ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

Πληροφορική

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Μελέτη Απόδοσης Συστήματος PIM (Process in Memory) ως προς την υλοποίηση πολλαπλασιασμού αραιών μητρώων επί διάνυσμα SPMV. Study of PIM (Process in Memory) System Performance regarding Sparse Matrix-Vector Multiplication SPMV Implementation.
Ονοματεπώνυμο Φοιτητή	Νικόλαος – Παναγιώτης Ανδρέου
Πατρώνυμο	Αθανάσιος
Αριθμός Μητρώου	ΜΠΠΛ21005
Επιβλέπων	Ιωάννης Βενέτης, Επίκουρος Καθηγητής

Ημερομηνία Παράδοσης Μάρτιος 2026

Τριμελής Εξεταστική Επιτροπή

Ιωάννης Βενέτης Επίκουρος Καθηγητής	Χαράλαμπος Κωνσταντόπουλος Καθηγητής	Ιωάννης Τασούλας Επίκουρος Καθηγητης

Copyright ©

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα
Επίκουρο καθηγητή Ιωάννη Βενέτη που μου έδωσε την ευκαιρία
να ασχοληθώ έμπρακτα με το κομμάτι του HPC
σε πραγματικό ερευνητικό επίπεδο, αναπτύσσοντας
δεξιότητες και λαμβάνοντας χρήσιμα εφόδια
που αφορούν την επιστήμη της πληροφορικής.

Περίληψη

Η υπολογιστική απόδοση αποτελεί διαχρονικά κρίσιμο ζήτημα στον πολλαπλασιασμό μεγάλων αραιών μητρώων με διανύσματα (Sparse Matrix–Vector Multiplication, SPMV), καθώς ο χρόνος εκτέλεσης αυξάνεται σημαντικά όσο μεγαλώνει το μέγεθος και η πολυπλοκότητα των δεδομένων. Το πρόβλημα καθίσταται εντονότερο στην περίπτωση αραιών μητρώων που περιέχουν στοιχεία κινητής υποδιαστολής διπλής ακρίβειας (64-bit double precision). Η αρχιτεκτονική και ο τρόπος λειτουργίας των κεντρικών μονάδων επεξεργασίας (CPU) συχνά δεν επαρκούν για την αποδοτική εκτέλεση τέτοιων πράξεων σε αποδεκτούς χρόνους. Αντιθέτως, οι μονάδες επεξεργασίας γραφικών (GPU), αξιοποιώντας τον υψηλό βαθμό παραλληλισμού που προσφέρουν, αποτελούν καθιερωμένη λύση για την επιτάχυνση αντίστοιχων υπολογιστικών εργασιών.

Σκοπός της παρούσας εργασίας είναι η μελέτη και αξιολόγηση μιας εναλλακτικής αρχιτεκτονικής επεξεργασίας τύπου Process-In-Memory (PIM), η οποία, πέραν της υποστήριξης παράλληλων υπολογισμών, ενσωματώνει τη μονάδα επεξεργασίας και τη μνήμη στο ίδιο ολοκληρωμένο κύκλωμα. Με τον τρόπο αυτό επιδιώκεται η μείωση των καθυστερήσεων που προκύπτουν από τη μεταφορά δεδομένων μεταξύ επεξεργαστή και μνήμης.

Για τον σκοπό αυτό, πραγματοποιείται σύγκριση της απόδοσης ενός PIM συστήματος με εκείνη μιας GPU. Ειδικότερα, μετράται ο χρόνος εκτέλεσης του πολλαπλασιασμού αραιού μητρώου με τυχαίο διάνυσμα για δώδεκα διαφορετικά αραιά μητρώα, αξιοποιώντας δύο διακριτές τεχνικές λειτουργίας για το σύστημα PIM. Τα πειράματα ενσωματώνονται σε πραγματική επιστημονική εφαρμογή που αφορά την προσομοίωση δικτύων βιολογικών νευρώνων.

Τα αποτελέσματα της μελέτης υποδεικνύουν ότι, παρότι η αρχιτεκτονική PIM παρουσιάζει θεωρητικά πλεονεκτήματα, απαιτούνται περαιτέρω βελτιώσεις ώστε η συγκεκριμένη τεχνολογία να καταστεί πλήρως αξιοποιήσιμη σε απαιτητικές επιστημονικές εφαρμογές.

Λέξεις Κλειδιά: SPMV, DPU, CPU, GPU, Αραιό μητρώο, Διάνυσμα, LIF.

Abstract

Computational performance is a frequent concern when it comes to sparse matrix-vector multiplication (SPMV). In such multiplications, the execution time increases significantly. The problem is intensified in sparse matrices that include 64-bit double-precision floating-point elements, which are decimal numbers with many digits before or after the decimal point. The architecture and, consequently, the mode of operation of central processing units (CPUs) are not sufficient to perform the operations in a reasonable amount of time. In contrast, graphics processing units (GPUs), with their ability to perform parallel computations, are now the main solution to the problem as they execute these operations more efficiently.

The purpose of this work is to study a new processor architecture, Process In Memory (PIM), which not only performs parallel computations but also differs in that it integrates a processing unit and memory into a single circuit. This avoids the delays that arise from the transfer of information between the two.

For this purpose, the performance of the PIM system was compared with that of a GPU. Specifically, the execution time was measured for the two alternatives (GPU and PIM) for sparse matrix-vector multiplication on thirteen different matrices using two alternative operating techniques for the PIM system. The above was applied to a real scientific application that studies the simulation of biological neuron networks (Leaky Integrated-and-Fire (LIF)).

This work suggests that this technology needs improvements to become usable for real scientific applications.

Keywords: SPMV, DPU, CPU, GPU, Sparse Matrix, Vector, LIF

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Εισαγωγή

Στα πλαίσια αυτής της Μεταπτυχιακής Διατριβής μελετάται απόδοση συστημάτων επεξεργασίας αναφορικά με τον πολλαπλασιασμό αραιών μητρών επί διάνυσμα (SPMV Sparse Matrix-Vector Multiplication) [1]. Συγκεκριμένα πρόκειται για συστήματα μονάδων επεξεργασίας CPU, GPU και μονάδων επεξεργασίας εντός της μνήμης (PIM Process In Memory) [2]. Το SPMV είναι ένας θεμελιώδης υπολογισμός με εκτεταμένη χρήση σε εφαρμογές τεχνητής νοημοσύνης, επεξεργασίας εικόνων, βίντεο καθώς και σε άλλες επιστημονικές εφαρμογές [1], [3]. Η διαδικασία του SPMV, λόγω του μεγάλου όγκου δεδομένων και της ανάγκης για πολλαπλές πράξεις, πρόσθεσης και πολλαπλασιασμού μη μηδενικών στοιχείων αραιών μητρών, αποτελεί έναν από τους πλέον απαιτητικούς υπολογισμούς που χρήζει την ανάγκη παράλληλης επεξεργασίας σε πολυπύρηνες. Η ποιο συνηθισμένη αρχιτεκτονική/λύση για αυτό το πρόβλημα είναι οι κάρτες γραφικών (GPU) [1], [4].

Παράλληλα με τη μελέτη της απόδοσης, στα πλαίσια της παρούσας διατριβής πραγματοποιήθηκε και η υλοποίηση της πράξης SPMV με διαφορετικές προσεγγίσεις και τεχνικές, με στόχο την πειραματική αξιολόγηση της συμπεριφοράς τους σε αρχιτεκτονικές CPU, GPU και PIM.

Μια μονάδα επεξεργασίας για να μπορέσει να πραγματοποιήσει πράξεις απαιτεί την αποθήκευση και διαχείριση των δεδομένων σε κάποια μνήμη. Σε αρχιτεκτονικές κεντρικών μονάδων επεξεργασίας (CPU Central Processor Unit) [5] και καρτών γραφικών (GPU Graphics Processor Unit) [4], [6] η μνήμη βρίσκεται σε ξεχωριστά κυκλώματα και απαιτούνται συνεχείς μεταφορές δεδομένων μεταξύ τους, προκαλώντας σημαντική επιβάρυνση στην ταχύτητα εκτέλεσης [1], [4], [5], [7].

Για την λύση του παραπάνω προβλήματος έρχονται οι αρχιτεκτονικές PIM (Process in Memory) [2], [7]. Σε αυτές τις αρχιτεκτονικές, οι μονάδες μνήμης και μονάδες επεξεργασίας βρίσκονται στο ίδιο κύκλωμα, περιορίζοντας έτσι τις μεταφορές δεδομένων από το ένα κύκλωμα στο άλλο βελτιώνοντας τη συνολική απόδοση [2]. Το ερευνητικό ερώτημα είναι κατά πόσο αυτή η προσέγγιση μπορεί να αποδώσει σε πραγματικές επιστημονικές εφαρμογές. Για την απάντηση αυτού η συγκεκριμένη διατριβή χρησιμοποιεί μια προσομοίωση δικτύων βιολογικών νευρώνων, χρησιμοποιώντας το μοντέλο LIF (Leaky Integrate-and-Fire) [8]. Ένα μοντέλο το οποίο βασίζεται σε πράξεις αραιού μητρώου επί διάνυσμα για μεγάλο όγκο δεδομένων.
Τα τελευταία χρόνια έχουν πραγματοποιηθεί διάφορες μελέτες τέτοιων αρχιτεκτονικών. Σημαντικό πλήθος ερευνητών όπως οι Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira Onur Mutlu παρουσίασαν εκτενείς πειραματικές αξιολογήσεις του PIM συστήματος και συγκεκριμένα της εταιρίας UpMem. Το σύστημα αυτό βρίσκει εφαρμογές σε τομείς όπως η γραμμική άλγεβρα, τα γραφήματα, οι βάσεις δεδομένων και τα νευρωνικά δίκτυα [2]. Παράλληλα, αναπτύχθηκε και η βιβλιοθήκη SparseP [7], η οποία αποτελεί την υλοποίηση του αραιού μητρώου επί διάνυσμα σε πραγματικές PIM πλατφόρμες, αξιοποιώντας διαφορετικές μορφές μητρώων (CSR, COO, BCSR, BCOO) [1] και τεχνικές κατανομής φορτίου για εκτέλεση πράξεων SPMV. Η συγκεκριμένη βιβλιοθήκη χρησιμοποιήθηκε εκτενώς σε αυτήν την εργασία, ώστε να πραγματοποιηθούν οι απαιτούμενες πράξεις από την αρχιτεκτονική του PIM. Συγκεκριμένα βιβλιοθήκη SparseP αποτελεί υπάρχον ερευνητικό εργαλείο που έχει αναπτυχθεί στο Εθνικό Μετσόβιο Πολυτεχνείο και δεν αποτελεί αντικείμενο ανάπτυξης

της παρούσας εργασίας. Στα πλαίσια της διατριβής χρησιμοποιήθηκε και παραμετροποιήθηκε κατάλληλα, ώστε να υποστηρίξει τις απαιτήσεις της προσομοίωσης του μοντέλου LIF και τη σύγκριση μεταξύ των αρχιτεκτονικών GPU και PIM.

Παρά τα σημαντικά αυτά βήματα, παραμένει περιορισμένη η μελέτη της εφαρμογής του PIM σε ρεαλιστικά ερευνητικά σενάρια με ιδιαίτερες απαιτήσεις πράξεων. Για παράδειγμα στις πράξεις κατά την προσομοίωση δικτύων βιολογικών νευρώνων που χρησιμοποιούν μοντέλα προσομοίωσης όπως το LIF. Το συγκεκριμένο μοντέλο χαρακτηρίζεται από έντονη εξάρτηση σε πράξεις αραιού μητρώου, γεγονός που το καθιστά ιδανικό πεδίο μελέτης για το εάν οι θεωρητικές βελτιώσεις του PIM μπορούν να έχουν πρακτικό όφελος.

Η παρούσα εργασία στοχεύει να καλύψει αυτό το κενό και να δείξει εάν αυτή η αρχιτεκτονική είναι σε θέση να σταθεί αντάξια τόσο στο πρόβλημα της απόδοσης, όσο και ως προς την πεπατημένη ως τώρα αρχιτεκτονική, δηλαδή τις κάρτες γραφικών. Μάλιστα, εκτός από μελέτη της απόδοσης της αρχιτεκτονικής PIM, πραγματοποιείται και συγκριτική μελέτη μεταξύ GPU και PIM στην υλοποίηση του SPMV στο πλαίσιο του μοντέλου LIF, με στόχο να αποτυπωθούν οι διαφορές απόδοσης και αποδοτικότητας μεταξύ των δύο αρχιτεκτονικών, να εντοπιστούν τα πλεονεκτήματα και οι περιορισμοί του PIM σε πραγματικές επιστημονικές εφαρμογές και να διατυπωθούν προτάσεις για βελτιώσεις για μελλοντική έρευνα.

Αναφορικά με την ερευνητική μέθοδο, οι μετρήσεις πραγματοποιηθήκαν τόσο για μεγάλα αραιά μητρώα και διανύσματα σε μέγεθος της τάξεως 10.000x10.000, όσο και για μικρότερα της τάξεως 5000x5000. Συγκεκριμένα, πραγματοποιείται η έρευνα σε τέσσερα μεγάλα μητρώα και εννιά μικρότερα. Οι εκάστοτε προσομοιώσεις LIF και οι αλγόριθμοι SPMV εφαρμοστήκαν για τα μητρώα σε αρχιτεκτονική GPU και PIM. Η GPU που χρησιμοποιήθηκε είναι η NVIDIA Tesla A100 [4] και το σύστημα PIM είναι το σύστημα της UpMem [2], [7].

Συνοψίζοντας, στα πλαίσια της παρούσας διατριβής πραγματοποιήθηκε η υλοποίηση και αξιολόγηση της πράξης SPMV σε αρχιτεκτονικές GPU και PIM, στο πλαίσιο προσομοίωσης δικτύων βιολογικών νευρώνων με χρήση του μοντέλου LIF. Πραγματοποιήθηκαν πειραματικές μετρήσεις σε αραιά μητρώα διαφορετικών μεγεθών και χαρακτηριστικών, με στόχο τη σύγκριση της απόδοσης, της αποδοτικότητας και των περιορισμών των δύο αρχιτεκτονικών σε ρεαλιστικά επιστημονικά σενάρια.

Η εργασία ακολουθεί την παρακάτω δομή: Στο πρώτο κεφάλαιο παρουσιάζονται οι αρχιτεκτονικές CPU, GPU και PIM καθώς και διαφορές μεταξύ τους. Στο δεύτερο κεφάλαιο αναλύεται η προσομοίωση των δικτύων βιολογικών νευρώνων στην οποία καλείται η συγκεκριμένη αρχιτεκτονική να ανταποκριθεί. Στο τρίτο κεφάλαιο, γίνεται αναφορά στην βιβλιοθήκη SparseP που χρησιμοποιήθηκε, σε τεχνικές κατανομής φορτίων και αναπροσαρμογής των μητρώων, προκειμένου αυτά να γίνουν αξιοποιήσιμα προς μελέτη. Στο τέταρτο κεφάλαιο παρουσιάζονται οι υλοποιήσεις αλγορίθμου SPMV στις αρχιτεκτονικές CPU, GPU και PIM. Στο πέμπτο κεφάλαιο παρουσιάζονται τα αποτελέσματα από τις μετρήσεις για PIM και GPU, οι διαφορές τους καθώς και το πως ανταποκρίθηκε το κάθε σύστημα. Τέλος παρατίθενται πλέον τα συμπεράσματα, καθώς και προτάσεις για μελλοντικές έρευνες πάνω στο αντικείμενο και στο πώς θα μπορούσε να βελτιωθεί η αρχιτεκτονική του PIM.

1 Η αρχιτεκτονική συστημάτων CPU, GPU & PIM

1.1 Αρχιτεκτονική CPU

Η Κεντρική Μονάδα Επεξεργασίας (Central Processing Unit, CPU) αποτελεί το βασικό υπολογιστικό στοιχείο κάθε συστήματος και είναι ξεχωριστό κύκλωμα από την προσωρινή μνήμη RAM. Ο επεξεργαστής αποθηκεύει και διαβάζει τις εντολές, τα δεδομένα και τις μεταβλητές από τη μνήμη RAM [9]. Η επικοινωνία αυτή μεταξύ μονάδας επεξεργασίας και μνήμης πραγματοποιείται μέσω διαύλου (bus) [5] . Σημειώνεται, ότι αυτά τα δυο κυκλώματα βρίσκονται πάνω σε μια τυπωμένη πλακέτα (Motherboard) [5].

Η διαδικασία εκτέλεσης μιας εντολής πραγματοποιείται με τρείς ενέργειες: fetch, decode και execute [5].

Fetch: Η CPU διαβάζει – παίρνει την εντολή ή τα δεδομένα που χρειάζεται από τη RAM με σκοπό να πραγματοποιήσει μια ενέργεια. Αυτά τα δεδομένα αποθηκεύονται προσωρινά σε καταχωρητές (registers) [5], [10] που είναι σημεία μνήμης τα οποία βρίσκονται εντός του κυκλώματος CPU. Οι καταχωρητές αποτελούν την ταχύτερη μορφή μνήμης και είναι αυτοί που ο επεξεργαστής χρησιμοποιεί με σκοπό να κάνει πράξεις.

Decode: Σε αυτό το στάδιο ο επεξεργαστής αποκωδικοποιεί την εντολή – δεδομένα που λαμβάνει, σε αποδοτικότερη για αυτόν μορφή, με σκοπό να εκτελέσει την ενέργεια που του ζητείται.

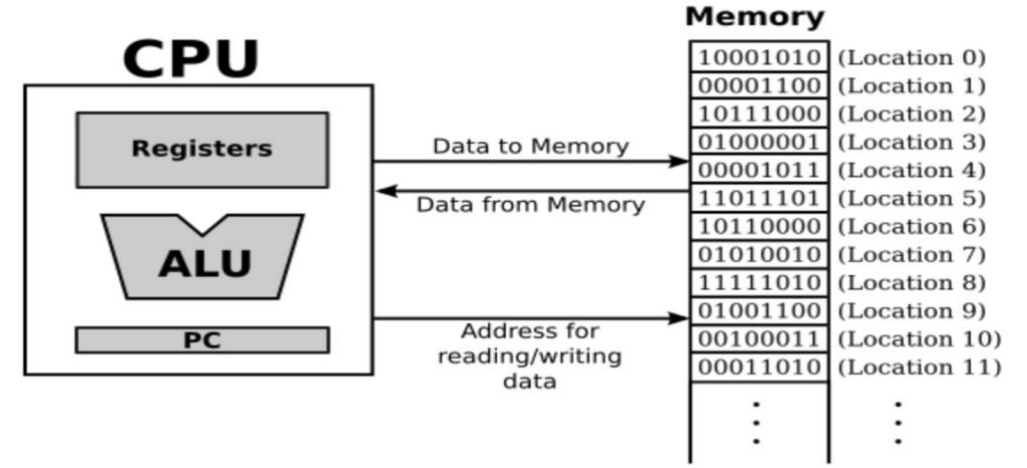
Execute: Εδώ εκτελείται πλέον η εντολή/πράξη και το αποτέλεσμα μεταφέρεται και καταγράφεται πίσω στην διεύθυνση μνήμης RAM.

Η συνεχής αυτή διαδικασία δημιουργεί καθυστερήσεις, καθώς οι χρόνοι πρόσβασης στη μνήμη είναι μεγαλύτεροι σε σχέση με την ταχύτητα λειτουργίας της CPU. Για την αντιμετώπιση του προβλήματος, χρησιμοποιείται ένα είδος προσωρινής μνήμης cache τριών επιπέδων (L1, L2, L3) [5], [10], η οποία είναι κομμάτι του επεξεργαστή. Εκεί αποθηκεύονται δεδομένα τα οποία χρησιμοποιεί η μονάδα επεξεργασίας συχνότερα. Η ύπαρξη cache μειώνει σημαντικά τις καθυστερήσεις, καθώς τα πιο συχνά χρησιμοποιούμενα δεδομένα παραμένουν πιο κοντά στον επεξεργαστή [5], [10]. Βέβαια το μέγεθος της συγκεκριμένης μνήμης δεν είναι επαρκές σε επίπεδο πράξεων με πολλά δεδομένα όπως είναι μια πράξη αραιού μητρώου επί διάνυσμα.

Σε επίπεδο παράλληλης εκτέλεσης, οι μονάδες επεξεργασίας CPU χαρακτηρίζονται από μικρό αριθμό πυρήνων αλλά μεγάλης ισχύος ανά πυρήνα σε σχέση με άλλες αρχιτεκτονικές π.χ. GPU, PIM. Για τον λόγο αυτό, οι μονάδες επεξεργασίας CPU είναι κατάλληλες για υπολογιστικά φορτία που απαιτούν υψηλή υπολογιστική ισχύ και δεν χρήζουν την ανάγκη τόσο πολύ παράλληλης επεξεργασίας, σε αντίθεση με τις GPU και PIM που εξειδικεύονται σε πολλές παράλληλες πράξεις που δεν απαιτούν υψηλή ισχύ ανά πυρήνα.

Οι κεντρικές μονάδες επεξεργασίας έχουν μια μονάδα που ονομάζεται Αριθμητική Λογική Μονάδα (Arithmetic Logic Unit - ALU) όπου πραγματοποιεί όλες της μαθηματικές πράξεις (προσθέσεις, αφαιρέσεις κ.τ.λ.) και λογικούς υπολογισμούς [5], [10].

Τέλος, υπάρχει και η μονάδα ελέγχου (Control Unit) [5], [10] , που είναι επίσης κομμάτι του επεξεργαστή. Σκοπός της είναι να συντονίζει και να καθοδηγεί όλες τις λειτουργίες, να ρυθμίζει τη ροή των δεδομένων και να κατευθύνει τις εντολές στα κατάλληλα τμήματα του επεξεργαστή, ώστε κάθε μέρος να εκτελεί τον ρόλο του σωστά και συγχρονισμένα. Είναι η μονάδα που είναι υπεύθυνη για την ομαλή λειτουργεία του CPU και των επιμέρους μονάδων του.



Εικόνα 1-1: Απεικόνιση σχέσης CPU με μνήμη

1.2 Αρχιτεκτονική GPU

Η Μονάδα Επεξεργασίας Γραφικών (Graphics Processing Unit, GPU) σε αντίθεση με την CPU είναι φτιαγμένη ώστε να εκτελεί μεγάλο πλήθος πράξεων ταυτόχρονα, αξιοποιώντας την παράλληλη επεξεργασία. Η διαφορά εδώ έγκειται στον αριθμό των πυρήνων. Μια GPU διαθέτει περισσότερους πυρήνες από την CPU, για παράδειγμα μια CPU μπορεί να διαθέτει 24 πυρήνες ενώ μια GPU μπορεί να διαθέτει πάνω από 10.000 πυρήνες [4]. Παρ' ότι, οι μεμονωμένοι πυρήνες της GPU είναι λιγότερο ισχυροί σε σύγκριση με αυτούς της CPU, η μαζικότητά τους καθιστά την GPU ιδανική για εφαρμογές που απαιτούν έντονη παράλληλη επεξεργασία, όπως η επίλυση μεγάλων αραιών μητρώων και οι υπολογισμοί σε επιστημονικές ή γραφικές εφαρμογές.

