ΑΤΟΜΙΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εκτέλεση εφαρμογών σε αρχιτεκτονικές PIM

Αναστάσιος Χατζηδημήτρης

ΠΑΝΕΠΙΣΤΙΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΝΕΠΙΣΤΙΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Εκτέλεση εφαρμογών σε αρχιτεκτονικές PIM (Process In Memory)

Αναστάσιος Χατζηδημήτρης

Επιβλέπων καθηγητής

Pedro Trancoso

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Ευχαριστίες

Με την περάτωση της παρούσης εργασίας θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέπων καθηγητή μου κ. Pedro Trancoso για την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα και ακολούθως για την υποστήριξη και καθοδήγηση που μου πρόσφερε όλο αυτό το διάστημα.

Επίσης θέλω να ευχαριστήσω τον διδακτορικό φοιτητή Ανδρέα Διαβαστό για την συνεισφορά του, τον πολύτιμο χρόνο που μου αφιέρωσε και τις συμβουλές του.

Θερμές ευχαριστίες θα πρέπει να δώσω και στην οικογένεια και φίλους για την στήριξη τους όλο αυτό το διάστημα. Συγκεκριμένα θα ήθελα να ευχαριστήσω τους συμφοιτητές και φίλους Πάτροκλο Πατρόκλου, Γιώργο Αντωνίου, Κωνσταντίνο Κυπριανού, Γιώργο Κακουλλή, Κωνσταντίνο Σολομωνίδη, Χρυστάλλα Τσουτσούκη, Ιωάννη Χρίστου και Κασιανή Πάρη.

Περίληψη

Από την στιγμή που κατασκευάστηκε ο πρώτος επεξεργαστής υπολογιστή κυριαρχεί μια σκέψη. Πώς να γίνει πιο γρήγορος, πιο αποδοτικός πιο αξιόπιστος. Για αρκετό καιρό κυριαρχούσε η ιδέα πως αυξάνοντας την συχνότητα του επεξεργαστή, αυτός γίνεται αυτομάτως καλύτερος. Αυτό όμως εδώ και αρκετό καιρό έχει σταματήσει να ισχύει. Μηχανές πολυπύρηνων και πολλαπλά επίπεδα μνήμης έχουν φτάσει να είναι το βασικό στοιχείο μελέτης για την αύξηση της απόδοσης. Όπως όλοι όμως γνωρίζουμε δεν υπάρχει η τέλεια λύση για κάποιο πρόβλημα, υπάρχουν όμως διαφορετικοί τρόποι για να προσεγγίσεις προβλήματα. Προβλήματα όπως η υπερθέρμανση των μηχανών, τα δίκτυα επικοινωνίας στο εσωτερικό του υλικού και προβλήματα όπως το *memory wall* είναι κάποια από τα οποία καθυστερούν την εξέλιξη των επεξεργαστών. Στην παρούσα διπλωματική εργασία θα μελετηθούν αρχιτεκτονικές που προσπαθούν να δώσουν λύση στο πρόβλημα του *memory wall*, ότι αν δηλαδή μια εφαρμογή εξαρτάται από την εξάρτησή της σε είσοδο/έξοδο για την ολοκλήρωση της, η διαφορά ταχύτητας μνήμης-επεξεργαστή δεν θα της επιτρέψει να είναι όσο αποδοτική επιθυμούμε.

Μια προσέγγιση στο πρόβλημα αυτό, είναι οι αρχιτεκτονικές *PIM – process in memory.* Η γενική ιδέα των αρχιτεκτονικών αυτών είναι μέσα στη μνήμη να προστεθούν μονάδες επεξεργασίας. Μεγάλο

πλεονέκτημα της προσέγγισης αυτής είναι το κατά πολύ αυξημένο bandwidth που επιτυγχάνεται μεταξύ μνήμης και μονάδα επεξεργασίας και οι πολύ λιγότερες καθυστερήσεις κατά την είσοδο/έξοδο δεδομένων.

Στη μελέτη αυτή παρουσιάζονται καταρχάς κάποια από αυτά τα μοντέλα οργάνωσης μηχανής που έχουν προταθεί μέχρι τώρα. Στη συνέχεια παρουσιάζονται κάποια εργαλεία τα οποία βοήθησαν στην εξομοίωση των αρχιτεκτονικών αυτών και στην ανάλυση των εφαρμογών οι οποίες επιλέχθηκαν για ανάλυση. Ακολουθεί επεξήγηση και ανάλυση των εφαρμογών αυτών (wordcount, k-means), ανάλυση των αποτελεσμάτων και βάση αυτών καταλήγουμε σε συμπεράσματα. Τέλος υπάρχει ένα κεφάλαιο για μελλοντική εργασία στο οποίο αναφέρονται κάποιες εισηγήσεις για το πώς θα ήταν καλό να συνεχιστεί η μελέτη των αρχιτεκτονικών αυτών.

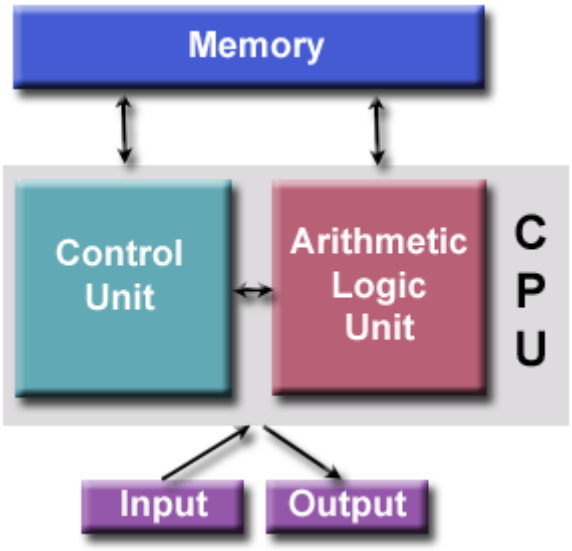
Περιεχόμενα

Περιεχόμενα

1.Εισαγωγή

1.1Κλασικές αρχιτεκτονικές υπολογιστών και αρχιτεκτονικές PIM

Μέχρι σήμερα η βάση για οποιοδήποτε αρχιτεκτονική υπολογιστή θεωρείτε ο κλασικός σχεδιασμός Von Neumann. Ο γνωστός τρόπος οργάνωσης αποτελείται από μονάδα εισόδου, μνήμη, μονάδα επεξεργασίας και μονάδες εξόδου. Όλα αυτά είναι ξεχωριστά κομμάτια που αποτελούν τον κλασικό υπολογιστή που όλοι γνωρίζουμε. Σε αυτό το μοντέλο έχουν χτιστεί και εξελιχθεί για δεκαετίες ακόμα και οι σύγχρονοι επεξεργαστές.



Εικόνα 1: αφαιρετικό σχέδιο αρχιτεκτονικής τύπου Von Neumann

Στο πιο πάνω σχέδιο φαίνεται ακριβώς αυτό που αναφέρθηκε, δηλαδή τα στοιχεία τα οποία αποτελούν την αρχιτεκτονική αυτή και πως αυτά συνδέονται μεταξύ τους.

Στα πλαίσια αυτής της εργασία ξεφεύγουμε αρκετά από αυτόν τον σχεδιασμό. Ο σχεδιασμός της έξυπνης μνήμης έχει αρκετές εκδοχές αλλά όλες έχουν την ίδια βασική ιδέα. Στο ίδιο υλικό όπου βρίσκετε η μνήμη ενσωματώνονται στοιχεία επεξεργασίας και τη θέση του *cpu* παίρνει ένας ή και περισσότεροι *host processors*. Τα *processing elements* αυτά είναι μικρότερης υπολογιστικής ισχύος από τον/τους *host processors* αλλά έχουν άμεση πρόσβαση σε κάποια κομμάτια μνήμης. Ο *host* αναλαμβάνει την ευθύνη να συντονίζει και να δίνει δουλειά στα processing elements, να εκτελεί το σειριακό κομμάτι μιας εφαρμογής καθώς επίσης και την διεκπεραίωση υπολογιστικά περίπλοκων πράξεων οι οποίες δεν μπορούν να γίνουν στην μνήμη.

Εικόνα 2: αφαιρετικός σχεδιασμός embedded DRAM

Στην εικόνα 2 φαίνεται σε αφαιρετικό επίπεδο πως θα είναι ένα chip DRAM σε αρχιτεκτονική Process in Memory.

1.2Κίνητρο

Οι αρχιτεκτονικές της έξυπνης μνήμης είναι μια καινοτόμος ιδέα, κάτι διαφορετικό από τα παραδοσιακά μοντέλα οργάνωσης που έχουμε συνηθίσει. Υπόσχεται πολλά αλλά πρέπει να μελετηθεί για να εξαχθούν σαφή συμπεράσματα για τις δυνατότητες και τους περιορισμούς της.

Στη συγκεκριμένη εργασία θα προσπαθήσουμε να δώσουμε μια αρχική απάντηση στο ερώτημα για το ποιο φάσμα εφαρμογών μπορεί να εκμεταλλευτεί αυτόν τον τρόπο οργάνωσης του υλικού. Τι χαρακτηριστικά δηλαδή πρέπει να έχει μια εφαρμογή για να αυξηθεί η επίδοσή της αν τρέξει σε μία Process in Memory αρχιτεκτονική.

1.3Στόχος και αναμενόμενα αποτελέσματα

Στόχος της εργασίας αυτής είναι να πετύχουμε μια αρκετά συγκλίνουσα εξομοίωση μιας αρχιτεκτονικής αυτής της μορφής με την βοήθεια του εξομοιωτή Marss x86 και σε αυτή να αναλύσουμε την επίδοση εφαρμογών για να μπορέσουμε να καταλήξουμε σε συμπεράσματα σχετικά με τα χαρακτηριστικά των εφαρμογών που «ταιριάζουν» και μπορούν να επωφεληθούν αν εκτελεστούν σε έξυπνη μνήμη.

Με το πέρας των πειραματικών δοκιμών αναμένουμε να έχουμε κάποια πρώτα συμπεράσματα σχετικά με τα χαρακτηριστικά της εφαρμογής που δοκιμάζεται όπως το ποσοστό χρόνου χρήσης της μνήμης, δηλαδή το ποσοστό χρόνου από τον συνολικό χρόνο εκτέλεσής της, και την πολυπλοκότητα του αλγορίθμου που πρέπει να έχει μια εφαρμογή για να μπορούμε να πούμε πως επωφελείται από την άμεση πρόσβαση στην ιεραρχία μνήμης που προσφέρει μια τέτοια αρχιτεκτονική.

Αναμένουμε πως θα επωφελούνται οι Memory-bound εφαρμογές, εφαρμογές που δεν είναι υπολογιστικά περίπλοκες και ο συνδιασμός αυτών των χαρακτηριστικών.

1.4Μεθοδολογία

Για την περάτωση της εργασίας αυτής αρχικά μελετήθηκαν διάφορες προηγούμενες έρευνες και προτάσεις που έγιναν για αρχιτεκτονικές έξυπνης μνήμης (κεφάλαιο 2 – ανάλυση σχετικών εργασιών από βιβλιογραφία) για κατανόηση της γενικής ιδέας της οργάνωσης της μηχανής σε ένα τέτοιο μοντέλο. Επόμενο βήμα ήταν να γίνει η επιλογή για το ποια από αυτές τις αρχιτεκτονικές θα χρησιμοποιηθεί ως βάση για την ανάλυση της επίδωσης των εφαρμογών. Στην εργασία αυτή ως βάση πήραμε την αρχιτεκτονική *FlexRAM*.

Στη συνέχεια έγινε εγκατάσταση του εξομοιωτή και αφού μελετήθηκαν τα *documentation* του και κατανοήθηκαν οι δυνατότητες και οι αδυναμίες του, με μια σειρά από βήματα γράφτηκαν τα *configuration files* τα οποία περιγράφουν 4 και 8 *PArrays* (Processing Elements).

Τέλος έγινε η επιλογή των εφαρμογών και αποφασίστηκε ο τρόπος με τον οποίο θα συλλεγούν και θα συγκριθούν τα αποτελέσματα. Ως μέτρο σύγκρισης της επίδοσης αποφασίστηκε να είναι οι παραδοσιακές Multicore και single core αρχιτεκτονικές.

1.5Περιγραφή Δομής Ατομικής Διπλωματικής Εργασίας

Η εργασία αυτή είναι χωρισμένη σε 8 κεφάλαια στο καθένα από τα οποία περιγράφεται με λεπτομέρεια ένας μέρος του έργου με το οποίο ασχοληθήκαμε για να φέρουμε εις πέρας τους στόχους μας. Το κεφάλαιο 1 είναι εισαγωγικό και στόχο έχει να φέρει τον αναγνώστη σε μια πρώτη επαφή με την γενική ιδέα των αρχιτεκτονικών Process in Memory και να τον βοηθήσει να κατανοήσει τους λόγους για τους οποίους έχουν σχεδιαστεί τα πειράματα που ακολουθούν. Γίνεται μια αναφορά στις παραδοσιακές αρχιτεκτονικές με τις οποίες όλοι είμαστε εξοικειωμένοι και εξηγεί τις βασικές τους διαφορές με τις αρχιτεκτονικές που μελετήθηκαν. Στο κεφάλαιο 2 περιγράφονται οι διάφορες αρχιτεκτονικές που έχουν προταθεί κατά καιρούς για υλοποιήσεις έξυπνης μνήμης και στο τέλος του κεφαλαίου συνοπτικά γίνεται μια σύγκρισή τους. Στο κεφάλαιο 3 παρουσιάζονται όλα τα εργαλεία τα οποία χρησιμοποιήθηκαν για την εξομοίωση μιας τέτοιας μηχανής καθώς και τα εργαλεία που χρησιμοποιήθηκαν για την ανάλυση της εκτέλεσης και επίδοσης των εφαρμογών που επιλέχθηκαν για δοκιμές. Στο κεφάλαιο 4 αναλύεται ο τρόπος με τον οποίο αξιοποιήθηκαν τα εργαλεία του κεφαλαίου 3 και το πώς το κάθε ένα από αυτά βοήθησε στην επίτευξη του τελικού στόχου. Στο επόμενο κεφάλαιο της εργασίας, κεφάλαιο 5 γίνεται λεπτομερής ανάλυση της κάθε μιας από τις εφαρμογές σχετικά με τον τρόπο εκτέλεσής της και την επίδοσή της. Η σύγκριση των αποτελεσμάτων και τα συμπεράσματα που μπορούν να εξαχθούν από αυτά παρουσιάζονται στο κεφάλαιο 6, ενώ στο κεφάλαιο 7 είναι κάποιες προτάσεις και εισηγήσεις για το πώς θα μπορούσε να συνεχιστεί η παρούσας εργασία. Τέλος το κεφάλαιο 8 είναι όλα τα references τα οποία μελετήθηκαν για να μπορέσει η εργασία αυτή να έρθει εις πέρας.

2. Ανάλυση σχετικών εργασιών από την βιβλιογραφία

Στο κεφάλαιο αυτό θα παρουσιάσω τις PIM αρχιτεκτονικές που έχω μελετήσει. Οι αρχιτεκτονικές αυτές έχουν διατεταγμένα μέσα στην μνήμη στοιχεία επεξεργασίας (*processing elements*) κατά γενική ομολογία μικρότερης υπολογιστικής ισχύος από τους κοινούς επεξεργαστές που κυκλοφορούν στην αγορά και μικρότερης υπολογιστικής ισχύος από τον *host processor* που τους συντονίζει. Τα *elements* αυτά λειτουργούν αυτόνομα και έχουν δικές τους λογικές και αριθμητικές μονάδες (ALU). Ακριβώς για το λόγο ότι είναι διατεταγμένα μέσα στη μνήμη κερδίζουν *bandwidth* και ελαχιστοποιούν τις καθυστερήσεις λόγο *input/output*. Οι PIM αρχιτεκτονικές δείχνουν να έχουν σημαντική βελτίωση στην απόδοση σε σχέση με του κοινούς υπολογιστές για εφαρμογές που έχουν μικρό *data locality* (τα δεδομένα στη μνήμη δεν είναι διατεταγμένα σε σειρά, γεγονός που αυξάνει δραματικά τα *cache miss*)στα δεδομένα που χρειάζονται. Ο τρόπος με τον οποίο είναι οργανωμένες αυτές οι μηχανές τις κάνει εύκολα επεκτάσιμες και για τον λόγο αυτό μπορούν μέσα από αυτές τις αρχιτεκτονικές να χτιστούν *massively-parallel systems*. Παράλληλες μηχανές με χιλιάδες ανεξάρτητους PIM κόμβους. Πολύ γενικά θα μπορούσαμε να πούμε πως ο κώδικας χωρίζεται σε ανεξάρτητα μέρη (*threats*), όσα από αυτά χρειάζονται περίπλοκες πράξεις και μεγάλη υπολογιστική δύναμη μένουν για επεξεργασία στο *host processor* και τα υπόλοιπα διανέμονται στα PIM nodes.

