται η επέκταση του διαχωρισμού των μπλοκ (*CSX-split*), καθώς και τα αποτελέσματα των μετρήσεων που επιβεβαιώνουν την βελτίωση στην επίδοση του SpM×V.

4.1 Σουίτα Αραιών Πινάκων

Οι πίνακες που χρησιμοποιούνται για τις μετρήσεις και την αξιολόγηση σε αυτό και στο επόμενο κεφάλαιο προέρχονται από την σουίτα *University of Florida Sparse Matrix Collection* [2]. Η σουίτα αυτή αποτελεί μία μεγάλη συλλογή αραιών πινάκων που έχουν προκύψει από πραγματικές εφαρμογές. Η συγκεκριμένη συλλογή χρησιμοποιείται πολύ από την ερευνητική κοινότητα που ασχολείται με την γραμμική άλγεβρα για την ανάπτυξη και την βελτίωση αλγορίθμων που έχουν να κάνουν με αραιούς πίνακες. Η σουίτα αυτή περιέχει πίνακες που χρησιμοποιούνται σε πραγματικές εφαρμογές με αποτέλεσμα να εξάγονται αποτελέσματα και συμπεράσματα που έχουν πρακτική αξία και χρησιμότητα. Πιο συγκεκριμένα, οι πίνακες καλύπτουν ένα μεγάλο φάσμα από πεδία και χωρίζονται σε δύο μεγάλες κατηγορίες, σε αυτούς που προκύπτουν από προβλήματα που έχουν να κάνουν με 2D ή 3D γεωμετρία (δομικής εφαρμοσμένης μηχανικής, υπολογιστικής ρευστοδυναμικής, ηλεκτρομαγνητικής, με ημι-αγώγιμες συσκευές, θερμοδυναμικής, υλικών, ακουστικής, όρασης υπολογιστών, με γραφικά υπολογιστών, κινηματικής και ρομποτικής) και σε αυτούς που δεν προκύπτουν από προβλήματα που έχουν να κάνουν με 2D ή 3D γεωμετρία (προσομοίωσης δικτύων, οικονομικών μοντέλων, θεωρητικής και κβαντικής χημείας, μαθηματικών και στατιστικών, δυναμικών δικτύων και γράφων). Στον Πίνακα 4.1 παρουσιάζονται οι 48 πίνακες που χρησιμοποιούνται για τις μετρήσεις.

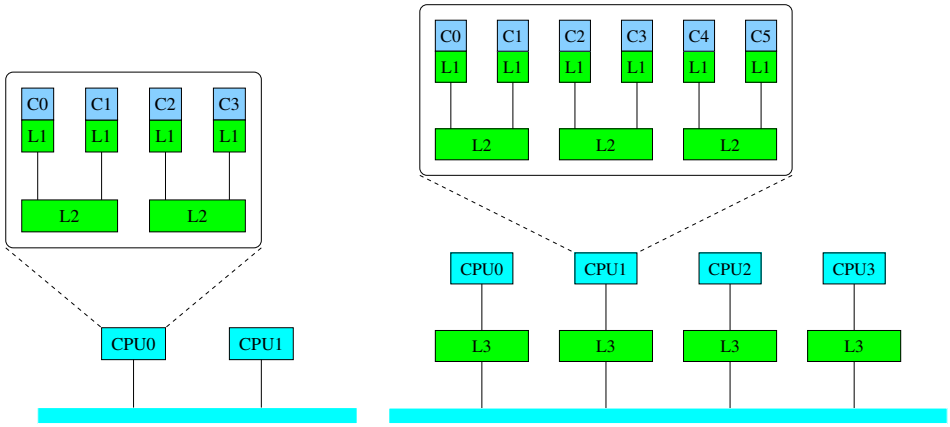
Α/Α	**Πίνακας**	**Γραμμές**	**Στήλες**	**Στοιχεία**	**Μέγεθος** (MB)	**Κατηγορία Πίνακα**
01	rma10	46835	46835	2374001	27.35	Υπολογιστική Ρευστοδυναμική
02	poisson3Db	85623	85623	2374949	27.51	Υπολογιστική Ρευστοδυναμική
03	ASIC_680ks	682712	682712	2329176	29.26	Προσομοίωση Δικτύων
04	helm2d03	392257	392257	2741935	32.88	2D/3D Γεωμετρία
05	thermomech_dK	204316	204316	2846228	33.35	Θερμοδυναμική
06	stomach	213360	213360	3021648	35.39	2D/3D Γεωμετρία
07	sme3Dc	42930	42930	3148656	36.20	Δομικής Εφαρμοσμένης Μηχανικής
08	FEM_3D_thermal2	147900	147900	3489300	40.49	Θερμοδυναμική
09	parabolic_fem	525825	525825	3674625	44.06	Υπολογιστική Ρευστοδυναμική
10	xenon2	157464	157464	3866688	44.85	Υλικά
11	barrier2-9	115625	115625	3897557	45.05	Ημι-αγώγιμες Συσκευές
13	ASIC_680k	682862	682862	3871773	46.91	Προσομοίωση Δικτύων
14	thread	29736	29736	4470048	51.27	Δομικής Εφαρμοσμένης Μηχανικής
15	torso3	259156	259156	4429042	51.67	2D/3D Γεωμετρία
16	ship_001	34920	34920	4644230	53.28	Δομικής Εφαρμοσμένης Μηχανικής
17	s3dkq4m2	90449	90449	4820891	55.52	Δομικής Εφαρμοσμένης Μηχανικής
18	apache2	715176	715176	4817870	57.86	Δομικής Εφαρμοσμένης Μηχανικής
19	Chebyshev4	68121	68121	5377761	61.80	Δομικής Εφαρμοσμένης Μηχανικής
20	largebasis	440020	440020	5560100	65.31	Βελτιστοποίηση
21	Hamrle3	1447360	1447360	5514242	68.62	Προσομοίωση Δικτύων
22	pre2	659033	659033	5959282	70.71	Προσομοίωση Δικτύων
23	rajat30	643994	643994	6175377	73.13	Προσομοίωση Δικτύων
24	cage13	445315	445315	7479343	87.29	Γράφοι
25	G3_circuit	1585478	1585478	7660826	93.72	Προσομοίωση Δικτύων
26	thermal2	1228045	1228045	8580313	102.9	Θερμοδυναμική
27	atmosmodj	1270432	1270432	8814880	105.7	Υπολογιστική Ρευστοδυναμική
28	bmwcra_1	148770	148770	10644002	122.4	Δομικής Εφαρμοσμένης Μηχανικής
29	Si87H76	240369	240369	10661631	122.9	Χημεία
30	hood	220542	220542	10768436	124.1	Δομικής Εφαρμοσμένης Μηχανικής
31	ohne2	181343	181343	11063545	127.3	Ημι-αγώγιμες Συσκευές
32	bmw3_2	227362	227362	11288630	130.1	Δομικής Εφαρμοσμένης Μηχανικής
33	pwtk	217918	217918	11634424	134.0	Δομικής Εφαρμοσμένης Μηχανικής
34	crankseg_2	63838	63838	14148858	162.2	Δομικής Εφαρμοσμένης Μηχανικής
35	nd12k	36000	36000	14220946	162.9	2D/3D Γεωμετρία
37	Si41Ge41H72	185639	185639	15011265	172.5	Χημεία
38	kkt_power	2063494	2063494	14612663	175.1	Βελτιστοποίηση
39	TSOPF_RS_b2383	38120	38120	16171169	185.2	Δυναμικά Δίκτυα
40	af_5_k101	503625	503625	17550675	202.8	Δομικής Εφαρμοσμένης Μηχανικής
41	af_shell9	504855	504855	17588875	203.2	Δομικής Εφαρμοσμένης Μηχανικής
42	Ga41As41H72	268096	268096	18488476	212.6	Χημεία
43	Freescale1	3428755	3428755	18920347	229.6	Προσομοίωση Δικτύων
44	msdoor	415863	415863	20240935	233.2	Δομικής Εφαρμοσμένης Μηχανικής
45	rajat31	4690002	4690002	20316253	250.4	Προσομοίωση Δικτύων
46	F1	343791	343791	26837113	308.4	Δομικής Εφαρμοσμένης Μηχανικής
47	fdif202x202x102	4000000	4000000	27840000	333.9	2D/3D Γεωμετρία
48	inline_1	503712	503712	36816342	423.3	Δομικής Εφαρμοσμένης Μηχανικής
49	ldoor	952203	952203	46522475	536.0	Δομικής Εφαρμοσμένης Μηχανικής
50	boneS10	914898	914898	55468422	638.3	Μείωση Μοντέλων

Πίνακας 4.1: Σουίτα αραιών πινάκων.

4.2 Πειραματική Πλατφόρμα

Οι μετρήσεις για την πειραματική αξιολόγηση έγιναν σε δύο διαφορετικά συστήματα υπολογιστών, σε ένα σύστημα 2 επεξεργαστών 4-πύρηνων αρχιτεκτονικής *Intel Harpertown* και σε ένα σύστημα 4 επεξεργαστών 6-πύρηνων αρχιτεκτονικής *Intel*

Dunnington. Στο Σχήμα 4.1 φαίνεται η ιεραρχία μνήμης για τις δύο αρχιτεκτονικές και στον Πίνακα 4.2 φαίνονται τα χαρακτηριστικά μεγέθη των δύο συστημάτων.



(α′) Αρχιτεκτονική Haprpertown (β′) Αρχιτεκτονική Dunnington

Σχήμα 4.1: Ιεραρχία μνήμης για τις αρχιτεκτονικές που χρησιμοποιούνται.

	Harpertown	**Dunnington**
Μοντέλο	E5405	X7460
Συχνότητα (GHz)	2.0	2.66
L1 (data/instr.)	32K/32K	32K/32K
L2 (unified)	6M	3M
L3 (unified)	–	16M
Αριθμός Πυρήνων	$2 \times 4 = 8$	$4 \times 6 = 24$

Πίνακας 4.2: Χαρακτηριστικά συστήματος για τις αρχιτεκτονικές.

