ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάλυση και Πειραματική Επαλήθευση του paper:

TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory

Εξαμηνιαία Εργασία στο μάθημα

Προηγμένα Θέματα Αρχιτεκτονικής Υπολογιστών

Δημητρίου Αγγελική (ΑΜ:03117106)

Τζομάκα Αφροδίτη (ΑΜ:03117107)

ΕΙΣΑΓΩΓΗ

Η παρούσα εργασία εκπονήθηκε στα πλαίσια του μαθήματος των Προηγμένων Θεμάτων Αρχιτεκτονικής Υπολογιστών. Ο στόχος της ήταν η ανάλυση και η κατανόηση ενός επιστημονικού paper με συγκεκριμένο θέμα και στη συνέχεια η αναπαραγωγή των αποτελεσμάτων αυτού και η επιβεβαίωσή τους. Το paper λοιπόν που θα αναλύσουμε στην συνέχεια ονομάζεται TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory (Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, Christos Kozyrakis @ Stanford University). Ακολουθεί η περιγραφή του paper, καταγραφή των αποτελεσμάτων του που συγκεκριμένα εμείς καλούμαστε να αναπαράξουμε, περιγραφή της πειραματικής διαδικασίας, παρουσίαση των δικών μας αποτελεσμάτων και συμπεράσματα πάνω σε αυτά.

ΠΕΡΙΓΡΑΦΗ ΤΟΥ PAPER

Τα τελευταία χρόνια, η ραγδαία ανάπτυξη στον τομέα της τεχνητής νοημοσύνης και της μηχανικής μάθησης έχει οδηγήσει φυσικά σε αυξημένες απαιτήσεις τόσο στο software όσο και στο hardware που απαιτείται για σχετικές υλοποιήσεις. Πιο συγκεκριμένα, οι σύγχρονες εφαρμογές που βασίζονται στα AI και ML απαιτούν αυξημένη ακρίβεια και συνεπώς την χρήση deep neural networks. Τα νευρωνικά αυτά δίκτυα είναι από τη μία πολύ βαριά υπολογιστικά, από την άλλη resource hungry όσον αφορά στη μνήμη που απαιτούν και επίσης κλιμακώνονται με σχετικά γρήγορο ρυθμό. Επομένως η προσεκτική και έξυπνη σχεδίαση των αντίστοιχων επιταχυντών είναι το ουσιαστικό ζητούμενο σε αυτό το πρόβλημα.

Ένα τυπικό βαθύ νευρωνικό δίκτυο αποτελείται από πολλαπλά CONV επίπεδα που πραγματοποιούν 2D συνέλιξη της εισόδου με φίλτρα (π.χ. μεταξύ εικόνων). Κάποια CONV επίπεδα ακολουθούνται από POOL επίπεδα που συμβάλλουν στην υποδειγματοληψία για καλύτερη εξαγωγή χαρακτηριστικών. Στο τέλος της στοίβας του feature extractor τα FC επίπεδα πραγματοποιούν την τελική ταξινόμηση. Τα feature maps που εισάγονται ως είσοδος σε CONV επίπεδα ή προκύπτουν ως έξοδος τους ονομάζονται i/o-fmaps και μπορούν να επεξεργάζονται σε batches για μεγαλύτερη αποδοτικότητα. Σύμφωνα με μελέτες και πειράματα, όσο περισσότερα από αυτά τα επίπεδα υπάρχουν και όσο μεγαλύτερος είναι ο αριθμός των fmaps αυτών τόσο πιο αποτελεσματικό είναι το νευρωνικό δίκτυο. Οι δύο αυτές τάσεις αυξάνουν το μέγεθος των νευρωνικών και ενισχύουν το πρόβλημα της δημιουργίας bottleneck στη μνήμη λόγω της επεξεργασίας όλο και περισσότερων δεδομένων.

Για το λόγο αυτό, το δεδομένο paper προτείνει έναν νέο επιταχυντή που ονομάζεται TETRIS. Ο ίδιος στοχεύει τόσο στην κλιμάκωση της επίδοσης όσο και στη μείωση του ενεργειακού κόστους των διαδικασιών συμπερασμού που εκπονούν NNs μεγάλης κλίμακας. Σε αντίθεση με προηγούμενες κοστοβόρες προσεγγίσεις σε κατανάλωση ενέργειας και χρήμα, το TETRIS αποφεύγει τη συγκέντρωση μεγάλων SRAM buffers στο chip και την ύπαρξη πολλαπλών DDRx καναλιών στη μνήμη για την υποστήριξη των processing elements (PE) των neural engines. Για να το πετύχει αυτό συνδυάζει την 3D σχεδίαση της μνήμης με τη χρήση βελτιστοποιημένων προγραμμάτων dataflow.