Πιο κάτω παρουσιάζονται οι αρχιτεκτονικές αυτές.

2.1 FlexRAM

[2,3] Η πρώτη από τις αρχιτεκτονικές που έχω μελετήσει είναι η *FlexRAM*. Κάθε chip ενός FlexRAM κόμβου περιέχει στον ίδιο χώρο μονάδες απλής μνήμης (DRAM) και μονάδες επεξεργασίας, τα *PArrays (64 από αυτά)*. Κάθε ένα από τα PArrays είναι μια 32-bit fixed-point 2-issue RISC μηχανή που δουλεύει στα 800Mhz. Χρησιμοποιεί 32 καταχωριτές και διασωλήνωση 5 σταδίων. Κάθε 4 *PArrays* μοιράζονται μια μονάδα πολλαπλασιασμού. Κάθε PArray έχει πρόσβαση στη μνήμη των γειτονικών του *PArrsays*. Για κάθε 4 *PArrays* υπάρχει μια κοινή 8-Kbyte *instruction memory.* Σε κάθε κόμβο υπάρχει και ένας *PMem*. *PMem* είναι ένας *low-issue RISC core* που δουλεία του είναι να συντονίζει τα PArrays και να εκτελεί τα σειριακά κομμάτια της εφαρμογής. Ο *PMem* υποστηρίζει και πράξεις *floating point*. Σε ένα σύστημα *FlexRAM* υπεύθυνος για τον συντονισμό των κόμβων είναι ο *PHost* που είναι ο κύριος επεξεργαστής της μηχανής. Ενώνοντας πολλούς κόμβους μαζί μπορούμε να επεκτείνουμε το σύστημα μας όσο θέλουμε και να του δώσουμε όση υπολογιστική δύναμη χρειάζεται.

Το κάθε PArray είναι υπεύθυνο για τα δικά του TBL misses. Σε τέτοια περίπτωση επικοινωνεί με τον host μέσο του PMem και αυτός του δίνει 2 words, στη μια είναι η διεύθυνση της διεργασίας που πρέπει να εκτελέσει και στην άλλη είναι η ένας δείκτης στα arguments που θα χρειαστεί.

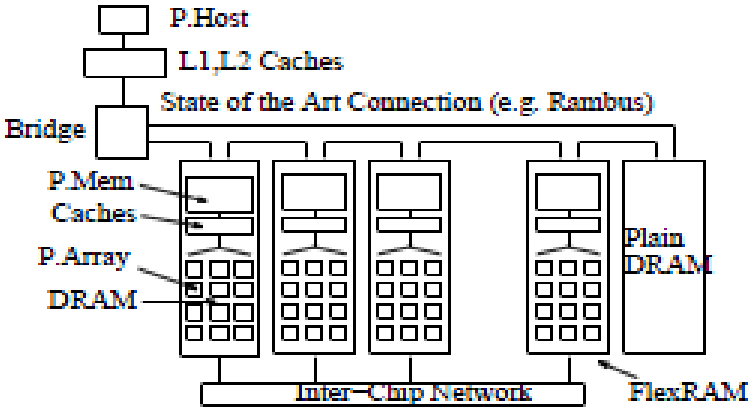


Figure 1: Overall organization of a *FlexRAM*-based memory system.

2.2 DIVA Emulator PIM-Based System

[4] DIVA (Data InensiVe Architecture) είναι ένα παράδειγμα embedded DRAM. Είναι βασισμένο σε τεχνολογίες FPGA και κάθε κόμβος μπορεί να έχει 2 λειτουργίες, *dumb and smart memory*. Η επικοινωνία μεταξύ του κάθε κόμβου γίνεται μέσω ενός ειδικού δικτύου χωρίς έτσι να επιβαρύνεται το δίκτυο μνήμης του συστήματος (*memory bus*). Κάθε κόμβος έχει δικούς του καταχωριτές, αριθμητική λογική μονάδα (*ALU*) και *floating point unit*.

Μοντέλα σαν αυτό (και γενικά PIM) φαίνεται να προσφέρουν μεγάλη βελτίωση σε εφαρμογές με ακανόνιστες προσβάσεις στη μνήμη και pointer-based εφαρμογές.

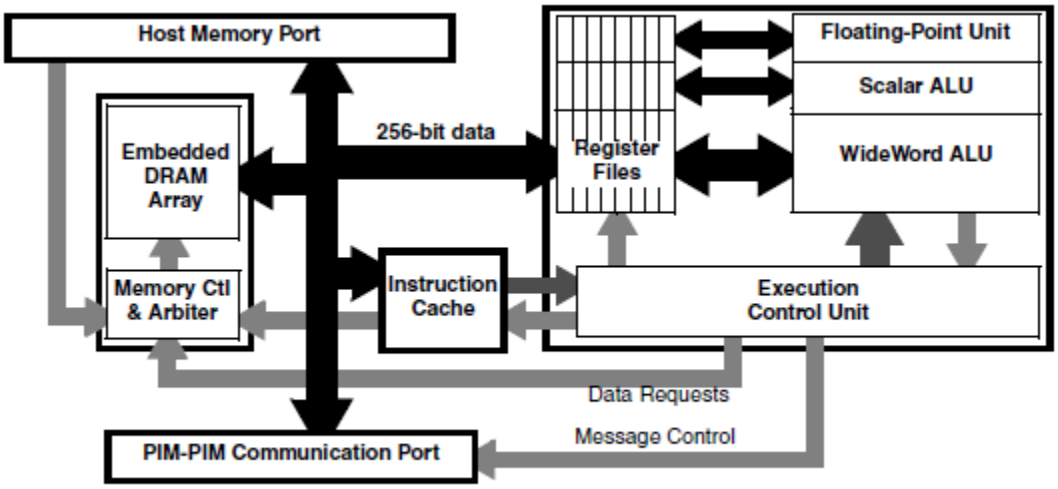


Fig. 1: DIVA PIM Node Organization

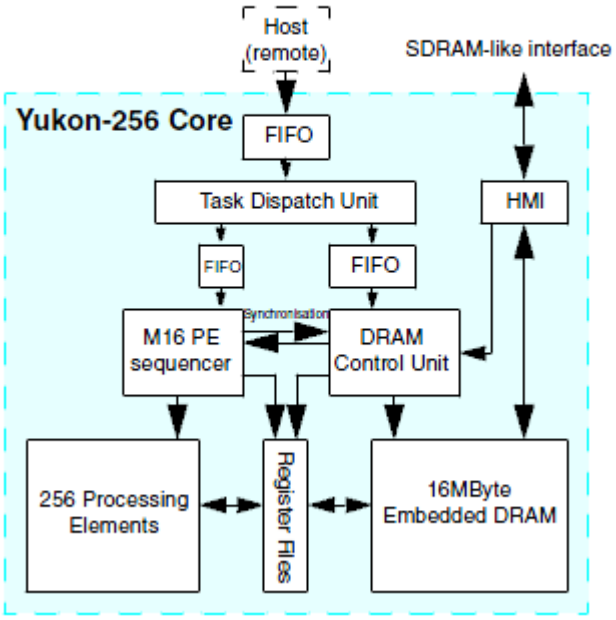
Στο σχέδιο φαίνεται πως οργανώνεται εσωτερικά ένας κόμβος. Πως επικοινωνούν μεταξύ τους οι μονάδες και το ανεξάρτητο από το *memory bus* του συστήματος *PIM-PIM Communication Port*. Η διαχείριση μνήμης στο μοντέλο αυτό γίνεται από τον host processor για το λόγο ότι ο κάθε κόμβος δεν μπορεί να έχει πρόσβαση σε όλη την μνήμη. Για το λόγο αυτό, σε κάθε κόμβο τρέχει ένας μικρός *run-time kernel* που δουλεία του είναι να επικοινωνεί με τον *host* σε περίπτωση κάποιου *page fault*. Ο kernel αυτός είναι ακόμα υπεύθυνος για την διαχείριση των buffers και για το context switching. Σε περίπτωση κάποιου page fault από τον PIM node, γίνεται suspend η διεργασία, ο kernel ενημερώνει τον host για τη νέα σελίδα που χρειάζεται, ο host ενημερώνει την μνήμη του κόμβου και τότε η διεργασία μπορεί να συνεχίσει.

2.3 Active Memory: Micron's Yukon

[6] Η Micron είναι μια εταιρία που ασχολείται κυρίως στη κατασκευή μνήμης. Το project Yukon της Micron είναι ένα chip στο οποίο ενσωματώνεται ένα μεγάλο ποσό DRAM μαζί με προγραμματιζόμενες μονάδες επεξεργασίας. Ο Yukon λειτουργεί σαν επιταχυντής (*accelerator*) της μηχανής η οποία τον φιλοξενεί. Ο host processor δηλαδή διαχειρίζεται το πρόγραμμα σαν κάτι ενιαίο και χρησιμοποιεί το Yukon σαν ένα επιπλέον πόρο του συστήματος όπου χρειάζεται. Υποστηρίζει *SIMD* (*single instruction multiple data*) μοντέλου παραλληλισμού, δηλαδή η ίδια εντολή να εκτελείται πάνω σε πολλά δεδομένα την ίδια στιγμή. Το Yukon έχει ένα πίνακα από *processing elements* που ελέγχονται από τον *host processor*. Όλες οι εργασίες (tasks) που δίνονται στο Yukon για εκτέλεση πρέπει να είναι υπολογιστικά εύκολες. Κάθε

processing element έχει μονάδα πρόσθεσης και πολλαπλασιασμού, 128 bytes αρχείο καταχωριτών, υποστηρίζει *data shifting*, και κάθε element είναι ενωμένο με τα γειτονικά του.

FIGURE 2. High Level View of Control Architecture



2.4 SAGE: A New Analysis and Optimization System for FlexRAM Architecture

[8] Το SAGE (*Statement analysis grouping evaluation*) είναι ένα σύστημα το οποίο μπορεί να χωρίσει τον κώδικα (*statement-based*) και να τον βάλει σε *wavefronts* (groups) για να αξιοποιεί όσο το δυνατό περισσότερο γίνεται τις δυνατότητες μια PIM αρχιτεκτονικής. Η αρχιτεκτονική που πήραν σαν σημείο αναφοράς για το SAGE ήταν η FlexRAM. Ο αλγόριθμός που χρησιμοποιεί χωρίζει τον κώδικα σε blocks (π.χ τα basic blocks του μεταγλωττισμένου κώδικα). Δίνει στη συνέχεια συγκεκριμένα «βάρη» στο κάθε ένα από αυτά τα blocks και καθορίζει την σειρά εκτέλεσής τους ανάλογα με το αν υπάρχουν αλληλεξαρτήσεις ή όχι. Τα blocks τα οποία είναι ανεξάρτητα και μπορούν να εκτελεστούν παράλληλα ξαναχωρίζοντας σε *wavefronts*. Αφού σε αυτό το στάδιο όλος ο κώδικας είναι χωρισμένος σε wavefronts γίνονται κάποιες τελευταίες βελτιστοποιήσεις και μοιράζονται ανάλογα με το βάρος τους και την υπολογιστική δύναμη που χρειάζεται το καθένα στον PHost, PMem ή PArray.

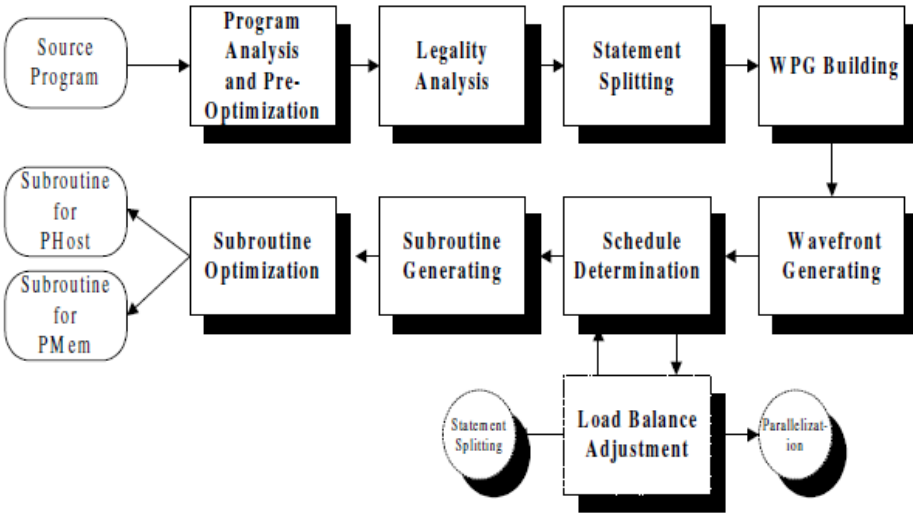


Fig. 2. Analysis and optimization stages of SAGE system.

2.5 Smart Memories

[9] Κάθε *smart memory chip* αποτελείται από πολλά *processing tiles* που το κάθε ένα από αυτά έχει δική του τοπική μνήμη, δίκτυο επικοινωνίας και μονάδα επεξεργασίας. Τα *tiles* αυτά είναι σχεδιασμένα για μπορούν να προγραμματίζονται με τρόπο ώστε να εξυπηρετούν ένα μεγάλο φάσμα εφαρμογών. Η μόνα επεξεργασίας που έχουν τα *tiles* είναι μια μηχανή *64-bit in-order 2 issue*. Ακόμα έχουν 2 *integer* και 1 *floating point cluster*, μονάδα για *load/store*, αριθμητική και λογική μονάδα (ALU) και αρχείο καταχωρητών. Τα tiles στο chip είναι οργανωμένα σε τετράδες. Με αυτό τον τρόπο αν για κάποια λειτουργία χρειάζεται μεγαλύτερη υπολογιστική δύναμή από όση προσφέρει ένα tile από μόνο του μπορούν να συνεργαστούν με πάρα πολύ μικρό κόστος επικοινωνίας. Γενικά η επικοινωνίας στο chip είναι packet-based και η δρομολόγηση γίνεται δυναμικά.

Για να δείξουν πως η συγκεκριμένη αρχιτεκτονική είναι ευέλικτη και εξυπηρετεί πολλές εφαρμογές, προσομοίωσαν (mapped) πάνω σε smart memory δύο εντελώς διαφορετικά μοντέλα μηχανών. Την *Imagine* , που είναι SIMD/vector μηχανή και το *Hydra multiprocessor*, με μικρή μείωση στην επίδοσή τους.

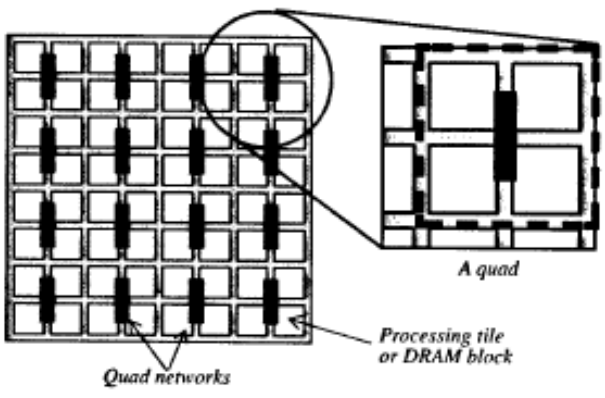


Figure 1. A Smart Memories chip

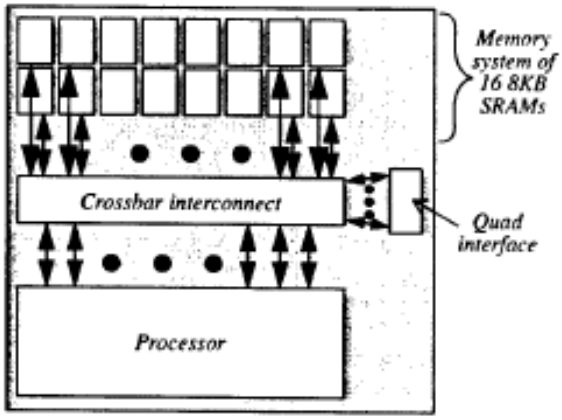


Figure 2. Tile floorplan

2.6 In-memory Parallelism for Database Workloads

[10] Στη συγκεκριμένη δουλεία εξετάζεται κατά πόσο οι αρχιτεκτονικές PIM μπορούν να βελτιώσουν την επίδοση σε εφαρμογές που έχουν να κάνουν με Βάσης Δεδομένων. Η αρχιτεκτονική που δοκιμάστηκε ονομάστηκε *PIM-n*. Κάθε ένα από τα *n blocks* της έχει ένα κομμάτι *PIM-M*, που είναι το κομμάτι μνήμης του *processing element* και ένα *PIM*-L που είναι η λογική μονάδα. Για να κρατηθεί όσο πιο απλός ο σχεδιασμός του *chip*, κάθε *block* έχει πρόσβαση στη δική του μνήμη και στα γειτονικά του *block* μόνο σε αντίθεση με τον *Host CPU* που μπορεί να έχει πρόσβαση σε όποια θέση μνήμης θέλει. Για την συνοχή στη μνήμη (*coherence*) αποκλειστική εύθηνη έχει ο προγραμματιστής και αυτό γιατί δεν υπάρχει καθόλου *hardware* για το σκοπό αυτό. Για τους σκοπούς της έρευνας χρησιμοποιήθηκε ένα μοντέλο PIM-32 όπου ο Host CPU ήταν μια *6-Issue out-of-order* μηχανή στα 1 Ghz και οι PIM-L μονάδες *in-order 2-issue* στα 500Mhz. Δοκιμάστηκαν 3 διαφορετικοί τύποι εφαρμογών, σειριακή αναζήτηση, hash joint or sort και index scan με μέση επιτάχυνση 43x.