Αναλυτικότερα, ως προς την αρχιτεκτονική, η GPU για να αποθηκεύει τα δεδομένα χρησιμοποιεί ένα είδος μνήμης (VRAM) [4], [6], [11], [12]. Η μνήμη οργανώνεται σε ανεξάρτητα κυκλώματα (memory controllers) [4], [6] που εξυπηρετούν τη γρήγορη πρόσβαση από τους πυρήνες της GPU στα δεδομένα. Όπως και στην CPU και εδώ οι πυρήνες των GPU παίρνουν τα δεδομένα μέσω δίαυλου BUS που υπάρχει μεταξύ αυτών και της μνήμης και τα αποθηκεύουν προσωρινά σε καταχωρητές, με σκοπό να γίνουν αξιοποιήσιμα. Επιπρόσθετα η GPU χρησιμοποιεί και αυτή μια μνήμη τύπου cache για παρόμοιους σκοπούς με την CPU.

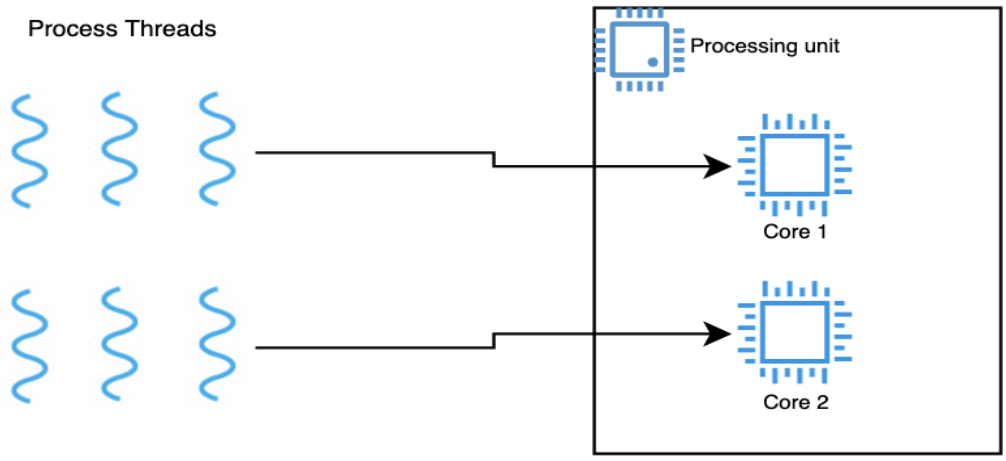
Η αρχιτεκτονική των σύγχρονων GPU έχει μια ιεραρχική οργάνωση σχετικά με το κύκλωμα στο οποίο βρίσκονται οι πυρήνες της. Συγκεκριμένα χωρίζονται σε επίπεδα μονάδων (εικ. 1-4). Πρώτα είναι τα GPC (Graphics Processor Clusters) [4], [6]. Εντός

των GPC υπάρχουν οι μονάδες SM (Streaming Multi Processors) [4], [6], στις οποίες περιλαμβάνονται οι πυρήνες της GPU. Οι κύριοι τύποι πυρήνων σε μια GPU είναι: CUDA Cores οι οποίοι είναι υπεύθυνοι για βασικές αριθμητικές πράξεις (προσθέσεις και αφαιρέσεις κ.τ.λ.), Tensor Cores οι συγκεκριμένοι πυρήνες πραγματοποιούν κατά κύριο λόγο πράξεις μητρώων και χρησιμοποιούνται αρκετά σε εφαρμογές μηχανικής μάθησης. Τέλος οι RT Cores (Ray Tracing Cores) οι οποίοι είναι υπεύθυνοι για την εκτέλεση αλγορίθμων ανίχνευσης ακτινών (ray tracing). Οι RT χρησιμοποιούνται για την παραγωγή φωτο-ρεαλιστικών εικόνων και άλλες λειτουργείες που αφορούν Graphics Rendering [4], [6], [11], [12] πεδίο που είναι εκτός του ενδιαφέροντος της παρούσας εργασίας. Οι πυρήνες του ενδιαφέροντος της διατριβής είναι τα CUDA Cores και Tensor Cores. Παρ' ότι η βασική αρχιτεκτονική παραμένει σταθερή, κάθε νέα γενιά GPU παρουσιάζει διαφοροποιήσεις, όπως αυξημένο αριθμό SMs, βελτιώσεις στους Tensor Cores καθώς και στη διαχείριση της μνήμης cache (π.χ. διπλή L2 cache στην A100 σε σχέση με την Ada) κ.τ.λ. [4], [6].

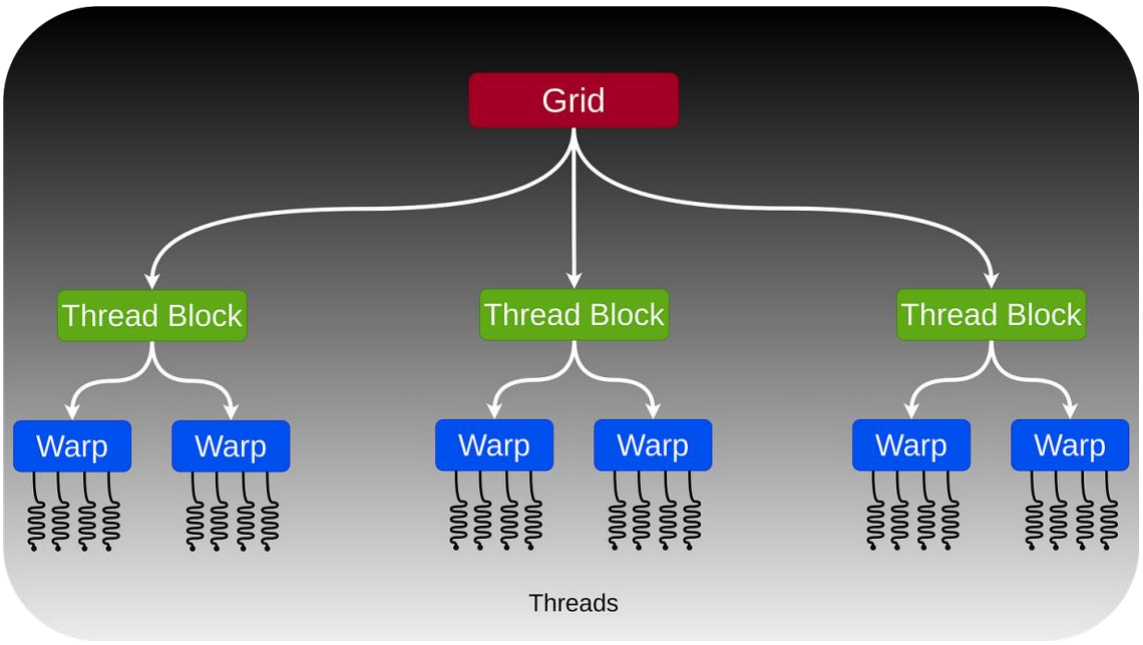
Για να μπορέσει ένα υπολογιστικό σύστημα, είτε πρόκειται για CPU, είτε για GPU, είτε για PIM (Processing-In-Memory**)**,να πραγματοποιήσει παράλληλες ενέργειες, αξιοποιεί μια τεχνολογία γνωστή ως νήματα (Threads) (εικ. 1-2) [9], [11], [12], [13], [14]**.** Τα νήματα αποτελούν μικρές ροές κώδικα, οι οποίες μπορούν να εκτελεστούν είτε παράλληλα σε πολλούς πυρήνες, είτε ψευδό-παράλληλα σε έναν μόνο πυρήνα μέσω τεχνικών χρόνο-προγραμματισμού (Context Switch) [9], [13]. Με αυτόν τον τρόπο, τμήματα ενός προγράμματος, όπως πράξεις σε ένα μεγάλο αραιό μητρώο, χωρίζονται σε επιμέρους νήματα, υπολογίζονται ταυτόχρονα και τα αποτελέσματα ενώνονται εκ νέου για την παραγωγή της τελικής λύσης.

Στην περίπτωση των GPU, η ιεραρχία ομαδοποίησης σχετικά με νήματα έχει ως εξής: Τα νήματα είναι η βασική μονάδα τα οποία συγκεντρώνονται σε ομάδες των 32 που ονομάζονται Warps [11], [12]. Τα Warps με τη σειρά τους οργανώνονται σε Thread Blocks και τα Thread Blocks σε Grid [11], [12]. Αυτή η ομαδοποίηση (εικ. 1-3) καθιστά δυνατή την εκτέλεση μαζικά παράλληλων πράξεων, όπου ο προγραμματιστής ορίζει τον τρόπο με τον οποίο τα νήματα δημιουργούνται, προγραμματίζονται και εκτελούνται στην GPU.

Τέλος στην GPU όπως και την CPU η εκτέλεση των εντολών γίνεται σε τρία στάδια, fetch, decode, execute.



Εικόνα 1-2: Απεικόνιση Νημάτων σε Πυρήνες.



Εικόνα 1-3: Απεικόνιση Νημάτων σε Wrap, Blocks, Grid.

Εικόνα 1-4: Απεικόνιση αρχιτεκτονικής GPU A100 Tensor Core [4].

1.3 Αρχιτεκτονική PIM UpMem

Στις συμβατικές CPU, GPU αρχιτεκτονικές, η μονάδα επεξεργασίας είναι χωρισμένη από τη μνήμη και απαιτούνται συνεχείς κύκλοι ανάγνωσης/εγγραφής για κάθε πράξη, κάτι που περιορίζει την απόδοση όταν το πλήθος των δεδομένων είναι μεγάλο. Οι GPU, από την άλλη, έχουν σχεδιαστεί ώστε να εκτελούν μαζικά παράλληλους υπολογισμούς, γεγονός που τις καθιστά ιδανικές για εφαρμογές όπως η επεξεργασία γραφικών, στα οποία χρειάζεται να πραγματοποιούνται πολλές πράξεις γραμμικής άλγεβρας σε πραγματικό χρόνο, ή στην εκπαίδευση νευρωνικών δικτύων.

Όπως προαναφέρθηκε, τα συστήματα PIM (Process In Memory) συνδυάζουν μονάδα επεξεργασίας και μνήμη στο ίδιο κύκλωμα/μονάδα [2], [7]. Αυτό μειώνει δραστικά την ανάγκη μεταφοράς δεδομένων μεταξύ μονάδας επεξεργασίας και μνήμης όπως θα συνέβαινε εάν αποτελούσαν διαφορετικά κυκλώματα, γεγονός που αποτελεί έναν από τους βασικότερους περιορισμούς στις συμβατικές αρχιτεκτονικές (CPU, GPU). Έτσι, αποφεύγεται η καθυστέρηση που προκαλείται από τη συνεχή επικοινωνία μεταξύ κεντρικής μονάδας επεξεργασίας και κεντρικής μνήμης.

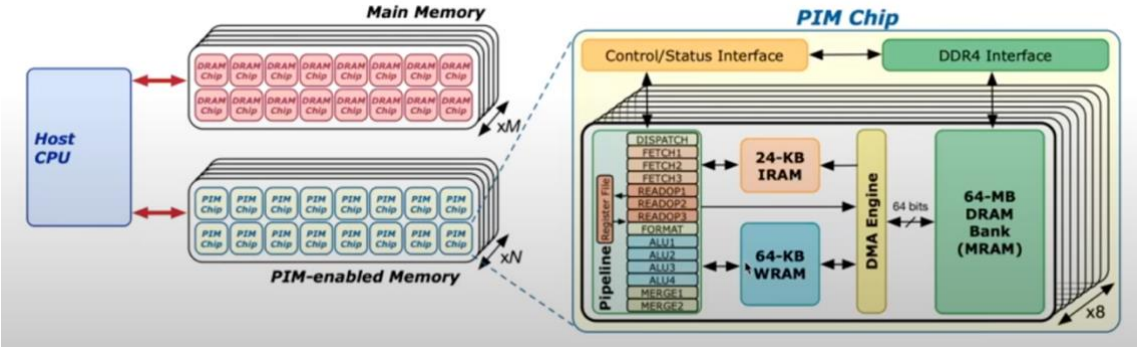
Πιο συγκεκριμένα, το σύστημα PIM αποτελείται από μονάδες επεξεργασίας με όνομα DRAM Processing Unit (DPU) και μονάδες μνήμης DRAM τύπου DDR4-2400 DIMM [5]. Το PIM χρειάζεται επιπλέον ένα εξωτερικό host σύστημα που αποτελείται από μια μονάδα επεξεργασίας (CPU) και κάποιες κλασσικές μνήμες (DRAM) [5]. Ο λόγος που το σύστημα PIM χρειάζεται ένα host σύστημα για να λειτουργήσει είναι ώστε να φορτωθούν/ανακτηθούν τα δεδομένα στις μνήμες του.

Μια ολοκληρωμένη λύση PIM της UpMem προσφέρει το host CPU σύστημα και το PIM σύστημα. Συγκεκριμένα, το PIM σύστημα δημιουργείται από ενότητες PIM κυκλωμάτων (PIM Chips) που ονομάζονται PIM DIMMs. Σε κάθε PIM Chip υπάρχουν 8 DPU τα οποία έχουν πρόσβαση στις τοπικές μνήμες DRAM του κυκλώματος. Αυτές οι μνήμες είναι οι εξής: Η MRAM των 64MB που είναι η βασική μνήμη αποθήκευσης δεδομένων, η IRAM (Instruction RAM) των 24MB που είναι η μνήμη στην οποία αποθηκεύονται οι εντολές εκτέλεσης και η WRAM (Working RAM) των 64KB, η οποία χρησιμοποιείται για την προσωρινή αποθήκευση δεδομένων και υπολογισμών που βρίσκονται σε εξέλιξη. Οι μνήμες DRAM των PIM chips, είναι προσβάσιμες από τον κεντρικό επεξεργαστή για την αντιγραφή δεδομένων εισόδου και την ανάκτηση αποτελεσμάτων μέσω του διαύλου μνήμης [2].

Σε μια ενότητα PIM DIMM υπάρχουν 2 ομάδες των 8 PIM Chip με σύνολο τα 64 DPU ανά PIM DIMM. Σε ένα σύστημα PIM υπάρχουν 20 PIM DIMMs (δηλαδή 64x2 DPUs για κάθε PIM DIMMs) με σύνολο τα 2560 DPUs [2], [7].

Το PIM της UpMem λειτουργεί με τον host CPU. Η εκτέλεση των ενεργειών πραγματοποιείται στα DPU και τα αποτελέσματα επιστρέφουν πίσω στο host σύστημα όταν ολοκληρώνεται η ενέργεια.

Επιπλέον για παράλληλες μεταφορές προς πολλές MRAM μνήμες υπάρχει περιορισμός ως προς τα μεγέθη των κομματιών (chunks) των μεταφορών. Τα chunks θα πρέπει να είναι το ίδιο μέγεθος αλλιώς η μεταφορά εκτελείται σειριακά από το host σύστημα [2], [7], κάτι το οποίο είναι σημαντική πληροφορία για τον μηχανικό HPC διότι μπορεί να δημιουργήσει παραπάνω καθυστερήσεις σε μεγάλο όγκο δεδομένων αν δεν βρεθεί ένας τρόπος να αντιμετωπιστεί προγραμματιστικά αυτός ο περιορισμός του συστήματος.



Εικόνα 1.7: Αρχιτεκτονική PIM UpMem [2].

Κάθε DPU είναι ένας 32-bit, RISC [5] πυρήνας της τάξεως των 500MHz [2], [7] που υποστηρίζει μέχρι 24 hardware threads (στην UpMem ονομάζονται tasklets) [9] και έχει 24 καταχωρητές των 32-bit. Εδώ θα πρέπει να αναφερθεί ότι η λογική fetch, decode, execute από τις μνήμες υπάρχει και σε αυτό το σύστημα. Με την διάφορα ότι υπάρχει αμελητέα μείωση στην ταχύτητα απόδοσης, διότι όλα τα κομμάτια ανήκουν στο ίδιο κύκλωμα. Η αρχιτεκτονική παρέχει την δυνατότητα πράξεων με 32-bit ακεραίων, όμως πολλές σύνθετες αριθμητικές πράξεις π.χ. 64-bit ακέραιοι, 32/64-bit αριθμών κινητής

υποδιαστολής μονής και διπλής ακρίβειας δεν υποστηρίζονται στο hardware και πρέπει να προσομοιωθούν/υλοποιηθούν αλγοριθμικά [2], [7], με συνέπεια το κόστος την ταχύτητα για εργασίες που απαιτούν τέτοιου τύπου υπολογισμούς.

Τα tasklets εκτελούν τον ίδιο κώδικα για διαφορετικά δεδομένα ο συγχρονισμός γίνεται με εργαλεία τύπου mutex, barriers [11], [12], [14] κ.λπ., που παρέχονται από τη βιβλιοθήκη της UPMEM. Οι μηχανικοί μπορούν να διαχειριστούν τα δεδομένα στις μνήμες MRAM μέσω συναρτήσεων που υπάρχουν στις βιβλιοθήκες, καθώς και άλλα πράγματα όπως το πλήθος των DPU που θα χρησιμοποιηθούν για τις εκάστοτε ενέργειες. Αυτό επιτυγχάνεται κατά την μεταγλώττιση του κώδικα σε γλώσσα μηχανής [2], [7]. Η UPMEM παρέχει κατάλληλες βιβλιοθήκες προγραμματισμού για C/C++, Python για την πραγματοποίηση αυτών των ενεργειών

Σύμφωνα με τον κατασκευαστή η συγκεκριμένη αρχιτεκτονική είναι κατάλληλη για εφαρμογές με πολύ μεγάλο όγκο δεδομένων όπως βάσεις δεδομένων, graph analytics, memory-bound, data-parallel εργασίες όπου ο τοπικός υπολογισμός δίνει μεγάλο όφελος [2]. Σαν μειονεκτήματα, παρατηρείται η ανάγκη ύπαρξης ενός host συστήματος που δουλειά του είναι να φορτώνει δεδομένα στα PIM κυκλώματα και να ανακτά τα αποτελέσματα. Αμέσως βγαίνει το συμπέρασμα πως αν ο εκάστοτε αλγόριθμος που τρέχει απαιτεί συνεχής, μικρές αλλαγές ή πολύ συχνό συγχρονισμό δεδομένων με το host σύστημα, η καθυστέρηση των μεταφορών αυτών από host σε PIM μπορεί να ακυρώσει τα οφέλη του συστήματος. Επίσης ένα άλλο πρόβλημα που μπορεί να προκύψει είναι, ότι οι πράξεις των 64-bit αριθμών κινητής υποδιαστολής μονής και διπλής ακρίβειας, θα απαιτήσουν περισσότερο χρόνο για υλοποιηθούν λόγω του ότι η αρχιτεκτονική δεν υποστηρίζει τους συγκεκριμένου τύπου αριθμούς. Σε αυτές τις περιπτώσεις επίσης μπορούν να ακυρώσουν τα οφέλη του συστήματος.

Αξίζει να σημειωθεί ότι τα συστήματα PIM δεν είναι διαθέσιμα ως αυτόνομες κάρτες υπολογιστών που τοποθετούνται σε μητρικές πλακέτες (όπως οι GPU). Αντιθέτως, παρέχονται ως ανεξάρτητες υπολογιστικές πλατφόρμες μέσω ειδικού hardware ή και μέσω cloud υπηρεσιών από εταιρείες όπως οι Intel, Samsung, Xilinx, UpMem, κ.ά.

2 Ερευνητικό αντικείμενο

2.1 Παρουσίαση Έρευνας Νευρώνων

Όπως προαναφέρθηκε, η συγκεκριμένη εργασία υλοποιείται στα πλαίσια μιας έρευνας προσομοίωσης βιολογικών νευρώνων εγκεφάλου. Η μελέτη του τρόπου με τον οποίο οι βιολογικοί νευρώνες εγκεφάλου επικοινωνούν, ανταλλάσσουν σήματα και εκτελούν σύνθετες λειτουργίες, είναι ένα θέμα που έχει απασχολήσει πολλούς στον ερευνητικό χώρο νευρολογίας και βιολογίας [15]. Λόγω της πολυπλοκότητας του προβλήματος απαιτείται η συμβολή πολλών επιστημών, όπως η Βιολογία, Ιατρική, Χημεία, Μαθηματικά και η Πληροφορική. Η Πληροφορική δημιουργεί εργαλεία τόσο σε επίπεδο λογισμικού (software) όσο και σε επίπεδο υλικού (hardware), με σκοπό την υλοποίηση προσομοιώσεων και περίπλοκων μαθηματικών υπολογισμών που βοηθούν στην εξέλιξη των εκάστοτε ερευνών [16].

Ενώ η έρευνα σχετικά με τον εγκέφαλο σε μοριακό επίπεδο αποτελεί αντικείμενο των θετικών επιστημών, το επίπεδο επεξεργασίας πληροφοριών και οι επικοινωνίες μεταξύ νευρώνων έχουν ιδιαίτερο ενδιαφέρον για τους ερευνητές της Πληροφορικής, ιδιαίτερα στο πεδίο του HPC (High Performance Computing). Λόγω της φύσης του εγκεφάλου αποστέλλονται δισεκατομμύρια σήματα μέσα στο δίκτυο των νευρώνων. Ο συνολικός αριθμός νευρώνων στον ενήλικο ανθρώπινο εγκέφαλο εκτιμάται ότι είναι ≈10^10 και κάθε νευρώνας συνδέεται περίπου με 7000 άλλους [8], [15]. Οι πληροφορίες μεταδίδονται με τη μορφή ηλεκτρικών και χημικών παλμών [15].

Η δραστηριότητα ενός μεμονωμένου νευρώνα μπορεί να μοντελοποιηθεί από διάφορα μοντέλα προσομοίωσης δράσης νευρωνικών δικτύων, όπως τα Hodgkin–Huxley [17], Hindmarsh–Rose [18], FitzHugh–Nagumo [19], [20], Van der Pol και Leaky Integrate-and-Fire (LIF) [5][65]. Το τελευταίο είναι και αυτό που χρησιμοποιείται στην παρούσα εργασία.

Πιο συγκεκριμένη η συγκεκριμένη έρευνα εξετάζει την κατάσταση της χίμαιρας , η οποία συμβαίνει σε ορισμένους ζωντανούς οργανισμούς και έχει συνδεθεί με βιολογικά φαινόμενα όπως ο ημισφαιρικός ύπνος. Αυτού του τύπου ο ύπνος έχει παρατηρηθεί σε πτηνά και θηλαστικά: το μισό του εγκεφάλου βρίσκεται σε βαθύ ύπνο ενώ το υπόλοιπο σε διαφορετική κατάσταση. Εκτιμάται πως η συγκεκριμένη κατάσταση επιτρέπει στον εγκέφαλο να ξεκουραστεί και να μένει επαγρύπνηση ταυτόχρονα, ενώ παρατηρείται σε ζώα που ζουν σε περιοχές με πολλά αρπακτικά αλλά και σε πουλιά κατά τη διάρκεια μακρών μεταναστευτικών πτήσεων [8].