Αξιοσημείωτο είναι ότι σημαντικό ρόλο στην επίδοση του SpM×V διαδραματίζουν οι διαθέσιμες κρυφές μνήμες (*caches*), επειδή είναι πρόβλημα περιορισμένης επίδοσης λόγω μνήμης (*memory-bound*) (Κεφάλαιο 1.2 σελ. 19). Είναι εύκολο να παρατηρηθεί στο Σχήμα 4.1 ότι για διαφορετικές επιλογές πυρήνων έχουμε διαφορετική διαθέσιμη μνήμη *L2*. Πράγματι, αν χρησιμοποιηθεί ο 1ος και ο 2ος πυρήνας της αρχιτεκτονικής *Intel Harpertown* παρατηρούμε ότι ενεργοποιείται μόνο μία μνήμη *L2*, ενώ στην περίπτωση που χρησιμοποιηθούν ο 1ος και ο 3ος ενεργοποιούνται δύο. Το ίδιο ακριβώς ισχύει και για την αρχιτεκτονική *Intel Dunnington*, όπου παίζει σημαντικό ρόλο πλέον και η μνήμη *L3*. Για τις μετρήσεις με συγκεκριμένο αριθμό νημάτων, γίνεται επιλογή των πυρήνων που θα χρησιμοποιηθούν με τρεις τρόπους, (α′) με αυτόν που έχουμε διαθέσιμες τις λιγότερες δυνατές κρυφές μνήμες και τους λιγότερους δυνατούς επεξεργαστές (ελάχιστοι πόροι), (β′) αυτόν που έχουμε διαθέσιμες τις περισσότερες δυνατές κρυφές μνήμες με τους λιγότερους επεξεργαστές ή ελάχιστες μνήμες και μέγιστος αριθμός επεξεργαστών (μέτριοι πόροι) και (γ′) αυτόν που έχουμε διαθέσιμες τις περισσότερες δυνατές κρυφές μνήμες, με μέγιστο αριθμό επεξεργαστών

(μέγιστοι πόροι). Στους Πίνακες 4.3 και 4.4 φαίνονται οι επιλογές επεξεργαστικών στοιχείων που χρησιμοποιούνται στις μετρήσεις για τις αρχιτεκτονικές *Intel Harpertown* και *Intel Dunnington* αντίστοιχα. Η αναπαράσταση των επεξεργαστών (CPUs) και των πυρήνων (Cores) που χρησιμοποιείται βασίζεται στο Σχήμα 4.1.

Νήματα	Ελάχιστοι Πόροι	Μέτριοι Πόροι	Μέγιστοι Πόροι
1	CPU0(C0)		
2	CPU0(C0,C1)	CPU0(C0,C2)	CPU0-CPU1(C0)
4	CPU0(C0,C1,C2,C3)	CPU0-CPU1(C0,C1)	CPU0-CPU1(C0,C2)
8	CPU0-CPU1(C0,C1,C2,C3)		

Πίνακας 4.3: Επιλογή επεξεργαστικών στοιχείων με χρησιμοποίηση ελάχιστων, μέτριων και μέγιστων πόρων σε Intel Harpertown αρχιτεκτονική.

Νήματα	Ελάχιστοι Πόροι	Μέτριοι Πόροι	Μέγιστοι Πόροι
1	CPU0(C0)		
2	CPU0(C0,C1)	CPU0(C0,C2)	CPU0-CPU1(C0)
6	CPU0(Όλοι)	CPU0-CPU1(C0,C2,C4)	CPU0-CPU1(C0,C2), CPU2-CPU3(C0)
12	CPU0-CPU1(Όλοι)	-	CPU0-CPU1-CPU2-CPU3(C0,C2,C4)
24	CPU0-CPU1-CPU2-CPU3(Όλοι)		

Πίνακας 4.4: Επιλογή επεξεργαστικών στοιχείων με χρησιμοποίηση ελάχιστων, μέτριων και μέγιστων πόρων σε Dunnington αρχιτεκτονική.

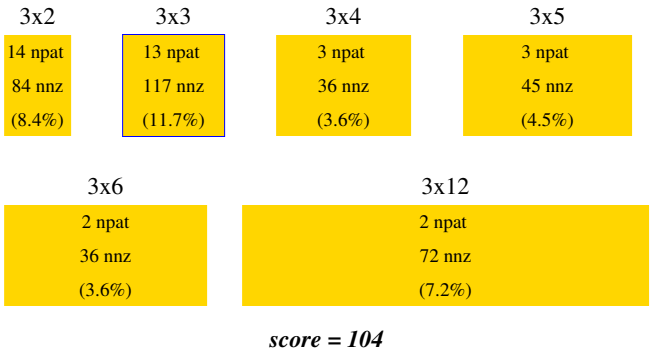
Οι μετρήσεις έγιναν σε λειτουργικό σύστημα *GNU/Linux 2.6* με χρήση του μεταγλωττιστή *g++ (v 4.5)*, ενώ για την δημιουργία του κώδικα στον χρόνο εκτέλεσης χρησιμοποιήθηκε η υποδομή *LLVM 2.7*. Η παραλληλοποίηση επιτυγχάνεται με απευθείας ανάθεση σε νήματα χρησιμοποιώντας *pthreads* και την κλήση συστήματος *sched_setaffinity()* για την ανάθεση των νημάτων σε καθορισμένους πυρήνες.

4.3 Επέκταση Διαχωρισμού Μπλοκ (*CSX-split*)

Στο σχήμα αποθήκευσης CSX, όπως ειπώθηκε και στην ενότητα 3.3.2, μετά την εξόρυξη των στατιστικών στοιχείων για κάθε τύπο κανονικοτήτων του αραιού πίνακα, οι υπο-τύποι κανονικοτήτων που δεν καλύπτουν ένα συγκεκριμένο ποσοστό των συνολικών μη-μηδενικών στοιχείων του πίνακα (min_perc) αφαιρούνται από τα στατιστικά. Έτσι, υπάρχει περίπτωση ένα μέρος των μη-μηδενικών στοιχείων που καλύπτει ένας τύπος κανονικοτήτων να μην ληφθεί υπόψιν για την επιλογή της κωδικοποίησης που θα εφαρμοστεί. Στις μονοδιάστατες κανονικότητες φαίνεται ότι αυτή η επίδραση δεν είναι πολύ σημαντική. Παρόλα αυτά στα μπλοκ μπορεί, εξαιτίας του κατωφλίου που εισάγεται, να κοπεί ένα μεγάλο μέρος από τα μη-μηδενικά στοιχεία που καλύπτει ένας τύπος κανονικοτήτων.

Στο Σχήμα 4.2 φαίνεται ένα χαρακτηριστικό παράδειγμα από την εξόρυξη στατιστικών στοιχείων με ελάχιστο κατώφλι 10% για τον τύπο $block\_r3$ σε ένα πίνακα με 1000 μη-μηδενικά στοιχεία. Παρατηρούμε ότι, ενώ συνολικά ο τύπος $block\_r3$ καλύπτει το 39% των μη-μηδενικών στοιχείων του πίνακα, στην αξιολόγηση του τύπου χρησιμοποιείται μόνο ένας υπο-τύπος που καλύπτει μόλις το $11,7\%$, αφού οι υπό-

λοιποι δεν ξεπερνούν το ελάχιστο κατώφλι. Αυτό έχει ως αποτέλεσμα ο συγκεκριμένος τύπος να υποβαθμίζεται από την ευριστική μέθοδο στην επιλογή κωδικοποίησης, εφόσον έχει πολύ χαμηλό σκορ. Καταληκτικά, μπορεί να ειπωθεί ότι δεν γίνεται αποτελεσματική αξιολόγηση στα μπλοκ, με αποτέλεσμα, σε πολλές περιπτώσεις παρόλο που η αξιοποίηση τους οδηγεί σε αποδοτική συμπίεση του αραιού πίνακα, να μην αποτελούν τις κυρίαρχες επιλογές για κωδικοποίηση.



Σχήμα 4.2: Στατιστικά στοιχεία και αξιολόγηση για τύπο $block\_r3$ στο CSX.

Το συγκεκριμένο πρόβλημα που προκύπτει επιλύεται σε μεγάλο βαθμό από την επέκταση *CSX-split*. Αφού συλλεχθούν τα στατιστικά στοιχεία για έναν δισδιάστατο τύπο κανονικοτήτων, εφαρμόζεται μια επιπλέον συνάρτηση *SplitBlocks()*. Πιο συγκεκριμένα, ξεκινώντας από τον υπο-τύπο κανονικοτήτων με τη μεγαλύτερη δευτερεύουσα διάσταση και πηγαίνοντας προς αυτόν με την μικρότερη, διαχωρίζονται τα μεγαλύτερα μπλοκ του συγκεκριμένου τύπου (για τα οποία υπάρχουν στατιστικά στοιχεία) σε μπλοκ μεγέθους ίσου με του υπο-τύπου που ελέγχεται και παρατηρείται αν ο συγκεκριμένος υπο-τύπος έχει ξεπεράσει το ελάχιστο κατώφλι min_perc. Σε περίπτωση που δεν έχει ξεπεραστεί το κατώφλι συνεχίζεται η διαδικασία με τον αμέσως μικρότερο σε μέγεθος υπο-τύπο. Σε αντίθετη περίπτωση ο συγκεκριμένος υπο-τύπος λαμβάνεται υπόψιν στην αξιολόγηση και τα μεγαλύτερα σε μεγέθη μπλοκ που δεν έχουν επιλεχθεί αφαιρούνται από τα στατιστικά στοιχεία. Στον Αλγόριθμο 18 φαίνεται αναλυτικά η συγκεκριμένη λειτουργία.

Στο Σχήμα 4.3 φαίνεται το προηγούμενο παράδειγμα για την επέκταση *CSX-split*. Στο Σχήμα (α′) φαίνονται οι αρχικές μετρήσεις που έχουν γίνει για τον τύπο $block\_r3$. Τα μπλοκ 3×12 δεν ξεπερνούν το ελάχιστο κατώφλι 10% που έχει τεθεί. Τα μπλοκ 3×6 αν συνυπολογιστούν και αυτά που υπάρχουν εσωτερικά των μπλοκ 3×12 ξεπερνούν το κατώφλι, οπότε διαχωρίζουμε τα 3×12 μπλοκ σε κομμάτια των 3×6, όπως φαίνεται και στο Σχήμα (β′). Ακολουθείται η ίδια διαδικασία και για τα υπόλοιπα μπλοκ μέχρι όλοι οι υπο-τύποι που παραμένουν να ξεπερνούν το 10%. Στο Σχήμα (ε′) φαίνεται ότι από το 39% των μη-μηδενικών στοιχείων του πίνακα που καλύπτει συνολικά ο τύπος $block\_r3$, στην αξιολόγηση του τύπου χρησιμοποιούνται πλέον τρεις υπο-τύποι που καλύπτουν το 38.3%. Το ποσοστό αυτό είναι πολύ καλύτερο από το αντίστοιχο για το CSX το οποίο ήταν 11.7%. Αυτό έχει ως αποτέλεσμα ο συγκεκριμένος τύπος να αξιολογείται αρκετά καλύτερα και να έχει περισσότερες πιθανότητες να επιλεχθεί.

Αλγόριθμος 18: Διαχωρισμός μπλοκ.