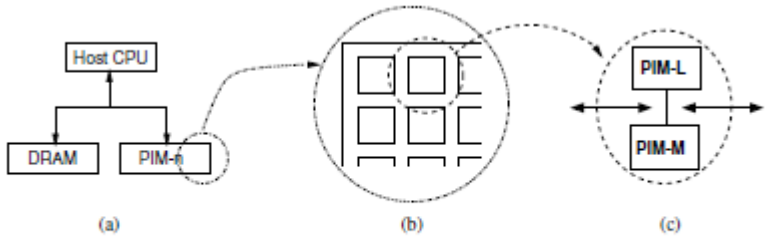


Fig. 1. (a) System configured with one PIM-n chip and regular DRAM; (b) Section of PIM-n chip; (c) Detail of PIM-n block.

2.7 Active Pages

[11] Το *Active Pages* είναι ένα μοντέλο επεξεργασίας το οποίο χωρίζει την εφαρμογή στον επεξεργαστή και στο σύστημα της έξυπνης μνήμης. Στόχος του είναι να κρατήσει τον επεξεργαστή να δουλεύει συνέχεια σε υψηλά επίπεδα απελευθερώνοντας τον από την εύθηνη του *data manipulation*. Αποτελείται ουσιαστικά από μια σελίδα με δεδομένα και ένα σετ από λειτουργίες οι οποίες εκτελούνται πάνω στα στοιχεία αυτής της σελίδας. Τα *processor-centric* κομμάτια (*partitions*) είναι αυτά που θα εκτελεστούν στον επεξεργαστή και περιέχουν όλους τους περίπλοκους αλγόριθμους και υπολογισμούς με *floating points*. Τα *memory-centric* κομμάτια έχουν να κάνουν με διαχείριση των δεδομένων και πράξεις *integers*. Ένα σύστημα *Active Pages* χτίζεται πάνω σε *RADRam* που είναι βασισμένα σε *FPGAs*. Εφαρμογές που

μπορούν να επωφεληθούν πολύ από αυτή την υλοποίηση είναι queries σε βάσης γιατί θεωρητικά μπορούμε να ελέγξουμε όλα τα records της βάσης ταυτόχρονα, εφαρμογές για επεξεργασία εικόνας και εφαρμογές τύπου sparse-matrix multiply. Στη τρίτη περίπτωση το σύστημα Active Pages χρησιμοποιείται για να φέρνει τα δεδομένα, να μαζεύει μόνο τα «χρήσιμα» σε blocks ανάλογα με το μέγεθος της *cache* και μόνο αυτά να φτάνουν στον επεξεργαστή.

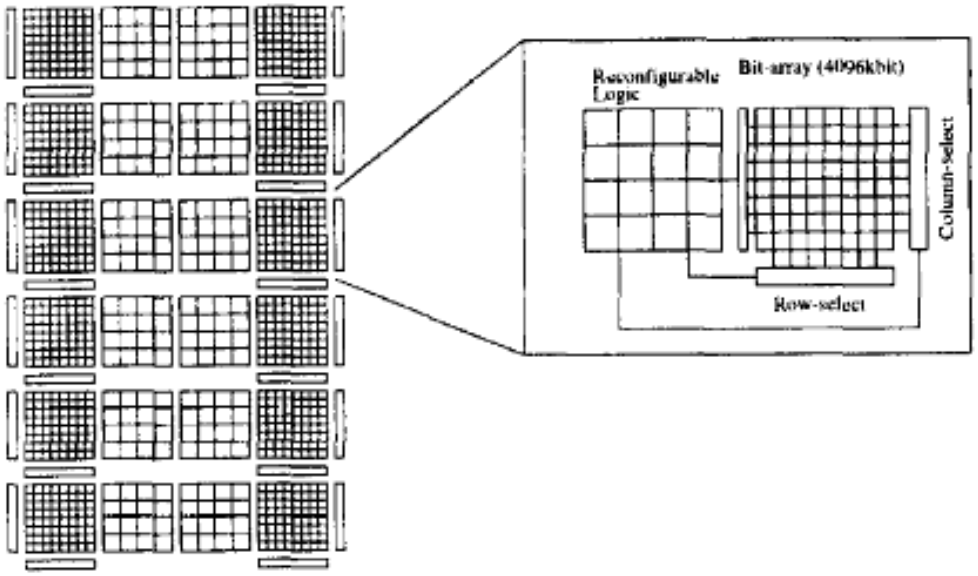


Figure 2: The RADram System

2.8 Memory Arithmetic Unit and Interface (MAUI)

[12] Η αρχιτεκτονική MAUI συνδυάζει ιδέες από προηγούμενες, παρόμοιου τύπου αρχιτεκτονικές που έχουν προταθεί όπως η Active Pages, η DIVA, και η ULMT. Στη MAUI, όλο το "intelligent" βρίσκεται στο memory controller. Σε αντίθεση με άλλες PIM αρχιτεκτονικές η συγκεκριμένη κρατά την μνήμη (DRAM) όπως είναι χωρίς καμία αλλαγή. Γενικά είναι μια μηχανή που εκτελεί SIMD εντολές σε vectors cache-line μεγέθους. Σε μια αρχιτεκτονική MAUI, όλα τα data flow περνούν μέσα από το hardware του MAU. Και πάλι όπως και στις άλλες PIM

αρχιτεκτονικές, ο host processor έχει τον απόλυτο έλεγχο και είναι αυτός που δίνει δουλειά στο MAUI. Για το λόγο αυτό, το ISA του επεξεργαστεί πρέπει να εμπλουτιστεί με τις εντολές που θα καθοδηγούν την εκτέλεση στη μονάδα αυτή. Μια μηχανή MAUI έχει δύο τύπους εντολών. Τις *setup* και τις *execution*. Οι *setup* είναι αυτές που θα καθορίσουν την διεύθυνση της πηγής των δεδομένων, την διεύθυνση-προορισμό και το μέγεθος των δεδομένων. Οι *execution* είναι οι εντολές που κάνουν πρόσθεση, πολλαπλασιασμό και vector scaling. Παρόλο που κάποιες εντολές στο MAUI μπορεί να πάρουν κάποιο χρόνο να εκτελεστούν, το υλικό είναι σχεδιασμένο με τρόπο ώστε το πρόγραμμα να «νομίζει» πως εκτελούνται στιγμιαία. Σε γενικές γραμμές όταν ο επεξεργαστής δώσει στη MAUI εντολές για εκτέλεση, οι ενέργειες που γίνονται είναι:

1. Στέλνει αίτημα στη μνήμη για ανάγνωση
2. Τα δεδομένα φορτώνονται στη MAUI cache
3. Γίνονται οι πράξεις
4. Στέλνει αίτημα στη μνήμη για γράψιμο

Για να διατηρηθεί η λογική σειρά των δεδομένων στη μνήμη, γίνεται χρήση *read and write locks*.

Κάποια προβλήματα που μπορούν να δημιουργηθούν από αυτή την αρχιτεκτονική είναι η συνοχή της cache. Πρόβλημα που όπως είδαμε και στη Active Pages, λύνεται σε επίπεδο software. Ο προγραμματιστής δηλαδή είναι υπεύθυνος για την συνοχή της μνήμης. Βασικό όμως για την MAUI είναι πως οι διευθύνσεις που παίρνει σαν είσοδο (*source* και *destination*) πρέπει να είναι φυσικές διευθύνσεις και όχι virtual addresses.

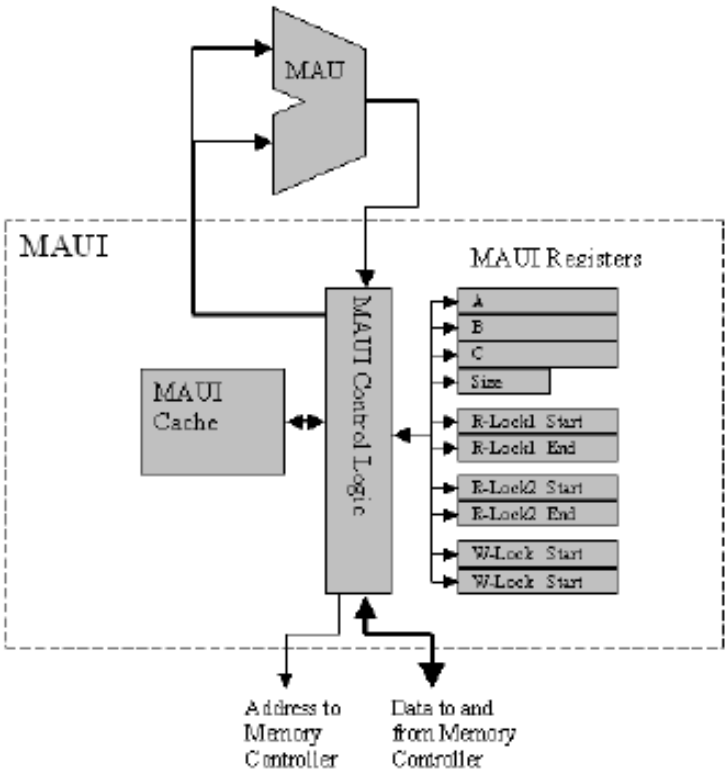
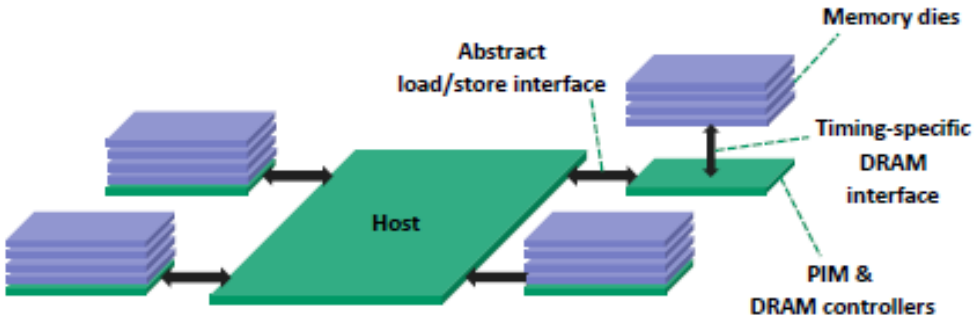


Figure 2: The block diagram of the MAUI architecture.

2.9 TOP-PIM: Throughput-Oriented Programmable Processing in Memory

[13] Η συγκεκριμένη προσέγγιση είναι ένας συνδυασμός του *memory 3D stacking*, *2.5D stacking* και *HMC (hybrid memory cube)* μαζί με την ιδέα του PIM. Το βασικό πρόβλημα με το *3D stacking* είναι πως αν βάλουμε μνήμη πάνω από ένα πολύ ισχυρό επεξεργαστή τότε η μνήμη θερμαίνεται πάρα πολύ και χρειάζεται πιο συχνά refresh, κάτι που κοστίζει σε χρόνο. Στη συγκεκριμένη μελέτη βάζουν την μνήμη πάνω από PIM nodes, και τοποθετούν αυτά τα blocks «κοντά» σε ένα ισχυρό host processor, ο οποίος χειρίζεται όλες τις υπολογιστικά περίπλοκες πράξεις. Η βασική διαφορά από τις προηγούμενες PIM αρχιτεκτονικές που αναφέρονται πιο πάνω, είναι πως τώρα στα PIM nodes, τα processing elements είναι GPUs και CPUs. Αυτό έχει σαν πλεονέκτημα πως το γενικό μοντέλο με CPU και GPU δίνει στους προγραμματιστές την δυνατότητα να το προγραμματίσουν εύκολα αφού είμαστε πλέον εξοικειωμένοι με αυτά τα υπολογιστικά μοντέλα. Επιπλέων αφού χρησιμοποιούμε και GPU και CPU, το φάσμα των εφαρμογών που μπορούν να αξιοποιήσουν αυτή την αρχιτεκτονική μεγαλώνει ακόμη περισσότερο. Επίσης για την υλοποίηση αυτής της ιδέας μπορούν να χρησιμοποιηθούν CPUs και GPUs που είδη υπάρχουν στην αγορά μειώνοντας έτσι το κόστος της υλοποίησης.

Η βασική πρόκληση του προγραμματιστή για αυτή τη μηχανή είναι να διανέμει τα δεδομένα σε πολλά *memory stacks*, κρατώντας παράλληλα το *locality* στους υπολογισμούς του κάθε PIM κόμβου. Η συγκεκριμένη αρχιτεκτονική δοκιμάστηκε σε εφαρμογές για γράφους και εφαρμογές για *high performance computers*.



2.10 Micron Automata Processor

[15, 16] Τα AP της Micron είναι μια pre-programmable μηχανή που βασικός της στόχος είναι να διενεργεί ολοκληρωμένες αναζητήσεις πάνω σε ροές δεδομένων (*unstructured data streams*). Μια βασική διαφορά των AP από τις υπόλοιπες αρχιτεκτονικές είναι πως η μονάδα επεξεργασίας (σε αυτή την περίπτωση ένα από τα *active elements* ,ένα αυτόματο δηλαδή) σαν είσοδο δεν θα πάρει μια διεύθυνση μνήμης αλλά ένα σύμβολο από την ροή δεδομένων (π.χ. ένα χαρακτήρα). Σε αντίθεση με τους γνωστούς επεξεργαστές τα AP είναι εξαιρετικά επεκτάσιμα από μερικές εκατοντάδες μέχρι χιλιάδες ή και εκατομμύρια ανάλογα με το μέγεθος του προβλήματος. Τα AP αποτελούνται από 3 βασικά στοιχεία (*active elements*) και τον συνδυασμό αυτών.

1. State Transition Element (STE). Στην ουσία το STE είναι αυτό που θα αποδεχτεί ή θα απορρίψει τον χαρακτήρα που δίνεται σαν είσοδος (η είσοδος στα AP είναι απλά δεδομένα) . Αν αποδεχτεί τον χαρακτήρα τότε το συγκεκριμένο αυτόματο προχώρα στην επόμενη κατάσταση και περιμένει τον επόμενο.
2. Counter Element (CNTR). Το CNTR είναι ένας μετρητής ο οποίος μπορεί να μετρήσει τιμές μέχρι 48-bits. Στα CNTR υπάρχει επιλογή για reset.
3. Boolean Element (Bool). Μπορεί να δώσει σαν αποτέλεσμα όλους τους πιθανούς συνδυασμούς λογικών τελεστών ή να σημαδέψει το «*end-of-dada*» στην έξοδο.

Κάθε chip AP έχει 49152 STE, 768 CNTR, και 2034 BOOl elements. Τα στοιχεία αυτά (elements) οργανώνονται σε σειρές μέσα στο chip, και οι σειρές σε blocks. Το κάθε ένα από αυτά τα στοιχεία μπορεί να ενωθεί με οποιοδήποτε από τα στοιχεία του block του ή με οποιοδήποτε από τα στοιχεία των 8 γειτονικών του blocks. Μπορεί ακόμα κάποιο element να είναι ενωμένο με περισσότερα από ένα στοιχεία ταυτόχρονα. Αυτό σημαίνει πως ένα αυτόματο μπορεί την ίδια στιγμή να είναι ενωμένο και να δίνει το output του σαν input σε 2304 άλλα αυτόματα συμπεριλαμβανομένου και του εαυτού του.