Το TETRIS συνίσταται από ένα logic die ως τελευταίο επίπεδο και πολλαπλά DRAM dies που στοιβάζονται πάνω από αυτό. Χρησιμοποιεί ως μνήμη ένα Hybrid Memory Cube (HMC) 8 επιπέδων οργανωμένο σε έναν αριθμό από 32-bit vaults - κάθετα κανάλια (τυπικά 16) καθένα από τα οποία σχετίζεται με μεγάλο αριθμό από PEs (τυπικά 14 x 14) και μία μόνο SRAM που λειτουργεί ως global buffer. Τα HMC, όπως προαναφέρθηκε, είναι τρισδιάστατες

μνήμες που βασίζονται στην τεχνολογία through silicon via (TSV), μια προηγμένη τεχνική interconnent που ενώνει dies κατακόρυφα δημιουργώντας την έννοια του vault όπως φαίνεται στο παρακάτω σχήμα. Τα κανάλια αυτά έχοντας δικό τους neural engine μπορούν να λειτουργούν ξεχωριστά και να επεξεργάζονται δεδομένα παράλληλα. Κάθε επίπεδο μνήμης τους διαθέτει 2 banks που επικοινωνούν μεταξύ τους μέσω κοινού διαδρόμου και αποθηκεύουν δεδομένα που έρχονται από ή κατευθύνονται στο logic layer μέσω των TSVs. Αντί για το κλασσικό crossbar δίκτυο για τη σύνδεση των vaults που υλοποιείται από ένα HMC, στο TETRIS υπάρχει 2D mesh NoC για καλύτερη προώθηση remote accesses μεταξύ καναλιών. Τέλος, όσον αφορά το neural engine προσομοιάζει τον Eyeriss και αποτελεί 2D array από PE καθένα από τα οποία περιέχει 16-bit fixed- point ALU και μικρό register file 512-1024 bytes.

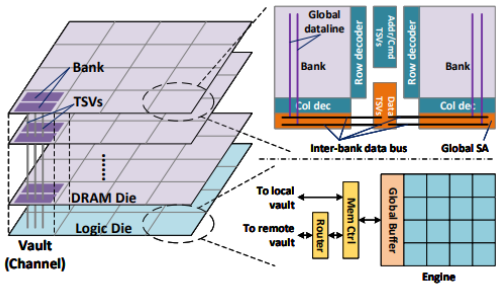


Figure 2. TETRIS architecture. Left: HMC stack. Right top: per-vault DRAM die structure. Right bottom: per-vault logic die structure.

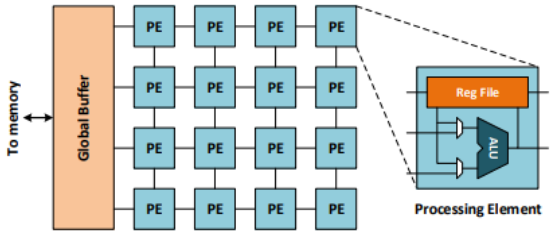


Figure 3. NN engine associated with one vault in TETRIS.

Ο επιταχυντής TETRIS χαρακτηρίζεται από βελτιστοποιήσεις τόσο στο hardware όσο και στο software.

Από άποψη hardware, η ύπαρξη 3D μνήμης συμβάλλει στην απελευθέρωση χώρου στο chip που θα καταναλώνονταν από SRAM buffers και στην απόδοση αυτού σε περισσότερα PE. Οι απομονωτές πάνω στο chip είναι μικρότεροι, λιγότεροι και διαφορετικού σκοπού. Ταυτόχρονα αυξάνεται το memory bandwidth, το οποίο σε ένα 2D design θα έπρεπε να επιτευχθεί με πολλαπλά DDRx channels, τα οποία καταναλώνουν μεγάλα ποσά ενέργειας. Η χρήση της 3D μνήμης οδηγεί επίσης σε αναθεώρηση της τοποθεσίας εκτέλεσης των πράξεων. Συγκεκριμένα, είναι ωφέλιμο κάποιες από αυτές να συμβαίνουν εντός της μνήμης μιας και σε αντίθεση με τα κανάλια DDRx που διαχέουν τις τιμές σε πολλαπλές DRAM, ένα vault στέλνει όλα τα bits στο ίδιο bank της DRAM επιτρέποντας τη διενέργεια του accumulation τοπικά. Το in-memory accumulation συμβάλλει στη μείωση της κίνησης που αφορά τη μνήμη αυξάνοντας το bandwidth και το performance αλλά μειώνοντας την ενέργεια στα κατακόρυφα κανάλια, ενώ επιτρέπει την back-to-back εκτέλεση read/write στη DRAM και άρα μειώνει την καθυστέρηση. Το TETRIS χρησιμοποιεί είτε bank είτε DRAM die accumulation που έχουν τα πιο ευνοϊκά αποτελέσματα.

Από άποψη software, το TETRIS αξιοποιεί ένα αναλυτικό μοντέλο για την εύρεση του βέλτιστου dataflow scheduling. Η βελτιστοποίηση του dataflow αποτελείται από δύο υποπροβλήματα, το mapping problem και το ordering problem. Το πρώτο αφορά την αντιστοίχιση της εισόδου, εξόδου και φίλτρων μιας 2D συνέλιξης στα PE και επιλύεται με το row stationary dataflow του Eyeriss. Από την άλλη, το δεύτερο που αφορά τη σειρά με την

οποία γίνεται η συνέλιξη και το buffering των δεδομένων (blocking/reordering τεχνικές) παραμένει άλυτο στη γενική περίπτωση με αποτέλεσμα προηγούμενες αρχιτεκτονικές να καταφεύγουν σε εξαντλητική αναζήτηση. Στην περίπτωση του TETRIS ωστόσο που διαθέτει μεγάλο αριθμό PE και μικρό global buffer, το πρόβλημα καθίσταται ευκολότερο και χρησιμοποιείται η τεχνική bypass ordering (προσπέραση του global buffer για το μεγαλύτερο κομμάτι της εισόδου και χρήση του για αποθήκευση δεδομένων για την υπόλοιπη είσοδο), η οποία εξαλείφει την επαναχρησιμοποίηση δεδομένων και επωφελείται από το in-memory accumulation.