Η συγκεκριμένη εργασία δεν ασχολείται τόσο με το φαινόμενο της χίμαιρας και με τους νευρώνες όσο με το υπολογιστικό κομμάτι που αφορά την προσομοίωση νευρώνων. Για μια ρεαλιστική προσομοίωση απαιτείται να ληφθούν υπόψη μεγάλα σε όγκο δίκτυα νευρώνων. Τέτοιου μεγέθους συστήματα απαιτούν τεράστια υπολογιστική ισχύ, τόσο σε μνήμη όσο και σε επεξεργασία, δημιουργώντας την ανάγκη ανάπτυξης αποδοτικών αλγορίθμων προσομοίωσης και παράλληλων υπολογιστικών μεθόδων [21]. Επίσης, η έρευνα απαιτεί την εκτέλεση πολλών προσομοιώσεων διαφορετικών δικτύων χρησιμοποιώντας το μοντέλο LIF για στατιστικά αξιόπιστα αποτελέσματα [8]. Επομένως, προκύπτει η ανάγκη για υπολογιστικά συστήματα ικανά να ανταπεξέλθουν σε αυτούς τους υπολογισμούς. Όπως αναφέρεται και παραπάνω οι κάρτες γραφικών (GPU) είναι πλέον πολύ δημοφιλείς για τέτοιου είδους επιστημονικές εφαρμογές λόγω της παράλληλης επεξεργασίας που προσφέρουν [21]. Χρησιμοποιούνται σε τομείς όπως

Νευροεπιστήμες, Φυσική και Βιολογία. Παράλληλα, η υποσχόμενη τεχνολογία στον χώρο του HPC είναι τα Process-In-Memory (PIM) συστήματα, τα οποία στοχεύουν στη μείωση του κόστους επικοινωνίας ανάμεσα σε μνήμης και επεξεργαστή.

2.2 Μοντέλο LIF

Το μοντέλο LIF παρουσιάστηκε για πρώτη φορά από τον Γάλλο νευροεπιστήμονα Louis O Lapicque το 1907 και περιγράφει την δραστηριότητα ενός μεμονομένου νευρώνα σε έναν εγκέφαλο και ένα δίκτυο νευρώνων [8]. Το μοντέλο προσεγγίζει τη δυναμική ενός νευρώνα κατά την διέγερση του σε τρία στάδια. Τα τρία στάδια αυτά είναι: πρώτο στάδιο μια εκθετική αύξηση δυναμικού (Διέγερση), έπειτα μια απότομη επαναφορά του δυναμικού και τέλος μια περίοδος ηρεμίας [8]. Εξηγώντας πιο μαθηματικοποιημένα συμβολίζοντας με u(t) την μεταβλητή που περιγράφεται το δυναμικό του νευρώνα την χρονική στιγμή t, η όλη διαδικασία περιγράφεται από τα ακόλουθα:

$$\frac{du(t)}{dt} = \mu - u(t)$$

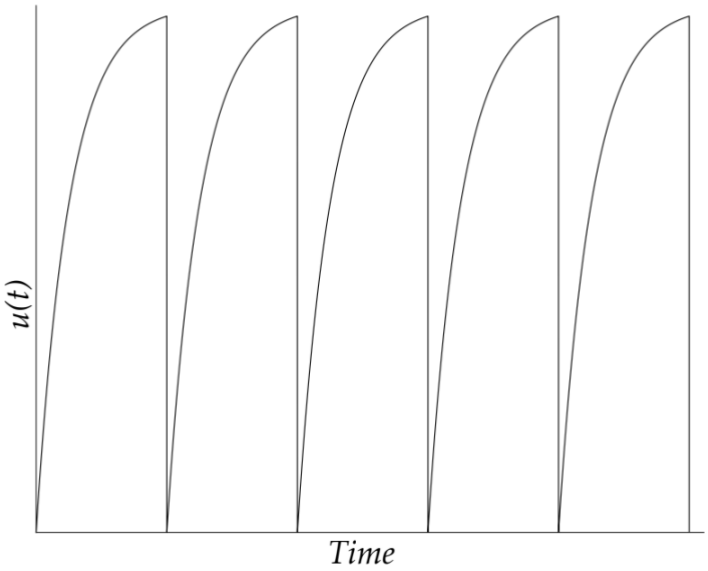
Εικόνα 2-1: Περιγραφή ρυθμού μεταβολής δυναμικού νευρώνα.

$$\lim_{\epsilon \to 0} u(t_q + \epsilon) \to u_{\text{rest}}, \quad \text{when } u(t_q) \geq u_{\text{th}}, \quad q = 1, 2, \ldots$$

Εικόνα 2-2: Περιγραφή στιγμιαίας επαναφοράς του δυναμικού όταν υπερβαίνει μια συγκεκριμένη τιμή κατωφλίου, που συμβολίζεται με uth. Το δυναμικό τότε επαναφέρεται στην κατάσταση ηρεμίας, που συμβολίζεται με urest, q = 1, 2, . . . είναι ένας μετρητής των επαναφορών

$$u(t) = u_{\text{rest}}, \quad \text{when } t_q \leq t < t_q + T_r,$$

Εικόνα 2-3: Εικόνα 2.3: Περιγραφή κατάστασης ηρεμίας μετά την επαναφορά του δυναμικού του νευρώνα για ένα χρονικό διάστημα Tr.



Εικόνα 2-4: Διαγραμματική αναπαράσταση δραστηριότητας νευρώνα.

Οι νευρώνες παρ' όλα αυτά έχουν την τάση να δημιουργούν δίκτυα και συζεύξεις. Αυτά τα δίκτυα και οι συζεύξεις πραγματοποιούνται μεταξύ γειτονικών και μη γειτονικών νευρώνων μέσα στους βιολογικούς εγκεφάλους, επηρεάζοντας έτσι τη συμπεριφορά του δυναμικού και την διέγερση κάθε νευρώνα [8], [22]. Συνεχίζοντας με το μοντέλο LIF για διασυνδεδεμένους νευρώνες, η παρακάτω παράσταση περιγράφει τη συμπεριφορά αυτών κατά την ενεργοποίηση.

$$\frac{du_i(t)}{dt} = \mu - u_i(t) + \frac{1}{N_i}\sum_{j=1}^{N} \sigma_{ij}\left[u_j(t) - u_i(t)\right]$$

Εικόνα 2-5: Περιγραφή ρυθμού μεταβολής νευρώνα κατά την αύξηση του δυναμικού.

Στην παραπάνω παράσταση, όπου N ο αριθμός όλων των γειτονικών νευρώνων, η δυναμική του νευρώνα *i* σε χρόνο *t* συμβολίζεται με *ui(t)*. Ο πίνακας σ αναπαριστά τις ενώσεις – συζεύξεις μεταξύ νευρώνων. Το στοιχείο *σij* υποδηλώνει την ισχύ σύζευξης μεταξύ των νευρώνων *i* και *j*. Λόγω των συνδέσεων και της ισχύος τους, το δυναμικό του νευρώνα *i* επηρεάζεται από το δυναμικό των γειτόνων του [8], [22].

Στην παρούσα εργασία μελετώνται νευρώνες που ανήκουν σε κάποιο δίκτυο, δηλαδή που ενώσεις – συζεύξεις μεταξύ τους. Για να μπορέσει η παραπάνω παράσταση να υπολογιστεί από υπολογιστικό σύστημα, χρησιμοποιείται η μέθοδος Euler για τη διακριτοποίηση της [16]. Σε επόμενο κεφάλαιο η εργασία θα παρουσιάσει περισσότερες λεπτομέρειες σχετικά με το πώς πραγματοποιούνται οι υπολογισμοί από τον υπολογιστή. Για την ώρα, πρέπει να σημειωθεί πως αυτό που απαιτεί σημαντικούς πόρους και υπολογιστική ισχύ είναι η πράξη αραιού μητρώου επι διάνυσμα σε μεγάλα νευρωνικά δίκτυα [8].

$$\sum_{j=1}^{N} \sigma_{ij} \left[u_j(t) - u_i(t) \right]$$

Εικόνα 2-6: Αριθμητική παράσταση SPMV.

2.3 Επιλογή μοντέλου LIF

Το μοντέλο Leaky Integrate-and-Fire (LIF) επιλέχθηκε στην παρούσα εργασία όχι μόνο λόγω της εκτεταμένης χρήσης του στη βιβλιογραφία για την προσομοίωση δικτύων βιολογικών νευρώνων, αλλά κυρίως λόγω των ιδιαίτερων υπολογιστικών χαρακτηριστικών που παρουσιάζει. Σε αντίθεση με πιο πολύπλοκα μοντέλα, όπως το Hodgkin–Huxley [17], το LIF προσφέρει μια απλούστερη μαθηματική περιγραφή της δυναμικής των νευρώνων, διατηρώντας ωστόσο τα βασικά χαρακτηριστικά της νευρωνικής δραστηριότητας. Η απλότητα αυτή από υπολογιστικής πλευράς, μεταφράζεται σε σημαντικά πλεονεκτήματα, καθώς το μεγαλύτερο μέρος των υπολογισμών κατά την προσομοίωση μεγάλων νευρωνικών δικτύων βασίζεται σε πράξεις αραιού μητρώου επί διάνυσμα (Sparse Matrix-Vector Multiplication – SPMV). Οι πράξεις αυτές αποτελούν το κυρίαρχο υπολογιστικό φορτίο του αλγορίθμου, ειδικά σε περιπτώσεις όπου ο αριθμός των νευρώνων και των συνδέσεων είναι ιδιαίτερα μεγάλος.

Η συγκεκριμένη ιδιότητα καθιστά το μοντέλο LIF ιδανικό υποψήφιο για μελέτη σε αρχιτεκτονικές παράλληλης επεξεργασίας, όπως οι GPU και τα συστήματα Process-In-Memory (PIM). Σε τέτοιες αρχιτεκτονικές, η απόδοση επηρεάζεται σε μεγάλο βαθμό από την αποτελεσματική διαχείριση της μνήμης και την ελαχιστοποίηση των μεταφορών δεδομένων, παράγοντες που είναι κρίσιμοι στις πράξεις SPMV.

Επιπλέον, η χρήση του LIF επιτρέπει την απομόνωση και την ανάλυση των επιδόσεων του υπολογιστικού υποσυστήματος (εικ. 2-6), χωρίς να παρεμβάλλεται υπερβολική πολυπλοκότητα από το ίδιο το μαθηματικό μοντέλο. Με τον τρόπο αυτό, η παρούσα εργασία επικεντρώνεται κυρίως στη μελέτη της αρχιτεκτονικής απόδοσης των GPU και PIM συστημάτων, αναδεικνύοντας τα πλεονεκτήματα και τους περιορισμούς τους σε ένα ρεαλιστικό, αλλά υπολογιστικά απαιτητικό, επιστημονικό σενάριο.

3 Βιβλιοθήκη SparseP, τεχνικές κατανομής φορτίων και μορφής πινάκων.

3.1 SPMV και Μορφές πινάκων υπολογισμού (CSR, COO, BCSR, BCOO)

Στον χώρο της Πληροφορικής και ειδικότερα στο πεδίο του High Performance Computing (HPC), τα μητρώα (matrices) αναπαρίστανται σε διάφορες μορφές με σκοπό την βελτιστοποίηση των υπολογιστικών πράξεων. Η επιλογή της κατάλληλης αναπαράστασης δεν αποσκοπεί μόνο στην βελτίωση της απόδοσης των πράξεων, στην μείωση της απαιτούμενης μνήμης, αλλά και στο να έρθουν τα μητρώα σε μια κατάσταση που εξυπηρετεί για την αποτελεσματική υλοποίηση των αλγορίθμων [1].

Οι πιο συνήθεις αναπαραστάσεις για τα αραιά μητρώα είναι:

Compressed Sparse Row (CSR) [1], [3], [7]**:**

Στη συγκεκριμένη αναπαράσταση, το αραιό μητρώο αποθηκεύεται με τη χρήση τριών μονοδιάστατων μητρώων. Το πρώτο μητρώο περιέχει όλες τις μη μηδενικές τιμές του αρχικού, διατηρώντας τη σειρά με την οποία εμφανίζονται γραμμικά. Το δεύτερο μητρώο (Row pointer array) έχει μέγεθος ίσο με τον αριθμό των γραμμών του πίνακα συν ένα. Στην κάθε θέση του περιέχει το άθροισμα του πλήθους των μη μηδενικών στοιχείων της γραμμής που αντιστοιχεί στην θέση, το τρίτο μητρώο (Column indices array) αποθηκεύει δείκτες που υποδεικνύουν την στήλη στην οποία βρίσκεται κάθε μη μηδενικό στοιχείο, διατηρώντας τη διάταξη που αντιστοιχεί στον πίνακα τιμών. Με αυτόν τον τρόπο επιτρέπει τον άμεσο εντοπισμό του σημείου εκκίνησης και λήξης των στοιχείων κάθε γραμμής στον πίνακα τιμών.

Η μορφή CSR είναι ιδιαίτερα αποδοτική σε πράξεις όπως ο πολλαπλασιασμός πίνακα-διανύσματος (SPMV), αφού μειώνει τόσο την απαίτηση μνήμης όσο και τον αριθμό των περιττών πράξεων που αφορούν μηδενικά στοιχεία. Αυτή είναι η μορφή πίνακα η οποία θα χρησιμοποιηθεί στην συγκεκριμένη εργασία.

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 30 \\ 40 & 0 & 50 & 60 \end{bmatrix}$$

Εικόνα 3-1: Παράδειγμα Αραιού Μητρώου (CSR)

Values = [10, 20, 30, 40, 50, 60]

Column indices = [0, 1, 3, 0, 2, 3]

Row pointer = [0, 1, 3, 6]

Coordinate List (COO) [1], [3], [7]**.**

Η μορφή Coordinate (COO) είναι μία από τις πιο απλές μεθόδους αναπαράστασης αραιών μητρώων. Η αναπαράσταση γίνεται με την βοήθεια τριών μονοδιάστατων μητρώων: Το πρώτο ονομάζεται (Row indices array**)** αποθηκεύει τους δείκτες των γραμμών όπου βρίσκονται τα μη μηδενικά στοιχεία στο αρχικό μητρώο. Το δεύτερο ονομάζεται (Column indices array) αποθηκεύει τους δείκτες στηλών των πινάκων που βρίσκετε η εκάστοτε μη μηδενική τιμή και το τρίτο (Values array) αποθηκεύει τις αριθμητικές τιμές των μη μηδενικών στοιχείων.

Η μορφή COO είναι ιδιαίτερα χρήσιμη για την εύκολη κατασκευή αραιών πινάκων από δεδομένα, καθώς και για την εισαγωγή/τροποποίηση στοιχείων. Ωστόσο, δεν είναι η πλέον αποδοτική για πράξεις μεγάλης κλίμακας, καθώς απαιτεί περισσότερους δείκτες συνεπώς περισσότερους υπολογιστικούς πόρους και δεν εκμεταλλεύεται την διάταξη των δεδομένων όπως οι πιο σύνθετες μορφές (π.χ. CSR).

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 30 \\ 40 & 0 & 50 & 60 \end{bmatrix}$$

Εικόνα 3-2: Παράδειγμα Αραιού Μητρώου (COO)

Row indices = [0, 1, 1, 2, 2, 2]

Column indices = [0, 1, 3, 0, 2, 3]

Values = [10, 20, 30, 40, 50, 60]

BCSR (Block Compressed Sparse Row) [1], [3], [7], [23]**:**

Η αναπαράσταση Block Compressed Sparse Row (BCSR) αποτελεί μια παραλλαγή της CSR, σχεδιασμένη να εκμεταλλεύεται την δομή των αραιών μητρώων που εμφανίζουν πολλά μη μηδενικά στοιχεία κοντά σε θέσεις χωρίζοντάς σε μικρότερα μητρώα (blocks).

Αντί να αποθηκεύονται μεμονωμένα μη μηδενικά στοιχεία, δείκτες γραμμών και στηλών, το BCSR χωρίζει το αρχικό μητρώο σε μικρότερα μητρώα διαστάσεων $r \times c$. Κάθε υπομητρώο που περιέχει τουλάχιστον ένα μη μηδενικό στοιχείο αποθηκεύεται στην μνήμη, ενώ οι δείκτες αναφέρονται πλέον σε block–γραμμές και block–στήλες αντί για μεμονωμένες θέσεις στοιχείων.

Η αναπαράσταση BCSR υλοποιείται με τρία μονοδιάστατα μητρώα, όπως και το CSR, με τη διαφορά ότι οι γραμμές και οι στήλες αφορούν πλέον blocks και όχι μεμονωμένα στοιχεία. Αυτό οδηγεί σε Μειωμένη χρήση μνήμης, λόγω αποθήκευσης σε συμπαγή blocks, βελτίωση επιδόσεων σε πράξεις όπως ο πολλαπλασιασμός αραιού μητρώου επί διάνυσμα (SPMV), ιδιαίτερα σε αρχιτεκτονικές υψηλών επιδόσεων [1], [3], [23].

Στο παρακάτω σχήμα, το αρχικό μητρώο έχει χωριστεί σε τέσσερα υπό μητρώα διαστάσεων 2×2. Τα μητρώα περιέχουν μη μηδενικά στοιχεία και επομένως αποθηκεύονται. Στη λογική της BCSR, το πρώτο μητρώο (επάνω αριστερά) αντιστοιχεί στη θέση [0,0] του μητρώου Values.

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 30 \\ 40 & 0 & 50 & 60 \end{bmatrix}$$

Εικόνα 3-3: Παράδειγμα Αραιού Μητρώου (BCSR)

$$A = \begin{bmatrix} \begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 30 \end{bmatrix} \\ \begin{bmatrix} 40 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 50 & 60 \\ 0 & 0 \end{bmatrix} \end{bmatrix}$$

Εικόνα 3-4: Αραιό Μητρώο χωρισμένο σε μικρότερά Μητρώα (BCSR)

$$\text{Block(0,0)} = \begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix}$$

$$\text{Block(0,1)} = \begin{bmatrix} 0 & 0 \\ 0 & 30 \end{bmatrix}$$

$$\text{Block(1,0)} = \begin{bmatrix} 40 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\text{Block(1,1)} = \begin{bmatrix} 50 & 60 \\ 0 & 0 \end{bmatrix}$$

Εικόνα 3-5: Δείκτες μητρώων (Blocks) (BCSR)

Values = [Block(0,0), Block(0,1), Block(1,0), Block(1,1)]

Column indices = [0, 1, 0, 1]

Row pointer = [0, 2, 4]

BCOO (Block Coordinate Format) [1], [3], [7], [23]**:**

Η αναπαράσταση BCOO είναι μια παραλλαγή της COO, στην οποία το μητρώο χωρίζεται σε και εδώ σε υπομητρώα, όπως ακριβώς και στην BCSR, αλλά η αποθήκευση ακολουθεί τη λογική της COO δηλαδή υπάρχουν, ένα μονοδιάστατο μητρώο με τους δείκτες γραμμών των μητρώων που περιέχει τουλάχιστον ένα μη μηδενικό στοιχείο, ένα μονοδιάστατο μητρώο με τους δείκτες στηλών που περιέχουν μη μηδενικά στοιχεία και ένα μητρώο που αποθηκεύει τις τιμές των μη μηδενικών στοιχειών**.**

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 30 \\ 40 & 0 & 50 & 60 \end{bmatrix}$$

Εικόνα 3-6:Παράδειγμα Αραιού Μητρώου (BCOO)

$$A = \begin{bmatrix} \begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 30 \end{bmatrix} \\ \begin{bmatrix} 40 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 50 & 60 \\ 0 & 0 \end{bmatrix} \end{bmatrix}$$

Εικόνα 3-7: Αραιό Μητρώο χωρισμένο σε μικρότερά Μητρώα (BCOO)

$$\text{Block(0,0)} = \begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix}$$

$$\text{Block(0,1)} = \begin{bmatrix} 0 & 0 \\ 0 & 30 \end{bmatrix}$$

$$\text{Block(1,0)} = \begin{bmatrix} 40 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\text{Block(1,1)} = \begin{bmatrix} 50 & 60 \\ 0 & 0 \end{bmatrix}$$

Εικόνα 3-8: Δείκτες μητρώων (Blocks) (BCOO)

Row block indices = [0, 0, 1, 1]

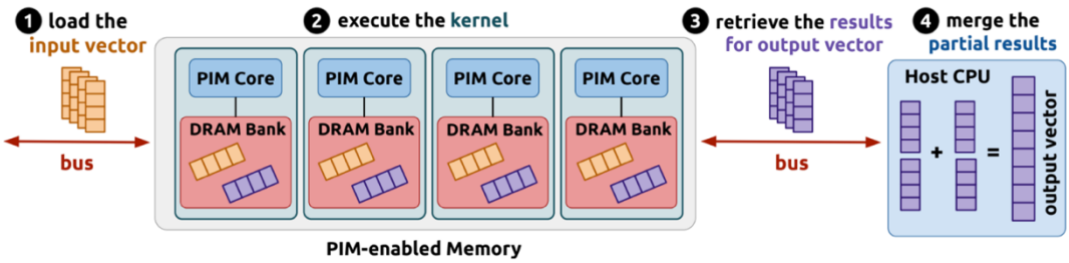
Column block indices = [0, 1, 0, 1]

Values = [Block(0,0), Block(0,1), Block(1,0), Block(1,1)]

3.2 SparseP Βιβλιοθήκη και κατανομή φορτίου στο PIM σύστημα.

Σε κάθε σύστημα παράλληλης επεξεργασίας και ιδιαίτερα σε παραδείγματα που αφορούν αραιά μητρώα, θα πρέπει να πραγματοποιηθεί μια κατανομή φορτίου (στοιχείων αραιών μητρώων και διανυσμάτων στην συγκεκριμένη περίπτωση), σε όλο το εύρος των πυρήνων της εκάστοτε αρχιτεκτονικής π.χ. GPU, PIM. Αυτό θα πρέπει να πραγματοποιηθεί με τέτοιο τρόπο ώστε να επιτευχθεί η αποτελεσματική χρήση ολόκληρου του συστήματος με σκοπό το να αποδώσει στο μέγιστο και χωρίς να κάνει λάθη. Το παραπάνω πραγματοποιείται μέσω διαφόρων τεχνικών στο σύστημα PIM με την χρήση της βιβλιοθήκης SparseP [7], η οποία εκτός από την κατανομή φορτίου στο σύστημα, μετατρέπει και τα μητρώα σε μορφές BCSR, CSR, COO, BCOO [1], [3], [7], [23], καθώς και πραγματοποιεί την πράξη SPMV με τις κατάλληλες συναρτήσεις που διαθέτει.

Συγκεκριμένα, τα αραιά μητρώα και τα διανύσματα για τα οποία πραγματοποιείται ο υπολογισμός του SPMV, κατανέμονται στις μονάδες επεξεργασίας DPU του PIM συστήματος μέσω του host συστήματος. Ύστερα πραγματοποιείται η πράξη από το PIM σύστημα και τα αποτελέσματα επιστρέφονται πίσω στο host σύστημα. Ουσιαστικά το host σύστημα, αναλαμβάνει την αρχική διαχείριση των δεδομένων του αραιού μητρώου και του διανύσματος για την πράξη SPMV (π.χ. μετατροπή αραιού μητρώου σε μορφή CSR), κατανέμοντας αραιά μητρώα και διανύσματα στις μνήμες DRAM του PIM συστήματος. Μετά την ολοκλήρωση της παραπάνω διαδικασίας, εκτελείται η πράξη του SPMV από τα DPU που βρίσκονται στα PIM Chip. Μετά την ολοκλήρωση των υπολογισμών από τις μονάδες PIM, το τελικό αποτέλεσμα συγκεντρώνεται και επιστρέφεται στην αρχιτεκτονική του host. Ο host μπορεί στη συνέχεια να χρησιμοποιήσει αυτά τα αποτέλεσμα για περαιτέρω επεξεργασία ή ανάλυση. Όλη αυτή η διαδικασία επιτυγχάνεται με την βοήθεια την βιβλιοθήκης SparseP.