Αν οργανώσουμε τα blocks σε ranks και κάνουμε ένα σχεδιασμό με πολλαπλά ranks θεωρητικά μπορούμε να επεκτείνουμε την μηχανή όσο θέλουμε.

Εφαρμογές που μπορούν να επωφεληθούν από τέτοιου είδους καινοτομία, όπως υποστηρίζει η Micron είναι:

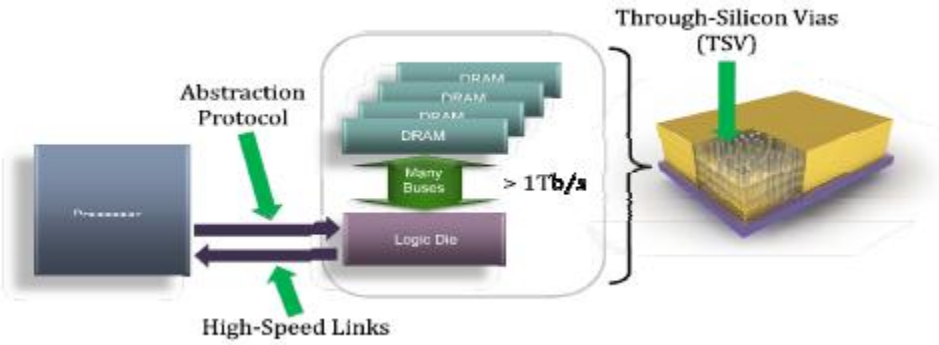
- *Bioinformatics*: τα νέα συστήματα γενετικών ακολουθιών δημιουργούν ένα μεγάλο όγκο από δεδομένα τα οποία πρέπει να τύχουν ανάλυσης. Τα AP δίνουν

τη λύση για υψηλής απόδοσης και ακρίβειας ανάλυσης των δεδομένων αυτών με χαμηλότερο κόστος.

- Network Security: Με την ταχύτητα στην ανάλυση που προσφέρουν τα AP, μας δίνεται η ευκαιρία να έχουμε το πλεονέκτημα απέναντι στους επιτιθέμενους γιατί μπορούμε να κάνουμε πιο αποτελεσματικό έλεγχο των πακέτων στις όλο και αυξανόμενες ταχύτητες των σημερινών και μελλοντικών δικτύων.
- *Signal Intelligence and Cryptography*: δίνει την δυνατότητα για πολύ πιο γρήγορες κρυπτοαναλύσης κάποιου κρυπτογραφημένου μηνύματος μπορώντας έτσι να αξιολογήσουμε κάποιο σύστημα κρυπτογράφησης αν είναι καλό ή όχι
- *Computational Finance*: τα AP δίνουν τη δυνατότητα για real-time ανάλυση στατιστικών για πράξεις που αφορούν συναλλαγές νομισμάτων και διακύμανση τιμών σε μετοχές. Μπορεί ακόμα να συνδυάσει πληροφορίες που βρίσκουμε σε περιβάλλοντα κοινωνικής δικτύωσης και έκτακτες ενημερώσεις (με κάποιο είδος data mining) για βοήθεια σε λήψη αποφάσεων.

2.11 Hybrid Memory Cube. New DRAM Architecture Increases Density and Performance

[18]Το HMC (Hybrid Memory Cube) είναι ένα παράδειγμα *3D memory stacking*. Μια στοίβα από ετερογενή die που στο πρώτο layer υπάρχει ένα logic die το οποίο εξειδικεύεται σε συγκεκριμένες εφαρμογές. Κάθε 1GB μνήμης, ένα υπόστρωμα δηλαδή, είναι βελτιστοποιημένο για υψηλό Bandwidth και συγχρονισμό (concurrency). Όλη η «λογική» της DRAM βρίσκεται στο logic layer. Αυτό είναι υπεύθυνο για τις ανανεώσεις της μνήμης, τον εντοπισμό λαθών και έχει διασύνδεση υψηλής ταχύτητας με τον host processor.



2.12 Σύγκριση Αρχιτεκτονικών

Κοιτάζοντας συγκριτικά τις αρχιτεκτονικές αυτές παρατηρούμε πως δεν διαφέρουν κατά πολύ. Τα βασικά τους χαρακτηριστικά είναι όμοια διαφέροντας μόνο λίγο στον τρόπο οργάνωσης. Στον πιο κάτω πίνακα φαίνονται τα χαρακτηριστικά των PIM nodes όπως αυτοί παρουσιάστηκαν.

ΑΡΧΙΤΕΚΤΟΝΙΚΗ	PIM node	Χρήση	Εφαρμογές
FlexRAM	32-bit fixed-point 2-issue RISC engine at 800MHz 5-stage pipeline 32-registers 8 Kbyte cache Shared multiplier 1 MByte DRAM Shared 8 Kbyte instruction memory	Αντικατάσταση των chip της κανονικής DRAM με chip FlexRAM	Vector map (with 1 or 2 vectors) Vector reduce Vector search
DIVA	Based on FPGAs Floating Point Unit WideWord Unit Private Registers	Κάθε DIVA node (ένα FPGA δηλαδη) σχετίζεται με μια DRAM και SRAM`	large-scale application exploiting processing with the wide datapaths
Micron’s Yukon	Εκτελεί μόνο απλές πράξεις. Τα 256 Processing elements του μοιράζονται 16 MByte DRAM	Συνυπάρχει μαζί με το σύστημα και ο κύριος επεξεργαστής τον χρησιμοποιεί σαν accelerator	υποστηρίζει SIMD μοντέλο παραλληλισμού
Smart Memories	64bit reconfigurable instruction format/decode 2 integer and 1 floating point cluster	Συνύπαρξη DRAM blocks και Processing tiles στο ίδιο chip	Στις δοκιμές έκαναν map πάνω σε αυτή την αρχιτεκτονική άλλες αρχιτεκτονικές

			με διαφορετικό σκοπό η κάθε μια (Imagine και Hydra)
Active Pages	Based on FPGAs RADRam (Reconfigurable Architecture DRAM) Data manipulation and Integer arithmetic	Αποτελείται από μια σελίδα με τα data sets και ένα σετ με τις πράξεις που θα εκτελέσει πάνω σε αυτά	Database queries (unindexed) Image processing Sparse matrix multiplication Διαλέγει τα «χρήσιμα» δεδομένα και τα στέλνει στον κύριο επεξεργαστή
MAUI	Integer arithmetic on cache-size vectors	Μεταξύ του επεξεργαστή και της μνήμης. Η DRAM παραμένει όπως είναι και ο επεξεργαστής στέλνει κάποιες δουλειές να γίνουν στη MAUI	SIMD εντολές σε vectors Πρόσθεση Πολλαπλασιασμό Vector scaling Διαχείριση δεδομένων στη μνήμη (π.χ. αντιγραφή)
Micron’s AP	Χιλιάδες αυτόματα που λειτουργούν αυτόνομα και είναι συνδεδεμένα μεταξύ τους. Station Transition Element Counter Element Boolean Element	Μπορεί να χρησιμοποιηθεί σαν ένα επιπλέον rescore στο σύστημα για pattern recognition	Big Data Bionformatics Network Security Signing and Crypto Finance

Παρατηρώντας τον πιο πάνω πίνακα μπορούμε να δούμε συνοπτικά τις διάφορες αρχιτεκτονικές που μελετήθηκαν. Αν και η κάθε μια από αυτές έχει τα δικά της χαρακτηριστικά οι εφαρμογές με τις οποίες δοκιμάστηκαν δεν διαφέρουν πολύ. Μπορούμε λοιπόν σαν ένα πρώτο συμπέρασμα να πούμε πως για να έχουμε μεγάλο κέρδος σε επίδοση από τις πιο πάνω αρχιτεκτονικές η εφαρμογή πρέπει να έχει πολλές απαιτήσεις από την μνήμη, και ακόμη καλύτερα εάν αναζητά σειριακά την μνήμη (πίνακες, vectors). Αυτό διαφοροποιείτε κάπως στην περίπτωση των AP της Micron, και αυτό γιατί τα αυτόματα αυτά δεν χρησιμοποιούν την μνήμη αλλά παίρνουν σαν είσοδο ένα data stream, μια σειρά από χαρακτήρες.

3.Micron's Automata Processor

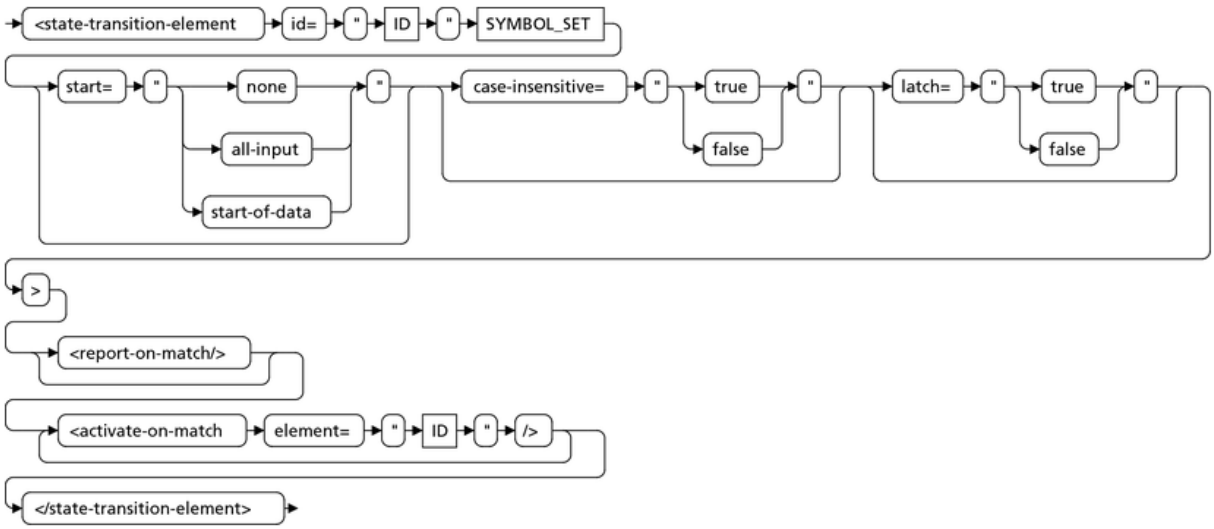
Τα Automata Processor είναι μια εντελώς νέα αρχιτεκτονική για επιτάχυνση στην επεξεργασία κανονικών εκφράσεων (regular expression accelerator). Κάτι που βρίσκει πολλές εφαρμογές σε αναλύσεις στατιστικών και λογικές πράξεις. Τα AP μπορούν εύκολα να επεκταθούν από δεκάδες μέχρι εκατομμύρια αυτόματα ανάλογα με το μέγεθος του κάθε προβλήματος. Χαρακτηριστικά θα πρέπει να αναφέρουμε πως εκτός από επίδοση, τα AP έχουν πολύ καλύτερη κατανάλωση ενέργειας από τα συμβατικά CPUs και GPUs.

Η κύρια πρόκληση και το κίνητρο της Micron για τα Automata Processor είναι πως τα περισσότερα από τα πιο υπολογιστικά δύσκολα και περίπλοκα προβλήματα στις μέρες μας απαιτούν έλεγχο και ανάλυση μη δομημένης ροής δεδομένων (unstructured data stream) σε επίπεδα petabyte. Αυτήν την ομάδα προβλημάτων δεν μπορούν να την χειριστούν οι παραδοσιακοί επεξεργαστές και οι παραδοσιακές αρχιτεκτονικές και για αυτό χρειάζεται μια εντελώς νέα προσέγγιση του προβλήματος.

Τα Automata processor είναι χτισμένα σύμφωνα με τους ορισμούς και την λογική τα μη ντετερμινιστικά πεπερασμένα αυτόματα (Non-deterministic finite automata). Αποτελούνται από 3 βασικά στοιχεία. Το State Transition Element (STE), Counter Element (CNTR), και το Boolean Element (Bool). Αυτά τα 3 στοιχεία μπορούν να συνδυαστούν μεταξύ τους για να δημιουργήσουν αυτόματα που εκφράζουν σχεδόν οποιαδήποτε κανονική έκφραση.

3.1State Transition Element (STE)

Το STE είναι ο πόρος (resource) στον οποίο φυλάγεται η κατάσταση ενός αυτόματου. Είναι επίσης το στοιχείο αυτό που θα αναγνωρίσει ένα σύμβολο εισόδου και είναι το μοναδικό από τα στοιχεία που δέχεται σαν είσοδο σύμβολα.

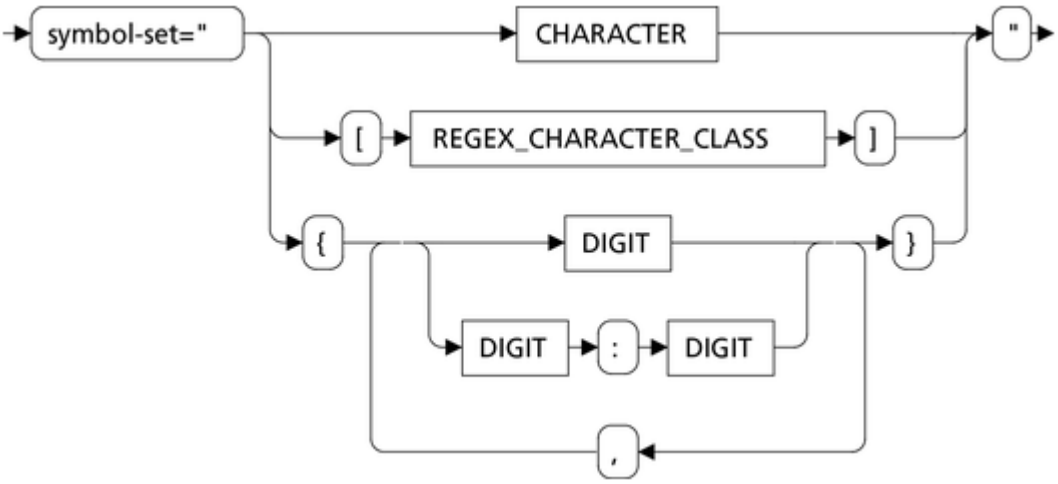


Στο πιο πάνω σχήμα φαίνεται ο τρόπος με τον οποίο ορίζεται ένα state transition element. Υπάρχουν τα υποχρεωτικά attributes που είναι ID και SYMBOL_SET και τα προαιρετικά όπως start, case insensitive, latch, καθώς επίσης και τα προαιρετικές ενέργειες κατά το output, report-on-match και activate-on-match.

3.1.1 STE attributes

ID: ένα μοναδικό αναγνωριστικό για το στοιχείο μέσω του οποίου θα μπορεί να κάνει αναφορά σε αυτό. Πρέπει να ξεκινά με χαρακτήρα και στη συνέχεια μπορεί να περιέχει χαρακτήρες, αριθμούς ή το σύμβολο της υπογράμμισης (underscore).

SYMBOL_SET: το σύνολο συμβόλων που θα αναγνωρίζει (match) το συγκεκριμένο στοιχείο. Όπως φαίνεται και στο πιο κάτω σχήμα, μπορεί να είναι είτε απλός χαρακτήρας, είτε αριθμός είτε κανονική έκφραση.



Start: το χαρακτηριστικό αυτό καθορίζει πότε θα ενεργοποιείται το συγκεκριμένο στοιχείο. Μπορεί να είναι είτε στην αρχή του input (μόνο στον πρώτο χαρακτήρα δηλαδή), είτε σε κάθε χαρακτήρα εσόδου, είτε μόνο όταν ενεργοποιηθεί από κάποιο άλλο στοιχείο.

Case-insensitive: καθορίζει αν θα λαμβάνεται υπόψη το case του χαρακτήρα

Latch: αν είναι ενεργοποιημένο το συγκεκριμένο χαρακτηριστικό, το στοιχείο από την πρώτη φορά που θα ενεργοποιηθεί μέχρι το τέλος του της ροής εισόδου, για κάθε κύκλο θα στέλνει match signal.