Τέλος, με σκοπό την αξιοποίηση της παραλληλίας που δύναται να προσφέρει μια 3D αρχιτεκτονική το TETRIS εφαρμόζει ένα υβριδικό partitioning των επιπέδων του νευρωνικού ανάμεσα στα ανεξάρτητα μεταξύ τους vaults. Αναλυτικότερα, συνδυάζει τις τεχνικές fmap και output partitioning (διαμοιρασμός των fmaps και της εξόδου μεταξύ vaults αντίστοιχα) επιτυγχάνοντας την ελαχιστοποίηση των remote accesses αλλά και την καλύτερη επαναχρησιμοποίηση των on-chip δεδομένων για λιγότερα DRAM accesses.

Στα πλαίσια του paper λοιπόν, με σκοπό την αξιολόγηση του προτεινόμενου επιταχυντή έγιναν πειράματα για τη γενική απόδοση και την κατανάλωση ενέργειας, το 3D NN hardware και το προτεινόμενο μοντέλο scheduling και partitioning. Για την διεξαγωγή των πειραμάτων χρησιμοποιήθηκαν state-of-the-art NNs, όπως AlexNet, ResNet, VGGNet, ZFNet. Τα τεχνικά χαρακτηριστικά του συστήματος αναφέρονται στο paper ενώ για την μοντελοποίηση χρησιμοποιήθηκαν: το CACTI-P για τον χώρο και την ενέργεια των register files και global buffers, το ORION για την ενεργεια του NoC και τελικά το zsim (επεκτεταμένο ώστε να υποστηρίζει την μοντελοποίηση του NoC interconnection) για το performance. Όσον αφορά στο scheduling και το partitioning έγινε χρήση του εργαλείου nn_dataflow.

Για τα αποτελέσματα των προαναφερθέντων πειραμάτων αξιολόγησης έχουμε τα εξής:

- Γενική απόδοση και κατανάλωση ενέργειας: Ένας 16-vault TETRIS επιταχυντής αυξάνει το Performance κατά 4.1x και μειώνει την ενέργεια κατά 1.48x σε σύγκριση με 2D baselines 4 καναλιών DDR3 ενώ καλυτερεύει τόσο την απόδοση όσο και την ενεργεία αλλά μόνο για 20% και 15% αντίστοιχα σε σχέση με το Neurocube (=> ανάγκη για καλύτερο scheduling και partition). Θα το δούμε αναλυτικότερα παρακάτω μιας και είναι το πείραμα με το οποίο ασχοληθήκαμε εμείς.
- 3D NN Hardware: Συγκρίνοντας μια baseline 2D αρχιτεκτονική LPDDR3 καναλιών με μια 3D projected αντίστοιχη αρχιτεκτονική (όπου χρησιμοποιείται η ίδια αναλογία PE-buffer ως προς τον χώρο), μια αρχιτεκτονική TETRIS χωρίς in-memory accumulation (Tn), μια αρχιτεκτονική TETRIS με DRAM accumulation (Td) και μια αρχιτεκτονική TETRIS με bank accumulation (Tb) παρατηρούμε αρχικά την υπεροχή των TETRIS αρχιτεκτονικών σε σχέση με την 3D projected εφόσον έχουμε περισσότερα PEs ενώ η δεύτερη αυξάνει σημαντικά την στατική ενέργεια. Τέλος ανάμεσα στις TETRIS αρχιτεκτονικές η Td μειώνει την δυναμική ενέργεια κατα 6.1x σε σχέση με την Tn ενώ η Tb επιφέρει μια καλυτέρευση το πολύ κατα 3.1% εφόσον έχουν ήδη ελαχιστοποιηθεί οι προσβάσεις στην DRAM.
- Dataflow scheduling & partitioning: Εδώ συγκρίνεται αρχικά το αναλυτικό μοντέλο scheduling που προτείνει το paper με το αντίστοιχο βέλτιστο που έχει προκύψει από exhaustive search και με διαφορές μικρότερες των 2.9% στο runtime και 1.8% στην ενέργεια βλέπουμε ότι τα δύο schedules είναι ισοδύναμα. Τέλος, συγκρίνεται το base partitioning του Neurocube με το hybrid partitiong του TETRIS, τόσο για το αναλυτικό όσο και για το βέλτιστο scheduling και αυτό που παρατηρείται είναι πως πέραν της ανεπαίσθητης αύξησης του NoC power, η δυναμική ενέργεια μειώνεται