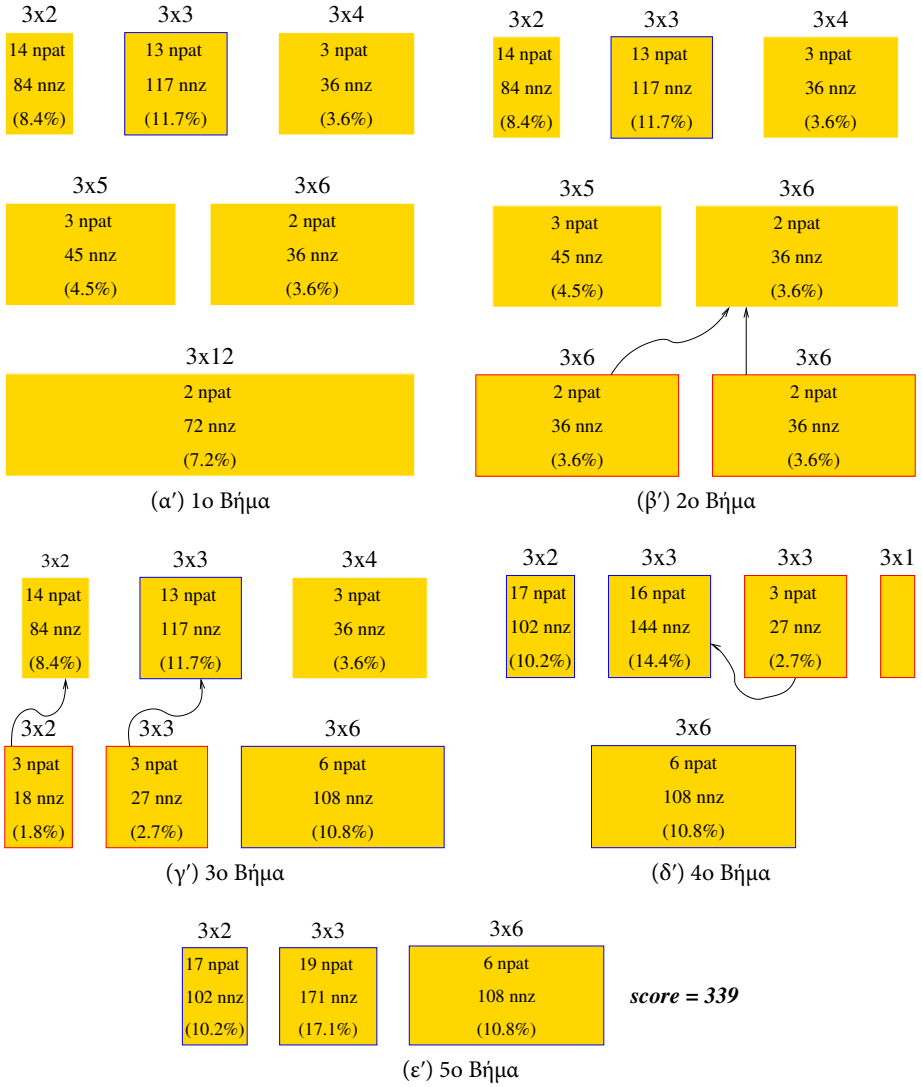
Input: $data$, statistical data of a block type
Input: min_perc, minimum treshold percentage for sub-types

$data \leftarrow Sort(data)$
$first\_subtype \leftarrow GetFirstElement(data)$
$current\_subtype \leftarrow first\_subtype$
while $current\_subtype \neq NULL$ **do**
  $nnz\_perc \leftarrow GetCoveragePerc(current\_subtype)$
  $cur\_dimsize \leftarrow GetOtherDimension(current\_subtype)$
  for $temp \leftarrow first\_subtype$ **to** $current\_subtype$ **do**
    $temp\_dimsize \leftarrow GetOtherDimension(temp)$
    $mul \leftarrow 1 - (temp\_dimsize \mod cur\_dimsize)/temp\_dimsize$
    $nnz\_perc \leftarrow nnz\_perc + GetCoveragePerc(temp) \times mul$
  if $nnz\_perc \geq min\_perc$ **then**
    for $temp \leftarrow first\_subtype$ **to** $current\_subtype$ **do**
      $SplitSubtypeTo(temp, current\_subtype)$
    $first\_subtype \leftarrow GetNextElement(current\_subtype)$
  $current\_subtype \leftarrow GetNextElement(current\_subtype)$

Συμπερασματικά, με την συγκεκριμένη επέκταση (CSX-split), κωδικοποιούνται στοιχεία τα οποία ανήκουν σε μεγάλα μπλοκ, που αποκόπτονται κανονικά από το απλό CSX. Αυτό συμβαίνει επειδή η ύπαρξη του κατωφλίου (min_perc) δεν επιτρέπει την κωδικοποίηση τους. Σε περίπτωση που μειωθεί το κατώφλι ώστε να συμπεριληφθούν όλα αυτά τα μπλοκ θα αυξηθεί πολύ ο αριθμός των υπο-τύπων, πράγμα το οποίο θα αυξήσει υπερβολικά των χρόνο υπολογισμών (Κεφάλαιο 3.3.4 σελ. 41). Το CSX-split έχει την δυνατότητα να αξιοποιεί αυτές τις κανονικότητες (μεγάλα μπλοκ) χωρίς να μειώνει το κατώφλι και αυξάνοντας ελάχιστα τον αριθμό των υπο-τύπων που χρησιμοποιούνται. Έτσι, οι αραιοί πίνακες συμπιέζονται ακόμα περισσότερο, γεγονός που επιφέρει κέρδος στην επίδοση του SpM×V. Επίσης, το CSX-split βρίσκει καλύτερα αποτελέσματα για μπλοκ και αυτό έχει ως αποτέλεσμα να αποτελούν σε περισσότερες περιπτώσεις (από το CSX) την πρώτη επιλογή της ευριστικής μεθόδου, το οποίο είναι πολύ θετικό εξαιτίας των πολύ καλών υπολογιστικών χαρακτηριστικών τους.

4.4 Μετρήσεις

Οι μετρήσεις που έγιναν έχουν ως σκοπό να δείξουν την βελτίωση στην επίδοση του SpM×V με την επέκταση CSX-split. Για τις μετρήσεις επιλέχθηκαν οι πίνακες από την σουίτα που περιγράφηκε στο Κεφάλαιο 4.1 που είχαν σημαντικές αλλαγές στην επιλογή των τύπων προς όφελος των μπλοκ. Πιο συγκεκριμένα, επιλέχθηκαν οι πίνακες για τους οποίους τα μπλοκ καλύπτουν τουλάχιστον 20% (επί των συνολικών) περισσότερα μη-μηδενικά στοιχεία στην επέκταση CSX-split σε σχέση με το απλό CSX. Συνολικά, για τις συγκεκριμένες μετρήσεις χρησιμοποιήθηκαν 10 από τους 48

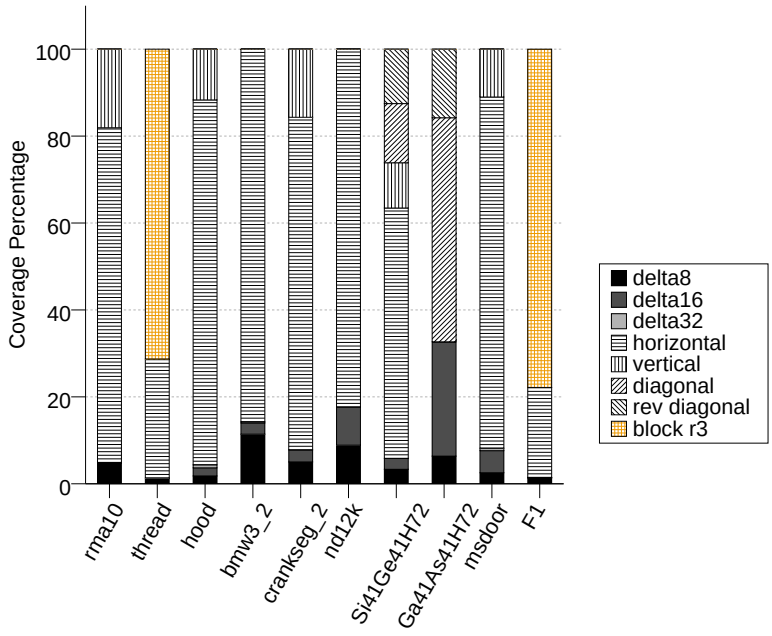


Σχήμα 4.3: Στατιστικά στοιχεία και αξιολόγηση για τύπο $block\_r3$ στο CSX-split.

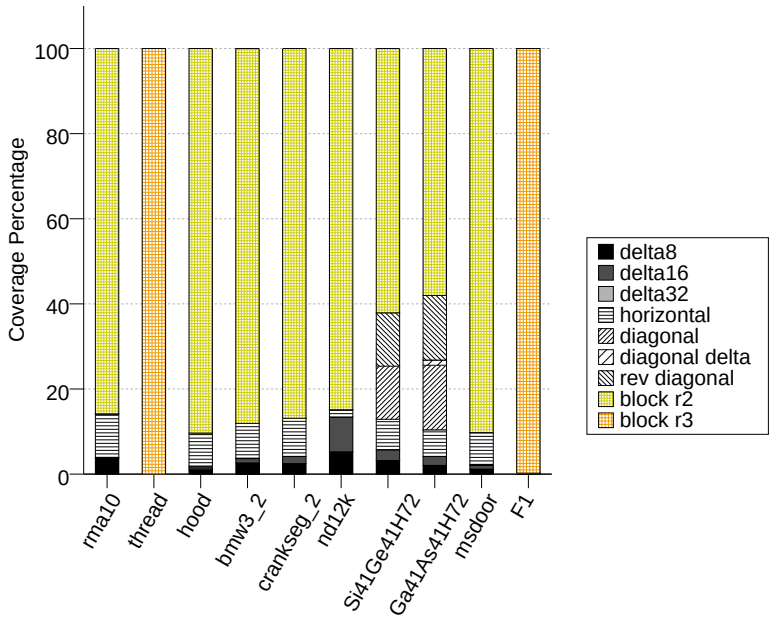
αραιούς πίνακες της σουίτας που έχουν το συγκεκριμένο χαρακτηριστικό. Οι πίνακες αυτοί, φαίνονται στο Σχήμα 4.4, μαζί με την κάλυψη από τύπους κανονικοτήτων που έχουν (α′) στο CSX και (β′) στο CSX-split. Αξίζει να σημειωθεί ότι το ποσοστό που χρησιμοποιείται στο CSX για κατώφλι είναι το 10%, ενώ στο CSX-split είναι 1%.

Στο Σχήμα 4.5 φαίνονται τα αποτελέσματα των μετρήσεων στους 10 πίνακες που επιλέχθηκαν. Οι μετρήσεις έγιναν σε αρχιτεκτονική *Harpertown* και στα 2 και 4 νήματα φαίνονται τα αποτέλεσμα όταν γίνονται επιλογές με χρησιμοποίηση μέγιστων δυνατών πόρων (επεξεργαστές και μνήμες). Φαίνεται ότι το CSR είναι καλύτερο στη σειριακή εκτέλεση, αλλά αυτό οφείλεται στον επιπλέον χρόνο που απαιτείται για την αποσυμπίεση των δεδομένων (Κεφάλαιο 1.2 σελ.19). Επίσης, είναι ξεκάθαρο ότι το

CSX-split είναι καλύτερο από το CSX σε επίδοση για οποιαδήποτε επιλογή νημάτων από τις τέσσερις δυνατές. Πιο συγκεκριμένα, με την επέκταση CSX-split αυξάνεται η επίδοση του CSX σχεδόν κατά 10% στα 1 και 2 νήματα και κατά 5% στα 4 και 8 νήματα και η διαφορά επί του σειριακού CSR είναι περίπου σταθερή και ίση με 12% για όλες τις επιλογές νημάτων. Τέλος, το γεγονός ότι ο χρόνος μετατροπής ενός αραιού πίνακα σε CSX μορφή δεν αυξήθηκε σχεδόν καθόλου με την επέκταση CSX-split κάνουν τα αποτελέσματα αυτά ακόμα πιο σημαντικά, εφόσον επιτυγχάνονται με μηδαμινό κόστος.