Report on match: καθορίζει ότι το στοιχείο είναι τερματικό για το αυτόματο

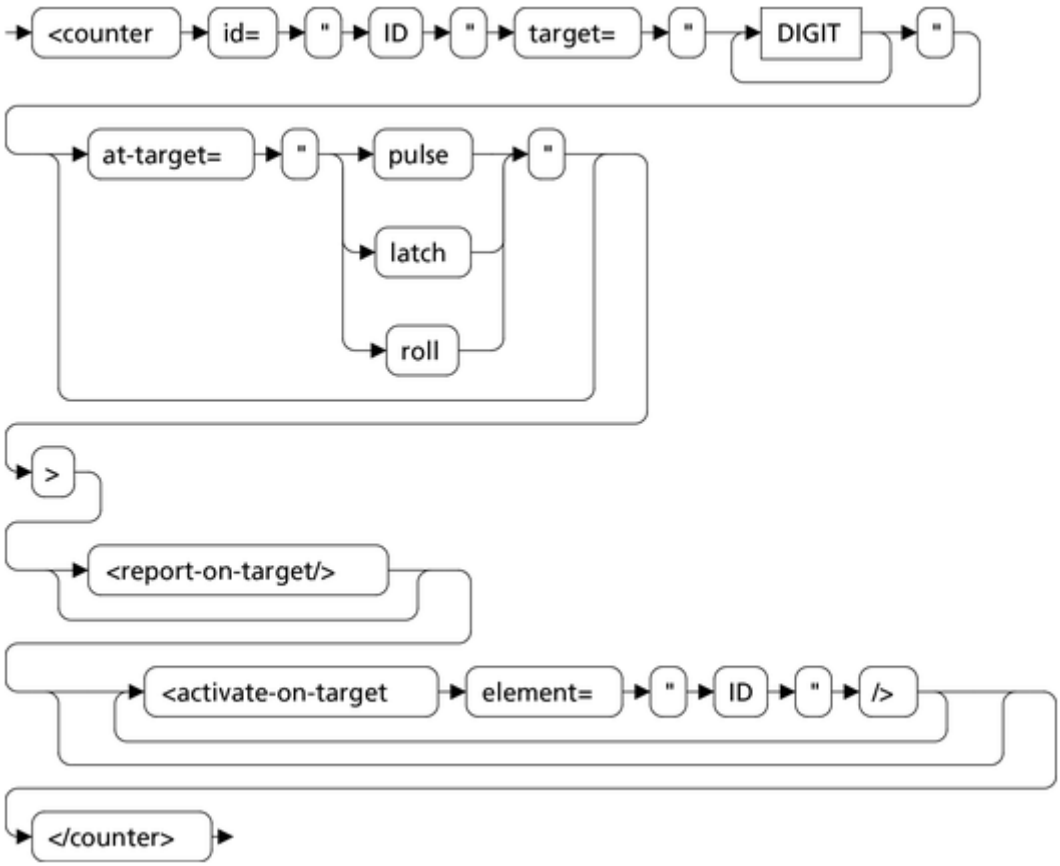
Active on match: καθορίζει ποιο στοιχείο ή σύνολο στοιχείων θα ενεργοποιηθούν αν ενεργοποιηθεί το συγκεκριμένο STE.

Πιο κάτω φαίνεται ένα παράδειγμα ορισμού STE:

```
<state-transition-element id="a" symbol-set="[a-z]" start="all-input">
    <report-on-match/>
</state-transition-element>
```

3.2 CNTR Element

Είναι ένα ειδικό στοιχείο στα αυτόματα στο οποίο καθορίζεται ένας αριθμός-στόχος και ενεργοποιείται όταν φτάσει σε αυτόν. Ποιο κάτω φαίνεται το πώς περιγράφετε ένα CNTR element.



3.2.1 CNTR attributes

Το CNTR element έχει κάποια υποχρεωτικά και κάποια προαιρετικά χαρακτηριστικά. Υποχρεωτικά είναι το ID και το target και προαιρετικά τα at_target, report-on-target και active-on-target.

ID: ένα μοναδικό αναγνωριστικό για το στοιχείο μέσω του οποίου θα μπορεί να κάνει αναφορά σε αυτό. Πρέπει να ξεκινά με χαρακτήρα και στη συνέχεια μπορεί να περιέχει χαρακτήρες, αριθμούς ή το σύμβολο της υπογράμμισης (underscore).

Target: ένας ακέραιος αριθμός που καθορίζει το στόχο του μετρητή. Όταν φτάσει ο μετρητής στον αριθμό αυτό γίνεται η ενέργεια που καθορίζεται στο at_target.

At_target: όταν φτάσει ο μετρητής στο στόχο μπορεί να κάνει 1 από τις 3 ενέργειες. Είτε latch, δηλαδή να συνεχίζει για κάθε επόμενο κύκλο να στέλνει σήμα, είτε pulse, δηλαδή ο μετρητής στέλνει το σήμα και παραμένει ανενεργός (σταματά να μετρά) μέχρι να ξαναγίνει reset, είτε roll, δηλαδή στέλνει το σήμα του και γίνεται reset αυτόματα.

Report on match: καθορίζει ότι το στοιχείο είναι τερματικό για το αυτόματο

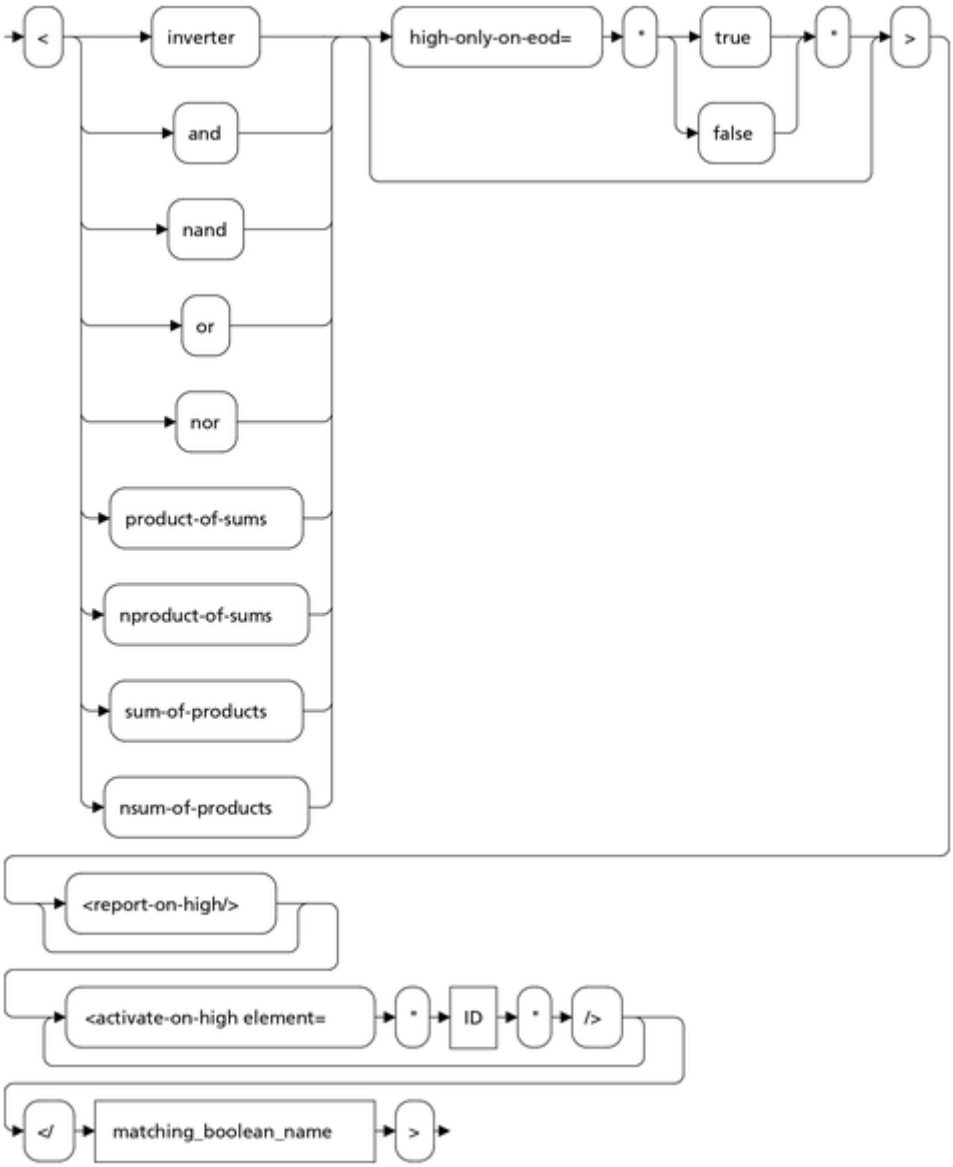
Active on match: καθορίζει ποιο στοιχείο ή σύνολο στοιχείων θα ενεργοποιηθούν αν φτάσει στο target το CNTR element

Ποιο κάτω φαίνεται ένα απλό παράδειγμα περιγραφής CNTR element:

```
<counter id="cnt_to_123" target="123" at-target="pulse">
</counter>
```

3.3 Bool element

Το Boolean στοιχείο μας δίνει την δυνατότητα να δημιουργήσουμε λογικές σχέσεις μέσα στο δίκτυο του αυτόματου χρησιμοποιώντας δυαδικές πράξεις (bitwise operations) όπως το AND και το OR. Ποιο κάτω φαίνεται το πώς περιγράφεται ένα Bool element.



Η πράξη που θα εκτελεί το στοιχείο μπορεί να είναι μια από τις invert, and, or, nand, nor, product-of-sums, inverted product of sums, sum-of-products και Inverted sum of products.

3.3.1 Bool attributes

Hight-only-on-eod: σημαδεύει το τέλος της ροής εισόδου. Μπορεί δηλαδή να παράγει σήμα μόνο με το πέρας του τελευταίου χαρακτήρα εισόδου.

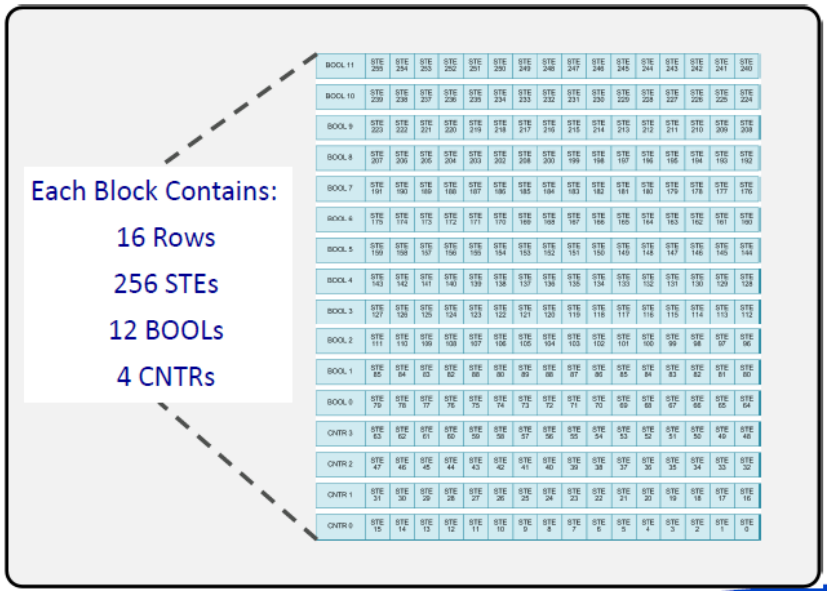
Report on high: καθορίζει ότι το στοιχείο είναι τερματικό για το αυτόματο

Active on high element: καθορίζει ποιο στοιχείο ή σύνολο στοιχείων θα ενεργοποιηθούν αν φτάσει στο target το CNTR element

3.4 Scalability

Κάθε chip AP έχει 49152 STE, 768 CNTR, και 2034 Bool elements. Τα στοιχεία αυτά (elements) οργανώνονται σε σειρές μέσα στο chip, και οι σειρές σε blocks. Το κάθε ένα από αυτά τα στοιχεία μπορεί να ενωθεί με οποιοδήποτε από τα στοιχεία του block του ή με οποιοδήποτε από τα στοιχεία των 8 γειτονικών του blocks. Μπορεί ακόμα κάποιο element να είναι ενωμένο με περισσότερα από ένα στοιχεία ταυτόχρονα. Αυτό σημαίνει πως ένα αυτόματο μπορεί την ίδια στιγμή να είναι ενωμένο και να δίνει το output του σαν input σε 2304 άλλα αυτόματα συμπεριλαμβανομένου και του εαυτού του.

Αν οργανώσουμε τα blocks σε ranks και κάνουμε ένα σχεδιασμό με πολλαπλά ranks θεωρητικά μπορούμε να επεκτείνουμε την μηχανή όσο θέλουμε.



Στην ποιο πάνω εικόνα φαίνεται ο τρόπος με τον οποίο οργανώνονται σε ένα chip όλα τα στοιχεία (STE, CNTR, Bool)

4.Tools / Simulator

4.1 QEMU:

Το QEMU είναι ένα λογισμικό ανοιχτού κώδικα που χρησιμοποιείται για την προσομοίωση μηχανών. Το QEMU μπορεί να τρέξει πάνω σε μια μηχανή λειτουργικά συστήματα ή προγράμματα που είναι σχεδιασμένα για διαφορετική μηχανή.

4.2 MARSS-x86:

Ο marss είναι ένας open source εξομοιωτής πλήρους συστήματος για αρχιτεκτονικές x86 χτισμένος πάνω σε QEMU που προσφέρει εξομοιώσεις για ομογενής και εταιρογενής μηχανές πολυπύρηνων με ακρίβεια κύκλων. Ο marss συμπεριλαμβάνει μοντέλα συνοχής μνήμης, δικτύων συνδέσεων των μονάδων, μνήμες και συσκευές I/O.

Για να προσομοιώσεις μια μηχανή στον marss πρέπει να μεταγλωττίσεις τον εξομοιωτή δίνοντας σαν παράμετρο το configuration file που περιγράφει την συγκεκριμένη μηχανή. Τα configuration files του είναι γραμμένα σε μορφή YAML και περιγράφουν τους πυρήνες, τα επίπεδα μνήμης και τις συνδέσεις μεταξύ τους.

Χτίσιμο μηχανής στον Marss:

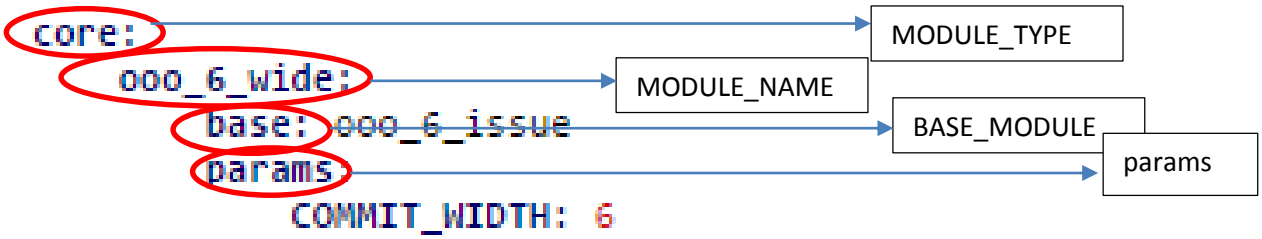
Τα configuration files στον Marss παρέχουν 3 βασικές ενότητες. Cores, Caches και Memory Controllers. Για κάθε μια από τις ενότητες ο marss προσφέρει διαφορετικούς τύπους από προκαθορισμένα μοντέλα (π.χ out-of-order core). Για το κάθε module (ενότητα) που θα χρησιμοποιήσει ο χρήστης πρέπει να δηλώσει MODULE_TYPE, MODULE_NAME, BASE_MODULE, params όπου:

MODULE_TYPE: ο τύπος του Module, μπορεί να είναι core, cache ή interconnection

MODULE_NAME: ένα μοναδικό όνομα για το Module

BASE_MODULE: σε ποιο από τα προεπιλεγμένα modules που παρέχει ο marss στηρίζεται

Params: παραμέτρους που θέλει να διαφοροποιήσει. Όσες παραμέτρους δεν αλλάξει διατηρούνται αυτές του BASE_MODULE



4.3 Valgrind:

Το valgrind είναι ένα σύνολο εργαλείων που βοηθούν στην ανάλυση του κώδικα και της εκτέλεσης μιας εφαρμογής. Τα εργαλεία valgrind ανιχνεύουν σφάλματα διαχείρισης μνήμης και μπορούν να δημιουργήσουν εύκολα το προφίλ των εφαρμογών με αρκετή λεπτομέρεια. Τα εργαλεία που παρέχει το valgrind είναι τα *memcheck, lackey, massif, helgrind, cachegrind, callgrind.*

Memcheck: ανιχνεύει προβλήματα σε σχέση με την μνήμη όπως διαρροές μνήμης, τιμές που δεν έχουν αρχικοποιηθεί, κακό χειρισμό της σωρού του προγράμματος. Οι συναρτήσεις ελέγχει είναι κυρίως οι *malloc, free, new, delete, memcopy*.