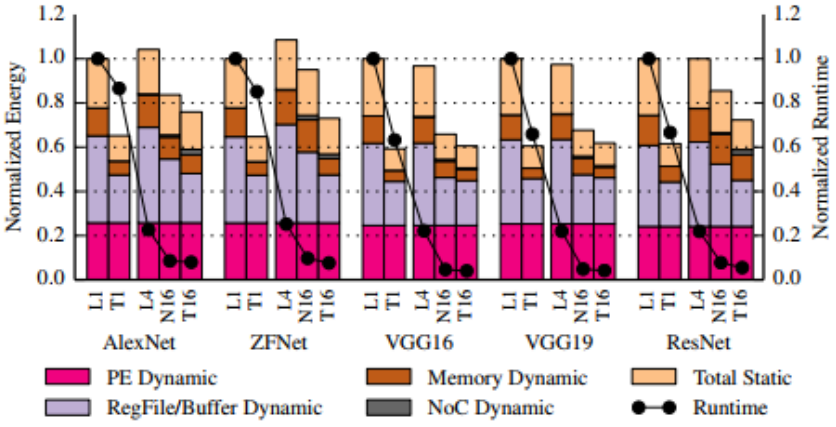
αρκετά χάριν του data reuse, οδηγώντας σε 13.3% αύξηση του performance και 10.5% μείωση των ενεργειακών απωλειών.

ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΥΠΟ ΕΞΕΤΑΣΗ ΠΕΙΡΑΜΑΤΟΣ

Το πείραμα που επιλέξαμε να αναπαράγουμε είναι το πρώτο που αφορά στη γενική αξιολόγηση της απόδοσης και ενέργειας του TETRIS σε σχέση με άλλες αρχιτεκτονικές. Συγκεκριμένα, οι αρχιτεκτονικές που συγκρίθηκαν είναι οι εξής:

- L1 - L4: 2D baseline αρχιτεκτονικές οι οποίες χρησιμοποιούν 1 και 4 κανάλια LPDDR3 αντίστοιχα. Το neural engine (τύπου Eyeriss) του L1 αποτελείται από μια συστοιχία 16 × 16 PE με 1024-byte register file το καθένα και global buffer των 576 kB. Το L4 διαθέτει 4 NN engines και άρα συνολικά 1024 PEs και 2.3 MB global buffer.
- T1 - T16: Πρόκειται για την αρχιτεκτονική του TETRIS που περιγράφηκε παραπάνω με 1 και 16 vaults αντίστοιχα. Όπως έχει ειπωθεί χρησιμοποιεί 14 × 14 PE array ανά vault, 512-byte register file ανά PE και global buffer των 133 kB ανά vault. Επιλέγεται bank in-memory accumulation και χρησιμοποιείται το μοντέλο scheduling και hybrid partitioning που προτείνεται στο paper.
- N16: 3D αρχιτεκτονική Neurocube των 16 vaults που προσομοιάζει εκείνη του TETRIS αλλά διαφέρει κατά βάση στο software (scheduling, partitioning schemes).

Οι παραπάνω αρχιτεκτονικές αξιολογήθηκαν με βάση 5 state-of-the-art νευρωνικά δίκτυα, τα AlexNet, ZFNet, VGG-16, VGG-19 και ResNet152. Στα πλαίσια της εργασίας, θα γίνει προσπάθεια αναπαραγωγής της παρακάτω γραφικής παράστασης που υπάρχει στο paper.



L1: LPDDR3-1, T1: TETRIS-1, L4: LPDDR3-4, N16: Neurocube-16, T16: TETRIS-16

Λόγω περιορισμών που οφείλονται στον προσομοιωτή και τα μηχανήματα μας (δημιουργία deadlocks κατά το simulation), η προσομοίωση της T16 αρχιτεκτονικής δεν ήταν δυνατή. Το simulation με τα περισσότερα vaults που μπορέσαμε να τρέξουμε ήταν το T4. Όσον αφορά το Neurocube δεν προσομοιώθηκε καθώς δεν είχαμε καν πρόσβαση στο σχετικό paper.
Επιπρόσθετα, τα στατιστικά ενέργειας που φαίνονται παραπάνω δεν προκύπτουν άμεσα από τον προσομοιωτή που χρησιμοποιήθηκε. Για την εξάγωγη τους χρειάζεται λεπτομερής διερεύνηση των αποτελεσμάτων και συγκέντρωση των event counts, τα οποία στη συνέχεια πρέπει να περάσουν από επιπλέον μοντέλα ενέργειας, πχ McPat. Κάτι τέτοιο ξεφεύγει από τα πλαίσια της εργασίας μας και δε θα προσομοιωθεί.

ΔΙΕΞΑΓΩΓΗ ΤΟΥ ΠΕΙΡΑΜΑΤΟΣ

Περιγραφή των εργαλείων

NN Dataflow: Ένα εργαλείο γραμμένο σε Python που βρίσκει ενεργειακά αποδοτικό dataflow scheduling για νευρωνικά δίκτυα το οποίο είναι συμβατό με τις αρχιτεκτονικές που θα προσομοιωθούν. Στα πλαίσια του πειράματος θα θεωρηθεί πως το json αρχείο dataflow που παράγει, αντιπροσωπεύει το βέλτιστο scheduling.

Zsim: Πρόκειται για τον x86-64 simulator που θα χρησιμοποιήσουμε για την προσομοίωση των αρχιτεκτονικών. Είναι γρήγορος, απλός και ακριβής και αποσκοπεί στην προσομοίωση ιεραρχιών μνήμης και μεγάλων ετερογενών συστημάτων. Εκμεταλλεύεται την παραλληλία και κλιμακώνει καλά σε πολυπύρηνα συστήματα.