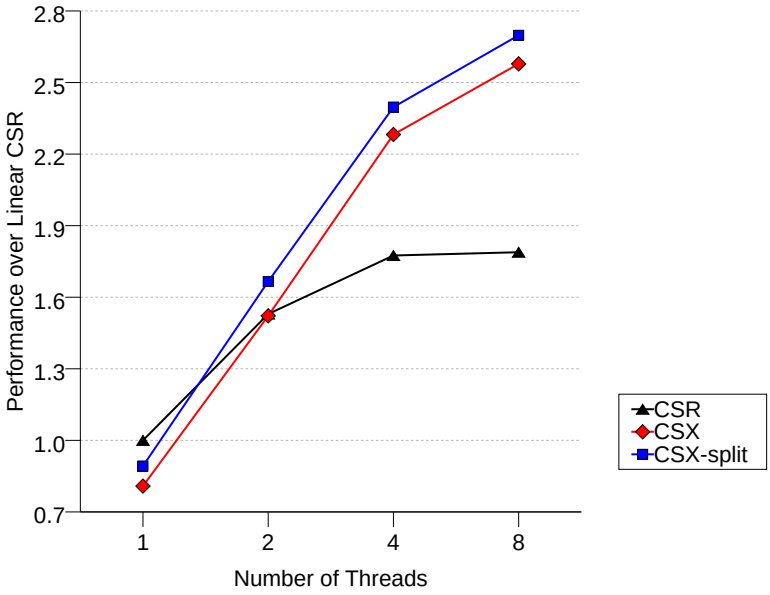


(α′) *CSX*



(β′) *CSX-split*

Σχήμα 4.4: Κάλυψη πινάκων με τύπους κανονικοτήτων.



Σχήμα 4.5: Επίδοση SpM×V για 1,2,4,8 νήματα επί της σειριακής εκτέλεσης του CSR.

Κεφάλαιο 5

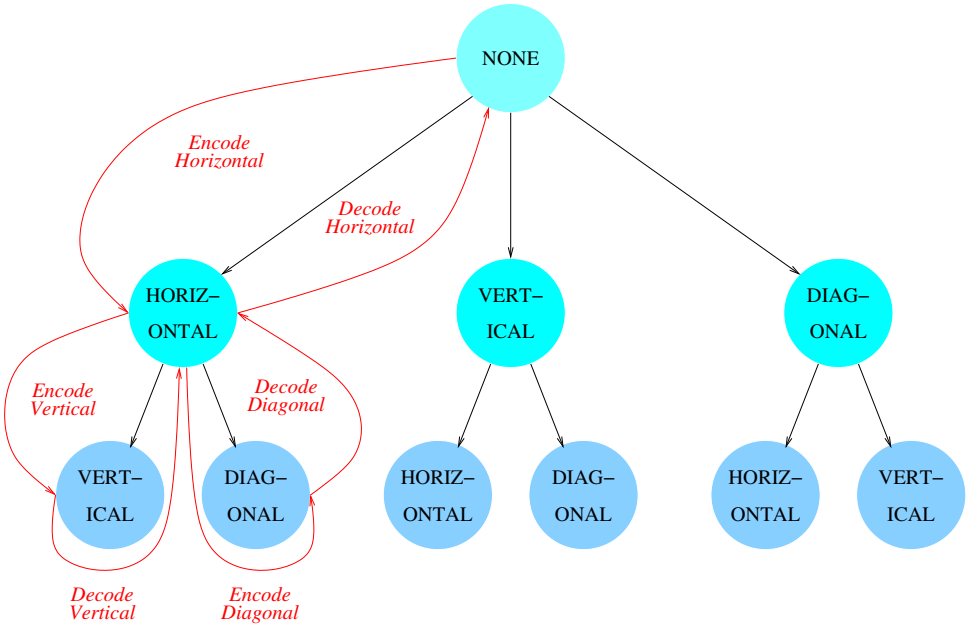
Μελέτη και Αξιολόγηση του Σχήματος Αποθήκευσης CSX

Στο Κεφάλαιο αυτό γίνεται μια αξιολόγηση του σχήματος αποθήκευσης CSX μέσα από εξαντλητικές μετρήσεις για όλες τις πιθανές κωδικοποιήσεις. Πιο συγκεκριμένα, παράγονται αρχικά όλες οι δυνατές ακολουθίες τύπων κανονικοτήτων του CSX για τους αραιούς πίνακες με μια *Depth-First Search* (*DFS*) αναζήτηση. Έπειτα, για όλες αυτές τις ακολουθίες γίνεται η αντίστοιχη κωδικοποίηση και μετριέται η επίδοση του SpM×V με απώτερο σκοπό την εξαγωγή χρήσιμων συμπερασμάτων που θα βοηθήσουν στην περαιτέρω κατανόηση του προβλήματος και, συνεπώς, στην βελτίωση του. Περισσότερο, εξετάζεται η ευριστική μέθοδος και το πόσο καλή είναι η αναπαράσταση του αραιού πίνακα που επιλέγει σε σχέση με τις υπόλοιπες.

5.1 Αναζήτηση DFS στο Πεδίο των Κωδικοποιήσεων του CSX

Η λογική πίσω από την υλοποίηση της αναζήτησης είναι πολύ απλή. Σκοπός της αναζήτησης αυτής είναι να βρει όλα τα δυνατά μονοπάτια τύπων κανονικοτήτων ενός πίνακα και όχι μόνο αυτό που επιλέγει η ευριστική. Αρχικά, ξεκινώντας από την ρίζα του δέντρου αναζήτησης (καμία κωδικοποίηση), συλλέγονται στατιστικά στοιχεία για τον πίνακα και δημιουργούνται κόμβοι με όλους τους τύπους κανονικοτήτων που έχουν βρεθεί. Έπειτα, γίνεται η κωδικοποίηση του τύπου κανονικοτήτων του πρώτου κόμβου που δημιουργήθηκε και επαναλαμβάνεται η διαδικασία μέχρι να βρεθεί κάποιος κόμβος στον οποίο δεν υπάρχουν άλλες δυνατές κωδικοποιήσεις (κόμβος-φύλλο). Σε αυτό το σημείο σημειώνεται το μονοπάτι που οδήγησε στο συγκεκριμένο κόμβο και γίνεται αποκωδικοποίηση του τελευταίου τύπου κανονικοτήτων και συνεχίζεται η διαδικασία από τον προηγούμενο κόμβο μέχρι να μην υπάρχει κόμβος που να μην έχει εξεταστεί.

Στο Σχήμα 5.1 φαίνεται αναλυτικά η διαδικασία για τον πρώτο κλάδο του δέντρου κωδικοποιήσεων του πίνακα *rajat30* (δες Πίνακα 4.1 σελ. 46). Στο συγκεκριμένο πίνακα, παρόλο που τα τελικά μονοπάτια του είναι μόνο έξι, χρειάζονται αρκετές κωδικοποιήσεις και αποκωδικοποιήσεις οι οποίες κοστίζουν πολύ σε χρόνο. Οι



Σχήμα 5.1: Αναζήτηση DFS στο πεδίο των κωδικοποιήσεων.

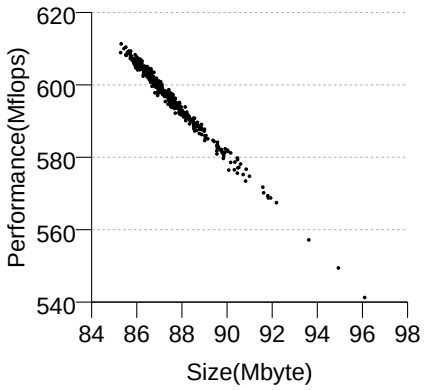
περισσότεροι πίνακες έχουν πολύ παραπάνω τελικά μονοπάτια (ξεπερνούν τις 1000 σε κάποιες περιπτώσεις) κάνοντας την διαδικασία αυτή πολύ χρονοβόρα, ιδιαίτερα για τους πολύ μεγάλους πίνακες. Ωστόσο, είναι θεμιτό ο πίνακας να έχει όσο το δυνατόν περισσότερα μονοπάτια, ώστε να υπάρχει ποικιλία μετρήσεων που οδηγεί σε καλύτερη εξαγωγή συμπερασμάτων. Γι'αυτόν το λόγο επιλέγονται πίνακες που έχουν πάνω από 100 μονοπάτια και που δεν ξεπερνούν ένα συγκεκριμένο μέγεθος έτσι ώστε να ολοκληρώνονται οι μετρήσεις σχετικά γρήγορα.

5.2 Μετρήσεις για Αξιολόγηση CSX

Τα αποτελέσματα των μετρήσεων έδειξαν ότι υπάρχουν τρεις διαφορετικές περιπτώσεις που θα εξεταστούν στη συνέχεια. Στην πρώτη περίπτωση, ανήκουν μετρήσεις που τρέχουν σε περισσότερα από 2 νήματα και που το μέγεθος του αραιού πίνακα ξεπερνά κατά πολύ (περισσότερο από 25%) το διαθέσιμο μέγεθος της κρυφής μνήμης $L2$. Έτσι, ο χρόνος προσπέλασης της μνήμης υπερκαλύπτει τον χρόνο των αριθμητικών υπολογισμών και μοναδικό ρόλο στην επίδοση παίζει το μέγεθος της αναπαράστασης του πίνακα (Σχήμα 5.2). Στην δεύτερη περίπτωση μετρήσεων, το μέγεθος εξακολουθεί να παίζει το σπουδαιότερο ρόλο, αλλά σημαντικό ρόλο πλέον παίζουν και οι αριθμητικοί υπολογισμοί. Παραδείγματα αυτής της κατηγορίας είναι πίνακες που τρέχουν με ένα ή δύο νήματα (Σχήμα 5.3), καθώς και πίνακες που το μέγεθος τους κυμαίνεται μεταξύ του 80% και του 125% της διαθέσιμης κρυφής μνήμης $L2$ (Σχήμα 5.4). Στην τελευταία κατηγορία, ανήκουν οι αραιοί πίνακες που δεν ξεπερνούν σε μέγεθος το 80% της κρυφής μνήμης $L2$ και το μέγεθος της αναπαράστασης παίζει ελάχιστο

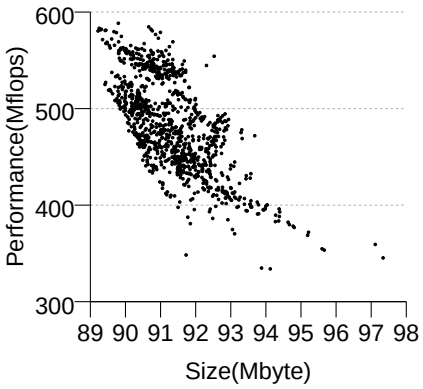
ή καθόλου ρόλο, με τους αριθμητικούς υπολογισμούς να είναι σχεδόν αποκλειστικά υπεύθυνοι για την επίδοση της εκάστοτε αναπαράστασης του πίνακα (Σχήμα 5.5). Στο σημείο αυτό αξίζει να σημειωθεί ότι υπάρχουν κάποιες κωδικοποιήσεις (ιδιαίτερα στα πολλά νήματα), οι οποίες λειτουργούν πολύ αρνητικά στην επίδοση. Όμως, επειδή δεν ήταν δυνατόν με την συγκεκριμένη ανάλυση να εξηγηθούν οι συγκεκριμένες συμπεριφορές, αποκόπτονται από τα διαγράμματα και γίνεται ανάλυση με τις υπόλοιπες κωδικοποιήσεις.