Lackey: κυρίως για σκοπούς δοκιμών και επιδείξεων.

Massif: δημιουργεί το προφίλ της σωρού της εφαρμογής. Παράγει ένα Γράφο ο οποίος δείχνει την χρήση της σωρού στον οποίο συμπεριλαμβάνονται και πληροφορίες για το πιο κομμάτι της εφαρμογής είναι υπεύθυνο για τις περισσότερες δεσμεύσεις μνήμης.

Cachegrind: εργαλείο που δημιουργεί το προφίλ της cache μνήμης. Παρέχει πληροφορίες για τα misses στην ιεραρχία της cache.

Callgrind: μια προέκταση του cachegrind, που δημιουργεί το call-graph της εφαρμογής.

4.4 KCachgrind:

Το συγκεκριμένο εργαλείο χρησιμοποιεί το αποτέλεσμα του *callgrid* (περιγραφή πιο από στο εργαλείο valgrind) για να αναπαραστήσει οπτικά τον γράφο εκτέλεσης της εφαρμογής.

5. Πειραματική Διάταξη

Για την μελέτη αρχιτεκτονικών έξυπνης μνήμης χρησιμοποιήθηκαν τα εργαλεία που αναφέρονται στο πιο πάνω κεφάλαιο.

Ο marss για να τρέξουν τα πειράματα, οι εφαρμογές δηλαδή πάνω στις διαφορετικές μηχανές, το *valgrind* για να παραχθεί ο γράφος εκτέλεσης της εφαρμογής και το *Kcachegrind* για την γραφική απεικόνισή του.

Οι εφαρμογές μεταγλωττίστηκαν στην μηχανή η οποία φιλοξενεί τον marss και στη συνέχεια αντιγράφηκαν μόνο τα εκτελέσιμα αρχεία στο *disk image* το οποίο χρησιμοποιεί ο εξομοιωτής. Στο κάθε αρχείο πριν την μεταγλώττιση προστέθηκαν εντολές εκκίνησης και τερματισμού του εξομοιωτή ανάλογα για το ποιο πείραμα θα έτρεχε (π.χ. μόνο συνάρτηση map με 4 out-of-order).

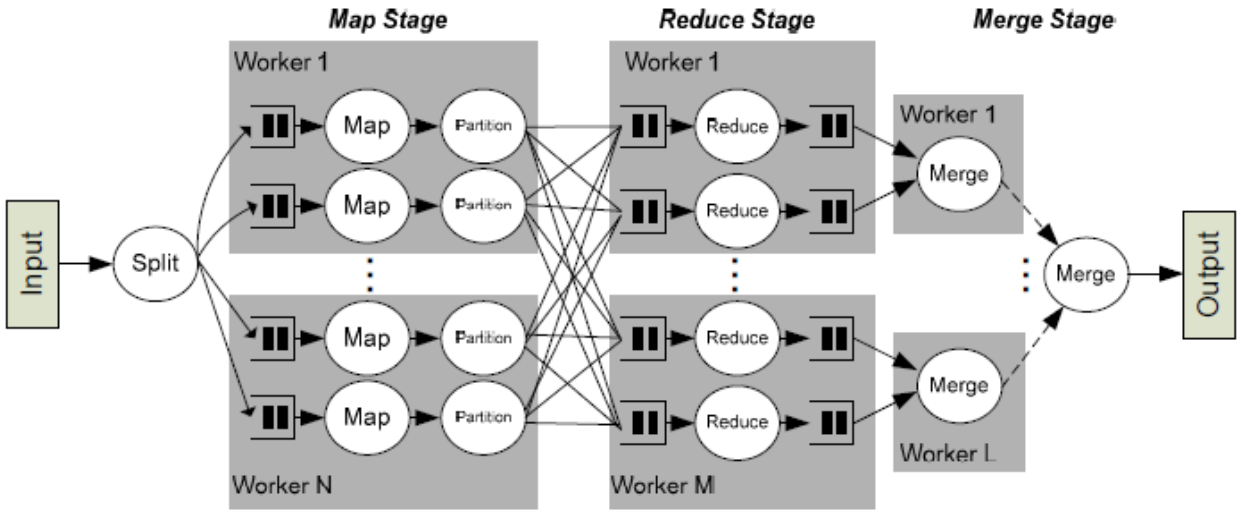
Για το λόγο ότι ο marss δεν υποστηρίζει αρχιτεκτονικές έξυπνης μνήμης τροποποιήθηκαν οι πυρήνες τους οποίους προσφέρει ο εξομοιωτής για να έχουμε μια καλή προσέγγιση της αρχιτεκτονικής αυτής. Δηλαδή σύμφωνα με τα στοιχεία που περιγράφονται στις δημοσιεύσεις για την οργάνωση *FlexRAM*, χρησιμοποιήθηκαν οι atom πυρήνες για την προσομοίωση των processing elements (*PArrays*). Τα βασικά χαρακτηριστικά τα οποία τροποποιήθηκαν ήταν οι χωρητικότητα, το accossiativity και οι χρόνοι πρόσβασης της L1 cache. Επειδή το κάθε processing element στη *FlexRAM* έχει άμεση πρόσβαση σε 1 MB μνήμης, χρησιμοποιήθηκε η L2 cache για να δείξουμε την «τοπική» μνήμη του κάθε *PArray*.

Με το πέρας των δοκιμών συλλέγηκαν τα αποτελέσματα και με την βοήθεια κάποιον scripts που προσφέρει ο ίδιος ο marss αναλύθηκαν τα αποτελέσματα. Τα αρχικά αποτελέσματα φυλάγονται σε μορφή YAML σε αρχείο που επιλέγει ο χρήστης με την επιλογή –stats πριν ξεκινήσει η προσομοίωση.

Στο παράρτημα Α επισυνάπτονται όλα τα configuration files που περιγράφουν με λεπτομέρεια τις μηχανές καθώς και οι εντολές εκτέλεσης ολόκληρης της πειραματικής διαδικασίας.

6. ΑΝΑΛΥΣΗ ΕΦΑΡΜΟΓΩΝ

Οι εφαρμογές που χρησιμοποιώ για την αξιολόγηση της έξυπνης μνήμης είναι οι εφαρμογές του project phoenix, οι οποίες είναι κάποιες map-reduce υλοποιήσεις διαφόρων εφαρμογών. Για την διπλωματική μου εργασία αναλύθηκαν οι wordcount και kmeans. Βασικά χαρακτηριστικά των εφαρμογών του phoenix είναι πως αντί για clusters χρησιμοποιεί νήματα, οι επικοινωνία γίνεται με shared-memory αωτι με network messages και οι υλοποιήσεις είναι σε γλώσσα προγραμματισμού C.



Στο πιο πάνω σχήμα παρουσιάζεται στη γενική του μορφή τα βήματα που ακολουθούν οι εφαρμογές του phoenix για να υλοποιήσουν το map-reduce μοντέλο προγραμματισμού σε μηχανές πολυπυρήνων.

Το πρώτο στάδιο, το *split,* χωρίζει τα δεδομένα εισόδου σε κομμάτια (chunks) για τους *map workers.* Το μέγεθος των κομματιών εξαρτάται από το μέγεθος της cache και ο λόγος είναι για να διατηρείται το *locality* των δεδομένων. Στη διαδικασία του map, κάθε map function παίρνει σαν είσοδο ένα chunk. Πριν τελειώσει το map stage γίνεται ένα intermediate partition για να βελτιωθεί η διαδικασία του reduce. Κακό partitioning μπορεί να οδηγήσει σε load imbalances στη συνέχεια (reduce). Στο reduce stage, κάθε worker αναλαμβάνει ένα "key set" από την ουρά. Τέλος στο merge stage, συνδυάζονται τα αποτελέσματα του reduce σε ένα ταξινομημένο output.

6.1 K-MEANS

Περιγραφή:

Η εφαρμογή Kmeans στοχεύει στον διαχωρισμό <p> σημείων σε <c> κλάσεις, όπου κάθε στοιχείο ανήκει στην κλάση με τον πλησιέστερο μέσο.

Ο αλγόριθμος του Kmeans χωρίζεται σε 2 βασικά βήματα, το *assignment step* και το *update step*. Στο πρώτο βήμα, τα σημεία ανατίθενται στις κλάσεις τις οποίες τα κεντρικά σημεία (means) είναι πιο «κοντά» στο σημείο. Αυτό καθορίζεται από την Ευκλήδεια Απόσταση. Τα αρχικά means υπολογίζονται τυχαία. Στο βήμα του update, υπολογίζονται τα καινούρια κεντρικά σημεία σύμφωνα με τα σημεία που έχουν ανατεθεί στην κάθε κλάση.

Ο αλγόριθμος του Kmeans βρίσκει εφαρμογές σε περιπτώσεις signal processing και data mining.

Για την εκτέλεση της εφαρμογής ο χρήστης δίνει τις παραμέτρους:

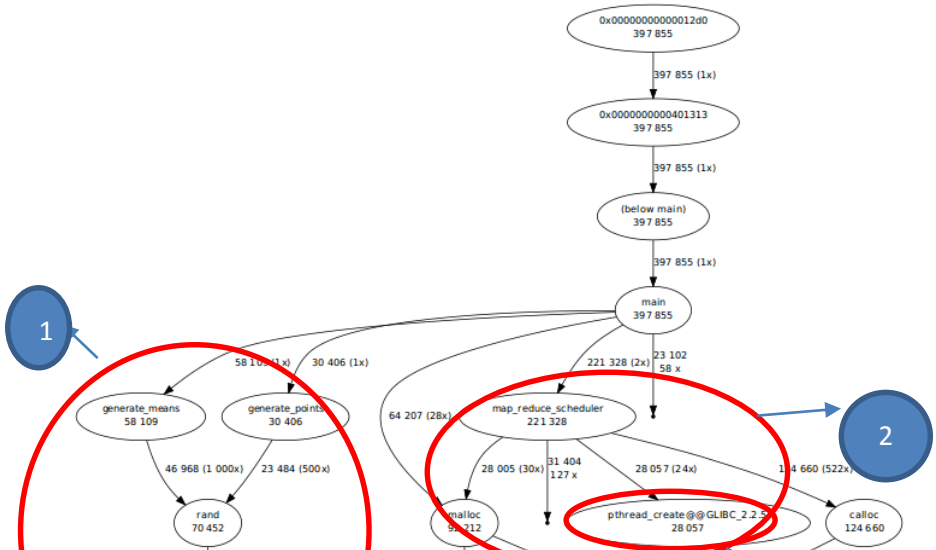
-d<number of dimensions>

-c<number of classes>

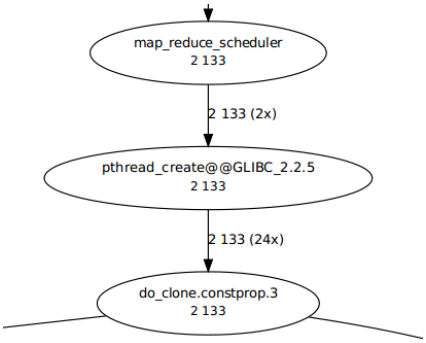
-p<number of points>

-s<max point value>

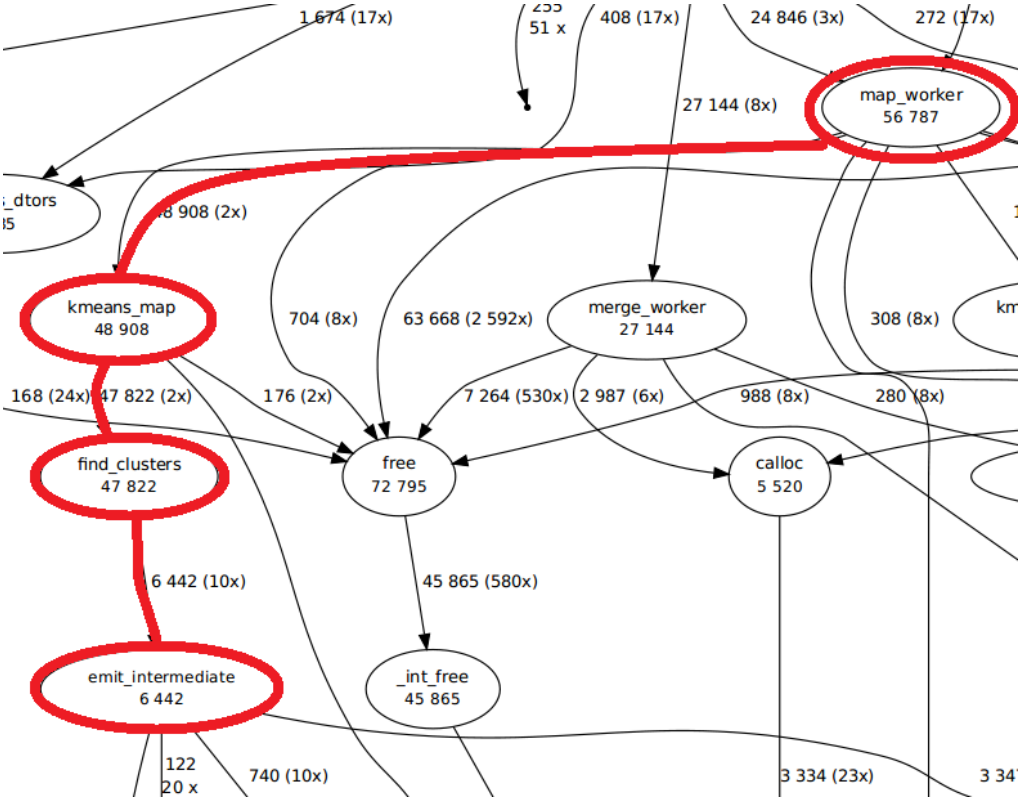
Γράφος εκτέλεσης:



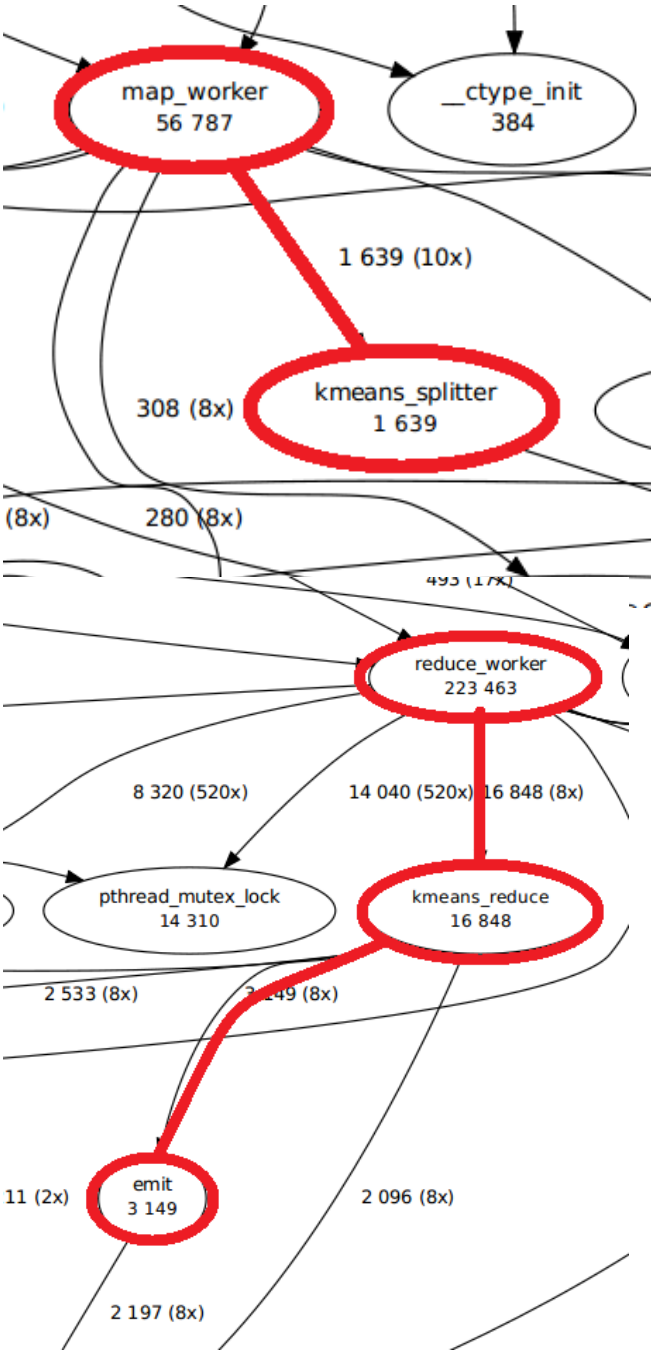
Σχήμα 1: Γενικός Γράφος εκτέλεσης εφαρμογής



Σχήμα 2: Συνέχεια μετά την συνάρτηση pthread_creat



Σχήμα 3: συνέχεια του σχήματος 2, η διαδικασία του map



Σχήμα 4: συνέχεια του σχήματος 2, διαδικασία του map

Σχήμα 5: συνέχεια του σχήματος 2, διαδικασία του reduce

Επεξήγηση γράφου εκτέλεσης:

Κατά την διάρκεια εκτέλεσης της εφαρμογής παρατηρούμαι 2 κύρια κλαδιά στον Γράφο εκτέλεσης (1 και 2 στο σχήμα 1). Στο κλαδί [1] είναι η διαδικασία όπου τυχαία παράγονται τα σημεία τα οποία στη συνέχεια θα χωριστούν στις κλάσεις, και ο καθορισμός των κλάσεων. Στο κλαδί [2] είναι η διαδικασία του map – reduce με την οποία τα σημεία χωρίζονται στις κλάσεις. Όλες οι παράμετροι (αριθμός σημείων, αριθμός κλάσεων, μέγιστη τιμή σημείου και διαστάσεις) δίνονται από τον χρήστη.