Zsim extensions: Μιας και το zsim προϋπήρχε του εν λόγω paper, χρειάστηκε να υλοποιηθούν ορισμένες επεκτάσεις οι οποίες δεν υπήρχαν διαθέσιμες online και στάλθηκαν σε εμάς μέσω των συγγραφέων του paper. Αρχικά, μοντελοποιήθηκε η καθυστέρηση που οφείλεται στα logic die NoC interconnects και προστέθηκε ένας trace generator ο οποίος ενσωματώνει το κατάλληλο scheduling αλλά και prefetching για να παράξει κατάλληλα memory address traces.

Διαδικασία

Αρχικά, προκειμένου να εξασφαλιστεί το κατάλληλο περιβάλλον με τα απαραίτητα για το zsim dependencies, χρησιμοποιήθηκε ένα docker image που ήταν διαθέσιμο online* το οποίο εμπλουτίστηκε για να υποστηρίξει το project μας.
Για να τρέξουμε μια επιτυχή προσομοίωση ακολουθούνται τα παρακάτω βήματα:

- Παραγωγή json αρχείου με το κατάλληλο scheduling και partitioning χρησιμοποιώντας το nn_dataflow.
- Παραγωγή των trace files που λαμβάνει ως είσοδο το zsim. Ο trace generator λαμβάνει ως είσοδο το προηγούμενο json αρχείο και παράγει traces για καθένα από τα επίπεδα του εν λόγω νευρωνικού.
- Δημιουργία του κατάλληλου patchRoot ανάλογα με την αρχιτεκτονική που θα χρησιμοποιηθεί. Εδώ προσδιορίζεται ο αριθμός των CPUs και NUMA nodes που θα θεωρήσει το zsim πως υπάρχουν στο σύστημα. Αυτό το βήμα δε χρειάζεται να ακολουθηθεί πριν από κάθε simulation.
- Συγγραφή του κατάλληλου configuration file για το zsim. Εδώ προσδιορίζονται οι λεπτομέρειες της αρχιτεκτονικής αλλά και η τοποθεσία των trace files (περισσότερα λόγια παρακάτω).
- Τρέξιμο bash script γραμμένου από εμάς το οποίο τρέχει ένα ένα τα simulations κάθε επιπέδου του νευρωνικού.

Με την ολοκλήρωση της προσομοίωσης παράγεται αρχείο zsim.out το οποίο διαθέτει αρκετές πληροφορίες για το simulation. Από αυτές, εκείνη που μας ενδιαφέρει είναι το contention time, το οποίο με κατάλληλο python script εξάγεται και αθροίζεται για όλα τα επίπεδα ενός νευρωνικού δίνοντας το runtime της συγκεκριμένης προσομοίωσης.

Όλα αυτά και μέχρις ότου να επιβεβαιωθεί η ορθότητα των configurations έγιναν simulate για το AlexNet, μιας και είναι το μικρότερο νευρωνικό και παίρνει σημαντικά λιγότερο χρόνο. Ύστερα προχωρήσαμε και στα υπόλοιπα που ήταν και πιο χρονοβόρα.

Configurations

Σε πρώτο επίπεδο, ασχοληθήκαμε με τα configurations που θα εισάγονταν στο εργαλείο για το scheduling. Με τη βοήθεια έτοιμων tests που υπήρχαν στο github του εργαλείου nn_dataflow με σκοπό την επαλήθευση του, βρέθηκαν τα configurations των αρχιτεκτονικών L1 και T16. Χρησιμοποιώντας το config του L1 κατασκευάσαμε εκείνο του L4 και αντίστοιχα με τη βοήθεια του T16 βρήκαμε εκείνα του T1 και T4. Τα τελικά configurations που προέκυψαν είναι τα παρακάτω:

Παράμετρος	**L1**	**L4**	**T1**	**T4**	**Επεξήγηση**
batch	16	16	16	16	
nodes	1 1	4 1	1 1	2 2	
array	16 16	16 16	14 14	14 14	PE array
regf	1024	1024	512	512	
gbuf	576056	2411725	133032	133032	
op-cost	2	2	2	2	cost of arithmetic operation
hier-cost (DRAM_COST, GBUF_COST, ITCN_COST, REGF_COST)	240 28 4 1	240 28 4 1	80 14 4 0.6	80 14 4 0.6	
hop-cost	0	0	0	40	cost of access through one NoC hop
unit-idle-cost	320	320	140	170	static cost over all nodes for unit execution time
mem-type	2D	2D	3D	3D	
solve-loopblocking	no	no	yes	yes	
hybrid-partition	no	no	yes	yes	
<net>	alex_net	alex_net	alex_net	alex_net	for example

Για να παραχθεί ένα json output τρέχουμε μια εντολή όπως η παρακάτω για το L1 του AlexNet:

```
python3 ./nn_dataflow_search.py --batch 16 --nodes 1 1 --array 16 16 --regf
1024 --gbuf 576056 --op-cost 2 --hier-cost 240 28 4 1 --hop-cost 0 --unit-
idle-cost 320 alex_net > l1.json
```