(α′) Harpertown, Si41Ge41H72, 8 νήματα

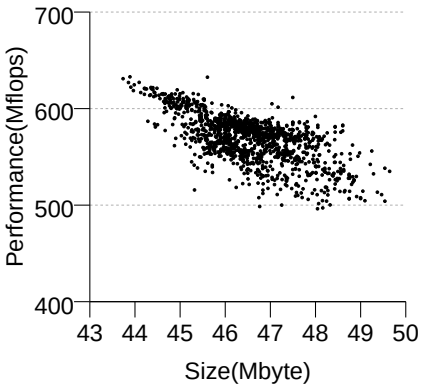


(β′) Dunnington, hood, 6 νήματα, ελάχιστοι πόροι

Σχήμα 5.2: Μετρήσεις που το μέγεθος παίζει κυρίαρχο ρόλο στην επίδοση.

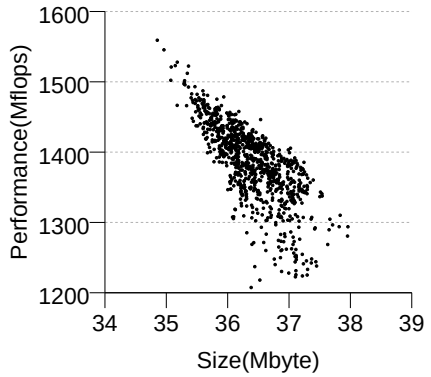


(α′) Harpertown, bmw3_2, 1 νήμα

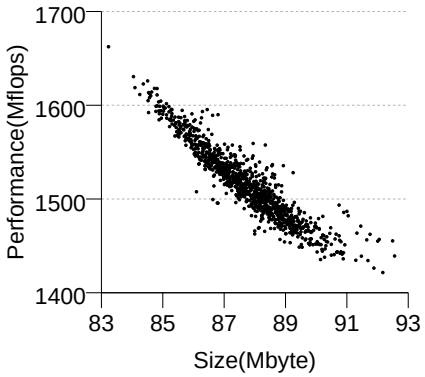


(β′) Dunnington, largebasis, 2 νήματα, μέγιστοι πόροι

Σχήμα 5.3: Μετρήσεις με λίγα νήματα.

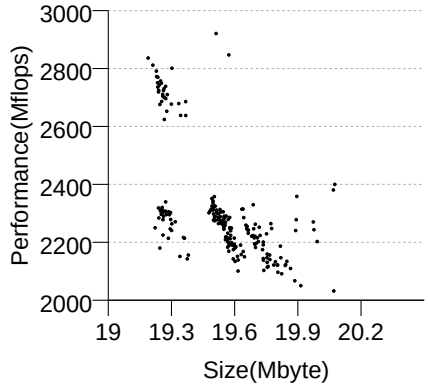


(α′) Harpertown, thread, 4 νήματα, μέγιστοι πόροι

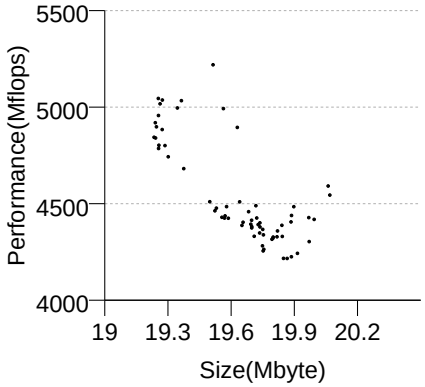


(β′) Dunnington, bmwcra_1, 6 νήματα, μέγιστοι πόροι

Σχήμα 5.4: Μετρήσεις που το μέγεθος του πίνακα πλησιάζει το διαθέσιμο μέγεθος κρυφής μνήμης.



(α′) Harpertown, rma10, 8 Νήματα



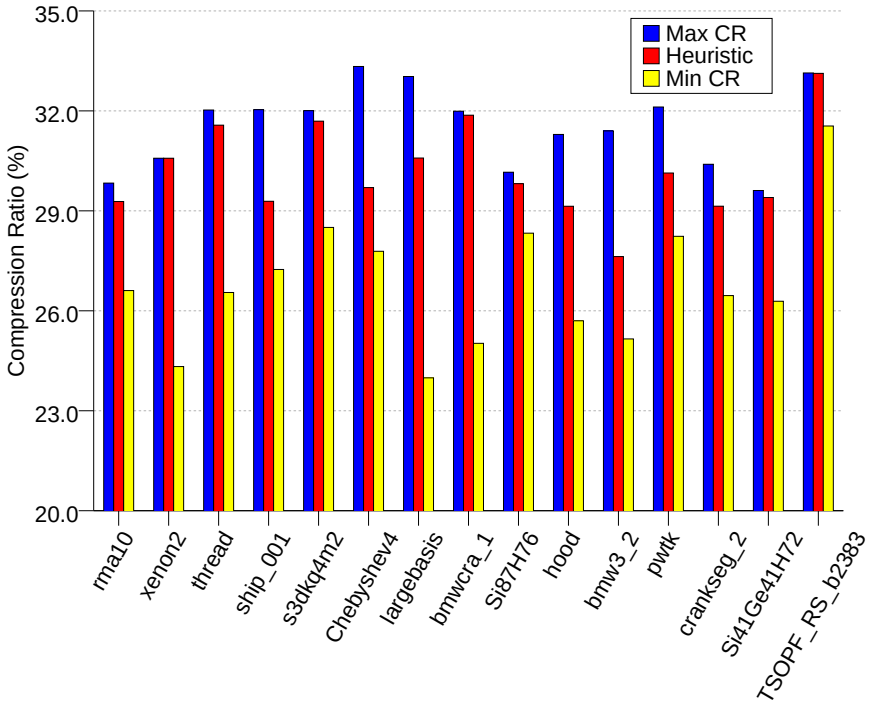
(β′) Dunnington, rma10, 24 Νήματα

Σχήμα 5.5: Μετρήσεις που το μέγεθος δεν παίζει σημαντικό ρόλο στην επίδοση.

Στόχος αποτελεί η βελτίωση του SpM×V, όταν ο αραιός πίνακας δεν χωράει στην κρυφή μνήμη. Ωστόσο, πιθανή βελτίωση και στην τρίτη περίπτωση δεν είναι ανεπιθύμητη. Οπότε, αρχικά μελετάται η επίδραση του μεγέθους της αναπαράστασης του πίνακα στην επίδοση και η επιλογή της ευριστικής με βάση το συγκεκριμένο χαρακτηριστικό. Έπειτα, αξιολογείται ο ρόλος που παίζουν οι αριθμητικοί υπολογισμοί των διάφορων τύπων κανονικοτήτων στην επίδοση του SpM×V και το πόσο καλά λαμβάνεται υπόψη αυτό από την ευριστική μέθοδο. Τέλος, παρουσιάζεται μια ανάλυση για την επιλογή των νημάτων, ώστε να έχουμε μια απόδοση πολύ κοντά στην βέλτιστη χωρίς να χρησιμοποιούνται όλοι οι πόροι για εξοικονόμηση ενέργειας.

5.2.1 Αξιολόγηση με Βάση το Μέγεθος

Το σημαντικότερο ρόλο στο SpM×V, όταν ο αραιός πίνακας δεν χωράει στην κρυφή μνήμη, παίζει το μέγεθος της αναπαράστασης του. Επομένως, το μέγεθος πρέπει να είναι η σημαντικότερη παράμετρος που θα πρέπει να υπολογίζει η ευριστική μέθοδος. Στο Σχήμα 5.6 φαίνεται πόσο καλά συμπιέζει τον πίνακα η αναπαράσταση που επιλέγεται από την ευριστική σε σχέση με την καλύτερη και τη χειρότερη δυνατή.

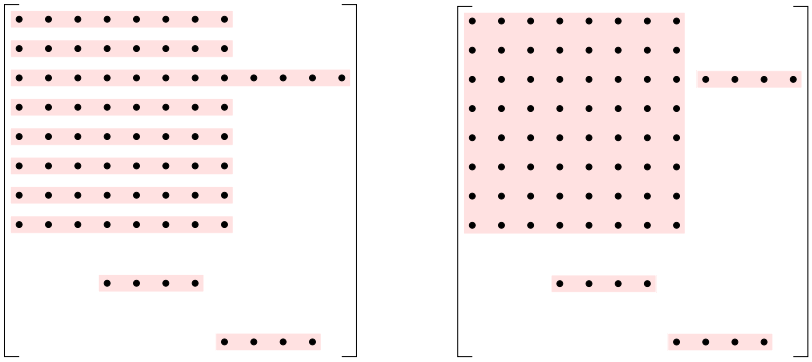


Σχήμα 5.6: Ποσοστό συμπίεσης για τις κωδικοποιήσεις της ευριστικής μεθόδου, της καλύτερης και της χειρότερης δυνατής περίπτωσης.

Είναι φανερό ότι η άπληστη λογική που υιοθετεί η ευριστική μέθοδος δεν πετυχαίνει πάντα την ιδανική συμπίεση. Σε μερικές περιπτώσεις μάλιστα απέχει αρκετά από την καλύτερη δυνατή συμπίεση. Πιο συγκεκριμένα, υπάρχουν πίνακες, όπως ο $Chebyshev4$ και ο $rajat30$, που μπορεί να επιτευχθεί παραπάνω από 5% καλύτερη συμπίεση από αυτήν της ευριστικής μεθόδου. Αυτό συμβαίνει διότι σε κάθε βήμα επιλέγεται να κωδικοποιηθεί ο τύπος κανονικοτήτων που επιφέρει την καλύτερη δυνατή συμπίεση με τα δεδομένα εκείνης της στιγμής (άπληστος αλγόριθμος), χωρίς αυτό να σημαίνει ότι η τελική ακολουθία τύπων που επιλέγεται είναι η ιδανικότερη.