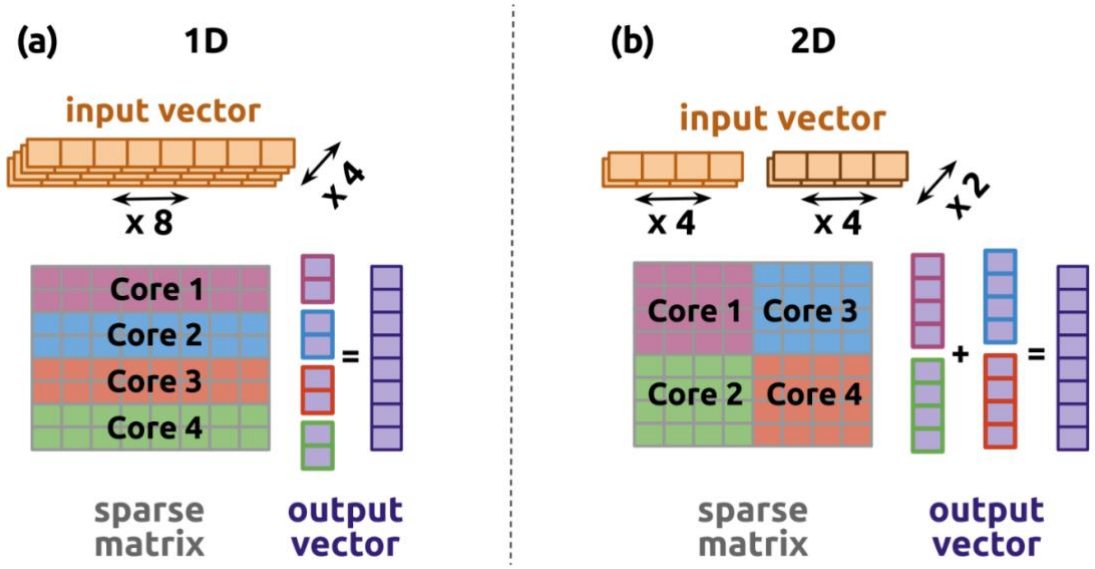


Εικόνα 3-9: Απεικόνιση κατανομής διανύσματος στο σύστημα PIM [7]

Αναφορικά με τις τεχνικές κατανομής στο σύστημα PIM για μορφές μητρώων BCSR, CSR, COO, BCOO, υπάρχουν δυο βασικών ειδών που υποστηρίζει η SparseP.

Τις τεχνικές μίας διάστασης στις οποίες το μητρώο χωρίζεται οριζόντια στους πύρινες PIM (εικ. 3-10), και ολόκληρο το διάνυσμα αντιγράφεται μέσα στις μνήμες DRAM του συστήματος. Κατά αυτήν την τεχνική, το μεγαλύτερο μέρος των υπολογιστικών πράξεων εκτελείται από τις DPU, ενώ το host σύστημα περιορίζεται στη μεταφορά του διανύσματος στις μνήμες DRAM, χωρίς ιδιαίτερη προ-επεξεργασία, πλην την μετατροπής του μητρώου στην εκάστοτε μορφή. Τα αποτελέσματα μετά την πράξη είναι ένας αριθμός διανυσμάτων ίσος με αυτών των πυρήνων, τα οποία ύστερα ενώνονται σε ένα από το host CPU ως προς το διανυσματικό αποτέλεσμα της πράξης [7].

Τις τεχνικές δύο διαστάσεων στις οποίες το μητρώο χωρίζεται στα DPU σε τμήματα δύο διαστάσεων, ο αριθμός των οποίων είναι ίσος με τον αριθμό των πυρήνων (εικ. 3-10). Με αυτό επιδιώκεται η επίτευξη ισορροπίας μεταξύ υπολογιστικού κόστους και κόστους μεταφοράς δεδομένων, καθώς μόνο ένα υποσύνολο των στοιχείων του διανύσματος εισόδου αντιγράφεται στη μνήμη DRAM του κάθε DPU. Ωστόσο, σε αυτή την τεχνική μετά τον υπολογισμό παράγεται μεγαλύτερος αριθμός διανυσμάτων σε σχέση με την τεχνική της μίας διάστασης. Αυτά τα διανύσματα μεταφέρονται στο host σύστημα και συγχωνεύονται από την μονάδα CPU, σχηματίζοντας το τελικό διάνυσμα εξόδου. Στη βιβλιοθήκη SparseP, το βήμα της συγχώνευσης που εκτελείται από τους πυρήνες CPU πραγματοποιούνται παράλληλα μέσω του OpenMP API [7], [24].

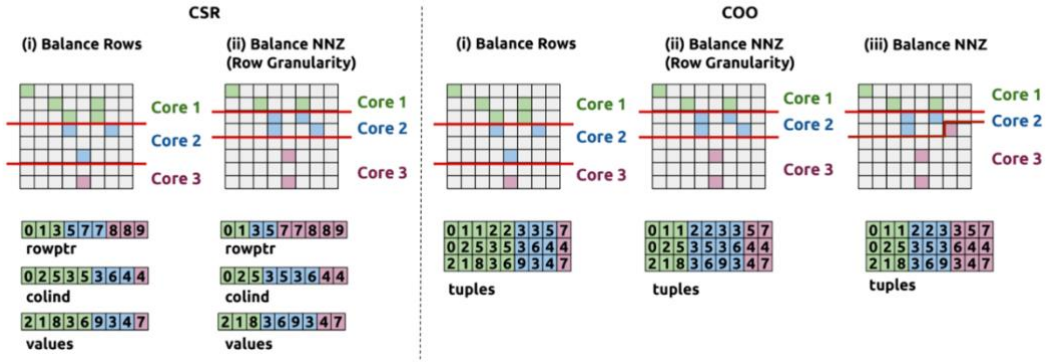


Εικόνα 3-10: Απεικόνιση τύπου κατανομής φορτίου στο σύστημα PIM

Παρακάτω η εργασία θα εμβαθύνει μόνο σε λεπτομέρειες τεχνικών κατανομής σε μια διάσταση για CSR και COO, διότι είναι και οι τεχνικές για τις οποίες πραγματοποιήθηκε η έρευνα (συγκεκριμένα CSRrow, CSRnnz [7]).

3.3 Τεχνικές κατανομής φορτίου μιας διάστασης CSR, COO

Για να κατανεμηθεί αποτελεσματικά το φορτίο για τον υπολογισμό του SPMV σε πολλαπλούς πυρήνες μέσω της τεχνικής διαμερισμού 1D, η βιβλιοθήκη SparseP παρέχει διάφορα σχήματα για κάθε υποστηριζόμενη συμπιεσμένη μορφή αραιού μητρώου. Στο παρακάτω σχήμα παρουσιάζονται παραδείγματα υλοποίησης σε πολλαπλούς πυρήνες χρησιμοποιώντας σχήματα κατανομής φορτίου για της μορφές CSR και COO. Η εξισορρόπηση φορτίου είτε ανά γραμμές (ROW), ώστε κάθε πυρήνας να επεξεργάζεται σχεδόν τον ίδιο αριθμό γραμμών, είτε ανά μη μηδενικων στοιχείων, ώστε κάθε πυρήνας να επεξεργάζεται σχεδόν τον ίδιο αριθμό μη μηδενικών στοιχείων. [7].



Εικόνα 3-11:Απεικόνιση τύπου κατανομής φορτίου με τεχνική μιας διάστασης στο σύστημα PIM [7]

Partitioning Technique	Compressed Format	Load Balancing Across PIM Cores
1D	CSR	rows (**CSR.row**) nnz* (**CSR.nnz**)
	COO	rows (**COO.row**) nnz* (**COO.nnz-rgrn**) nnz (**COO.nnz**)
	BCSR	blocks† (**BCSR.block**) nnz† (**BCSR.nnz**)
	BCOO	blocks (**BCOO.block**) nnz (**BCOO.nnz**)

Εικόνα 3-12: Υποστηριζόμενες Τεχνικές ταξινόμησης μητρώων από βιβλιοθήκη SparseP [7]

3.4 Τεχνικές κατανομής νημάτων στις μονάδες επεξεργασίας

Όπως προαναφέρθηκε κάθε στο σύστημα PIM υποστηρίζεται η χρήση πολλαπλών νημάτων. Όπως για τα δεδομένα έτσι και για τα νήματα η βιβλιοθήκη SparseP πραγματοποιεί τεχνικές εξισορρόπησης φορτίου για κάθε συμπιεσμένη μορφή αραιού μητρώου, με σκοπό την μέγιστη απόδοση και ορθότητα υπολογισμών. Με παρόμοιο τρόπο όπως εξηγείται και παραπάνω, για τις μορφές CSR και COO, εξισορροπείται η εκτέλεση των νημάτων είτε βάση γραμμών (rows), ώστε κάθε νήμα να επεξεργάζεται σχεδόν τον ίδιο αριθμό γραμμών (rows), είτε τα μη μηδενικών στοιχείων (nnz) [7].

Συγκεκριμένα για την μορφή CSR**,** το μητρώο αποθηκεύεται με βάση τη σειρά των γραμμών του. Αυτό σημαίνει ότι ολόκληρες οι γραμμές του πίνακα ανατίθενται σε κάθε νήμα για επεξεργασία, προκειμένου να επιτευχθεί ομοιόμορφη κατανομή του υπολογιστικού έργου. Στο σύστημα PIM της UPMEM, στα στοιχεία για τα διανύσματα εξόδου πραγματοποιείται προσπέλαση σε κλίμακα των 64-bit στη μνήμη DRAM. Έτσι, όταν η εξισορρόπηση πραγματοποιείται σε κλίμακα γραμμών, αναθέτονται γραμμές στα νήματα σε τμήματα των 8/sizeof (datatype (int32, int64, fp32, fp64 etc.)) [7].

Ο συγχρονισμός μεταξύ των νημάτων κατά τις εγγραφές στα στοιχεία του διανύσματος εξόδου μπορεί να υλοποιηθεί με τρεις διαφορετικές προσεγγίσεις:

Coarse-Grained Locking (lb-cg): Ένα καθολικό mutex [23], [25] προστατεύει όλα τα στοιχεία του διανύσματος εξόδου συνολικά. Αυτή η μέθοδος είναι απλή στην υλοποίηση, αλλά περιορίζει σημαντικά την παραλληλία, καθώς τα νήματα έρχονται σε συχνή αναμονή [7].

Fine-Grained Locking (lb-fg): Πολλαπλά mutexes [23], [25] προστατεύουν επιμέρους στοιχεία του διανύσματος εξόδου. Η βιβλιοθήκη SparseP υλοποιεί έναν μηχανισμό που αντιστοιχίζει mutexes στα στοιχεία του διανύσματος εξόδου με κυκλικό τρόπο (Round Robin). Το API του UPMEM υποστηρίζει έως 56 mutexes [2], [7]. Στην συγκεκριμένη εφαρμογή χρησιμοποιούνται 32 *mutexes*, ώστε η αντιστοίχιση να μπορεί να υλοποιηθεί με μία μόνο πράξη μετατόπισης στη διεύθυνση MRAM, αποφεύγοντας δαπανηρές πράξεις διαίρεσης [7].

Lock-Free (lf): Δεδομένου ότι οι μορφές αποθήκευσης είναι ταξινομημένες κατά γραμμές ή κατά μπλοκ-γραμμές, οι συνθήκες race condition [5], [23], [25] στα στοιχεία του διανύσματος εξόδου προκύπτουν μόνο σε περιορισμένο αριθμό περιπτώσεων.· Συγκεκριμένα, είτε όταν μια γραμμή (ή μπλοκ-γραμμή για τις μορφές BCSR/BCOO) κατανέμεται μεταξύ διαφορετικών νημάτων, είτε όταν συνεχόμενα στοιχεία του διανύσματος εξόδου που επεξεργάζονται από διαφορετικά νήματα ανήκουν στην ίδια διεύθυνση DRAM. Στην προτεινόμενη προσέγγιση χωρίς mutex, τα νήματα αποθηκεύουν προσωρινά μερικά αποτελέσματα για αυτά τα λίγα στοιχεία στη μνήμη δεδομένων WRAM που διαθέτει το σύστημα. Στη συνέχεια, ένα μόνο νήμα συγχωνεύει τα αποτελέσματα και γράφει το τελικό αποτέλεσμα για το αντίστοιχο στοιχείο του διανύσματος εξόδου στην τράπεζα DRAM [7].

Compressed Format	Load Balancing Across Threads	Synchronization Approach
CSR	rows (**CSR.row**)	-
	nnz* (**CSR.nnz**)	-
COO	rows (**COO.row**)	-
	nnz* (**COO.nnz-rgrn**)	-
	nnz (**COO.nnz**)	lb-cg / lb-fg / lf
BCSR	blocks† (**BCSR.block**)	lb-cg / lb-fg (only for int8 and small block sizes)
	nnz† (**BCSR.nnz**)	lb-cg / lb-fg (only for int8 and small block sizes)
BCOO	blocks (**BCOO.block**)	lb-cg / lb-fg / lf
	nnz (**BCOO.nnz**)	lb-cg / lb-fg / lf

Εικόνα 3-13: Υποστηριζόμενες τεχνικές συγχρονισμού νημάτων από την βιβλιοθήκη SparseP [7]

3.5 Εκτέλεση SPMV με SparseP

Συνεχίζοντας με την εκτέλεση του SPMV αφού πραγματοποιηθούν τα παραπάνω οι πυρήνες του PIM (DPU) περιλαμβάνουν τρεις τύπους δομών δεδομένων:

Τα μητρώα που αποθηκεύουν τα μη μηδενικά στοιχεία, δηλαδή τις τιμές (values[]) και τις θέσεις των μη μηδενικών στοιχείων rowptr[], colind[] για CSR, tuples[] για COO, browptr[], bcolind[] για BCSR, browind[], bcolind[] για BCOO.

Το μητρώο που πρακτικά είναι το διάνυσμα εισόδου προς στο σύστημα.

Τα μητρώα που είναι τα διανυσματικά αποτελέσματα που δημιουργούνται, τα οποία ενοποιούνται ύστερα από το host σύστημα ή από ένα νήμα εντός του PIM συστήματος.

Για αρχή ο αλγόριθμος εκτελεί σειριακές προσπελάσεις μνήμης στα αραιά μητρώα στα οποία αποθηκεύονται τα μη μηδενικά στοιχεία και οι θέσεις τους. Προκειμένου να μπορέσει γίνει η εκμετάλλευση μνημών τύπου scratchpad ή cache για καλύτερη απόδοση, κάθε νήμα διαβάζει τα μη μηδενικά στοιχεία με ανάκτηση μεγάλων τμημάτων δεδομένων (chunks) από τη DRAM του host συστήματος, προς τις μνήμες του PIM (coarse-grained [5], [14], [23], [26]). Στη συνέχεια, τα στοιχεία διαβάζονται σε πιο λεπτομερές επίπεδο (fine-grained [5], [14], [23], [26]). Στο σύστημα UPMEM PIM, ανακτώνται τμήματα δεδομένων μεγέθους 256 byte για την αναγνώριση των μη μηδενικών στοιχείων, όπως προτείνεται από το UPMEM API, καθώς οι μεταφορές των 256 byte αξιοποιούν πλήρως το διαθέσιμο εύρος ζώνης της μνήμης DRAM [2]. Για τις μορφές BCSR και BCOO, μόνο για τον πίνακα που αποθηκεύει τις τιμές των μη μηδενικών στοιχείων, η ανάκτηση από τη DRAM στη μνήμη δεδομένων γίνεται σε τμήματα μεγέθους $r \times c \times$ *sizeof(data_type)* byte, με την προϋπόθεση ότι το μητρώο αποθηκεύεται σε μητρώο διαστάσεων $r \times c$. *[7]*

Αναφορικά με τα διανύσματα, προκαλούνται μη κανονικές προσπελάσεις μνήμης στα στοιχεία του διανύσματος εισόδου. Συγκεκριμένα, οι προσπελάσεις στα στοιχεία του διανύσματος εισόδου καθορίζονται από τις θέσεις στηλών (column indexes) των μη μηδενικών στοιχείων κάθε συγκεκριμένου μητρώου. Δεδομένου ότι τα μητρώα που εμπλέκονται στον SPMV είναι ιδιαίτερα αραιά, δηλαδή οι δείκτες στηλών των μη μηδενικών στοιχείων παρουσιάζουν μεγάλη διακύμανση, οι προσπελάσεις στη μνήμη για το διάνυσμα εισόδου δεν χρήζουν τόσο μεγάλης ανάγκης μνήμης scratchpad, cache. Έτσι, στις υλοποιήσεις SPMV, τα νήματα ενός πυρήνα PIM προελαύνουν απευθείας τα

στοιχεία του διανύσματος εισόδου μέσω της DRAM, για τις μορφές CSR και COO σε 64-bit μεγέθους ανά ανάγνωση, ενώ για τις μορφές BCSR και BCOO σε μέγεθος *c* × *sizeof(data_type)* byte, όπου *c* είναι η οριζόντια διάσταση του μεγέθους του μπλοκ. [7]

Όσον αφορά το διάνυσμα εξόδου, τα νήματα αποθηκεύουν προσωρινά τα αποτελέσματα για τα ίδια στοιχεία του διανύσματος εξόδου στη μνήμη δεδομένων (*scratchpad* ή *cache*) ώστε να εκμεταλλευτούν την γρήγορη πρόσβαση δεδομένων από τις συγκεκριμένες μνήμες, μέχρι να γίνει η προσπέλαση σε όλα τα μη μηδενικά στοιχεία της ίδιας γραμμής. Στη συνέχεια, τα παραγόμενα αποτελέσματα γράφονται στην τράπεζα DRAM σε λεπτομερές επίπεδο (fine-granularity) [5], [14], [23], [26]. Για τις μορφές CSR και COO σε 64-bit μεγέθους ανά ανάγνωση, και για τις μορφές BCSR και BCOO σε μέγεθος *r* × *sizeof(data_type)* byte, όπου *r* είναι η κατακόρυφη διάσταση του μεγέθους του μπλοκ. [7]

3.6 Σχεδίαση και πειραματική υλοποίηση

Η βιβλιοθήκη SparseP υποστηρίζει πληθώρα μορφών αραιών μητρώων και τεχνικών κατανομής φορτίου, γεγονός που προσφέρει μεγάλη ευελιξία, αλλά ταυτόχρονα απαιτεί προσεκτικές επιλογές σχεδίασης. Στο πλαίσιο της παρούσας εργασίας, πραγματοποιήθηκε αξιολόγηση των διαθέσιμων μορφών αποθήκευσης και τεχνικών κατανομής, με στόχο τη βέλτιστη ισορροπία μεταξύ υπολογιστικού κόστους, κατανάλωσης μνήμης και κόστους επικοινωνίας με το host σύστημα.

Αν και η βιβλιοθήκη SparseP υποστηρίζει μορφές όπως BCSR και BCOO, οι οποίες μπορούν να προσφέρουν πλεονεκτήματα σε συγκεκριμένα πρότυπα αραιότητας, τελικά επιλέχθηκε η μορφή CSR. Η επιλογή αυτή βασίστηκε στο γεγονός ότι τα μητρώα που προκύπτουν από τη μοντελοποίηση των νευρωνικών δικτύων παρουσιάζουν αραιότητα που ευθυγραμμίζεται καλύτερα με γραμμική αποθήκευση, χωρίς έντονη block δομή.

- **Τυχαία Αραιότητα:** Τα μητρώα βαρών στα νευρωνικά δίκτυα συχνά στερούνται έντονης δομής μπλοκ (block structure), καθιστώντας τα block formats λιγότερο αποδοτικά λόγω του "padding" με μηδενικά στοιχεία.
- **Sequential Access:** Η CSR ευθυγραμμίζεται με τη γραμμική επεξεργασία ανά σειρά, γεγονός που διευκολύνει τη σειριακή προσπέλαση στη μνήμη της DPU (MRAM), μειώνοντας τις σύνθετες κατανομές φορτίων στην μνήμη.

Παράλληλα, επιλέχθηκαν τεχνικές κατανομής φορτίου μίας διάστασης (1D partitioning), καθώς επιτρέπουν την εκτέλεση του κύριου όγκου των υπολογισμών εντός των DPU, περιορίζοντας τη συμμετοχή του host συστήματος σε απλές λειτουργίες διανομής και συγχώνευσης αποτελεσμάτων. Οι τεχνικές CSRrow και CSRnnz χρησιμοποιήθηκαν με στόχο την εξισορρόπηση του υπολογιστικού φόρτου μεταξύ των DPU, ανάλογα με τη δομή του κάθε αραιού μητρώου.

Οι παραπάνω επιλογές σχεδίασης αποσκοπούσαν στη δημιουργία ενός πειραματικού πλαισίου που να αναδεικνύει τις πραγματικές δυνατότητες και περιορισμούς της αρχιτεκτονικής PIM, χωρίς να εισάγονται επιπλέον επιβαρύνσεις από μη αναγκαία πολύπλοκες μορφές αποθήκευσης ή τεχνικές κατανομής.

4 Υλοποιήσεις Μοντέλου LIF

4.1 Υλοποιήσεις CPU

Η υλοποίηση του αλγορίθμου SPMV σε περιβάλλον CPU αποτέλεσε το θεμέλιο της πειραματικής διαδικασίας, λειτουργώντας ως το απαραίτητο σημείο αναφοράς (baseline) για την συγκριτική αξιολόγηση των επιδόσεων. Η αρχιτεκτονική της CPU, αν και προσφέρει ένα περιβάλλον υψηλής προγραμματιστικής ευελιξίας και εύκολης επαλήθευσης της ορθότητας των αποτελεσμάτων, παρουσιάζει από την φύση της σαν αρχιτεκτονική περιορισμούς σε τέτοιου είδους εφαρμογές υπολογισμών. Έτσι η εκτέλεση σε CPU ανέδειξε γρήγορα τους περιορισμούς της σειριακής ή περιορισμένα παράλληλης επεξεργασίας, ειδικά σε περιπτώσεις μεγάλων αραιών μητρώων. Η απόδοση του αλγορίθμου επηρεάζεται σε μεγάλο βαθμό από την πρόσβαση στη μνήμη και δυνατότητα παράλληλης επεξεργασίας, καθώς οι πράξεις SPMV χαρακτηρίζονται από χαμηλή υπολογιστική ένταση και έντονη εξάρτηση από τη χωρική και χρονική τοπικότητα των δεδομένων. Ως αποτέλεσμα, η CPU λειτουργεί κυρίως ως baseline υλοποίηση, προσφέροντας μεν σταθερότητα και προβλεψιμότητα, αλλά περιορισμένες δυνατότητες απόδοσης σε σύγκριση με πιο εξειδικευμένες αρχιτεκτονικές παράλληλης επεξεργασίας.

Συνεχίζοντας με τις υλοποιήσεις LIF για την CPU, υπάρχουν τρείς: Sequential (Σειριακή) (εικ. 4-1), Reorganized (εικ. 4-2) και gemv (εικ. 4-3) [8].

Αν και στην συγκεκριμένη διατριβή δεν μελετάται η απόδοση της CPU, οι υλοποιήσεις για αυτήν εξηγούνται με σκοπό να γίνει πιο κατανοητή η εφαρμογή και υλοποίηση των κατάλληλων αλγορίθμων.