Όσον αφορά στις παραπάνω τιμές, η αναλογία είναι προφανής εκτός από τις παραμέτρους που αφορούν τα κόστη. Αρχικά, αποφασίσαμε να κρατήσουμε ίδια τα κόστη ιεραρχίας μνήμης; μεταξύ των δύο Lx και Tx αρχιτεκτονικών. Αναφορικά με το καθένα από αυτά, τα κόστη των DRAM, register file και interconnects είναι σχετικά προφανές πως θα παραμείνουν ίδια, ενώ για το κόστος πρόσβασης στον global buffer καταλήξαμε μετά από πειράματα με το εργαλείο πως μπορεί να θεωρηθεί επίσης ίδιο. Στη συνέχεια, το hop cost δεν είναι μηδενικό μόνο στην περίπτωση του T4 όπου χρειάζεται να λειτουργήσει το NoC και τελικά το στατικό κόστος προσδιορίσθηκε στις παραπάνω τιμές μέσω της παρατήρησης αντίστοιχων γραφικών παραστάσεων και πειραμάτων.

Μετά τον προσδιορισμό των παραπάνω configurations, προχωρήσαμε στην εύρεση των configurations του zsim. Είχαμε ήδη στη διάθεση μας τα configurations για τα T1 και T4 από τον κώδικα που μας στάλθηκε, οπότε χρειάστηκε να προσδιορίσουμε μόνο εκείνα των Lx.

Παρακάτω φαίνονται τα εν λόγω configurations για τις αρχιτεκτονικές L1 και T1 αντίστοιχα. Με σκούρο μπλε σημειώνονται οι αλλαγές που αντιστοιχούν στις αρχιτεκτονικές L4 και T4.

L1 - L4

```
sys = {
        cores = {
        c = {
        cores = 1;
        type = "Null";
        };
        };

        logic = {
        l = {
        units = 1;              #4
        frequency = 500;
        traceFilePrefix = <path to trace file>;
        rdbuf = "rdbuf";
        wrbuf = "wrbuf";
        };
        };

        lineSize = 32;
        frequency = 2000;

        caches = {
        rdbuf = {
        array = { ways = 1 };
        caches = 1;             #4
        size = 32;
        latency = 0;
        };

        wrbuf = {
        array = { ways = 1 };
        caches = 1;             #4
        size = 32;
        latency = 0;
        };

        gbuf = {
        latency = 1;
        array = {
        type = "IdealLRU";
        ways = 64;
        };
        caches = 1;
        banks = 8;
        size = 576256;          #2411776
        children = "rdbuf|wrbuf";
        };

        # coherent = false;
        };

        mem = {
```

```
            #splitAddrs = False;
            controllers = 1;          #4

            type = "Channel";

            channelType = "DDR";

            channelFreq = 1000;
            pageSize = 256;
            pagePolicy = "close";
            deviceIOWidth = 32;
            channelWidth = 32;
            queueDepth = 64;

            banksPerRank = 2;
            ranksPerChannel = 8;

            timing = {
            tCAS = 11;
            tRCD = 11;
            tRP = 11;
            tRAS = 22;
            tWR = 12;
            tCCD = 4;
            tRTP = 6;
            tRRD = 4;
            tWTR = 6;
            tFAW = 20;
            tRTRS = 0;
            tRFC = 117;
            tREFI = 7800;
            };
            power = {
            VDD = 1.2;
            IDD0 = 15.9;
            IDD2N = 7.9;
            IDD2P = 7.9;  # no power-down
            IDD3N = 8.4;  # IDD2N + 0.5 mA
            IDD3P = 8.4;  # no power-down
            IDD4R = 289.4;
            IDD4W = 296.3;
            IDD5 = 32.1;

            channelWirePicoJoulePerBit = 0.6;
            };
            };
};

sim = {
            gmMBytes = 1024;
            deadlockDetection = false;
};

process0 = {
            command = "/root/zsim/misc/hooks/test_logic_unit";
            startFastForwarded = true
};
```

T1 - T4

```
sys = {
        cores = {
        c = {
        cores = 1;              #4
        type = "Null";
        };
        };

        logic = {
        l = {
        units = 1;              #4
        frequency = 500;
        traceFilePrefix = <path to trace file>,
        rdbuf = "rdbuf";
        wrbuf = "wrbuf";
        };
        };

        lineSize = 32;
        frequency = 2000;

        caches = {
        rdbuf = {
        array = { ways = 1 };
        caches = 1;             #4
        size = 32;              #64
        latency = 0;
        };

        wrbuf = {
        array = { ways = 1 };
        caches = 1;             #4
        size = 32;              #64
        latency = 0;
        };

        gbuf = {
        latency = 1;
        array = {
        type = "IdealLRU";
        ways = 64;
        };
        caches = 1;             #4
        banks = 8;
        size = 131072;
        children = "rdbuf|wrbuf";
        };

       #coherent = false;
        #};

        #itcn = {
        #nodes = 4;
        #layer = "nch";

        #type = "NUMA";
        #addressMap = "NUMA";

        #routingAlgorithm = {
        #type = "Mesh2DDimensionOrder";
        #dimX = 2;
        #dimY = 2;
```

```
        #};

        #routers = {
        #type = "Simple";
        #latency = 5;
        #};
        };


        mem = {
        controllers = 1;        #4

        type = "Channel";

        channelType = "DDR";

        channelFreq = 1000;
        pageSize = 256;
        pagePolicy = "close";
        deviceIOWidth = 32;
        channelWidth = 32;
        queueDepth = 64;

        banksPerRank = 2;
        ranksPerChannel = 8;

        timing = {
        tCAS = 11;
        tRCD = 11;
        tRP = 11;
        tRAS = 22;
        tWR = 12;
        tCCD = 4;
        tRTP = 6;
        tRRD = 4;
        tWTR = 6;
        tFAW = 20;
        tRTRS = 0;
        tRFC = 117;
        tREFI = 7800;
        };
        power = {
        VDD = 1.2;
        IDD0 = 15.9;
        IDD2N = 7.9;
        IDD2P = 7.9;  # no power-down
        IDD3N = 8.4;  # IDD2N + 0.5 mA
        IDD3P = 8.4;  # no power-down
        IDD4R = 289.4;
        IDD4W = 296.3;
        IDD5 = 32.1;

        channelWirePicoJoulePerBit = 0.6;
        };
        };
};

sim = {
        gmMBytes = 4096;
        deadlockDetection = false;
};

process0 = {
```