Στο Σχήμα 5.7 παρουσιάζεται ένα απλό παράδειγμα στο οποίο η ευριστική μέθοδος δεν κάνει την καλύτερη δυνατή επιλογή. Είναι φανερό ότι η αναπαράσταση του πίνακα που προκύπτει από την ευριστική μέθοδο καλύπτει όλα τα στοιχεία με δέκα κανονικότητες, ενώ στην καλύτερη δυνατή περίπτωση καλύπτονται μόλις από τέσσερις. Αυτό γίνεται διότι, αρχικά, η ευριστική βρίσκει σκορ για τις οριζόντιες κανονικότητες

$nnz_{enc} - npatterns = 76 - 10 = 66$, ενώ για τα μπλοκ μεγέθους 8 (είτε είναι ευθυγραμμισμένα κατά γραμμές είτε κατά στήλες) βρίσκει $nnz_{enc} - npatterns = 64 - 1 = 63$. Οπότε κωδικοποιούνται οι οριζόντιες κανονικότητες, που καλύπτουν όλα τα στοιχεία του πίνακα, και προκύπτει η τελική αναπαράσταση του πίνακα. Παρόλα αυτά αν επιλεγούν τα μπλοκ, κωδικοποιηθούν και σε ένα δεύτερο βήμα επιλεγούν και κωδικοποιηθούν και οι εναπομείναντες οριζόντιες κανονικότητες, τα αποτελέσματα είναι πολύ καλύτερα.

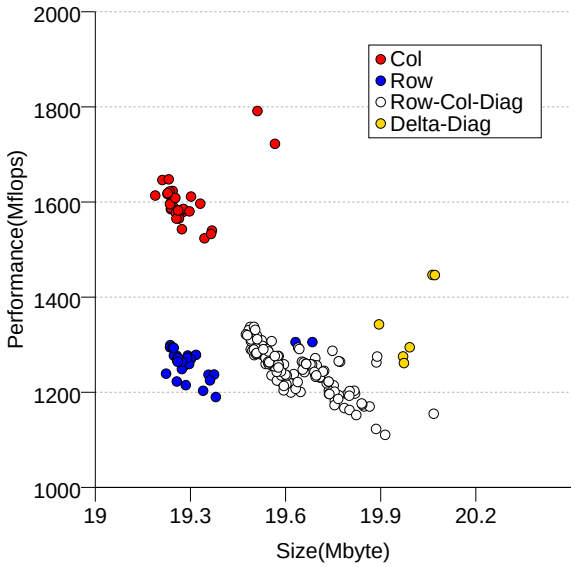


(α′) Ευριστική μέθοδος (β′) Ιδανική περίπτωση

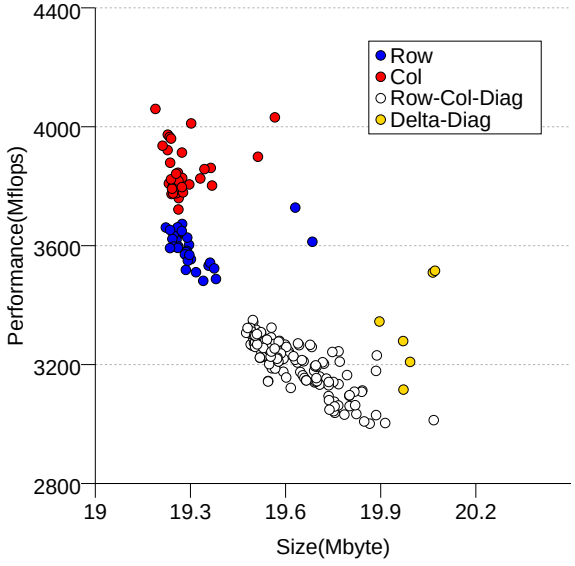
Σχήμα 5.7: Επιλογές για κωδικοποίηση πάνω σε έναν αραιό πίνακα.

5.2.2 Αξιολόγηση με Βάση τους Αριθμητικούς Υπολογισμούς

Σημαντικό ρόλο σε πίνακες που πλησιάζουν το μέγεθος της διαθέσιμης κρυφής μνήμης ή τρέχουν με λίγα νήματα, παίζουν οι αριθμητικοί υπολογισμοί. Για να μελετηθεί καλύτερα η επίδραση της ακολουθίας κωδικοποιήσεων, που επιλέγεται, στον χρόνο των αριθμητικών υπολογισμών, μελετάμε τους πίνακες που το μέγεθος της αναπαράστασής τους δεν ξεπερνά το μέγεθος της διαθέσιμης κρυφής μνήμης (τρίτη κατηγορία μετρήσεων). Για να γίνει η μελέτη αυτή, τα μη-μηδενικά στοιχεία του πίνακα χωρίζονται σε στοιχεία που διατρέχονται (α′) κατά γραμμές (οριζόντιες κανονικότητες και μπλοκ ευθυγραμμισμένα κατά στήλες), (β′) κατά στήλες (κατακόρυφες κανονικότητες και μπλοκ ευθυγραμμισμένα κατά γραμμές), (γ′) κατά διαγώνιες (διαγώνιες και αντι-διαγώνιες) και (δ′) σε στοιχεία που δεν έχουν κωδικοποιηθεί (αποστάσεις Δ). Ο διαχωρισμός προέκυψε μετά από την παρατήρηση ότι οι περιπτώσεις αυτές, μπορούν να διαδραματίσουν σημαντικό ρόλο στον χρόνο της εκτέλεσης του SpM×V. Για τον διαχωρισμό, υλοποιήθηκε αλγόριθμος ομαδοποίησης (clustering), που παίρνει ως παραμέτρους τα τέσσερα ποσοστά εμφάνισης των στοιχείων που ανήκουν στις παραπάνω περιπτώσεις, και με βάση την ευκλείδεια απόσταση μεταξύ των ακολουθιών τις κατατάσσει σε ομάδες. Στο Σχήμα 5.8 φαίνεται η επίδραση που έχει η επιλογή των κωδικοποιήσεων στην επίδοση του SpM×V σε δύο τέτοιες περιπτώσεις.



(α′) Harpertown, rma10, 4 νήματα, μέγιστοι πόροι



(β′) Dunnington, rma10, 12 νήματα, μέγιστοι πόροι

Σχήμα 5.8: Επίδραση της ακολουθίας κωδικοποιήσεων στον χρόνο εκτέλεσης του SpM×V.

Παρατηρούμε ότι η κωδικοποίηση πολλών οριζόντιων ή πολλών ευθυγραμμισμένων κατά στήλες μπλοκ ή και των δύο (μπλε χρώμα στα διαγράμματα), δεν επιφέρει καλά αποτελέσματα. Αυτό συμβαίνει, διότι, υπάρχει διάβασμα μετά από εγγραφή (εξάρτηση τύπου *Read After Right* ή *RAW*) στη μεταβλητή εξόδου y (Αλγόριθμοι 12,

17 σελ. 43), γεγονός που δημιουργεί πρόβλημα στην ροή των αριθμητικών πράξεων και αυξάνει τον χρόνο εκτέλεσης. Αξίζει να σημειωθεί ότι, προς το παρόν, το CSX δεν αξιοποιεί όλα τα υπολογιστικά χαρακτηριστικά των μπλοκ (vectorization κ.ο.κ.), πράγμα που σημαίνει ότι είναι πολύ πιθανό τα μπλοκ να επιφέρουν καλύτερη επίδοση σε σχέση με τις οριζόντιες κανονικότητες.

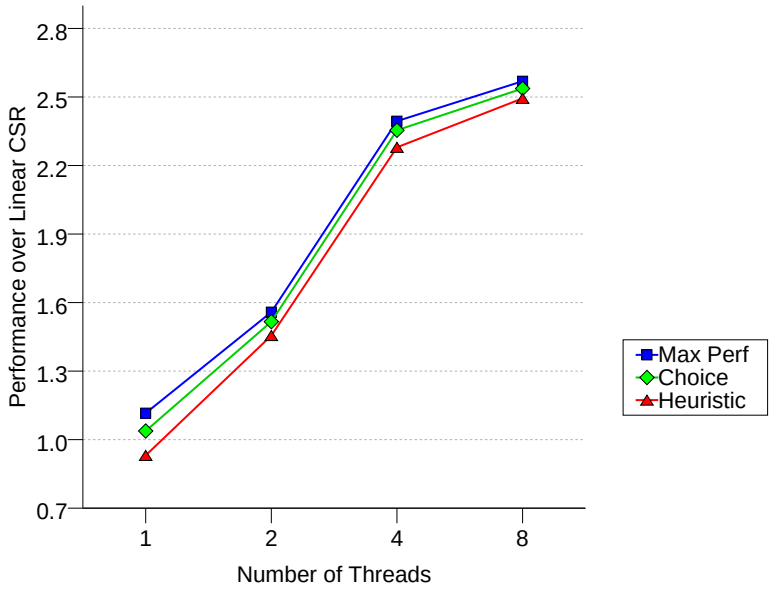
Η κωδικοποίηση πολλών διαγώνιων κανονικοτήτων, επίσης, δεν παράγει πολύ καλά αποτελέσματα (κίτρινο χρώμα στα διαγράμματα). Το γεγονός ότι, για τον υπολογισμό ενός στοιχείου, χρειάζονται περισσότερες προγραμματιστικές εντολές και φορτώνονται περισσότερες νέες μεταβλητές από την μνήμη (Αλγόριθμοι 14, 15 σελ. 43), μειώνει αρκετά την επίδοση.

Τα καλύτερα αποτελέσματα φαίνεται ότι έχουν οι κατακόρυφες κανονικότητες και τα ευθυγραμμισμένα κατά γραμμές μπλοκ (κόκκινο χρώμα στα διαγράμματα). Οι κανονικότητες της συγκεκριμένης κατηγορίας δεν αντιμετωπίζουν κανένα από τα προβλήματα των προηγούμενων δύο (Αλγόριθμοι 13, 16 σελ. 43) και πετυχαίνουν την καλύτερο χρόνο. Όπως και στην περίπτωση των ευθυγραμμισμένων κατά στήλες μπλοκ, και εδώ υπάρχει η δυνατότητα βελτίωσης των υπολογισμών των ευθυγραμμισμένων κατά γραμμές μπλοκ, κάνοντας ακόμα καλύτερη την επίδοση τους.