4.1.1 Σειριακή υλοποίηση

Algorithm 1 Sequential implementation of the LIF model

```
Input: time_steps                          ▷ Number of time-steps to simulate
Input: N                                   ▷ Number of neurons
Input: Δt
Input: μ                                   ▷ Exponential integration factor
Input: u_th                                ▷ Neuron firing threshold
Input: u_rest                              ▷ Neuron rest state
Input: u[N]                                ▷ Potential of neurons at time t
Input: N_i[N]                              ▷ Number of neighbours per neuron
Input: σ[N][N]                             ▷ Connectivity matrix

Var: u_next[N]                             ▷ Potential of neurons at time t + Δt

u ← Initialize potential of each neuron in the range [0, u_th)
σ ← Initialize connectivity matrix

for t ← 1, time_steps do                   ▷ Iteration over time-steps
    for i ← 1, N do                        ▷ Iteration over neurons
        u_next[i] ← u[i] + Δt · (μ − u[i])
        sum ← 0
        for j ← 1, N do                    ▷ Iteration over neighbours
            sum ← sum + σ[i][j] · (u[j] − u[i])
        end for
        u_next[i] ← u_next[i] + Δt · sum/N_i[i]
        if u_next[i] > u_th then           ▷ Neuron fires
            u_next[i] ← u_rest
        end if
    end for
    Exchange u and u_next
end for
```

Εικόνα 4-1:Σειρακή υλοποίηση του μοντέλου LIF σε CPU [8]

Η παραπάνω υλοποίηση αλγορίθμου (εικ. 4-1), είναι μια σειριακού τύπου υλοποίηση του μοντέλου LIF σε αρχιτεκτονική CPU.

Πιο συγκεκριμένα ο αλγόριθμος προσομοιώνει τη δυναμική πολλών νευρώνων σε ένα δίκτυο για έναν αριθμό χρονικών βημάτων (*time_steps*). Κάθε νευρώνας έχει ένα δυναμικό *u[i]*, το οποίο εξελίσσεται ανά βήμα υπολογισμών της εκάστοτε εξίσωσης.

Μεταβλητές αλγορίθμου:

1. *time_steps* → Αριθμός χρονικών βημάτων.
2. *N* → Συνολικός αριθμός νευρώνων.
3. *Δt* → Χρονικό βήμα της προσομοίωσης.
4. *μ* → Εκθετικός παράγοντας ολοκλήρωσης (ρυθμίζει την εξέλιξη του δυναμικού).
5. *uth* → Κατώφλι ενεργοποίησης του νευρώνα.
6. *urest* → Τιμή στην οποία επανέρχεται το δυναμικό όταν ο νευρώνας "εκφορτίζεται" (fires).
7. *u[N]* → Πίνακας με τα δυναμικά των νευρώνων στη χρονική στιγμή t.
8. *Ni[N]* → Αριθμός γειτόνων κάθε νευρώνα.
9. *σ[N][N]* → Πίνακας συνδεσιμότητας (απεικόνιση σχέσης νευρώνων).

Βήματα αλγορίθμου:

- Ο αλγόριθμος ξεκινάει δημιουργώντας το διάνυσμα *u* που αφορά την διέγερση των νευρώνων για το εύρος τιμών από 0-1 (αριθμοί κινητής υποδιαστολής διπλής ακρίβειας).
- Ο αλγόριθμος ξεκινάει την επανάληψη για κάθε χρονικό βήμα (*time_step*).
- Μέσα σε κάθε βήμα γίνεται η προσπέλαση για κάθε νευρώνα αφού γίνει ο υπολογισμός του *unext* προχωράει στην προσπέλαση του κάθε γείτονα του νευρώνα και γίνονται οι απαραίτητες πράξεις.
- Έπειτα υπολογίζεται το *unext* μετά την αλληλεπίδραση με τους γείτονες.
- Ύστερα γίνεται έλεγχος ενεργοποίησης του νευρώνα.
- Αν ο νευρώνας έχει ενεργοποιηθεί τότε γυρνάει σε κατάσταση ηρεμίας.
- Επανάληψη των παραπάνω μέχρι την ολοκλήρωση των χρονικών βημάτων.

Επιστρέφοντας στο πρόβλημα της απόδοσης και την μνήμη, πέρα από το γεγονός του ότι απαιτείται παραλληλισμός και υπολογιστική ισχύ λόγω της φύσης των πράξεων ο υπολογισμός σχετίζεται και με την ενημέρωση του δυναμικού ενός νευρώνα σε ένα νευρωνικό δίκτυο. Για να ενημερωθεί το δυναμικό του νευρώνα, πρέπει να διαβαστούν και να εγγραφούν δεδομένα από τις εκάστοτε μεταβλητές (*σ, u*) από και προς την μνήμη. Το μητρώο συνδεσιμότητας *σ* περιέχει τους συντελεστές συζεύξεων μεταξύ του νευρώνα i και των γειτονικών του νευρώνων και πρέπει να προσπεραστούν συνολικά N συντελεστές συζεύξεων. Το διάνυσμα δυναμικού *u* περιέχει τα δυναμικά των νευρώνων και πρέπει να προσπεραστούν συνολικά από την μνήμη N δυναμικά [8]. Επιπλέον, ορισμένες τιμές (όπως το τρέχον δυναμικό *ui(t),* το νέο δυναμικό *ui(t+Δt),* και άλλες σταθερές όπως *Δt, μ,* και *uthu*) ιδανικά θα πρέπει να και αυτές να διαβαστούν από τη μνήμη και να αποθηκευτούν σε καταχωρητές (Registers) για να μπορεί η μονάδα CPU να τις αξιοποιήσει.

Η συνολική κίνηση δεδομένων (σε bytes) για κάθε νευρώνα περιλαμβάνει την ανάγνωση και εγγραφή 2N + 6 τιμών διπλής ακρίβειας, καθεμία από τις οποίες απαιτεί 8 bytes. Επομένως, η συνολική πρόσβαση στη μνήμη είναι 8(2N+6) bytes [8]. Ο συνολικός αριθμός των πράξεων αριθμών κινητής υποδιαστολής διπλής ακρίβειας που απαιτούνται είναι 3N+6.

Η Αριθμητική Ένταση (AI: Arithmetic Intensity) υπολογίζεται ως ο λόγος των πράξεων κινητής υποδιαστολής προς την κίνηση δεδομένων:

$$AI = \frac{N \cdot (3N+6)}{N \cdot 8(2N+6)} = \frac{3}{16} - \frac{3}{16 \cdot (N+3)} \approx \frac{3}{16}$$

Εικόνα 4-2:Arithmetic Intensity Αλγορίθμου [8]

Το αποτέλεσμα αυτό υποδηλώνει ότι το πρόβλημα περιορίζεται από τη μνήμη, δηλαδή η απόδοση περιορίζεται περισσότερο από την ταχύτητα πρόσβασης στη μνήμη παρά από τον αριθμό των πράξεων δεκαδικών. Ως εκ τούτου, η βελτιστοποίηση των προσβάσεων στη μνήμη είναι κρίσιμη για τη βελτίωση της απόδοσης [8].

4.1.2 Reorganized υλοποίηση

Algorithm 2 *"Reorganized"* CPU implementation

```
1: Input: time_steps                                   ▷ Number of time-steps to simulate
2: Input: N                                            ▷ Number of neurons
3: Input: Δt
4: Input: μ                                            ▷ Exponential integration factor
5: Input: u_th                                         ▷ Neuron firing threshold
6: Input: u_rest                                       ▷ Neuron rest state
7: Input: u[N]                                         ▷ Potential of neurons at time t
8: Input: N_i[N]                                       ▷ Number of neighbours per neuron
9: Input: σ[N][N]                                      ▷ Connectivity matrix
10:
11: Var: u_next[N]                                     ▷ Potential of neurons at time t + Δt
12: Var: sum_of_σ[N]                                   ▷ Sum of rows of σ
13:
14: u ← Initialize potential of each neuron in the range [0, u_th)
15: σ ← Initialize connectivity matrix
16:
17: for i ← 1, N do                                    ▷ Initialize sum_of_σ
18:     sum_of_σ[i] ← 0
19:     for j ← 1, N do
20:         sum_of_σ[i] ← sum_of_σ[i] + σ[i][j]
21:     end for
22: end for
23:
24: for t ← 1, time_steps do                           ▷ Iteration over time-steps
25:     for i ← 1, N do                                ▷ Iteration over neurons
26:         u_next[i] ← u[i] + Δt · (μ − u[i])
27:         sum ← −u[i] · sum_of_σ[i]
28:         for j ← 1, N do                            ▷ Iteration over neighbours
29:             sum ← sum + σ[i][j] · u[j]
30:         end for
31:         u_next[i] ← u_next[i] + Δt · sum/N_i[i]
32:         if u_next[i] > u_th then                   ▷ Neuron fires
33:             u_next[i] ← u_rest
34:         end if
35:     end for
36:     Exchange u and u_next
37: end for
```

Εικόνα 4-3:Reorganized υλοποίηση του μοντέλου LIF σε CPU [8].

Εδώ η διαφορά με τον παραπάνω αλγόριθμο είναι η προσπάθεια μείωσης της πολυπλοκότητας, διαχωρίζοντας τα αθροίσματα και υπολογίζοντας ανεξάρτητα, λαμβάνοντας υπόψη το γεγονός ότι το δυναμικό ui(t) παραμένει σταθερό στη χρονική στιγμή t [8]. Πιο συγκεκριμένα, η παραπάνω αριθμητική παράσταση (εικ. 2-6) μπορεί να γραφεί ως εξής:

$$\sum_{j=1}^{N} \sigma_{ij} \cdot u_j(t) - u_i(t) \cdot \sum_{j=1}^{N} \sigma_{ij}$$

Εικόνα 4-4: Νέα Αριθμητική παράσταση SPMV [8]

Δεδομένου ότι το δεύτερο άθροισμα *σij* παραμένει σταθερό κατά τη διάρκεια μιας προσομοίωσης, μπορεί να προϋπολογιστεί στην αρχή της διαδικασίας. Για τον σκοπό αυτό, ορίζεται με έναν επιπλέον διανυσματικό μητρώο ***sum_of_σ*** με N στοιχεία, που κάθε στοιχείο του αποθηκεύει το αποτέλεσα από το παραπάνω άθροισμα για έναν νευρώνα στο δίκτυο [8].

Έτσι καταλήγει το εξής:

$$\sum_{j=1}^{N} \sigma_{ij} \cdot u_j(t) - u_i(t) \cdot sum\_of\_\sigma_i$$

Εικόνα 4-5: Νέα Αριθμητική παράσταση SPMV [8]

Αυτή η αλλαγή μειώνει σημαντικά τον αριθμό των πράξεων που απαιτούνται για τον υπολογισμό. Αυτή η υλοποίηση του αλγόριθμου ονομάζεται ως ¨***Reorganized***¨**.**

4.1.3 gemv υλοποίηση

Algorithm 3 *"gemv"* CPU implementation

```
1: for t ← 1, time_steps do
2:     cblas_dgemv(CblasRowMajor, CblasNoTrans, N, N, 1, σ, N, u, 1, 0, u_next, 1)
3:
4:     for i ← 1, N do                ▷ Iteration over neurons to update result of vector-matrix
5:         sum ← Δt · (u_next[i] − u[i] · sum_of_σ[i]) /N_i[i]
6:         u_next[i] ← u[i] + Δt · (μ − u[i]) + sum
7:         if u_next[i] > u_th then
8:             u_next[i] ← u_rest
9:         end if
10:    end for
11:    Exchange u and u_next
12: end for
```

Εικόνα 4-6: gemv υλοποίηση του μοντέλου Leaky Integrate-and-Fire (LIF) [8]

Στον συγκεκριμένο αλγόριθμο για το υπολογισμό της αριθμητικής παράστασης στην εικόνα 4-5, χρησιμοποιείται η συνάρτηση ***cblas_dgemv()*** [27] από τη βιβλιοθήκη *Intel MKL [27]*, η οποία περιλαμβάνει υλοποιήσεις υψηλής απόδοσης για εφαρμογές γραμμικής άλγεβρας, βελτιστοποιημένες για επεξεργαστές *Intel*.

Εδώ παρουσιάζεται μόνο το σημείο των χρονικών βημάτων για αυτήν την υλοποίηση, καθώς τα υπόλοιπα μέρη είναι ίδια με τον παραπάνω αλγόριθμο (reooganized).

Το αποτέλεσμα της πράξης SPMV αποθηκεύεται στον πίνακα *unext* και στη συνέχεια ενημερώνεται περαιτέρω με τον δεύτερο όρο της εξίσωσης και τις υπόλοιπες πράξεις της εξίσωσης:

$$u_i(t+\Delta t) = u_i(t) + \Delta t \left[\mu - u_i(t) + \frac{1}{N_i} \sum_{j=1}^{N} \sigma_{ij} \left[u_j(t) - u_i(t) \right] \right]$$

Εικόνα 4-7: Αριθμητική παράσταση LIF [8]

Παραπάνω αποτυπώνεται η βασική λογική της υλοποίησης σε CPU, όπως αυτή σχεδιάστηκε και υλοποιήθηκε στην παρούσα εργασία. Ιδιαίτερη έμφαση δόθηκε στη σαφή αντιστοίχιση των δομών δεδομένων της μορφής CSR με τις πράξεις του αλγορίθμου, ώστε να διασφαλιστεί η ορθότητα και η αναγνωσιμότητα του κώδικα.

Κατά την υλοποίηση, επιλέχθηκε η σειριακή εκτέλεση των βρόχων ως σημείο αναφοράς (baseline), επιτρέποντας την εύκολη επαλήθευση των αποτελεσμάτων και τη σύγκριση με τις αντίστοιχες υλοποιήσεις σε GPU και PIM. Η επιλογή αυτή διευκόλυνε τον εντοπισμό σφαλμάτων και την κατανόηση της συμπεριφοράς του αλγορίθμου πριν τη μετάβαση σε πιο σύνθετες αρχιτεκτονικές.

4.2 Υλοποιήσεις GPU

Η υλοποίηση του αλγορίθμου σε αρχιτεκτονική GPU εκμεταλλεύεται τον υψηλό βαθμό παραλληλισμού που προσφέρουν τα σύγχρονα συστήματα που χρήζουν την επεξεργασία γραφικών στοιχείων. Η αντιστοίχιση γραμμών ή μη μηδενικών στοιχείων του αραιού μητρώου σε νήματα επιτρέπει την ταυτόχρονη επεξεργασία μεγάλου όγκου δεδομένων, γεγονός που οδηγεί σε σημαντική επιτάχυνση σε σχέση με την υλοποίηση σε CPU.

Η απόδοση της GPU στο συγκεκριμένο πρόβλημα δεν είναι γραμμική, καθώς επηρεάζεται από δύο κρίσιμους παράγοντες:

- **Thread Divergence & Load Balancing:** Η βασική πρόκληση προκύπτει από την ανομοιομορφία των συνάψεων ανά νευρώνα. Όταν διαφορετικά νήματα threads ενός Warp επεξεργάζονται γραμμές με σημαντικά διαφορετικό αριθμό μη μηδενικών στοιχείων, δημιουργείται ένα φαινόμενο που νήματα με λιγότερα δεδομένα παραμένουν αδρανή (idle) περιμένοντας τα υπόλοιπα να ολοκληρώσουν, γεγονός που μειώνει το συνολικό utilization της μονάδας SM (Streaming Multiprocessor). [28]
- **Memory Coalescing:** Η απόδοση του SPMV εξαρτάται άμεσα από την δυνατότητα της GPU να ομαδοποιεί τις δεσμεύσεις μνήμης. Στην περίπτωση του CSR format, ενώ η ανάγνωση των πινάκων values και column_indices είναι συχνά ευθυγραμμισμένη, η πρόσβαση στο διάνυσμα των δυναμικών των νευρώνων μπορεί να είναι ακανόνιστη, προκαλώντας καθυστερήσεις [29]

Παρά τις παραπάνω προκλήσεις, η GPU παραμένει ιδιαίτερα αποδοτική για το συγκεκριμένο πρόβλημα, κυρίως λόγω του υψηλού υψηλούς εύρους ζώνης στην μνήνη και της δυνατότητας απόκρυψης καθυστερήσεων μνήμης μέσω μαζικού παραλληλισμού. Το συγκριτικό πλεονέκτημα της GPU έγκειται στη δυνατότητά της για latency hiding [29]. Ακόμη και αν μια δέσμευση στη μνήμη VRAM καθυστερεί, ο scheduler της GPU εναλλάσσει ακαριαία την εκτέλεση σε άλλα έτοιμα Warps. Με αυτόν τον τρόπο, η GPU καταφέρνει να διατηρεί την απόδοση της.

Σχετικά με την υλοποίηση για την GPU έχουν δημιουργηθεί 3 τρεις υλοποιήσεις ομοίως με CPU (“Baseline”, “Reorganized” and “gemv”) εικ. 4.8 εικ. 4.10 εικ. 4.12.

4.2.1 Baseline υλοποίηση

Algorithm 4 *“Baseline”* GPU implementation

1: Allocate memory on the GPU for u, u_{next}, N_i, σ
2: Copy u, σ, N_i from the CPU to the GPU
3: Copy N, Δt, μ, u_{th}, u_{rest} to the constant memory of the GPU
4:
5: $T \leftarrow Threads\ Per\ Block$
6: $B \leftarrow \lceil N/T \rceil$ ▷ Number of blocks in CUDA grid
7:
8: **for** $t \leftarrow 1, time\_steps$ **do** ▷ Iteration over time-steps
9: UPDATE<<<B, T>>>(u, u_{next}, σ, N_i)
10:
11: Exchange u and u_{next}
12: **end for**

Εικόνα 4-8: Baseline Υλοποίηση [8]

Στον Αλγόριθμο 4 (εικ. 4-8) παρουσιάζεται η προσέγγιση που ακολουθήθηκε για την υλοποίηση του LIF από την μονάδα GPU στην συγκεκριμένη έρευνα. Στο συγκεκριμένο δεν αποτυπώνονται πεδία όπως δηλώσεις μεταβλητών, εισαγωγή δεδομένων κ.τ.λ.π. καθώς είναι πανομοιότυπες με τους παραπάνω Αλγορίθμους της GPU.

Εδώ δημιουργήθηκε ένα μονοδιάστατο CUDA grid [11], [12] στο οποίο δημιουργείται ένας επαρκής αριθμός από blocks με νήματα, έτσι ώστε κάθε νήμα να εκτελεί τους υπολογισμούς για έναν μόνο νευρώνα. Κατά τη διάρκεια κάθε χρονικού βήματος, απαιτείται η ενημέρωση των δυναμικών των γειτονικών νευρώνων από το ακριβώς προηγούμενο χρονικό βήμα. Εδώ ακόμα και αν οι υπολογισμοί για ορισμένους νευρώνες ολοκληρώνονται ταχύτερα για το εκάστοτε χρονικό βήμα, η ανταλλαγή των τρεχουσών και νέων τιμών των δυναμικών (Αλγόριθμος 4, Γραμμή 11) δεν μπορεί να προχωρήσει πριν ολοκληρωθούν όλοι οι υπολογισμοί των νέων δυναμικών. Με άλλα λόγια, απαιτείται ένα η προσθήκη ενός φράγματος (Barrier) [25], [26] σε αυτό το σημείο για να μπορέσει να πραγματοποιηθεί η ανταλλαγή τιμών μέχρι να ολοκληρωθούν οι κατάλληλοι υπολογισμοί για όλους τους νευρώνες.

Το μοντέλο προγραμματισμού CUDA δεν παρέχει Global Barrier Synchronization για όλα τα threads του grid σε GPUs με Compute Capability < 7.0 [11], [12], [26]. Εάν απαιτούνται τέτοια σημεία Global Synchronization, συνήθως δημιουργούνται πολλαπλά υπολογιστικά νήματα στην εφαρμογή, τα οποία εκτελούνται διαδοχικά στην ίδια ροή πράξεων. Σε αυτήν την περίπτωση, ένα υπολογιστικό νήμα πρέπει να ολοκληρωθεί πριν εκτελεστεί το επόμενο, δημιουργώντας έτσι ένα σημείο Global Synchronization. Μια άλλη προσέγγιση για την αντιμετώπιση του ζητήματος είναι η δημιουργία ενός καθολικού φράγματος. Ωστόσο, αυτό επιβάλλει έναν περιορισμό στον μέγιστο αριθμό blocks που μπορούν να δημιουργηθούν σε ένα grid, επηρεάζοντας έτσι την δομή του κώδικα του υπολογιστικού πυρήνα [8]. Τέλος, αν η GPU έχει Compute Capability ≥ 7.0, μπορούν να χρησιμοποιηθούν τα Cooperative Groups που παρέχει η CUDA [11], [12]. Αυτά, ωστόσο, έχουν παρόμοιους περιορισμούς με τη λύση του καθολικού φράγματος. Οι GPU που μελετώνται έχουν υλοποιήσεις που χρησιμοποιούν την απλούστερη προσέγγιση, η

οποία είναι η αναμονή για την ολοκλήρωση της υπολογιστικής διαδικασίας μετά από κάθε χρονικό βήμα. Επομένως, η κλήση της επόμενης υπολογιστικής διαδικασίας (Αλγόριθμος 4, Γραμμή 9) βρίσκεται μέσα στον βρόχο των χρονικών βημάτων. Μόνο μετά την ολοκλήρωση της εκτέλεσης της υπολογιστικής διαδικασίας και την ενημέρωση των δυναμικών για όλους τους νευρώνες εκτελείται η επόμενη κλήση**.** Μετά την αντιγραφή δεδομένων στη γενική μνήμη στην αρχή του προγράμματος, δεν πραγματοποιούνται περαιτέρω ανταλλαγές δεδομένων μεταξύ του host συστήματος και της GPU (εκτός από το τέλος του προγράμματος, για την ανάκτηση των τελικών αποτελεσμάτων). Η ανταλλαγή των διανυσμάτων *u* και *unext* υλοποιείται ως ανταλλαγή δεικτών μνήμης, και μόνο αυτοί οι δείκτες μεταβιβάζονται στην επόμενη κλήση του υπολογιστικού πυρήνα. Επομένως, δεν υπάρχει η δυνατότητα επικάλυψης υπολογισμών με μεταφορές δεδομένων, μια δημοφιλής μέθοδος για τη βελτίωση της απόδοσης εφαρμογών που στοχεύουν σε GPUs. Επιπλέον, είναι σαφές ότι κατά την εκτέλεση του υπολογιστικού πυρήνα, τα περισσότερα δεδομένα που διαβάζονται από τη μνήμη χρησιμοποιούνται μόνο μία φορά. Συνεπώς, η κοινή μνήμη L1, L2 κ.τ.λ. της GPU δεν μπορεί να αξιοποιηθεί για τη μείωση της πίεσης στη μνήμη. Για παραμέτρους που παραμένουν σταθερές κατά τη διάρκεια της προσομοίωσης, πραγματοποιείται αντιγραφή τους στην κοινή μνήμη L1, L2 κ.τ.λ. της GPU, η οποία έχει ταχύτερο χρόνο πρόσβασης σε σύγκριση με τη μνήμη VRAM [8].

Τέλος, ο Αλγόριθμος 5 (εικ. 4-9) παρουσιάζει την ενημέρωση δυναμικού ενός νευρώνα.