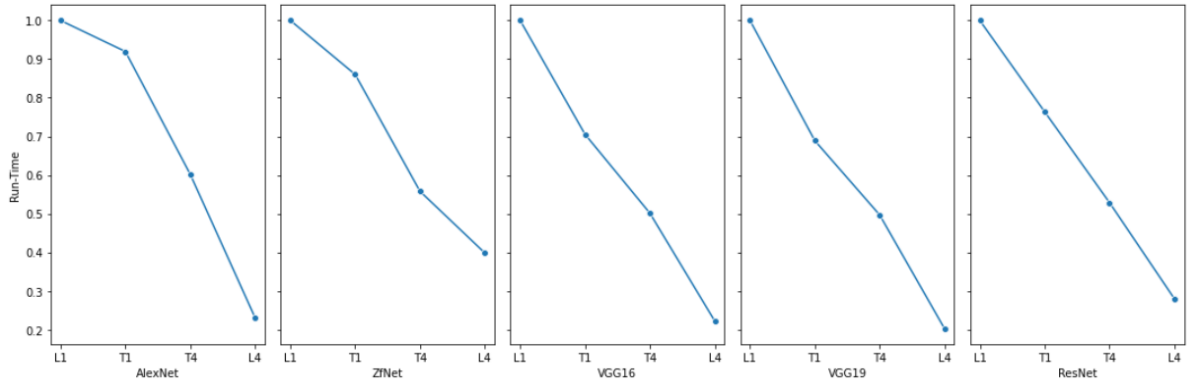
```
        command = "/root/zsim/misc/hooks/test_logic_unit";
        #command = "/root/zsim/misc/hooks/test_logic_unit_numa 4294967296";

        startFastForwarded = true;
        #patchRoot = "/root/zsim/misc/patchRoot/PatchRoot_4"

};
```

Μερικές παρατηρήσεις σχετικά με την επιλογή των χαρακτηριστικών και των αντίστοιχων τιμών τους στα παραπάνω configurations είναι οι εξής:

- Για τα T1-T4: Εκείνα που προφανώς αλλάζουν πηγαίνοντας από 1 node σε 4 είναι αρχικά τα cores του επεξεργαστή και τα logic units του accelerator από 1 σε 4. Αντίστοιχα οι caches για read/write/global buffer αυξάνονται σε 4 και επιπλέον καθώς διπλασιάζονται οι διαστάσεις διπλασιάζεται και το size αυτών. Επιπλέον για την προσομοίωση της λειτουργίας των 4 vaults σε μία δομή TETRIS (2D NoC), προστίθεται στο config file το κομμάτι που αφορά τα interconnects (itcn). Τέλος, στο κομμάτι που αφορά την διεργασία της προσομοίωσης, η εντολή αλλάζει σε test_logic_unit_numa όπως προσδιορίστηκε και παραπάνω στο itcn ενώ προστίθεται και η εντολή για την ενσωμάτωση του κατάλληλου patch root για την προσομοίωση των επιπλέον cores και numa nodes.
- Από το Tx στο Lx: Το layout παρέμεινε ίδιο όπως και τα τεχνικά χαρακτηριστικά της μνήμης καθώς δεν είχαμε στη διάθεση μας τα specifications των LPDDR3 καναλιών. Εκείνο που αλλάζει είναι το κομμάτι που αφορά στα interconnects το οποίο αφαιρέθηκε από configurations των Lx.
- Για τα L1-L4: Ακολουθείται παρόμοια λογική με τις αλλαγές των T1-T4 ωστόσο εδώ δεν έχουμε τετραπλασιασμό των cores διότι δεν έχουμε την παράλληλη επεξεργασία των vaults (συνακόλουθα δεν χρειάζεται και κάποιο patch root). Στη συνέχεια έχουμε την επιπλέον αλλαγή του μεγέθους του global buffer, ο οποίος τετραπλασιάζεται σύμφωνα και με τις οδηγίες του paper από 576kB σε 2.3MB ενώ τέλος, τετραπλασιάζονται και οι memory controllers λόγω του τετραπλασιασμού και των LPDDR3 καναλιών.