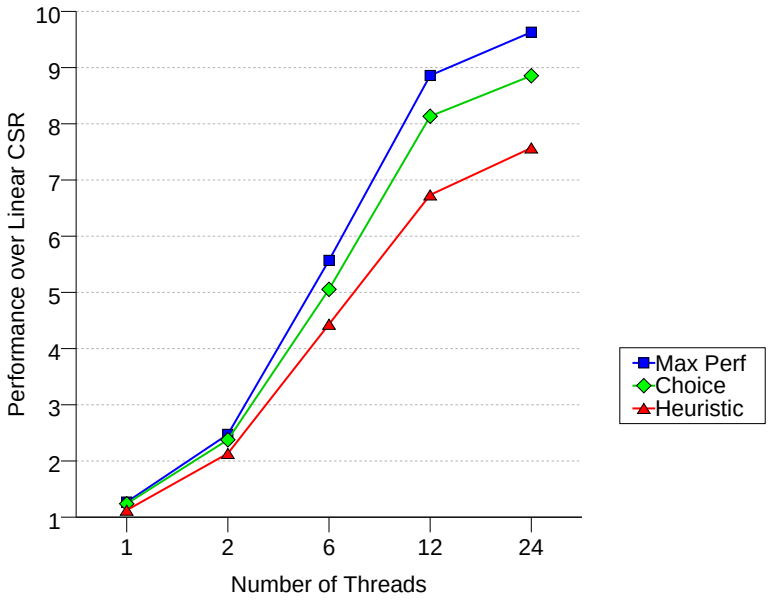
Η ευριστική μέθοδος, που χρησιμοποιεί το CSX, επιλέγει καθαρά με βάση το μέγεθος της αναπαράστασης χωρίς να λαμβάνει καθόλου υπόψη της, τους αριθμητικούς υπολογισμούς, με αποτέλεσμα η επίδοση του SpM×V με το σχήμα CSX να μην είναι αρκετά καλή σε λίγα νήματα και σε περιπτώσεις που η διαθέσιμη κρυφή μνήμη ξεπερνά ή πλησιάζει το μέγεθος του πίνακα (δεύτερη και τρίτη κατηγορία).

Για να αξιολογηθεί αν έχουν βρεθεί οι παράγοντες, που επηρεάζουν την επίδοση του SpM×V, με επιτυχία και για τις τρεις κατηγορίες μετρήσεων, επιλέγεται σε κάθε περίπτωση η ακολουθία κωδικοποιήσεων, η οποία έχει τα επιθυμητά χαρακτηριστικά σε πολύ μεγάλο βαθμό και συγκρίνεται με το καλύτερο αποτελέσματα και με το αποτέλεσμα της ευριστικής μεθόδου. Πιο συγκεκριμένα, για την πρώτη κατηγορία μετρήσεων (πίνακες που τρέχουν σε πολλά νήματα και με μεγάλο μέγεθος αναπαράστασης σε σχέση με την διαθέσιμη κρυφή μνήμη) επιλέγεται η κωδικοποίηση με το μικρότερο μέγεθος αναπαράστασης του πίνακα. Στην δεύτερη κατηγορία μετρήσεων (πίνακες που το μέγεθος της αναπαράστασης τους είναι κοντά στην διαθέσιμη κρυφή μνήμη και πίνακες που τρέχουν σε πολύ λίγα νήματα) επιλέγονται, αρχικά, οι ακολουθίες κωδικοποιήσεων οι οποίες έχουν μέγεθος το πολύ 2% παραπάνω από το μικρότερο δυνατό και μετά από αυτές επιλέγεται η μια με τα καλύτερα υπολογιστικά χαρακτηριστικά με βάση την αξιολόγηση που έγινε προηγουμένως. Αναλυτικότερα, για αρχιτεκτονική Harpertown επιλέγεται η ακολουθία που έχει τα περισσότερα μη-μηδενικά στοιχεία που ανήκουν σε κατακόρυφες κανονικότητες και σε ευθυγραμμισμένα κατά γραμμές μπλοκ (col_elems), ενώ, στην αρχιτεκτονική Dunnington τα στοιχεία αυτά μετριούνται ως διπλά, τα στοιχεία που ανήκουν σε οριζόντιες κανονικότητες και σε ευθυγραμμισμένα κατά στήλες μπλοκ μετριούνται ως μονά, τα υπόλοιπα δεν λαμβάνονται υπόψιν και επιλέγεται η ακολουθία με το μεγαλύτερο σκορ ($2 \times col\_elems + row\_elems$). Στην τρίτη κατηγορία (πίνακες που η διαθέσιμη κρυφή μνήμη ξεπερνά το μέγεθος της αναπαράστασης), γίνεται η επιλογή καθαρά με βάση το σκορ χωρίς πρώτα να έχει γίνει το φιλτράρισμα με βάση το μέγεθος.

Στα Σχήματα 5.9 και 5.10 φαίνονται τα αποτελέσματα της παραπάνω αξιολόγησης για τους 15 πίνακες που έγιναν οι μετρήσεις για αρχιτεκτονικές Harpertown και Dunnington αντίστοιχα. Από το διάγραμμα φαίνεται ότι η επιλογή που γίνεται είναι πολύ κοντά στην καλύτερη δυνατή και ότι οι παράμετροι που επηρεάζουν την επίδοση του SpM×V έχουν προσδιοριστεί με πολύ μεγάλη ακρίβεια και επιτυχία. Αναλυτικότερα, στην αρχιτεκτονική Harpertown η επίδοση της επιλογής κυμαίνεται από το 93% της μέγιστης δυνατής για το 1 νήμα μέχρι το 98, 75% για τα 8 νήματα, ενώ για αρχιτεκτονική Dunnington η αντίστοιχη επίδοση κυμαίνεται από 96 έως 97% της μέγιστης δυνατής για το 1 και τα 2 νήματα και από 90 έως 92% για τα υπόλοιπα. Αντίθετα, η ευριστική είναι αρκετά χειρότερη από την μέγιστη δυνατή περίπτωση και η επίδοση που επιτυγχάνεται με την χρήση της κυμαίνεται από το 83% της μέγιστης δυνατής για το 1 νήμα μέχρι το 97% για τα 8 νήματα, ενώ για αρχιτεκτονική Dunnington η αντίστοιχη επίδοση κυμαίνεται από 86 έως 88% της μέγιστης δυνατής για το 1 και τα 2 νήματα και από 76 έως 78% για τα υπόλοιπα. Αξιοσημείωτο είναι το γεγονός ότι η ευριστική έχει το επιπλέον πλεονέκτημα ότι μπορεί και επεξεργάζεται το κάθε νήμα ξεχωριστά και μπορεί να επιβάλει διαφορετική ακολουθία κωδικοποιήσεων ανά νήμα. Αυτό δεν είναι εφικτό να γίνει, ούτε στις μετρήσεις για την αξιολόγηση του CSX (επειδή δημιουργείται τεράστιος αριθμός ακολουθιών), ούτε κατά την διάρκεια της επιλογής ακολουθίας (επειδή στις μετρήσεις τα δεδομένα μαζεύονται συνολικά και όχι από κάθε νήμα ξεχωριστά). Οπότε, ενδέχεται να κάνει αρκετά χειρότερη επιλογή σε σχέση με την πραγματικά καλύτερη δυνατή ειδικά για πολλά νήματα.



Σχήμα 5.9: Επίδοση της καλύτερης δυνατής περίπτωσης, της ευριστικής και της επιλογής που γίνεται με βάση τις παραμέτρους που βρέθηκε ότι επηρεάζουν την επίδοση του SpM×V σε αρχιτεκτονική Harpertown.



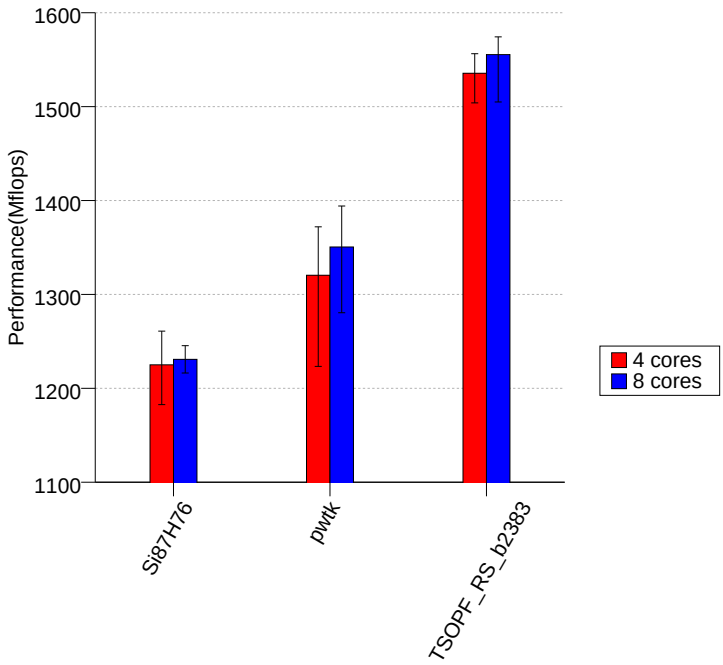
Σχήμα 5.10: Επίδοση της καλύτερης δυνατής περίπτωσης, της ευριστικής και της επιλογής που γίνεται με βάση τις παραμέτρους που βρέθηκε ότι επηρεάζουν την επίδοση του SpM×V σε αρχιτεκτονική Dunnington.

5.3 Ενεργειακά Ζητήματα στο CSX

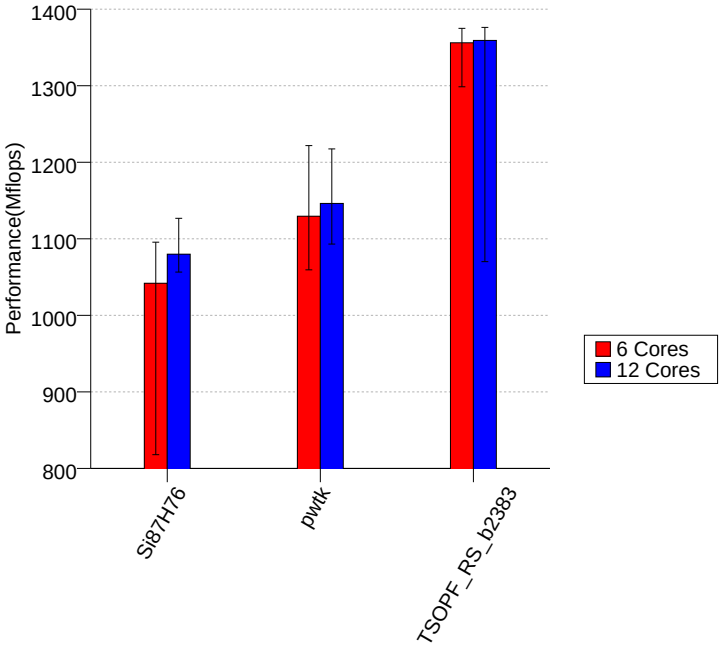
Όταν τρέχουμε το SpM×V δεν επιτυγχάνεται ικανοποιητική επιτάχυνση με την χρήση πολλών νημάτων. Επομένως, είναι λογικό κάποιος από τη στιγμή που δεν πετυχαίνει πολύ καλή επιτάχυνση να θέλει να χρησιμοποιήσει λιγότερους πόρους, ώστε να καταναλώσει λιγότερη ενέργεια. Η ανάγκη αυτή γίνεται ακόμα πιο έντονη σε έναν υπολογιστικό πυρήνα, όπως είναι αυτός του SpM×V, ο οποίος είναι χρονοβόρος καταναλώνοντας, επομένως, αρκετή ενέργεια. Η υλοποίηση που υπάρχει αυτή την στιγμή αφήνει αποκλειστικά στον χρήστη την επιλογή των νημάτων και κατ' επέκταση των πόρων του συστήματος. Παρόλα αυτά, θα είναι πολύ χρήσιμο, ο χρήστης να δηλώνει πόση επίδοση θέλει να έχει επί της μέγιστης και το σύστημα να επιλέγει την καλύτερη δυνατή επιλογή ενεργειακά και το αντίστροφο.