Algorithm 5 Update of a single neuron on the GPU

```
1: procedure UPDATE(u, u_next, σ, N_i)
2:     i ← blockIdx.x · blockDim.x + threadIdx.x + 1
3:
4:     if i <= N then                                  ▷ GPU thread must participate
5:         u_next[i] ← u[i] + Δt · (μ − u[i])
6:         sum ← 0
7:         for j ← 1, N do                             ▷ Iteration over neighbours
8:             sum ← sum + σ[i][j] · (u[j] − u[i])
9:         end for
10:        u_next[i] ← u_next[i] + Δt · sum/N_i[i]
11:        if u_next[i] > u_th then                    ▷ Neuron fires
12:            u_next[i] ← u_rest
13:        end if
14:    end if
15: end procedure
```

Εικόνα 4-9: Αλγοριθμική απεικόνιση ενημέρωσης δυναμικού ενός νευρώνα [8]

4.2.2 Reorganized υλοποίηση

Algorithm 6 *"Reorganized"* GPU implementation

```
1: Allocate memory on the GPU for u, u_next, N_i, σ, sum_of_σ
2: Copy u, σ, N_i, sum_of_σ from the CPU to the GPU
3: Copy N, Δt, μ, u_th, u_rest to the constant memory of the GPU
4:
5: T ← Threads Per Block
6: B ← ⌈N/T⌉
7:
8: for t ← 1, time_steps do
9:     UPDATE<<<B, T>>>(u, u_next, σ, N_i, sum_of_σ)
10:
11:    Exchange u and u_next
12: end for
```

Εικόνα 4-10: Reorganized Υλοποίηση [8]

Εδώ υπάρχει η ίδια παράληψη σε πεδία όπως: δηλώσεις μεταβλητών, εισαγωγή δεδομένων κ.τ.λ.π. καθώς είναι πανομοιότυπες με τους παραπάνω Αλγορίθμους.

Πέρα από την αναδιοργάνωση των υπολογισμών για τα αθροίσματα όπως και πραγματοποιείται και στην GPU, η υλοποίηση έχει τα ίδια χαρακτηριστικά και ζητήματα με την παραπάνω ¨*Baseline*¨.

Algorithm 7 GPU kernel for *"Reorganized"*

```
1: procedure UPDATE(u, u_next, σ, N_i, sum_of_σ)
2:     i ← blockIdx.x · blockDim.x + threadIdx.x + 1
3:
4:     if i <= N then
5:         u_next[i] ← u[i] + Δt · (μ − u[i])
6:         sum ← −u[i] · sum_of_σ[i]
7:         for j ← 1, N do
8:             sum ← sum + σ[i][j] · u[j]
9:         end for
10:        u_next[i] ← u_next[i] + Δt · sum/N_i[i]
11:        if u_next[i] > u_th then
12:            u_next[i] ← u_rest
13:        end if
14:    end if
15: end procedure
```

Εικόνα 4-11: Αλγοριθμική απεικόνιση ενημέρωσης δυναμικού ενός νευρώνα για Reorganized μέθοδο [8]

Τέλος, ο Αλγόριθμος 7 (εικ. 4-11) παρουσιάζει την ενημέρωση δυναμικού ενός νευρώνα για την Reorganized υλοποίση.

4.2.3 gemv υλοποίηση

Algorithm 8 *"gemv"* GPU implementation

1: **for** $t \leftarrow 1, time\_steps$ **do**
2: cublasDgemv(OP_T, N, N, 1, σ, N, u, 1, 0, u_{next}, 1)
3:
4: UPDATE$<<<B,T>>>$(u, u_{next}, σ, N_i, sum_of_σ)
5:
6: Exchange u and u_{next}
7: **end for**

Εικόνα 4-12: Αλγοριθμική απεικόνιση ενημέρωσης δυναμικού ενός νευρώνα για gemv μέθοδο [8]

Εδώ υπάρχει η ίδια παράληψη σε πεδία όπως: δηλώσεις μεταβλητών, εισαγωγή δεδομένων κ.τ.λ.π. καθώς είναι πανομοιότυπες με τους παραπάνω Αλγορίθμους.

Η διαφορά εδώ σε σχέση με το CPU είναι πως η συνάρτηση *cblas_dgemv()* [27] από τη βιβλιοθήκη *Intel MKL [27]* έχει αντικατασταθεί με τη συνάρτηση *cublasDgemv() [11], [12]* από τη βιβλιοθήκη *cuBLAS* για την εκτέλεση της πράξης πίνακα-διανύσματος στη GPU. Τα αποτέλεσμα της *cublasDgemv()* αποθηκεύονται στο *unext* τα οποία στη συνέχεια μεταβιβάζονται σε μια υπολογιστική διαδικασία για την τελική ενημέρωση, και πάλι στη GPU.

Στην περίπτωση της GPU, η προσαρμογή φορτίου έγινε με τέτοιο τρόπο ώστε να εκμεταλλεύεται τον μαζικό παραλληλισμό της αρχιτεκτονικής. Συγκεκριμένα, η αντιστοίχιση των γραμμών του αραιού μητρώου σε ανεξάρτητα threads αποτέλεσε συνειδητή επιλογή, με στόχο τη μεγιστοποίηση της ταυτόχρονης εκτέλεσης.

Κατά την υλοποίηση, δόθηκε ιδιαίτερη προσοχή στη διάταξη των δεδομένων στη μνήμη, ώστε να επιτευχθεί όσο το δυνατόν καλύτερο memory coalescing. Παράλληλα, εξετάστηκαν οι επιπτώσεις της ανομοιομορφίας στον αριθμό μη μηδενικών στοιχείων ανά γραμμή, καθώς αυτή μπορεί να οδηγήσει σε ανισορροπία φορτίου και divergence μεταξύ των threads.

4.3 Υλοποίηση PIM

Η υλοποίηση του αλγορίθμου SPMV σε αρχιτεκτονική Process-In-Memory διαφοροποιείται ουσιαστικά από τις κλασικές αρχιτεκτονικές CPU και GPU, καθώς ο υπολογισμός μεταφέρεται κοντά στη μνήμη. Η προσέγγιση αυτή στοχεύει στη μείωση του κόστους μεταφοράς δεδομένων, το οποίο αποτελεί βασικό περιοριστικό παράγοντα στις πράξεις SPMV.

Η προσέγγιση αυτή στοχεύει στην, εξάλειψη του κόστους μεταφοράς τεράστιων όγκων δεδομένων μέσω των διαύλων PCIe ή της μνήμης συστήματος. Ωστόσο, η υλοποίηση αποκάλυψε σημαντικές προκλήσεις:

- **Περιορισμένο Σύνολο Εντολών (ISA):** Οι τρέχουσες υλοποιήσεις PIM (όπως της UPMEM) διαθέτουν ένα απλοποιημένο σύνολο εντολών. Η έλλειψη hardware υποστήριξης για πράξεις κινητής υποδιαστολής (Floating-Point) σημαίνει ότι οι υπολογισμοί του μοντέλου LIF πρέπει να υλοποιηθούν μέσω προσομοίωσης, γεγονός που αυξάνει τον αριθμό των κύκλων ρολογιού ανά πράξη.
- **MRAM Latency vs Performance:** Παρόλο που η απόσταση επεξεργαστή-δεδομένων εκμηδενίζεται, η πρόσβαση στην τοπική μνήμη MRAM της DPU παραμένει πιο αργή σε σχέση με την cache της CPU, απαιτώντας βελτιστοποιημένη χρήση της WRAM (Working RAM) ως buffer.
- **Inter-DPU Communication:** Η αρχιτεκτονική PIM υπερέχει όταν οι υπολογισμοί και οι ενέργειες πραγματοποιούνται μόνο στης μονάδες DPU. Κάθε ανάγκη για επικοινωνία μεταξύ διαφορετικών DPU ή επιστροφή δεδομένων στον Host εισάγει σημαντικές καθυστερήσεις, καθιστώντας τον προσεκτικό διαχωρισμό των δεδομένων (1D Partitioning) επιτακτική ανάγκη.

Αναφορικά με τη διαχείριση της μνήμης στο σύστημα, απαιτείται ιδιαίτερη προσοχή λόγω των περιορισμών που επιβάλλει η αρχιτεκτονική του PIM. Σε αντίθεση με τους παραδοσιακούς επεξεργαστές, η μεταφορά δεδομένων από την MRAM (Main RAM του DPU) προς την WRAM (Working RAM) πραγματοποιείται ρητά από τον προγραμματιστή, μέσω ειδικών εντολών όπως οι *mram_read* και *mram_write*.

Στο πλαίσιο της παρούσας εργασίας, δίνεται ιδιαίτερη έμφαση στο μέγεθος και στην ευθυγράμμιση των μεταφερόμενων τμημάτων δεδομένων στην μνήμη. Δεδομένου ότι η διαθέσιμη WRAM είναι περιορισμένη (64KB), τα στοιχεία του αραιού μητρώου φορτώνονται τμηματικά στην εκάστοτε μνήμη. Τα δεδομένα οργανώνονται σε τμήματα μεγέθους πολλαπλάσιου των 8 bytes, ώστε να επιτυγχάνεται η μέγιστη δυνατή ευθυγράμμιση και να βελτιστοποιείται η ταχύτητα μεταφοράς.

Σχετικά με τα νήματα εκτέλεσης (tasklets) των DPU, για την αποδοτική αξιοποίηση των υπολογιστικών πόρων κάθε DPU, η κατανομή των δεδομένων πραγματοποιήθηκε με τρόπο ώστε να ελαχιστοποιείται ο ανταγωνισμός για κοινόχρηστους πόρους. Συγκεκριμένα, υλοποιήθηκε ένας στατικός διαμερισμός των δεδομένων, όπου κάθε tasklet αναλαμβάνει την επεξεργασία ενός συγκεκριμένου εύρους μη μηδενικών στοιχείων (NNZ) του αραιού μητρώου. Η συγκεκριμένη προσέγγιση επιλέχθηκε προκειμένου να αποφευχθεί η χρήση μηχανισμών συγχρονισμού, όπως mutexes, τα οποία θα εισήγαγαν σημαντικό υπολογιστικό κόστος στην εκτέλεση του μοντέλου LIF. Το κόστος αυτό είναι ιδιαίτερα κρίσιμο στην αρχιτεκτονική PIM, καθώς οι επεξεργαστές DPU δεν διαθέτουν εκτεταμένη υποστήριξη σε επίπεδο υλικού για σύνθετους μηχανισμούς συγχρονισμού. [2], [7]

Οι παραπάνω διαδικασίες υλοποιούνται με την χρήση συναρτήσεων της βιβλιοθήκης SparseP, μέσω της κατάλληλης παραμετροποίησης, επιτρέποντας τον αποτελεσματικό έλεγχο της μνήμης και τη βελτίωση της συνολικής απόδοσης στα DPU.

Παρότι η αρχιτεκτονική PIM δεν πέτυχε τις επιδόσεις της GPU στις συγκεκριμένες δοκιμές, προσφέρει σημαντικά ερευνητικά συμπεράσματα σχετικά με τις δυνατότητες και τα όρια της υπολογιστικής κοντά στη μνήμη, ειδικά για εφαρμογές με έντονη εξάρτηση από την πρόσβαση στη μνήμη.

Algorithm 9 DPU-Accelerated Leaky Integrate-and-Fire (LIF) Simulation

Inputs:
  n: Number of neurons
  s: Connectivity matrix (CSR format)
  dt: Time step size
  μ: Exponential integration factor
  uth: Neuron firing threshold
  $total\_time\_steps$: Total simulation time steps
  $ttransient$: Transient time steps
  $sampling$: Sampling rate
  nr_of_dpus: Number of DPUs
Initialization:
  $u[n] \leftarrow$ Initialize neuron potentials with random values in $[0, uth)$
  $u\_next[n], omega[n], omega1[n], sum\_s[n], divide[n] \leftarrow 0$
  $part\_info \leftarrow$ Initialize partition data for DPUs
  $dpu\_info \leftarrow$ Initialize DPU information
  $input\_args \leftarrow$ Initialize input arguments for DPUs
  $max\_rows\_per\_dpu, max\_nnz\_ind\_per\_dpu, max\_nnz\_val\_per\_dpu, max\_rows\_per\_tasklet \leftarrow 0$
  $y \leftarrow$ Allocate memory for output vector
  Calculate $sum\_s[i]$ and $divide[i]$ for each neuron i
  Partition Data:
for $k \leftarrow 0$ to $nr\_of\_dpus - 1$ **do**
  $rows\_per\_dpu \leftarrow$ Calculate rows per DPU
  $prev\_rows\_dpu \leftarrow$ Calculate previous rows per DPU
  $rows\_per\_dpu\_pad \leftarrow$ Calculate padded rows per DPU
  $nnz \leftarrow$ Calculate number of non-zero elements
  $nnz\_ind\_pad \leftarrow$ Calculate padded non-zero indices
  $nnz\_val\_pad \leftarrow$ Calculate padded non-zero values
  Update $max\_rows\_per\_dpu$, $max\_nnz\_ind\_per\_dpu$, $max\_nnz\_val\_per\_dpu$
  $dpu\_info[k] \leftarrow$ Store DPU information
  $input\_args[k].nrows \leftarrow rows\_per\_dpu$
  $input\_args[k].tcols \leftarrow s.ncols$
  Partition Tasklets:
  for $t \leftarrow 0$ to $NR\_TASKELTS - 1$ **do**
    $input\_args[k].start\_row[t] \leftarrow$ Calculate start row for tasklet
    $input\_args[k].rows\_per\_tasklet[t] \leftarrow$ Calculate rows per tasklet
    Update $max\_rows\_per\_tasklet$
  end for
end for
Re-allocate Memory with Padding:
Re-allocate $s.rowptr$, $s.colind$, $s.values$ with padding
Copy Input Arguments to DPUs:
for $k \leftarrow 0$ to $nr\_of\_dpus - 1$ **do**
  $input\_args[k].max\_rows \leftarrow max\_rows\_per\_dpu$
  $input\_args[k].max\_nnz\_ind \leftarrow max\_nnz\_ind\_per\_dpu$
  Copy $input\_args[k]$ to DPU k
end for
Copy Input Matrix to DPUs:
Copy $s.rowptr$, $s.colind$, $s.values$ to DPUs
Simulation Loop:
for $it \leftarrow 0$ to $total\_time\_steps - 1$ **do**
  Copy u to DPUs
  Launch kernel on DPUs
  Copy y from DPUs
  Update $u\_next[i]$ for each neuron i
  Update $omega1[i]$ if $u\_next[i] > uth$
  Exchange u and u_next
  if $(it + 1) \mod sampling = 0$ **then**
    Accumulate average times
    Reset average times
    if $it > ttransient$ **then**
      Calculate $omega[i]$
    end if
  end if
end for

Εικόνα 4-13: Υλοποίηση του μοντέλου LIF σε PIM.

Τελειώνοντας με την περιγραφή των υλοποιήσεων σε CPU και GPU ήρθε η ώρα να προχωρήσει αυτή η εργασία στο βασικό της αντικείμενο που είναι η υλοποίηση που αφορά το σύστημα PIM (Process in Memory).

Για την υλοποίηση στο PIM σύστημα υπάρχει ένας αλγόριθμος με 2 διαφορετικές υλοποιήσεις ως προς την κατανομή φορτίου στους πύρινες του PIM συστήματος, όπως αναφέρεται και σε παραπάνω κεφάλαιο. Αυτές διαφοροποιούνται ως προς τον τρόπο κατανομής του υπολογιστικού φορτίου. Συγκεκριμένα, η κατανομή μπορεί να βασίζεται είτε στον αριθμό των μη μηδενικών στοιχείων του αραιού μητρώου είτε στον αριθμό των γραμμών του. Πιο συγκεκριμένα εξισορροπείται το φορτίο βάση μη μηδενικών στοιχεία, έτσι ώστε κάθε νήμα να επεξεργάζεται σχεδόν τον ίδιο αριθμό μη μηδενικών στοιχείων ή βάση γραμμών έτσι ώστε κάθε νήμα να επεξεργάζεται σχεδόν τον ίδιο αριθμό γραμμών.

Όπως φαίνεται στον παραπάνω αλγόριθμο (εικ. 4-13), ξεκινάει με τις αναγκαίες μεταβλητές που θα πρέπει να δοθούν, με σκοπό υλοποιηθεί η διαδικασία προσομοίωσης των νευρώνων (Γραμμή 1-10). Ύστερα δίνονται οι αρχικές τιμές στις εκάστοτε μεταβλητές (Γραμμή 11-19). Παρακάτω πραγματοποιείται ο ορισμός κατανομής φορτίου – δεδομένων του αραιού μητρώου και των Tasklet στα DPU(Γραμμή 21-31). Αφού ολοκληρωθούν αυτά πραγματοποιείται μια αναπροσαρμογή των στοιχείων, που αφορούν την κατανομή χρησιμοποιώντας τεχνικές padding [7], [23] με σκοπό την κατανομή ισόποσου φορτίου στους πυρήνες DPU, για την μέγιστη απόδοση του συστήματος (Γραμμή 32-41). Ύστερα ο αλγόριθμος και αφού ολοκληρωθούν οι κατάλληλες ενέργειες ορισμού, περνάει όλα τα δεδομένα για το μητρώο σχέσεων *σ* και διανύσματος *u* από το host σύστημα (CPU) στους πυρήνες DPU(Γραμμή 41-48). Ύστερα ξεκινάει η πραγματοποίηση των πράξεων στα DPU ανά βήμα προσομοίωσης (*time_step*) και η έναρξη της προσομοίωσης (Γραμμή 49-τέλος).

Κάπου εδώ θα πρέπει να αναφερθεί ότι η συγκεκριμένη προσομοίωση χρησιμοποιεί μια και μόνο σχέση νευρώνων, δηλαδή ένα αραιό μητρώο το οποίο δεν αλλάζει ανά χρονικό βήμα προσομοίωσης, σε αντίθεση με το διάνυσμα που αφορά τις διεγέρσεις. Αυτό σημαίνει πως στην περίπτωση που θα χρειαζόταν να αλλάζει και το μητρώο κατά την προσομοίωση, όλη αυτή η διαδικασία προσαρμογής-μεταφοράς-κατανομής θα έπρεπε να επαναλαμβάνεται ανά βήμα. Κάτι πολύ αρνητικό για την απόδοση του SPMV, διότι πρακτικά θα πρόκυπτε ένα άλλου είδους πρόβλημα, όμοιο με αυτό της μνήμης για τις αρχιτεκτονικές των CPU. Στην συγκεκριμένη περίπτωση πραγματοποιείται μεταφορά των καινούριων τιμών του διανύσματος μετά από κάθε βήμα κάτι που επίσης ενδέχεται να δημιουργεί καθυστερήσεις στην υλοποίηση του SPMV.

Οι περισσότερες από τις παραπάνω ενέργειες που αφορούν τα DPU γίνονται με την βοήθεια την βιβλιοθήκης SparseP [8] όπως αναφέρεται παραπάνω στην εργασία.

Αξίζει να σημειωθεί ότι, παρότι οι ψευδοκώδικες παρουσιάστηκαν σε υψηλό επίπεδο, η πραγματική υλοποίηση περιλαμβάνει επιπλέον λεπτομέρειες που σχετίζονται με τη διαχείριση μνήμης, τον συγχρονισμό και τον χειρισμό οριακών περιπτώσεων, οι οποίες δεν αποτυπώνονται πλήρως για λόγους απλότητας.

4.4 Σύγκριση υλοποιήσεων και επιπτώσεις σχεδίασης

Η σύγκριση των υλοποιήσεων σε CPU, GPU και PIM αναδεικνύει τη σημαντική επίδραση της αρχιτεκτονικής σχεδίασης στην απόδοση του αλγορίθμου SPMV. Ενώ η CPU προσφέρει απλότητα και ευκολία υλοποίησης, αδυνατεί να κλιμακώσει αποτελεσματικά για μεγάλα προβλήματα λόγω περιορισμένου παραλληλισμού και μνήμης.

Αντίθετα, η GPU εκμεταλλεύεται τον μαζικό παραλληλισμό και το υψηλό εύρος ζώνης μνήμης, επιτυγχάνοντας σημαντικά καλύτερη απόδοση. Ωστόσο, αυτή εξαρτάται έντονα από τη δομή του αραιού μητρώου, γεγονός που καθιστά το SPMV μια από τις πιο απαιτητικές πράξεις για βελτιστοποίηση.

Η αρχιτεκτονική PIM προσφέρει μια εναλλακτική προσέγγιση, μεταφέροντας τον υπολογισμό κοντά στα δεδομένα. Παρότι οι επιδόσεις δεν ξεπέρασαν αυτές της GPU στο παρόν πειραματικό πλαίσιο, τα αποτελέσματα υποδεικνύουν ότι η ίσως στο μέλλον η προσέγγιση αυτή μπορεί να αποδειχθεί ιδιαίτερα αποδοτική σε σενάρια όπου το κόστος μεταφοράς δεδομένων κυριαρχεί.

Χαρακτηριστικό	**CPU (Βασική Υλοποίηση)**	**GPU (Υψηλής Ροής Δεδομένων)**	**PIM (Υπολογισμός Κοντά στη Μνήμη)**
Μοντέλο Εκτέλεσης	Γενικής χρήσης, περιορισμένος παραλληλισμός	Μαζικός παραλληλισμός	Κατανεμημένος υπολογισμός σε μονάδες μνήμης
Κύριο Πλεονέκτημα	Ευελιξία και χαμηλή καθυστέρηση	*Μεγάλο εύρος ζώνης μνήμης,* παραλληλισμός	Ελαχιστοποίηση μεταφοράς δεδομένων
Κύριο Σημείο Συμφόρησης	Περιορισμένο εύρος ζώνης μνήμης	Απόκλιση νημάτων και kernels περιορισμένα από μνήμη	Περιορισμοί συνόλου εντολών και υπολογιστικής υποστήριξης
Διαχείριση Μνήμης	Ιεραρχία cache (L1–L3)	Υψηλό εύρος ζώνης VRAM και συνεκτική πρόσβαση	Τοπική μνήμη DPU (υπολογισμός κοντά στα δεδομένα)
Συμπεριφορά SPMV	Υψηλό ποσοστό χρήσης cache και ανάγνωσής από μνήμη RAM	Εξαρτώμενη από την αραιότητα	Αποδοτική πρόσβαση στη μνήμη με περιορισμένη υπολογιστική ισχύ
Δυνατότητα Παραλληλίας	Περιορισμένη	Υψηλή	Εξαρτάται από το πλήθος των DPU
Καταλληλότητα για μοντέλο LIF	Βασική υλοποίηση, έλεγχος ορθότητας	Υψηλής απόδοσης προσομοιώσεις	Ερευνητική προσέγγιση

5 Αποτελέσματα μετρήσεων

5.1 Ανάλυση αποτελεσμάτων

Το συγκεκριμένο κεφάλαιο εστιάζει στην ανάλυση αποτελεσμάτων για τα συστήματα PIM και GPU.

Αναφορικά με το σύστημα PIM θα αναλυθούν τα αποτελέσματα σχετικά με τον χρόνο κατανομής δεδομένων στις μονάδες DPU, συγκεκριμένα ως προς στον μέσο χρόνο ανά βήμα προσομοίωσης και τον τελικό χρόνο κατανομής δεδομένων στα DPU, καθώς επίσης τον χρόνο εκτέλεσης της προσομοίωσης για όλα τα βήματα για τα παρακάτω αραιά μητρώα (εικ.5-1). Για το σύστημα GPU η ανάλυση θα γίνει ως προς τον τελικό χρόνο εκτέλεσης.