ΑΠΟΤΕΛΕΣΜΑΤΑ

Μετά την ολοκλήρωση της διαδικασίας που περιγράφηκε στην προηγούμενη ενότητα, προέκυψαν ορισμένες τιμές για το runtime κάθε αρχιτεκτονικής οι οποίες κανονικοποιήθηκαν ως προς την L1 για κάθε νευρωνικό.
Οπτικοποιώντας τα αποτελέσματα σε γραφικές παραστάσεις με την βοήθεια python script στο Google Colab έχουμε την ακόλουθη εικόνα:



ΣΧΟΛΙΑΣΜΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

Με μία πρώτη ματιά βλέπουμε πως η φθίνουσα συμπεριφορά του διαγράμματος 8 του paper επιβεβαιώνεται. Ωστόσο πρέπει να τονιστεί ότι το T4 είναι μια αρχιτεκτονική που απουσιάζει από το αρχικό διάγραμμα όμως ακολουθώντας μία νοητή γραμμή απο το T1 στο T16 του αρχικού θεωρούμε ότι οι τιμές που έχουν προκύψει είναι θεμιτές.
Μερικές παρατηρήσεις ακόμη είναι οι παρακάτω:

- Όπως και στο αρχικό διάγραμμα, το T1 φαίνεται να αποδίδει καλύτερα για μεγαλύτερου μεγέθους νευρωνικά (VGG, ResNet). Η ίδια παρατήρηση μπορεί να γίνει και για το T4 χωρίς ωστόσο η βελτίωση εδώ να είναι τόσο εμφανής.
- Για μεγαλύτερα NNs οι γραφικές γίνονται πιο ομαλές και συγκεκριμένα των δύο VGG μοιάζουν αρκετά.
- Εμφανείς είναι επίσης οι αστοχίες στο T1 για το AlexNet - όπου περιμέναμε μια κατάτι μικρότερη τιμή - και για το L4 του ZF - όπου περιμέναμε πολύ καλύτερη συμπεριφορά.

Συμπερασματικά, συγκρίνοντας τα αποτελέσματά μας και με αυτά του paper βλέπουμε πως για να πετύχουμε τις επιθυμητές αποδόσεις χρειάζονται σίγουρα παραπάνω από 4 vaults προκειμένου να αξιοποιηθούν στο έπακρο οι δυνατότητες για παράλληλη επεξεργασία και data reuse που προσφέρει το TETRIS τόσο σε hardware όσο και σε software επίπεδο. Επιπλέον, τα διαφορετικά NNs φαίνεται πως επηρεάζουν την απόδοση των αρχιτεκτονικών όπως ήταν αναμενόμενο ενώ και πάλι για να δούμε σημαντική βελτίωση χρειαζόμαστε NNs μεγαλύτερων μεγεθών - όπως άλλωστε είναι και το ζητούμενο: η κλιμάκωση δηλαδή για μεγαλύτερα και πιο περίπλοκα νευρωνικά. Τέλος, τα αποτελέσματα που παρουσιάσαμε δεν είναι τέλεια καθώς όπως είναι προφανές τα configurations δεν είναι απόλυτα ακριβή. Κάτι τέτοιο προϋποθέτει την πρόσβαση σε αρκετά datasheets και εργαλεία καθώς και την λεπτομερή και εκτεταμένη έρευνα σε πολλά πεδία (ενέργεια, δίκτυα διασύνδεσης, αλγόριθμοι κ.α.) και ξεφεύγει από τα πλαίσια της εργασίας μας. Επιπλέον, ο κυρίως simulator, zsim, σε κάποια layers των μεγαλύτερων νευρωνικών δεν παράγει αποτελέσματα (κενά .out αρχεία) και αυτό είναι ένα ακόμα σημείο που παρατηρήσαμε, μας προβλημάτισε και εικάζουμε ότι συνευθύνεται για τις προαναφερθείσες αστοχίες.
Συνοψίζοντας, είναι πλέον πασιφανής η ανάγκη για ανάπτυξη αρχιτεκτονικών όπως του TETRIS που να ανταποκρίνονται στις απαιτήσεις των συνεχώς αυξανόμενων σε μέγεθος και πολυπλοκότητα νευρωνικών δικτύων. Με την εκπόνηση της παρούσας εργασίας αποκτήσαμε γνώσεις και εμπειρία για state-of-the-art ζητήματα όπως το παραπάνω που σχετίζονται άμεσα με τα ερευνητικά μας ενδιαφέροντα και πιθανά θα αξιοποιήσουμε στο μέλλον.

ΑΝΑΦΟΡΕΣ

- https://web.stanford.edu/~mgao12/pubs/tetris.asplos17.pdf
- https://github.com/s5z/zsim
- https://web.stanford.edu/group/mast/cgi-bin/drupal/content/zsim-fast-and-accurate-microarchitectural-simulation-thousand-core-systems
- https://github.com/stanford-mast/nn_dataflow
- https://leepoly.com/2019/08/25/Build-zsim-nvmain/
- https://hub.docker.com/r/leepoly/nvmain
- https://www.micron.com/%20support/tools-and-utilities/power-calc
- Το github μας με τα simulations: https://github.com/aggeliki-dimitriou/zsim-simulations