Κατά την προσομοίωση αρχιτεκτονικής έξυπνης μνήμης, οι συναρτήσεις οι οποίες θα εκτελεστούν σε αυτή είναι όσες είναι κάτω από το κλαδί [2] του γράφου εκτέλεσης (map – reduce).

Σχήμα 1: ο γενικός Γράφος εκτέλεσης του προγράμματος. Παρατηρούμε 2 κύρια κλαδιά. Στο κλαδί [1] γίνεται κλήση των συναρτήσεων οι οποίες δημιουργούν τυχαία τα σημεία (ο αριθμός των οποίων δίνεται σαν παράμετρος από τον χρήστη στην εφαρμογή) τα οποία θα χωριστούν στις κλάσεις. Ακόμα σε αυτό το κλαδί παράγονται και τα “means” τα οποία θα είναι τα κεντρικά σημεία των κλάσεων (ο αριθμός των κλάσεων επίσης δίνεται σαν παράμετρος από τον χρήστη στην εφαρμογή). Και τα 2 (points and means) παράγονται τυχαία.

Σχήμα 2: στο σχήμα 2 φαίνεται η συνάρτηση [f1] η οποία μπορεί να θεωρηθεί ως ρίζα του γράφου εκτέλεσης όλων των συναρτήσεων που πραγματοποιούν την διαδικασία του map-reduce.

Σχήμα 3/ Σχήμα 4: στο σχήμα 3 και 4 φαίνονται όλες οι συναρτήσεις των οποίων γίνεται κλήση για την διαδικασία του map. Οι συναρτήσεις αυτές θα εκτελεστούν στην προσομοίωση της έξυπνης μνήμης.

Σχήμα 5: στο σχήμα 5 φαίνονται οι συναρτήσεις των οποίων γίνεται κλήση κατά την διαδικασία του reduce. Οι συναρτήσεις αυτές θα εκτελεστούν στην προσομοίωση της έξυπνης μνήμης.

Ανάλυση χρόνου εκτέλεσης εφαρμογής:

Παράμετροι που δόθηκαν στην εφαρμογή για συλλογή αποτελεσμάτων:

./kmeans –d2 –c10 –p1000 –s100

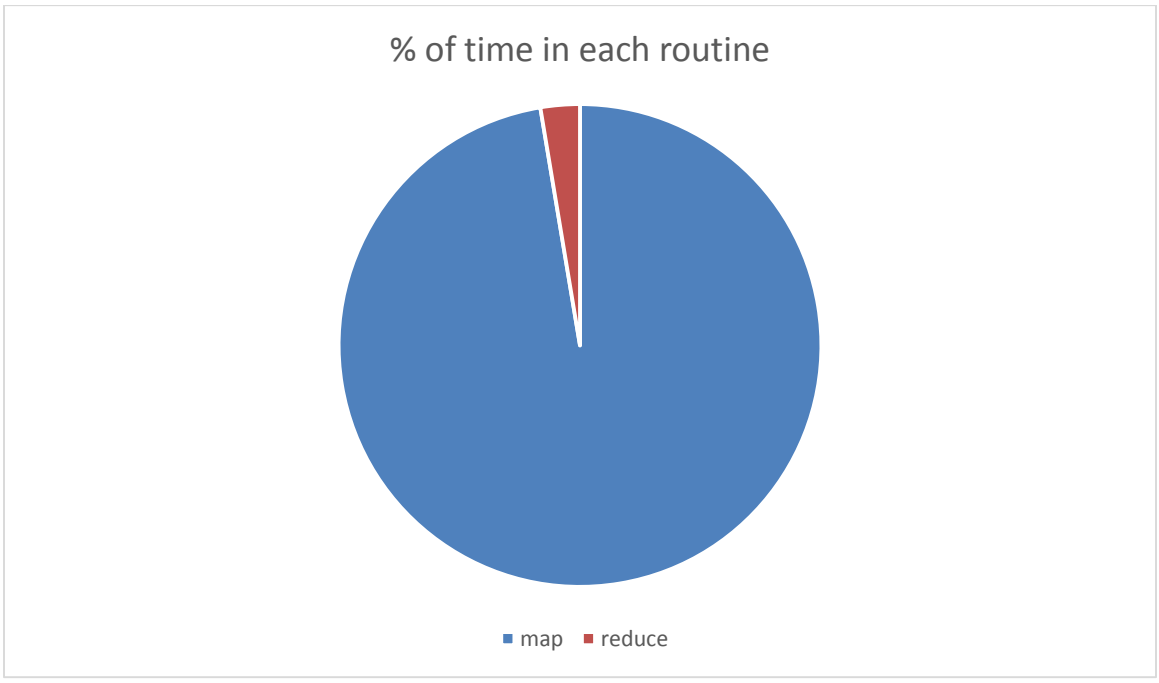
Στο πιο κάτω γράφημα (Γράφημα 1) φαίνεται σε ποσοστά ο χρόνος εκτέλεσης του κάθε σταδίου της εφαρμογής:

Ο χρόνος εκτέλεσης για το κομμάτι του map είναι για τις συναρτήσεις που φαίνονται στα σχήματα 3 και 4.

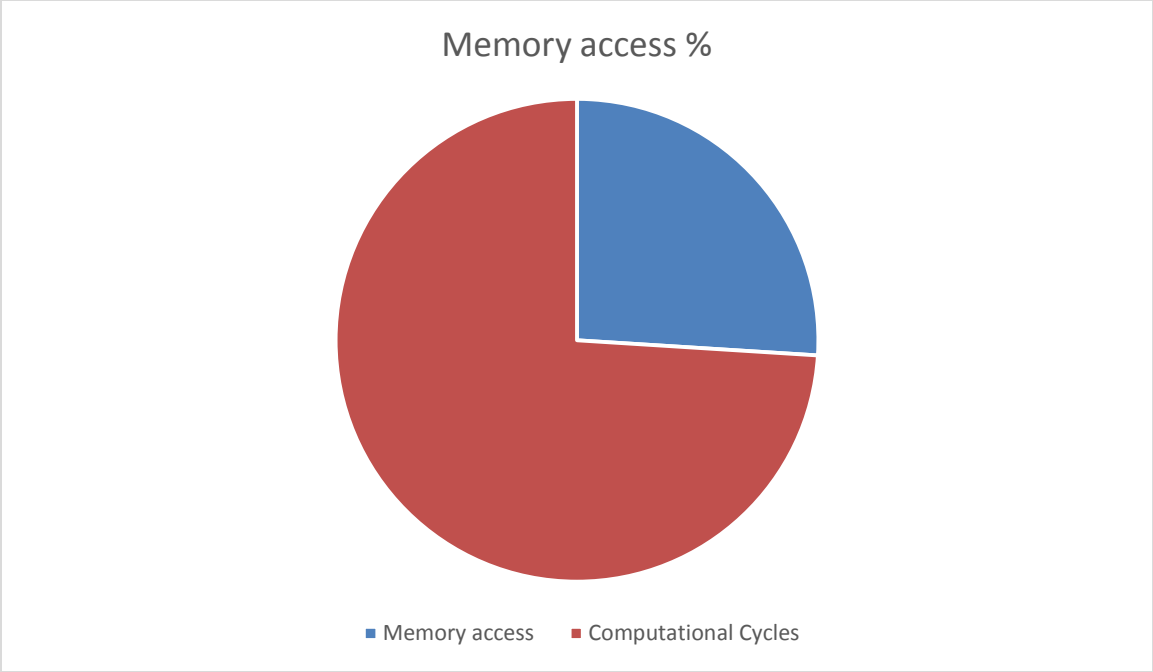
Ο χρόνος εκτέλεσης για το κομμάτι του reduce είναι για τις συναρτήσεις που φαίνονται στο σχήμα 5.

Οι χρόνοι υπολογίστηκαν με *ptlcalls* (ptlcall_nop(), ptlcall_kill(), ptlcall_switch_to_sim(), ptlcall_switch_to_native()) που υποστηρίζει ο mars.

(για τα αποτελέσματα χρησιμοποιήθηκε η μηχανή 4xooo -4 out of order cores- η οποία περιγράφεται αναλυτικά πιο κάτω)



Γράφημα 1: ποσοστό κάθε συνάρτησης από τον συνολικό χρόνο εκτέλεσης εφαρμογής



Γράφημα 2: Memory access % out of total time

Γράφημα 2: στο γράφημα 2 παρουσιάζεται το ποσοστό χρόνου το οποίο καταναλώνει η εφαρμογή για όλα τα I/O και για τους υπολογισμούς. Ο υπολογισμός των ποσοστών αυτών έγινε με βάση τα cache miss και cache hits της εφαρμογής (στατιστικά από mars simulator). Από τα configuration files γνωρίζουμε τον χρόνο που χρειάζεται για μια πρόσβαση στο κάθε επίπεδο μνήμης.

Άρα ο χρόνος για υπολογιστεί ο συνολικός χρόνος για τα memory requests χρησιμοποιήθηκε η φόρμουλα:

$$total_{memory_{access_{time}}} = \left((L1_{miss} + L1_{hit}) * L1_{latency}\right) + \left((L2_{miss} + L2_{hit}) * L2_{latency}\right)$$

Περιγραφή configuration files για marss:

MACHINE	APPLICATION PART	DESCRIPTION			
		Core	Cache		
			L1	L2	L3
4 x out of order cores (ooo)	ALL	ISSUE WIDTH: 4 COMMIT WIDTH: 4	SIZE: 128K LINE_SIZE: 64 ASSOC: 8 LATENCY: 2 READ_PORTS:2 WRITE_PORTS:1	SIZE: 2M LINE_SIZE:64 ASSOC: 8 LATENCY: 5 READ_PORTS: 2 WRITE_PORTS:2	
	Serial				
8 x out of order cores (ooo)	ALL				
	Serial				
Xeon core	Serial	ISSUE WIDTH: 5 COMMIT WIDTH: 4 ROB SIZE: 128 ISSUE Q SIZE: 36 ALU FU COUNT: 6 FPU FU COUNT: 6 LOAD FU COUNT: 1 STORE FU COUNT: 1 LOAD Q SIZE: 48 STORE Q SIZE: 32	SIZE: 128K LINE_SIZE: 64 ASSOC: 8 LATENCY: 2 READ_PORTS:2 WRITE_PORTS:1	SIZE: 2M LINE_SIZE:64 ASSOC: 8 LATENCY: 5 READ_PORTS:2 WRITE_PORTS:2	SIZE: 12M LINE_SIZE:64 ASSOC: 16 LATENCY: 27 READ_PORTS: 2 WRITE_PORTS :2
4 x atom cores	Map-reduce-merge	DISPATCH Q SIZE: 16 ISSUE PER CYCLE: 2	SIZE: 8K LINE_SIZE:32 ASSOC: 2 LATENCY: 1 READ_PORTS:2 WRITE_PORTS:1	SIZE: 1M LINE_SIZE:32 ASSOC: 4 LATENCY: 1 READ_PORTS:2 WRITE_PORTS:2	
8 x atom cores	Map-reduce-merge				

Πίνακας 1: περιγραφή μηχανών που χρησιμοποιήθηκαν

Πίνακας 1:

Στον πίνακα 1 παρουσιάζονται τα κομμάτια από τα configuration file που κάνουν την κάθε μηχανή να ξεχωρίζει. Χαρακτηριστικά για την αναπαράσταση της έξυπνης μνήμης ο χρόνος πρόσβασης στην ιεραρχία μνήμης για τις μηχανές 4xatom και 8xatom έχει οριστεί ως 1 κύκλος. Με τον τρόπο αυτό δείχνουμε ότι οι πυρήνες βρίσκονται μέσα στην μνήμη. Ακόμα για το λόγο ότι τα *Processing Elements* μιας έξυπνης μνήμης πρέπει να κρατηθούν μικρά και όσο πιο απλά γίνεται έχουν μόνο ένα επίπεδο cache (L1). Η L2 στη προσομοίωση, έχει οριστεί να έχει μέγεθος 1MB και παίρνει την θέση της

«κοντινής» RAM, το κομμάτι δηλαδή της μνήμης όπου το *Processing Element* έχει άμεση πρόσβαση χωρίς καθυστερήσεις.

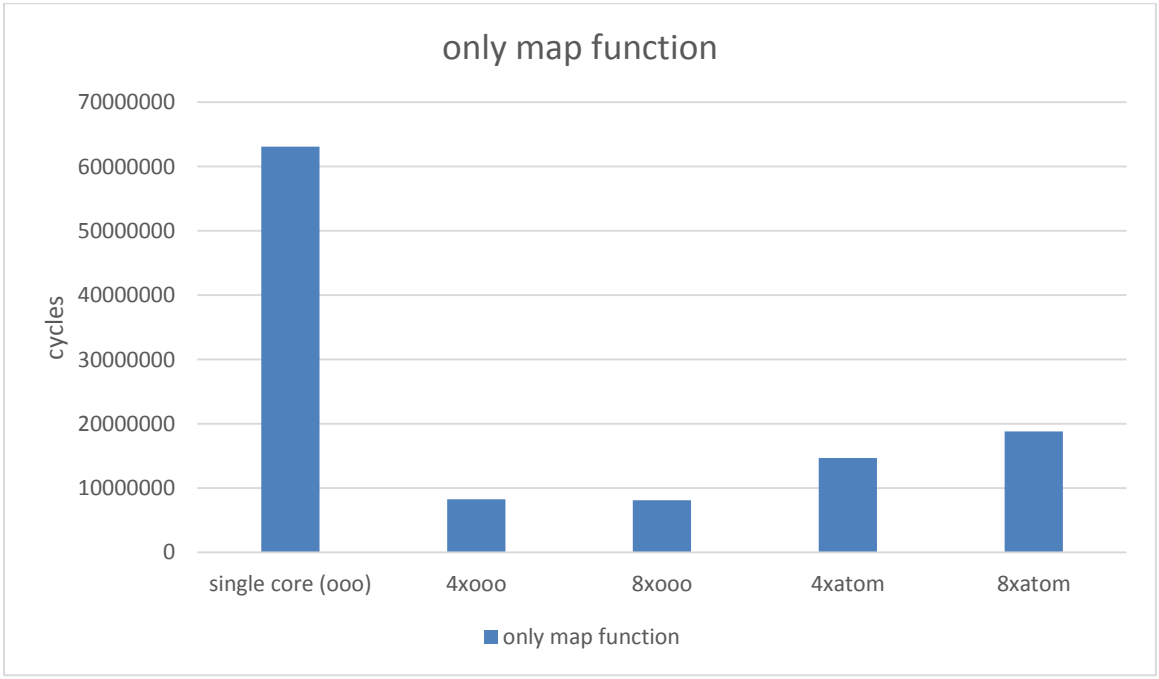
Οι μηχανές 4xatom και 8xatom είναι αυτές που χρησιμοποιήθηκαν για την προσομοίωση της έξυπνης μνήμης και προσομοιώνουν αντίστοιχα 4 και 8 *Processing Elements.* Οι δυνατότητες του πυρήνα και οι παράμετροι της ιεραρχίας μνήμης είναι οι ίδιες με αυτές που περιγράφονται στην *FlexRAM*.