Η μελέτη πραγματοποιείται για σύνολο δώδεκα αραιά μητρώα, καθώς επίσης για χρόνους προσομοίωσης των 500 βημάτων (Simulation time 0.05s), που σημαίνει 500 πράξεις SPMV ανά προσομοίωση για κάθε αραιό μητρώο, όπως φαίνεται παρακάτω:

Matrices	Rows	Columns	NNZ (Non-Zero)
epb1	14734	14734	95053
HEP-th	27240	27240	342437
pkustk02	10800	10800	410400
tube1	21498	21498	459277
adder_dcop_01	1813	1813	11156
bayer06	3008	3008	27576
bcsstk10	1086	1086	11578
ex7	1633	1633	54543
G31	2000	2000	19990
gemat11	4929	4929	33185
rdb5000	5000	5000	29600
t2dal_a	4257	4257	20861

Εικόνα 5-1: Αραιά Μητρώα Μελέτης

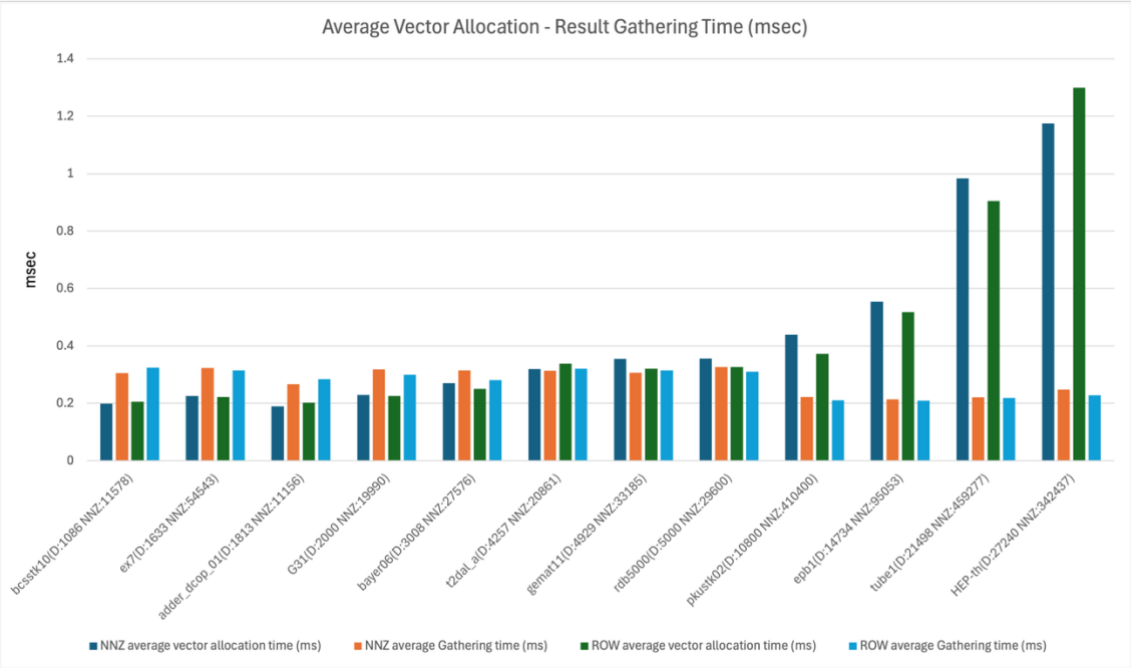
```
[INFO] Allocated 2048 DPU(s)
[INFO] Allocated 16 TASKLET(s) per DPU
[INFO] ../matrices/small/adder_dcop_01.mtx: 1813 Rows,
1813 Cols, 11156 NNZs
Running simulation with following parameters:
  Number of neurons   : 1813
  Connectivity matrix :
../matrices/small/adder_dcop_01.mtx
  Simulation time     : 0.050000 seconds (500 time steps)
  Transient time      : 0 seconds (250 time steps)
  Sampling rate       : 100 time steps
  dt                  : 1.0e-04 seconds
  mu                  : 1.000000000000000
  uth                 : 0.980000000000000
  s_min               : 0.700000000000000
  s_max               : 0.700000000000000
```

Εικόνα 5-2: Πληροφορίες προσομοίωσης

5.2 Ανάλυση αποτελεσμάτων PIM

5.2.1 Μέσοι Χρόνοι κατανομής και συλλογής φορτίων διανυσμάτων ανά βήμα.

Στο παρακάτω διάγραμμα (Average Vector Allocation Time – Average Result Gather Time Per Iteration (ms) (εικ.5-3), απεικονίζεται ο μέσος χρόνος που απαιτείται για την κατανομή των διανυσμάτων στα DPU του συστήματος PIM (Average Allocation time), καθώς και ο μέσος χρόνος συλλογής των διανυσμάτων (Average Gathering time) αποτελεσμάτων ανά επανάληψη σε κλίματα χρόνου χιλιοστών του δευτερολέπτου (ms), για εύρος 0-1.4. Από την στιγμή που στην πράξη SPMV, το μέγεθος του αραιού μητρώου καθορίζει και το μέγεθος του διανύσματος, η μελέτη πραγματοποιείται εδώ ανά αραιό μητρώο εισόδου στην προσομοίωση.



Εικόνα 5-3: Average Vector Allocation – Result Gather Time Per Iteration (ms)

Παρατηρήσεις για μέσους χρόνους κατανομής φορτίων ανεξάρτητα τεχνικών (κατανομή ανά γραμμές ROW, ανά μη μηδενικά στοιχεία NNZ):

Για αραιά μητρώα με μικρό έως μεσαίο πλήθος στοιχείων, της τάξεως 1.000x1.000 – 5.000x5.000, ο χρόνος κατανομής και συλλογής διατηρείται σε χαμηλά και σχετικά σταθερά επίπεδα 0.3ms – 0.4ms ανά βήμα προσομοίωσης. Σχετικά με μεγαλύτερα αραιά μητρώα με πλήθος στοιχείων μεγαλύτερο από 10.000x10.000, υπάρχει μια απότομη αύξηση στον μέσο χρόνο κατανομής των στοιχείων στα DPU. Αυτό υποδηλώνει ότι το μέγεθος του αραιού μητρώου, επηρεάζει αρκετά την απόδοση του συστήματος, πόσο μάλλον στην συγκεκριμένη προσομοίωση που πραγματοποιείται η ανακατανομή διανύσματος σε κάθε βήμα. Παρά το γεγονός ότι αυτό ήταν ένα αναμενόμενο αποτέλεσμα, αυτό το οποίο έχει ιδιαίτερο ενδιαφέρον, είναι το κατά πόσο αυτός ο χρόνος αυξάνεται απότομα από ένα μέγεθος και μετά και πόσο σημαντική γίνεται η απόκλιση.

Πράγμα που δηλώνει ότι η συγκεκριμένη αρχιτεκτονική για μεγάλη σε ποσότητα φορτίο είναι πολύ πιθανό να υπάρξουν σημαντικές καθυστερήσεις λόγω του προβλήματος της μνήμης όμοιες με τις αρχιτεκτονικές CPU.

Αναφορικά με τα μη μηδενικά στοιχεία των αραιών μητρώων και το πλήθος αυτών παρατηρείται πως δεν επηρεάζεται ο χρόνος της κατανομής. Για παράδειγμα τα μητρώα adder_cop, G31 και bayer06 κυμαίνονται σε κοντινούς χρόνους παρά το γεγονός της μεγάλης απόκλισης σε αριθμό μη μηδενικών στοιχείων. Ομοίως και tube1 και HEP-th παρατηρείται ότι η διαφορά που έχουν στους χρόνους αφορά το μέγεθος τους παρά τα μη μηδενικά στοιχεία. Για παράδειγμα ο HEP-th είναι ένα αραιό μητρώο με περισσότερα στοιχειά από το tube1 αλλά με μικρότερο σε αριθμό μη μηδενικών στοιχειών.

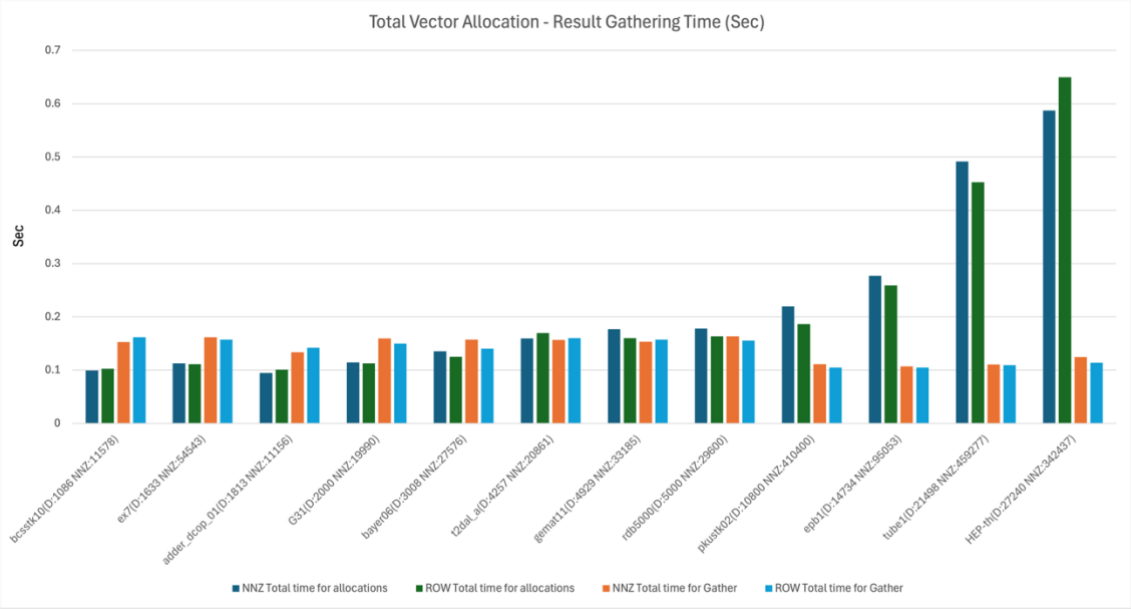
Παρατηρήσεις για μέσους χρόνους κατανομής φορτίων αναφορικά με την τεχνική κατανομής (κατανομή ανά γραμμές ROW, ανά μη μηδενικά στοιχεία NNZ):

Αναφορικά με τις διαφορές χρόνου μεταξύ των τεχνικών κατανομής, παρατηρείται ότι όσο αυξάνεται το μέγεθος των αραιών μητρώων, το ποσοστό των μη μηδενικών στοιχείων επηρεάζει σημαντικά το κόστος εκτέλεσης. Συγκεκριμένα, για μητρώα όπου ο αριθμός των μη μηδενικών στοιχείων είναι σχετικά μικρός σε σχέση με το συνολικό τους μέγεθος, η τεχνική κατανομής βάση μη μηδενικών στοιχείων (NNZ) αποδεικνύεται καταλληλότερη σε σχέση με την τεχνική κατανομής ανά γραμμές (ROW). Το αντίστροφο ισχύει για αραιά μητρώα με μεγαλύτερο αριθμό μη μηδενικών στοιχείων σε σχέση με το μέγεθος. Για παράδειγμα για *tube1* και *HEP-th* που υπάρχει σημαντική διαφορά σε ποσοστό μη μηδενικών στοιχείων φαίνεται την διαφορά χρόνων μεταξύ NNZ και ROW για τα δύο μητρώα έχει ακριβώς την αντίθετη συμπεριφορά.

Παρατηρήσεις για μέσους χρόνους συλλογής αποτελεσμάτων ανά βήμα (Average Result Gather Time per iteration):

Οι χρόνοι συλλογής αποτελεσμάτων διατηρούνται γενικά χαμηλότεροι και πιο σταθεροί για όλα τα μεγέθη αραιών μητρώων. Πράγμα που δηλώνει πως το κόστος για το στάδιο της συλλογής αποτελεσμάτων δεν επηρεάζεται από το μέγεθος τους μητρώου ούτε από των αριθμό των μη μηδενικών στοιχείων. Μια τελευταία παρατήρηση είναι: πως οι χρόνοι συλλογής αποτελεσμάτων είναι μεγαλύτεροι από τους χρόνους κατανομής φορτίων για τα μικρότερα αραιά μητρώα μεγέθους 1.000x1.000 – 3.000x3.000.

5.2.2 Τελικοί Χρόνοι κατανομής και συλλογής φορτίων διανυσμάτων ανά βήμα.



Εικόνα 5-4: Total Vector Allocation – Result Gather Time (sec)

Εδώ πλέον είναι τα αποτελέσματα των χρόνων κατανομής ύστερα από το τέλος της προσομοίωσης των νευρώνων. Όπως ειπώθηκε και παραπάνω, αυτή η συνεχής κατανομή δεδομένων διανύσματος στα DPU προσθέτει ένα κόστος χρόνου της τάξεως των δευτερολέπτων (sec) όπως φαίνεται και στο διάγραμμα (εικ. 5-4), με εύρος από 0.1s – 0.7s. Ένα σημαντικό κόστος για μεγάλα αραιά μητρώα και βάση και των επιδόσεων συνολικής εκτέλεσης της GPU που θα παρουσιαστούν και παρακάτω.

Παρατηρήσεις για τελικούς χρόνους κατανομής φορτίων ανεξάρτητα τεχνικών (κατανομή ανά γραμμές ROW, ανά μη μηδενικά στοιχεία NNZ):

Οι παρατηρήσεις είναι όμοιες με παραπάνω με την διαφορά ότι εδώ απεικονίζονται οι χρόνο ύστερα από την ολοκλήρωση της προσομοίωσης

Παρατηρήσεις για μέσους χρόνους κατανομής φορτίων αναφορικά με την τεχνική κατανομής (κατανομή ανά γραμμές ROW, ανά μη μηδενικά στοιχεία NNZ):

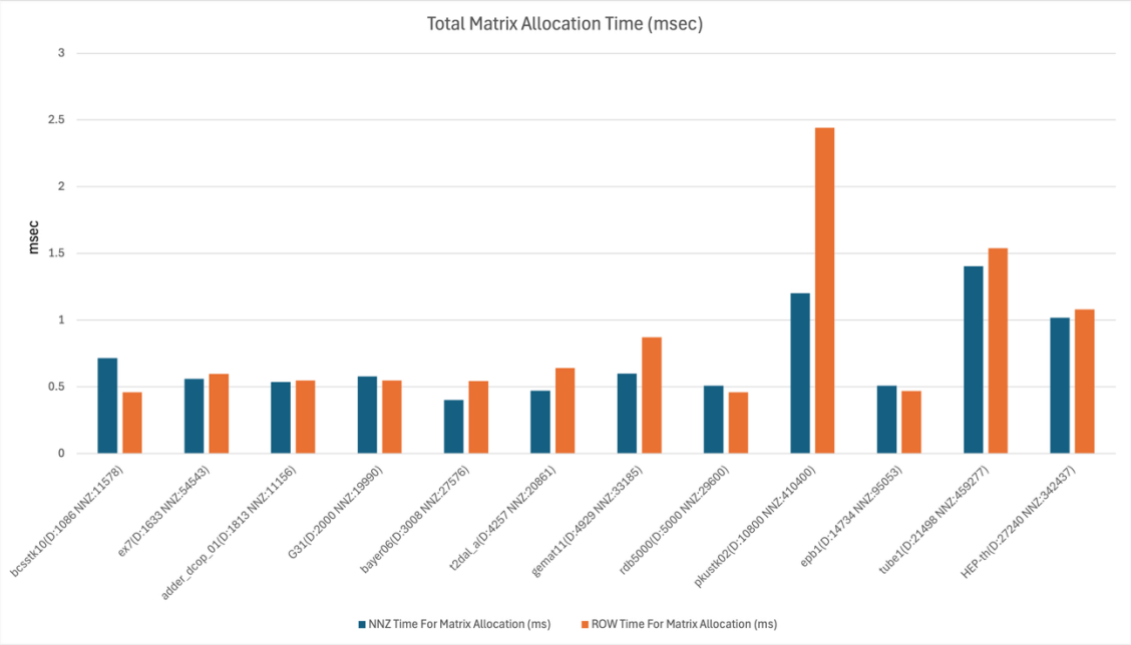
Οι παρατηρήσεις είναι όμοιες με παραπάνω με την διαφορά ότι εδώ απεικονίζονται οι χρόνο ύστερα από την ολοκλήρωση της προσομοίωσης.

Παρατηρήσεις για τελικούς χρόνους συλλογής αποτελεσμάτων ανά βήμα (Average Result Gather Time per iteration):

Οι παρατηρήσεις είναι όμοιες με παραπάνω με την διαφορά ότι εδώ απεικονίζονται οι χρόνο ύστερα από την ολοκλήρωση της προσομοίωσης.

5.2.3 Τελικοί χρόνοι κατανομής αραιών μητρώων.

Στο παρακάτω διάγραμμα *Total Matrix Allocation Time (ms)* (εικ.5-5), εμφανίζεται ο συνολικός χρόνος κατανομής του αραιού μητρώο στις μονάδες των DPU για εύρος τιμών χρόνου από 0ms - 3ms. . Εδώ δεν παρουσιάζεται ο μέσος χρόνος κατανομής, διότι στην συγκεκριμένη εφαρμογή το αραιό μητρώο κατανέμεται μόνο μια φορά στις μονάδες και χρησιμοποιείται σε όλα τα βήματα της προσομοίωσης. Δεν απαιτείτε δηλαδή ενημέρωση στοιχείων του αραιού μητρώου.



Εικόνα 5-5: Total Matrix Allocation Time (msec)

Παρατηρήσεις για τελικούς χρόνους κατανομής φορτίων ανεξάρτητα τεχνικών (κατανομή ανά γραμμές ROW, ανά μη μηδενικά στοιχεία NNZ):

Για τα μικρά σε μέγεθος μητρώα ο χρόνος κατανομής του αραιού μητρώου είναι σχεδόν ο ίδιος και κυμαίνεται σε εύρος 0.5ms – 1ms. Όπως συμβαίνει και παραπάνω στα διανύσματα, όσο αυξάνεται το μέγεθος του μητρώου τόσο αυξάνεται και ο χρόνος της κατανομής στα DPU.

Βέβαια εδώ υπάρχουν κάποιες εξαιρέσεις για ορισμένα μητρώα. Συγκεκριμένα για το αραιό μητρώο *epb1* παρά των περισσότερων στοιχείων σε ποσότητα που διαθέτει, ο

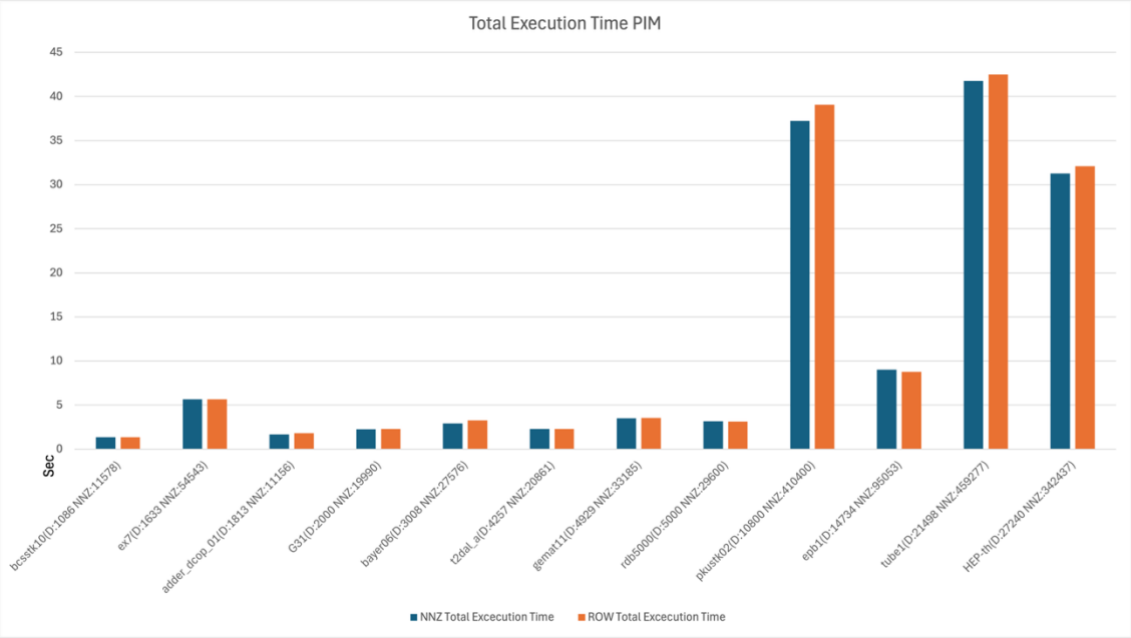
χρόνος κατανομής στο σύστημα είναι αισθητά μικρότερος έναντι των άλλων μεγάλων και μικρότερων αραιών μητρώων. Εδώ αυτό οφείλεται στην δομή του αραιού μητρώου. Συγκεκριμένα το αραιό μητρώο *epb1* έχει λίγα μη μηδενικά στοιχεία συγκριτικά με το μέγεθος του και την σχέση μεγέθους και μη μηδενικών στοιχείων που έχουν τα υπόλοιπα αραιά μητρώα. Από εδώ βγαίνει το πόρισμα του πως ο χρόνος κατανομής αραιών μητρώων στο σύστημα, εξαρτάται σε ένα μεγάλο βαθμό από τον αριθμό των μη μηδενικών στοιχείων των αραιών μητρώων. Επίσης σε περίπτωση που θα χρειαστεί να ενημερώνονται συνέχεια οι τιμές του μητρώου, τότε θα υπήρχε ένα ακόμα μεγαλύτερο χρονικό κόστος κατά στην προσομοίωση, όπως και στα διανύσματα της τάξεως 0.5ms*500=0.250s – 2.5ms*500=1.25s.

Παρατηρήσεις για μέσους χρόνους κατανομής φορτίων αναφορικά με την τεχνική κατανομής (κατανομή ανά γραμμές ROW, ανά μη μηδενικά στοιχεία NNZ):

Η αρχιτεκτονική παρουσιάζει γενικά μια σχετική σταθερότητα στο χρονικό κόστος μεταξύ των τεχνικών NNZ και ROW, με τα περισσότερα μητρώα να εμφανίζουν ένα ελαφρά μεγαλύτερο κόστος χρόνου για στην τεχνική κατανομής ανά γραμμές. Ωστόσο, παρατηρούνται ορισμένες εξαιρέσεις. Συγκεκριμένα, για το μητρώο *pkustk02*, η διαφορά στο κόστος χρόνου μεταξύ των τεχνικών κατανομής ίσου αριθμού μη μηδενικών στοιχείων και κατανομής ανά γραμμές στα DPU είναι ιδιαίτερα αυξημένη, με την τεχνική ανά γραμμές να εμφανίζει επιπλέον καθυστέρηση περίπου 1,5 ms, διαφορά που δεν παρατηρείται σε κανένα άλλο μητρώο. Η απόκλιση αυτή ενδέχεται να οφείλεται σε παράγοντες της αρχιτεκτονικής, όπως πιθανούς κατακερματισμούς μνήμης (memory fragmentation) που προέκυψαν κατά την εκτέλεση. Επιπλέον, μια μικρότερη διαφορά παρατηρείται και στο μητρώο *bcsstk10*, όπου ο χρόνος εκτέλεσης είναι αυξημένος κατά περίπου 0,5 ms για την τεχνική κατανομής ίσης ποσότητας μη μηδενικών στοιχείων ανά DPU.

5.2.4 Τελικοί εκτέλεσης μοντέλου LIF PIM.

Αφού αναλύθηκαν οι χρόνοι κόστους για τις μεταφορές – κατανομές δεδομένων στο σύστημα PIM. Κάτι το οποίο όπως αναφέρεται και παραπάνω είναι κάτι ιδιαίτερα σημαντικό και αφορά άμεσα στον σχεδιασμό λειτουργείας της αρχιτεκτονικής. Η ανάλυση προχωράει στο τελικό στάδιο για την αρχιτεκτονική PIM, που έχει να κάνει με την μελέτη του συνολικού χρόνου εκτέλεσης της προσομοίωσης που πραγματοποιεί το μοντέλο LIF για τα βήματα επαναλήψεων και πράξεων SPMV.