Στα Σχήματα 5.11 και 5.12 φαίνονται τα αποτελέσματα για αρχιτεκτονικές Harpertown και Dunnington για επιλογές νημάτων που διαχειρίζονται το ίδιο μέγεθος κρυφής μνήμης σε όλα τα επίπεδα, αλλά διαφέρουν σε αριθμό. Για τις μετρήσεις που έγιναν σε αρχιτεκτονική Dunnington, παρατηρούμε ότι έχουν πολύ μεγάλο εύρος επίδοσης, πράγμα που οφείλεται κυρίως στις παθητικές περιπτώσεις που αναφέρθηκαν στο Κεφάλαιο 5.2 (σελ. 56). Επίσης, φαίνεται στα διαγράμματα ότι σε όλες τις περιπτώσεις υπάρχουν ακολουθίες κωδικοποιήσεων, για λιγότερα νήματα, που παρουσιάζουν καλύτερη επίδοση, όχι μόνο από την χειρότερη, αλλά και από τη μέση τιμή των επιδόσεων για περισσότερα νήματα. Μάλιστα στον πίνακα $Si87H76$ για αρχιτεκτονική Harpertown και στον πίνακα $pwtk$ για αρχιτεκτονική Dunnington επιτυγχάνεται κα-

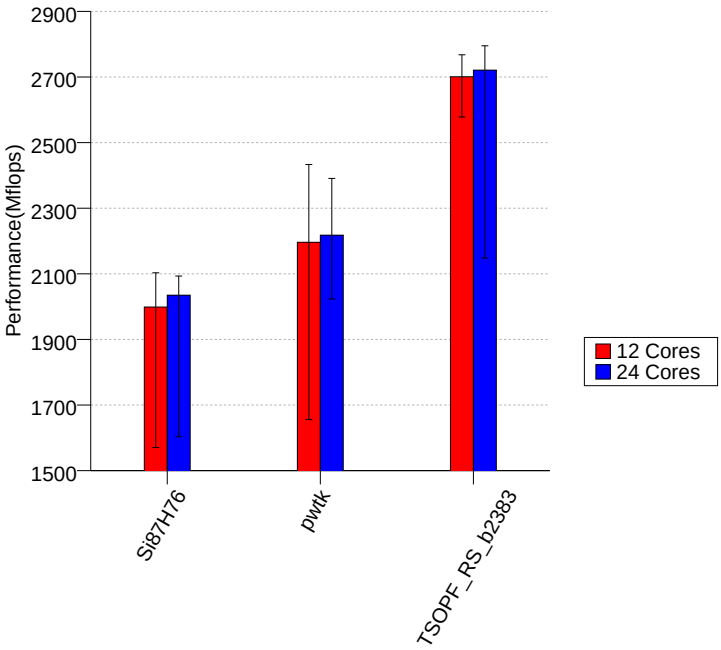
λύτερη μέγιστη επίδοση με τα λιγότερα νήματα. Οπότε, γίνεται εύκολα αντιληπτό ότι χρησιμοποιώντας λιγότερα νήματα δεν χάνουμε σχεδόν καθόλου σε επίδοση και μπορούμε να κερδίσουμε αρκετά σε ενέργεια εφόσον χρησιμοποιούνται λιγότεροι πόροι.



Σχήμα 5.11: Εύρος της επίδοσης του SpM×V για όλες τις ακολουθίες κωδικοποιήσεων σε αρχιτεκτονική Harpertown (4,8 νήματα, 24 MB L2).



(α′) 6,12 νήματα, 18 MB L2, 32 MB L3



(β′) 12,24 νήματα, 36 MB L2, 64 MB L3

Σχήμα 5.12: Εύρος της επίδοσης του SpM×V για όλες τις ακολουθίες κωδικοποιήσεων σε αρχιτεκτονική Dunnington.

Κεφάλαιο 6

Μελλοντική Έρευνα για το SpMxV

Το SpM×V δεν επιταχύνει ακόμα αρκετά και είναι θεμιτή η περαιτέρω βελτίωση της επίδοσης του, ιδιαίτερα σε πολυνηματικές αρχιτεκτονικές. Οπότε, παρακάτω παρουσιάζονται μερικές ιδέες για το σχήμα αποθήκευσης CSX, που επιτυγχάνει πολύ καλή επίδοση σε σχέση με τα υπόλοιπα σχήματα, ώστε να βελτιωθεί ακόμα περισσότερο.

6.1 Βελτίωση Υπολογιστικών Χαρακτηριστικών για Κανονικότητες του CSX

Στα πλαίσια της παρούσας διπλωματικής εργασίας, υλοποιήθηκε η επέκταση CSX-split (Κεφάλαιο 4) στην οποία η επιλογή που γίνεται από την ευριστική περιλαμβάνει περισσότερα μπλοκς. Επίσης, από την αξιολόγηση, που έγινε στους αριθμητικούς υπολογισμούς (Κεφάλαιο 5.2.2), φαίνεται ότι τα μπλοκς αποτελούν ένας σημαντικός παράγοντας για την βελτίωση της επίδοσης του SpM×V. Οπότε, η υλοποίηση αριθμητικών βελτιστοποιήσεων για μπλοκς και ιδιαίτερα του vectorization, που μπορεί να υλοποιηθεί και για τις υπόλοιπες κανονικότητες, αναμένεται να δώσει σημαντικά αποτελέσματα.

6.2 Αξιολόγηση Παθητικών Περιπτώσεων

Στο Κεφάλαιο 5.2 βρέθηκαν κάποιες ακολουθίες κωδικοποιήσεων που λειτουργούν παθητικά για το CSX και δεν επιτυγχάνουν καθόλου καλή επίδοση. Οι ακολουθίες αυτές αφαιρέθηκαν από την διαδικασία αξιολόγησης του CSX, αλλά η περαιτέρω μελέτη τους είναι απαραίτητη για την ολοκλήρωση της αξιολόγησης και για τον καλύτερο προσδιορισμό των παραμέτρων που επηρεάζουν το CSX. Μετά την ολοκλήρωση της συγκεκριμένης μελέτης είναι πολύ πιθανό η ανάπτυξη μιας τεχνικής μηχανικής μάθησης (machine learning), η οποία με βάση τις συγκεκριμένες παραμέτρους θα επιλέγει την ακολουθία κωδικοποίησης αντικαθιστώντας την παρούσα ευριστική μέθοδο, να δώσει πολύ καλύτερα αποτελέσματα από τα τωρινά.

6.3 Συμπίεση στο Πεδίο Τιμών

Το CSX αποτελεί ένα σχήμα αποθήκευσης που συμπιέζει μόνο το πεδίο col_ind. Ωστόσο, το πεδίο val είναι το μεγαλύτερο κομμάτι του πίνακα και περαιτέρω συμπίεση στο συγκεκριμένο πεδίο θα βελτιώσει την επίδοση του SpM×V. Μια προσπάθεια, που έχει γίνει για συμπίεση στο συγκεκριμένο πεδίο, είναι το CSR-VI (Κεφάλαιο 2.3.1 σελ. 28). Επομένως, η περαιτέρω ανάπτυξη της συγκεκριμένης υλοποίησης ή η δημιουργία μιας καλύτερης και η συγχώνευση τους με το CSX αποτελούν πρόκληση για το μέλλον.

6.4 Διεύρυνση των Κανονικοτήτων που Ανιχνεύονται

Οι τύποι κανονικοτήτων, που ανιχνεύονται από το CSX σχήμα, είναι πολλοί. Αυτό, όμως, δεν σημαίνει ότι η αξιοποίηση επιπλέον κανονικοτήτων δεν θα επιφέρει καλύτερη συμπίεση των αραιών πινάκων και, επομένως, καλύτερη επίδοση για το SpM×V. Οπότε, σημαντική βελτίωση για το CSX θα είναι να διευρυνθεί το πεδίο των τύπων κανονικοτήτων. Επιπρόσθετα, ενδιαφέρων παρουσιάζει και η υλοποίηση τεχνικών ανίχνευσης και κωδικοποίησης κανονικοτήτων, ώστε να μειωθεί ο χρόνος μετατροπής ενός πίνακα σε μορφή CSX.

Καταληκτικά, το SpM×V παρουσιάζει έντονο ενδιαφέρον και αποτελεί ανοικτό ερευνητικό πεδίο. Η μελλοντική έρευνα πάνω στο SpM×V, μέσω του σχήματος αποθήκευσης CSX, είναι πολλά υποσχόμενη και αναμένεται να δώσει πολύ καλά αποτελέσματα.

Βιβλιογραφία

[1] R. C. Agarwal, F. G. Gustavson, and M. Zubair. A high performance algorithm using pre-processing for the sparse matrix-vector multiplication. In *Supercomputing'92*, pages 32–41, Minn., MN, November 1992. IEEE.

[2] T. Davis. University of Florida sparse matrix collection. *NA Digest*, 97(23):7, 1997.

[3] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris. Performance evaluation of the sparse matrix-vector multiplication on modern architectures. *The Journal of Supercomputing*, 2008.

[4] E. Im and K. Yelick. Optimizing sparse matrix computations for register reuse in SPARSITY. *Lecture Notes in Computer Science*, 2073:127–136, 2001.

[5] V. Karakasis, G. Goumas, and N. Koziris. A comparative study of blocking storage methods for sparse matrices on multicore architectures. In *12th IEEE International Conference on Computational Science and Engineering (CSE-09)*, Vancouver, Canada, 2009. IEEE Computer Society.

[6] V. Karakasis, G. Goumas, and N. Koziris. Exploring the effect of block shapes on the performance of sparse kernels. In *IEEE International Symposium on Parallel and Distributed Processing (Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications)*, Rome, Italy, 2009. IEEE.

[7] K. Kourtis, G. Goumas, and N. Koziris. Improving the performance of multithreaded sparse matrix-vector multiplication using index and value compression. In *37th International Conference on Parallel Processing (ICPP'08)*, pages 511–519, Sept. 2008.

[8] K. Kourtis, G. Goumas, and N. Koziris. Optimizing sparse matrix-vector multiplication using index and value compression. In *CF '08: Proceedings of the 2008 conference on Computing frontiers*, pages 87–96, New York, NY, USA, 2008. ACM.

[9] K. Kourtis, V. Karakasis, G. Goumas, and N. Koziris. CSX: An extended compression format for SpMV on shared memory systems. In *16th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP'11)*, San Antonio, TX, USA, 2011. ACM. *To appear*.

[10] A. Pinar and M. T. Heath. Improving performance of sparse matrix-vector multiplication. In *Supercomputing'99*, Portland, OR, November 1999. ACM SIGARCH and IEEE.

[11] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations, 1994.

[12] R. W. Vuduc and H. Moon. Fast sparse matrix-vector multiplication by exploiting variable block structure. In *High Performance Computing and Communications*, volume 3726 of *Lecture Notes in Computer Science*, pages 807–816. Springer, 2005.

[13] Wikipedia. Sparse matrix, February 2011.