Εικόνα 5-6: Total Execution Time PIM (sec)

Παρατηρήσεις για τελικούς χρόνους εκτέλεσης προσομοίωσης LIF ανεξάρτητα τεχνικών (κατανομή ανά γραμμές ROW, ανά μη μηδενικά στοιχεία NNZ):

Το PIM σύστημα βγάζει έναν χρόνο εκτέλεσης οποίος κυμαίνεται από εύρος 1s – 45s, όπως φαίνεται και στο παραπάνω διάγραμμα (εικ. 5-6). Παρατηρείται πως ο χρόνος εκτέλεσης της προσομοίωσης, δεν έχει να κάνει καθαρά και μόνο με το μέγεθος των μητρώων. Αλλά περισσότερο με την ποσότητα των μη μηδενικών στοιχείων του εκάστοτε μητρώου. Συγκεκριμένα το μητρώο *ex7,* παρά το γεγονός του ότι είναι ένα σχετικά μικρό σε μέγεθος αραιό μητρώο 1633x1633, έχει μεγαλύτερο χρόνο εκτέλεσης που ξεπερνάει τα 5s από τα υπόλοιπα μικρά – μεσαία αραιά μητρώα, ακόμα και από αυτά που έχουν το διπλάσιο μέγεθος του. Αυτό συμβαίνει διότι το συγκεκριμένο αραιό μητρώο έχει πολύ μεγάλο αριθμό μη μηδενικών στοιχειών έναντι των άλλων στην κατηγορία του (π.χ. ex7,NNZ=54543 και rdb5000,NNZ=29600). Ομοίως και για μεγαλύτερα αραιά μητρώα το αραιό μητρώο *epb1* παρά το μεγάλο μέγεθος του 14737x14737, έχει πολύ γρηγορότερο χρόνο εκτέλεσης με διαφορές μεγαλύτερες των 30s για την κατηγορία του, πόσο μάλλον ο χρόνος εκτέλεσης του είναι πολύ κοντά στο μικρότερο αραιό μητρώο *ex7*. Το συγκεκριμένο μητρώο έχει μικρότερο αριθμό μη μηδενικών στοιχείων έναντι των άλλων, πράγμα το οποίο επίσης αποδεικνύει ότι αυτό που παίζει μεγαλύτερο ρόλο πέρα από το μέγεθος του αραιού μητρώου για την ταχύτητα στην εκτέλεση, είναι το πόσα μη μηδενικά στοιχεία υπάρχουν.

Τέλος θα πρέπει να αναφερθεί ότι το γεγονός του ότι ο χρόνος εκτέλεσης κυμαίνεται σε χρόνους έως +40s, δεν έχει να κάνει τόσο με την αναγκαιότητα κατανομής και μεταφορών δεδομένων από το host σύστημα στο PIM. Διότι όπως φάνηκε και παραπάνω οι τελικοί χρόνοι για τις κατανομές είναι πολύ μικρότεροι της τάξεως από 0.1s – 0.7s από αυτούς για όλη την εκτέλεση. Εδώ πρόκειται για αποτέλεσμα απόδοσης του PIM συστήματος. Θα πρέπει επίσης να συμπεριληφθεί πως η αρχιτεκτονική δεν υποστηρίζει πράξεις με αριθμούς κινητής υποδιαστολής διπλής ακρίβειας τα αποτελέσματα τα βγάζει με κάποιο είδος αλγορίθμου που χρησιμοποιείται από το ίδιο το σύστημα [2], [7], πράγμα το οποίο παίζει σημαντικό ρόλο για την απόδοση. Καθώς επίσης και το ότι μονάδες των DPU έχουν μια σχετικά μικρή ταχύτητα ρολογιού στα 500MHz [2], [7].

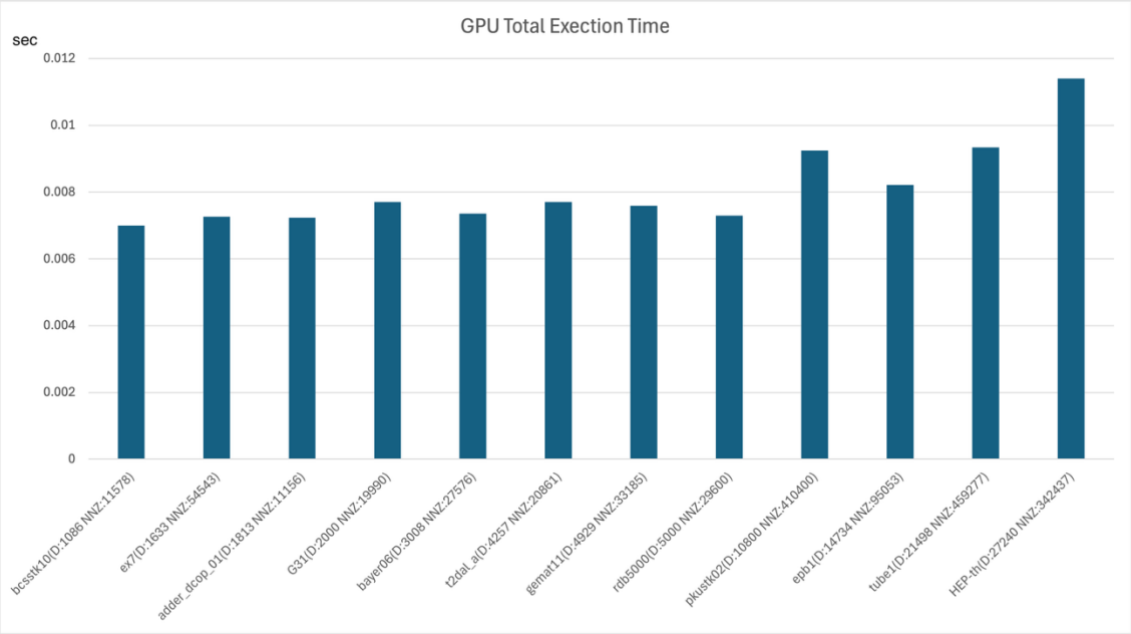
Παρατηρήσεις για τελικούς χρόνους εκτέλεσης προσομοίωσης LIF αναφορικά με την τεχνική κατανομής (κατανομή ανά γραμμές ROW, ανά μη μηδενικά στοιχεία NNZ):

Αναφορικά με τους τελικούς χρόνους εκτέλεσης παρατηρείται μια σχετικά ίδια απόδοση για τα μικρά – μεσαία αραιά μητρώα. Σχετικά με τα μεγαλύτερα αραιά μητρώα ο χρόνος εκτέλεσης εμφανίζει διαφορές για τις δύο τεχνικές κατά περίπου 2s, με την τεχνική κατανομής ανά ίσο αριθμό γραμμών στα DPU να εμφανίζει μεγαλύτερο χρόνο εκτέλεσης. Αυτό έχει ως συμπέρασμα ότι η αρχιτεκτονική για μεγάλα φορτία αποδίδει καλύτερα σε κατανομές που αφορούν ίσο αριθμό μη μηδενικών στοιχείων στις μονάδες.

5.3 Ανάλυση αποτελεσμάτων GPU

Η ανάλυση των αποτελεσμάτων για την GPU πραγματοποιείται μόνο ως προς τον τελικό χρόνο εκτέλεσης του μοντέλου LIF και όχι για επιμέρους χρόνους, όπως η κατανομή φορτίων ή οι διαφορές εκτέλεσης βάσει επιμέρους τεχνικών. Αυτό συμβαίνει διότι, στην αρχιτεκτονική GPU δεν απαιτείται συνεχής μεταφορά δεδομένων από κάποιο σύστημα host προς τη GPU. Επιπλέον, η παρούσα εργασία εστιάζει κυρίως στην ανάλυση του συστήματος PIM και στη σύγκριση της τελικής του απόδοσης με αυτήν μιας αρχιτεκτονικής τύπου GPU.

5.3.1 Τελικοί χρόνοι εκτέλεσης μοντέλου LIF GPU.



Εικόνα 5-7: Total Execution Time GPU (sec)

Ξεκινώντας με τον συνολικό χρόνο εκτέλεσης της GPU παρατηρείται πως η εκτέλεση της προσομοίωσης πραγματοποιείται για ένα εύρος χρόνου από 0.006s – 0.012s. Με λίγα λόγια εδώ ο χρόνος εκτέλεσης πλέον φτάνει να είναι τόσο γρηγορότερος που πλέον μπορεί να μετρηθεί εύκολα και στην κλίματα των msec. Σχετικά με τις διαφορές μεταξύ των αραιών μητρώων, παρατηρείται μια ομοιομορφία ως προς τους χρόνους εκτέλεσης μεταξύ των μητρώων. Βέβαια και εδώ όπως ήταν αναμενόμενο για μεγαλύτερα μητρώα που έχουν πολλά μη μηδενικά στοιχεία ο χρόνος εκτέλεσης είναι μεγαλύτερος, με την διαφορά ότι δεν εμφανίζονται τόσο μεγάλες διακυμάνσεις όπως στο σύστημα PIM παραπάνω. Για παράδειγμα για το αραιό μητρώο *bcsstk10* και το αραιό μητρώο HEP-th, δηλαδή σε σύγκριση ενός μικρού μητρώου με λιγότερα μη μηδενικά στοιχεία με ένα μεγαλύτερο με περισσότερα μη μηδενικά στοιχεία, οι διαφορές χρόνων κυμαίνονται στα 0.003s - 0.004s εν’ αντιθέσει το PIM σύστημα που είναι στα +40s.

Εδώ βγαίνει ως αποτέλεσμα ότι και στην GPU έχει σημασία το μέγεθος τον μητρών αλλά περισσότερο ο αριθμός των μη μηδενικών στοιχείων για το εκάστοτε αραιό μητρώο.

Απλά οι διαφορές στους χρόνους είναι λιγότερο έντονες και το σύστημα GPU έχει μια σαφώς καλύτερη απόδοση στην ταχύτητα.

5.4 Διαφορές GPU, PIM

Η συγκριτική μελέτη των χρόνων εκτέλεσης μεταξύ της αρχιτεκτονικής PIM και της GPU αποκαλύπτει μεγάλες διαφορές στην αποδοτικότητα των δύο συστημάτων. Συγκεκριμένα, οι χρόνοι εκτέλεσης της GPU κυμαίνονται σε επίπεδα χιλιοστών του δευτερολέπτου (msec**)**, για εύρος τιμών περίπου 0.011 s (11 ms), ενώ οι αντίστοιχοι χρόνοι του συστήματος PIM κυμαίνονται σε δευτερόλεπτα (s), φθάνοντας +40s. Η διαφορά αυτή αποδεικνύει ότι η GPU είναι κατά ένα μεγαλύτερο ποσοστό ταχύτερη από την αρχιτεκτονική PIM για τη συγκεκριμένη επιστημονική εφαρμογή. Πέρα από το κόστος χρόνου για ενημέρωση στοιχείων στους πύρινες DPU, που όπως φαίνεται και παραπάνω επηρεάζουν σαφώς λιγότερο. Αυτή η διαφορά οφείλεται, στην ταχύτητα και στην ποσότητα των πυρήνων GPU και PIM. Η GPU εμφανίζει μεγαλύτερες ταχύτητες ανά πυρήνα της τάξεως 1.4GHz έναντι της μονάδας DPU που εμφανίζει 500MHz. Παράλληλα, η GPU αξιοποιεί πολύ μεγαλύτερο αριθμό υπολογιστικών μονάδων, πάνω από 10.000+ πυρήνες – έναντι των 2560 DPUs του PIM, γεγονός που της επιτρέπει να εκτελεί μεγαλύτερο όγκο παράλληλων πράξεων. Επιπλέον, πρέπει να σημειωθεί ότι το σύστημα PIM δεν υποστηρίζει πράξεις αριθμών κινητής υποδιαστολής διπλής ακρίβειας**,** σε αντίθεση με την GPU. Για τον λόγο αυτό, οι αντίστοιχες πράξεις στο PIM υλοποιούνται μέσω αλγοριθμικών προσεγγίσεων ή προσομοιώσεων, γεγονός που προσθέτει επιπλέον υπολογιστικό κόστος και αυξάνει τον συνολικό χρόνο εκτέλεσης.

Επίσης και η συμπεριφορά των δύο αρχιτεκτονικών ως προς την αύξηση του αριθμού των μη μηδενικών στοιχείων (NNZ) και του μεγέθους των μητρών παρουσιάζει σημαντικές αποκλίσεις. Η GPU διατηρεί μια ομαλότητα (π.χ., από 7 ms σε 11.5 ms). σε χρόνους εκτέλεσης για την σχέση μεγέθους και ποσότητας μη μηδενικών στοιχείων ανά αραιά μητρώο. Εν αντιθέσει το PIM παρουσιάζει έντονες διαφορές από ένα μέγεθος φορτίου και μετά. Τα αραιά μητρώα με μεγάλο αριθμό σε μη μηδενικά στοιχεία και μέγεθος όπως οι *pkustk*02 και *tub4*, εμφανίζουν δραματική αύξηση του χρόνου εκτέλεσης, με διαφορές που μπορεί να φθάσουν σε δεκάδες φορές μεγαλύτερες μεταξύ διαφορετικών μητρών. Επίσης η επίδραση της μεθόδου κατανομής των δεδομένων (βάσει NNZ ή ROW) διαφέρει αισθητά μεταξύ των δύο αρχιτεκτονικών. Στο PIM αν και ο χρόνος της ίδιας της κατανομής είναι συγκριτικά μικρότερος με αυτόν της εκτέλεσης, η δομή διάταξης των δεδομένων επηρεάζει την αποδοτικότητα της εκτέλεσης, με την τεχνική κατανομής ανά μη μηδενικά στοιχεία να εμφανίζει καλύτερη απόδοση αντί αυτήν ανά γραμμών.

Αναφορικά με την κατανομή των φορτίων στο σύστημα PIM, η GPU πάλι εδώ είναι μια καλύτερη επιλογή για εφαρμογές στις οποίες χρειάζεται συνεχείς ενημέρωση δεδομένων στην μνήμη. Αυτό διότι δεν έγκειται στην αναγκαιότητα μιας host ξεχωριστής αρχιτεκτονικής για την συγκεκριμένη ενέργεια, αλλά χρησιμοποιεί την δικιά της μνήμη που διαθέτει στην πλακέτα της. Εν αντιθέσει στην αρχιτεκτονική PIM προστίθεται ένα κόστος χρόνου της τάξεως των δευτερολέπτων. Στην συγκεκριμένη εφαρμογή, μόνο η ενημέρωση του διανύσματος κυμαίνεται σε εύρος από 0.1s – 0.7s, σε περίπτωση που θα χρειαζόταν και ενημέρωση του μητρώου ο χρόνος θα ήταν ακόμα μεγαλύτερος. Πράγμα που θα δημιουργούσε τουλάχιστον +2s κόστος χρόνου για το συγκεκριμένο παράδειγμα.

Εν’ κατακλείδι, η GPU παρουσιάζει υψηλή, σταθερή και προβλέψιμη απόδοση, καθιστώντας την εξαιρετικά αποδοτική για τέτοιου πράξεις και εφαρμογές μεγάλης

κλίμακας. Αντιθέτως, η PIM, εμφανίζει σημαντικές προκλήσεις στην επίτευξη χαμηλών χρόνων εκτέλεσης, ιδιαίτερα σε εφαρμογές τέτοιου τύπου.

Συμπεράσματα

Η αρχιτεκτονική GPU, με την ταχύτητα της και την σταθερή απόδοση της σε διάφορες δομές φορτίου αραιών μητρώων – διανυσμάτων, παραμένει η καλύτερη λύση στην απόδοση για τις εξεταζόμενες πράξεις SPMV. Το σύστημα PIM, αν και φιλόδοξο, παρουσιάζει σημαντικά εμπόδια που σχετίζονται με την απόδοση. Οι χρόνοι εκτέλεσης είναι εξαιρετικά υψηλοί, με την διαδικασία και του υπολογισμού, πέρα της κατανομής, να αποτελεί τον κυρίαρχο παράγοντα καθυστέρησης. Επίσης η απόδοση του PIM είναι ασταθής ανάλογα με το μέγεθος του αραιού μητρώου και την ποσότητα των μη μηδενικών στοιχείων του, εμφανίζοντας σε μεγάλο όγκο αυτών πολύ μεγαλύτερες αποκλίσεις. Κάτι άλλο που επηρεάζει την απόδοση είναι η επιλογή της τεχνικής κατανομής δεδομένων στην μνήμη, οδηγώντας σε μετρήσιμες διαφορές σε μονάδα δευτερολέπτων (sec) στον συνολικό χρόνο εκτέλεσης. Αυτό αποδεικνύει ότι η βελτιστοποίηση της διάταξης δεδομένων παίζει επίσης ρόλο για την αρχιτεκτονική PIM. Επίσης η εξάρτηση που εμφανίζει από μια εξωτερική αρχιτεκτονική host, την κάνει ακατάλληλη για εφαρμογές που χρήζουν συνεχείς ενημερώσεις δεδομένων, καθώς προστίθεται σημαντικό κόστος χρόνου για επιστημονικές εφαρμογές. Η έλλειψη υποστήριξης αριθμών κινητής υποδιαστολής διπλής ακρίβειας σε επίπεδο κυκλώματος, καθιστούν την υλοποίηση εκτός από πιο σύνθετη, καθώς απαιτείται η χρήση λογισμικού προσομοίωσης, αλλά και λιγότερο αποδοτική καθώς απαιτείται υπολογιστική ισχύ για την υλοποίηση. Καθώς επίσης σε περιπτώσεις που τα αραιά μητρώα υπερβαίνουν την συνολική μνήμη MRAM των διαθέσιμων DPUs, θα πρέπει να εφαρμοστούν τεχνικές ανταλλαγής δεδομένων μεταξύ εξωτερικής μνήμης (swapping) π.χ. από host σύστημα σε πραγματικό χρόνο. Κάτι το οποίο επίσης θα μπορούσε να επηρεάσει την απόδοση του συστήματος. Παρ’ όλα αυτά, η θεμελιώδης αρχή της επεξεργασίας εντός μνήμης παραμένει ελπιδοφόρα, καθώς υπόσχεται δραστική μείωση της κίνησης δεδομένων και βελτίωση της ενεργειακής αποδοτικότητας σε μελλοντικά συστήματα.

Σχετικά με μελλοντική έρευνα και ανάπτυξη, για να καταστεί ανταγωνιστικό το PIM σε επιστημονικές εφαρμογές, όπως η συγκεκριμένη, θα πρέπει να εστιάσει στην βελτιστοποίηση των πυρήνων DPU, κάνοντας τους πιο ισχυρούς και περισσότερους σε αριθμό, προσαρμόζοντας την αρχιτεκτονική να είναι συμβατή με πράξεις αριθμών κινητής υποδιαστολής διπλής ακρίβειας όπως και στις GPU. Καθώς θα πρέπει η αρχιτεκτονική να επανασχεδιαστεί με έναν τρόπο που θα έχει μικρή έως αμελητέα εξάρτηση από τον επεξεργαστή αναφορικά με τα δεδομένα και την ενημέρωση τους στην μνήμη. Αυτήν την στιγμή, μπορεί η αρχιτεκτονική να μην μπορεί να είναι σε θέση να αποδώσει σε τέτοιου τύπου έντονες υπολογιστικά εφαρμογές όπως ο πολλαπλασιασμός αραιού μητρώου επί διάνυσμα, αλλά για εφαρμογές τύπου caching [5], [9] και εφαρμογές που χρειάζονται μικρό-υπολογισμοί με σταθερά και μη δυναμικά δεδομένα π.χ. δεδομένα βάσεων δεδομένων, ενδεχομένως να είναι μια καλή λύση.

Βιβλιογραφία

[1] N. Bell and M. Garland, “Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors.”

[2] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, “Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture,” May 2022, [Online]. Available: http://arxiv.org/abs/2105.03814

[3] Y. Saad, “Iterative Methods for Sparse Linear Systems Second Edition.”

[4] Nvidia, “NVIDIA A100 Tensor Core GPU Architecture UNPRECEDENTED ACCELERATION AT EVERY SCALE ii NVIDIA A100 Tensor Core GPU Architecture.”

[5] *Hennessy, J. L., & Patterson, D. A. (2017). Computer Architecture: A Quantitative Approach. Elsevier.*

[6] Nvidia, “NVIDIA ADA GPU ARCHITECTURE Designed to deliver outstanding gaming and creating, professional graphics, AI, and compute performance. *Updated to include information on NVIDIA L40 and L4 Data Center GPUs ii NVIDIA Ada GPU Architecture.”

[7] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, “SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Systems,” May 2022, [Online]. Available: http://arxiv.org/abs/2201.05072

[8] I. E. Venetis and A. Provata, “Analysis of the Leaky Integrate-and-Fire neuron model for GPU implementation.”

[9] A. S. Tanenbaum, “MODERN OPERATING SYSTEMS FIFTH EDITION.”

[10] “Intel Architecture Software Developer’s Manual Volume 1: Basic Architecture,” 1999. [Online]. Available: http://www.intel.com.

[11] “CUDA by Example.” [Online]. Available: www.wowebook.com

[12] “CUDA C++ Programming Guide Release 13.0 NVIDIA Corporation,” 2025.

[13] B. Ward, “B R I A N W A R D H O W L I N U X,” 2021.

[14] D. Patterson, “In Praise of Programming Massively Parallel Processors: A Hands-on Approach.”

[15] E. R. . Kandel, John. Koester, Sarah. Mack, and Steven. Siegelbaum, *Principles of neural science*. McGraw Hill, 2021.

[16] “Dynamical Systems in Neuroscience.”

[17] A. L. Hodgkin and A. F. Huxley, “I952) I I7,” 1952.

[18] “A Model of Neuronal Bursting Using Three Coupled First Order Differential Equations”.

[19] R. Fitzhugh, “IMPULSES AND PHYSIOLOGICAL STATES IN THEORETICAL MODELS OF NERVE MEMBRANE.”

[20] J. Nagumot, S. Arimotot, and S. Yoshizawat, “PROCEEDINGS OF THE IRE An Active Pulse Transmission Line Simulating Nerve Axon*.”

[21] B. Golosio, G. Tiddia, C. De Luca, E. Pastorelli, F. Simula, and P. S. Paolucci, “Fast simulations of highly-connected spiking cortical models using GPUs,” Nov. 2020, doi: 10.3389/fncom.2021.627620.

[22] W. Gerstner and W. M. Kistler, “Spiking Neuron Models Single Neurons, Populations, Plasticity,” 2002.

[23] “Introduction to High Performance Scientific Computing.” [Online]. Available: http://www.saylor.org.

[24] L. Dagum and R. Menon, “OpenMP: An Industry-Standard API for Shared-Memory Programming,” 1998. [Online]. Available: www.openmp.org.

[25] Herlihy, Maurice, Shavit, and Nir, “The Art of Multiprocessor Programming.”

[26] Ananth. Grama, *Introduction to parallel computing*. Addison-Wesley, 2003.

[27] “Developer Reference for Intel ® oneAPI Math Kernel Library for C Chapter 1: Developer Reference for Intel ® oneAPI Math Kernel Library-C.”

[28] D. Merrill and M. Garland, “Single-pass Parallel Prefix Scan with Decoupled Look-back.”

[29] V. Volkov, “Better Performance at Lower Occupancy,” 2010.