4xooo: 4 out of orders cores - default marss' parameters

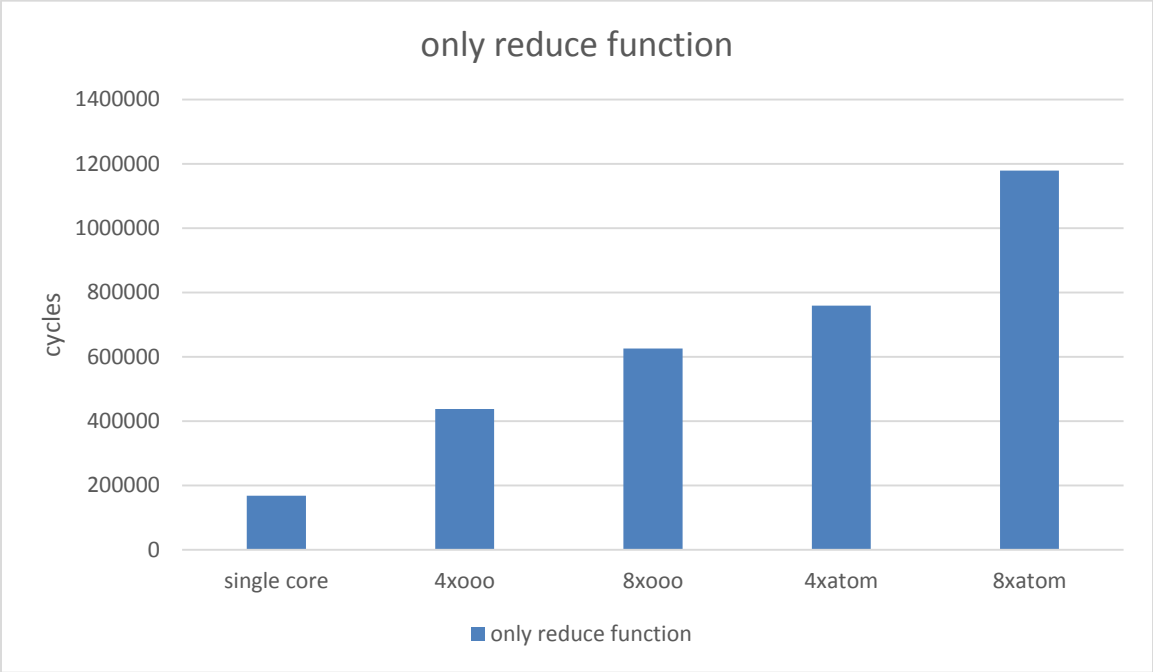
8xooo: 8 out of orders cores – default marss' parameters

Xeon: xeon single core – default marss' parameters

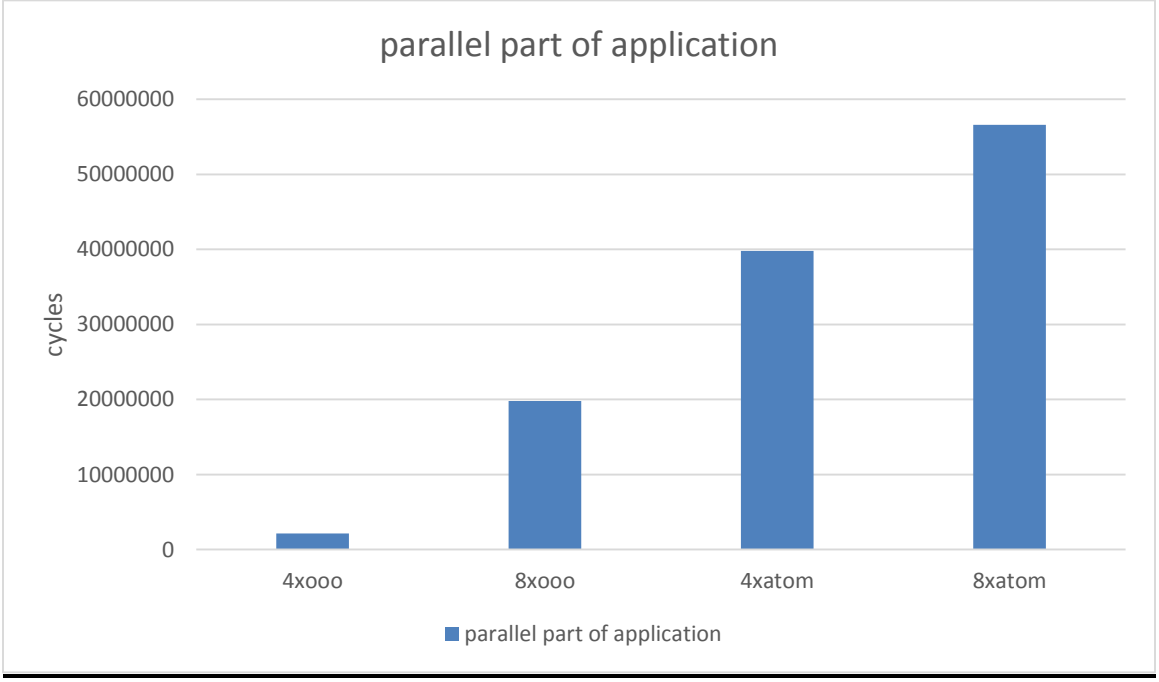
Αποτελέσματα:



Γράφημα 3: χρόνος εκτέλεσης της συνάρτησης map σε κύκλους



Γράφημα 4: χρόνος εκτέλεσης της συνάρτησης reduce σε κύκλους



Γράφημα 5: χρόνος εκτέλεσης του παράλληλου κομματιού της εφαρμογής σε κύκλους

Επεξήγηση γραφικών παραστάσεων αποτελεσμάτων:

Γράφημα 3:

Στη γραφική παράσταση (γράφημα 3) παρατηρούμε το χρόνο εκτέλεσης της συνάρτησης *map* στις διαφορετικές μηχανές. Παρατηρώντας τα αποτελέσματα μπορούμε να πούμε πως το κέρδος στη συγκεκριμένη περίπτωση προέρχεται από τον παραλληλισμό και όχι από την χρήση έξυπνης μνήμης. Αυτό είναι αναμενόμενο αφού όπως φαίνεται και στο γράφημα 2 (ποσοστό χρόνου για computational και I/O), η εφαρμογή που εξετάζουμε δεν είναι memory-bound αλλά computational. Ακόμα στο σχήμα 3 φαίνεται το κάλεσμα της συνάρτησης *find_cluster* η οποία είναι υπολογιστικά αρκετά περίπλοκη ~$O(n^2)$. Άρα χρησιμοποιώντας περισσότερα από 1 cores, βελτιώνουμε τον χρόνο αφού το συγκεκριμένο κομμάτι της εφαρμογής (map) μπορεί να εκτελεστεί παράλληλα, αλλά αλλάζοντας τα cores σε πιο αργά (atoms αντί ooo), αν και έχουμε άμεση και χωρίς καθυστερήσεις πρόσβαση στη μνήμη, η εκτέλεση γίνεται πιο αργή γιατί είναι βασισμένη στους υπολογισμούς και όχι στα I/O.

Γράφημα 4:

Στη γραφική παράσταση (γράφημα 4) παρατηρούμε το χρόνο εκτέλεσης της συνάρτησης *reduce* στις διαφορετικές μηχανές που δοκιμάστηκε η εφαρμογή. Παρατηρώντας τα αποτελέσματα βλέπουμε πως το single_core έχει την καλύτερη επίδοση και από τα απλά παράλληλα και από τα παράλληλα atoms. Αυτό δικαιολογήται μέσα από τον κώδικα και τον τρόπο εκτέλεσης της εφαρμογής γιατί η συνάρτηση reduce κάνει μόνο 2 απλές πράξεις, μια πρόσθεση (μέσα σε ένα for loop) και μια διαίρεση. Λόγο του μικρού αριθμού σημείων (-p runtime option) που δίνουμε σαν παράμετρο, τα overheads για την δημιουργία και την χρονοδρομολόγιση των threads ξεπερνούν τον χρόνο εκτέλεσης των πράξεων που έχει να κάνει η συνάρτηση reduce. Οι κύκλοι για να ολοκληρωθεί η συνάρτηση γίνονται όλο και περισσότεροι με τα atoms (έξυπνη μνήμη) γιατί αφού όπως προαναφέρθηκε, το k-means δεν είναι μια memory-bound εφαρμογή, αλλάζοντας τα γρήγορα out-of-order με πιο αργά atom, παρόλο που μειώνονται οι χρόνοι πρόσβασης στην ιεραρχία μνήμης στο τέλος χάνουμε στην επίδοση.

Γράφημα 5:

Στο γράφημα 5 παρουσιάζεται η συνολική εικόνα της εφαρμογής στις διάφορες μηχανές. Είναι δηλαδή ένας συνδυασμός του γραφήματος 3 και 4.

Γενική παρατήρηση για τα γραφήματα:

Στα πιο πάνω γραφήματα παρατηρούμε πως οι μηχανές με τα 8 atom cores είναι πιο αργές από αυτές με τα 4. Αυτό υποθέτουμε πως οφείλετε στα overheads του χρονοπρογραμματισμού των threads και το αποτέλεσμα αυτό αναμένετε να αλλάξει (τα 8 atom να γίνουν πιο γρήγορα από τα 4) με μεγαλύτερο

input file γιατί εκεί ο πραγματικός ωφέλιμος χρόνος λειτουργίας του core θα επισκιάζει τις καθυστερήσεις του χρονοπρογραμματισμού. Υπόθεση την οποία δεν μπορέσαμε να αποδείξουμε στην συγκεκριμένη πειραματική διαδικασία.

6.2 Word Count Application

Περιγραφή:

Η εφαρμογή του wordcount μετρά την συχνότητα εμφάνισης της κάθε μιας μοναδικής λέξης μέσα σε ένα αρχείο κειμένου. Η συγκεκριμένη υλοποίηση της εφαρμογής το κάνει χρησιμοποιώντας μοντέλο προγραμματισμού map-reduce.

Διαδικασία εκτέλεσης εφαρμογής:

Η εφαρμογή του wordcount του phoenix εκτελείτε και ολοκληρώνεται με τα ακόλουθα βήματα.

1. Open file
2. Map file to memory (uses mmap())
3. Fill struct for splitter function
4. Fill scheduler struct
    - task-data: file stats
    - map function pointer
    - reduce function pointer
    - splitter function pointer
    - compare function pointer
    - unit-size (approximately bytes per word)
    - cache size
    - # of map threads
    - # of reduce threads
    - # of merge threads

Scheduler:

5. Find # of available processors
6. Creates threads for map
7. MAP
8. Creates threads for reduce
9. REDUCE
10. Creates threads for merge
11. MERGE

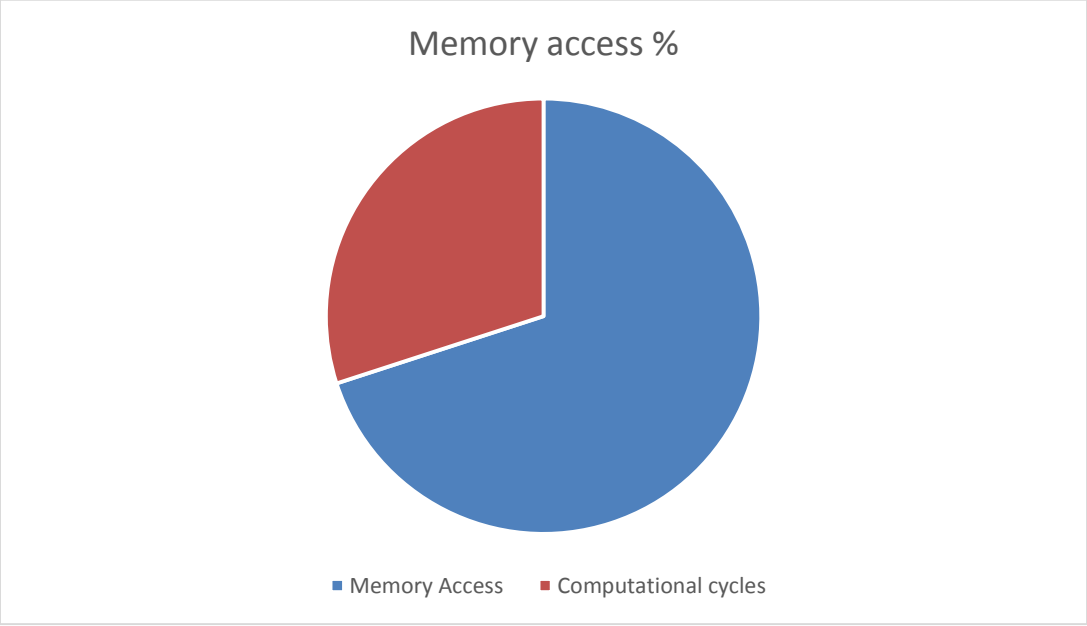
Για την εξαγωγή αποτελεσμάτων μετρώ μόνο των χρόνο που χρειάζεται η διαδικασία του scheduler, γιατί το υπόλοιπο κομμάτι του προγράμματος είναι σειριακό και δεν υπάρχει λόγος το μετρήσουμε αφού θα είναι το ίδιο σε όλες τις μηχανές.

Περιγραφή configuration files για marss:

Τα configuration files που περιγράφουν τις μηχανές στις οποίες έτρεξε η εφαρμογή για την συλλογή των αποτελεσμάτων περιγράφονται στον [πίνακα 1]. (εφαρμογή kmeans).

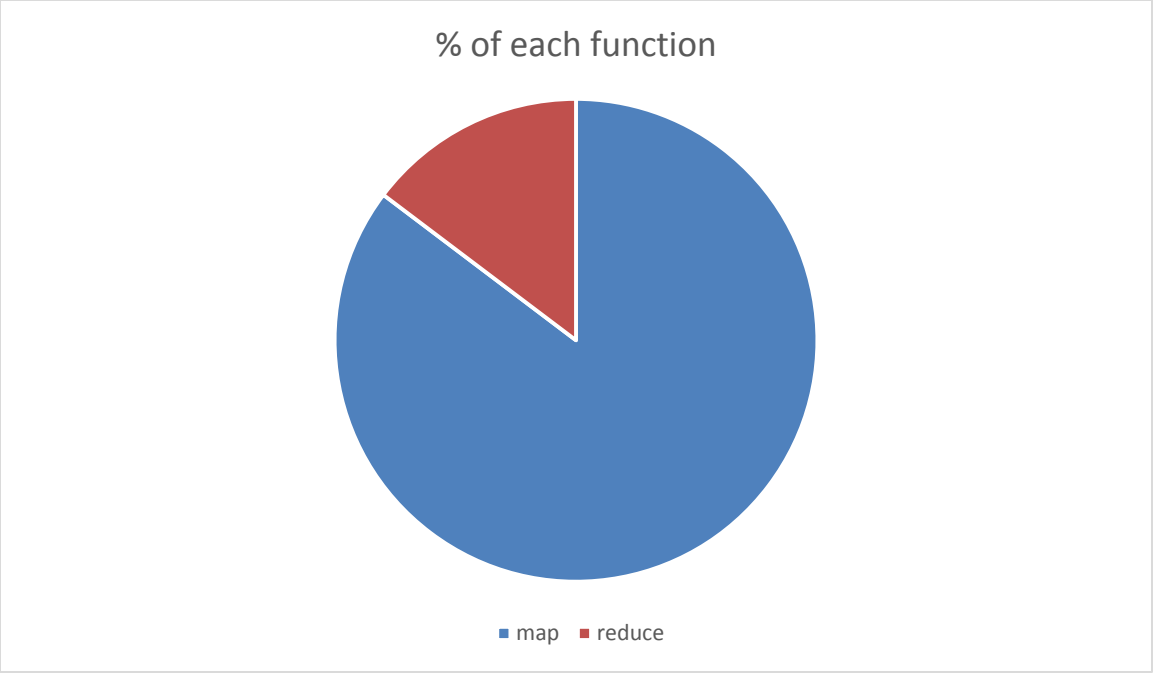
Γιατί wordcount:

Ο λόγος που επιλέγουμε την εφαρμογή αυτή για ανάλυση σε αρχιτεκτονικές έξυπνης μνήμης, είναι γιατί το wordcount είναι μια memory-centric εφαρμογή χωρίς πολλές και υπολογιστικά περίπλοκες πράξεις.



Γράφημα 6: ποσοστό χρόνου εκτέλεσης για I/O

Όπως φαίνεται στο γράφημα 6, το ποσοστό του συνολικού χρόνου εκτέλεσης που ξοδεύεται για είσοδο/έξοδο είναι σχεδόν 75%. Άρα σαν αναμενόμενο αποτέλεσμα περιμένουμε να δούμε σημαντική βελτίωση στο συνολικό χρόνο αφού μειώνονται στο ελάχιστο οι χρόνοι πρόσβασης στην ιεραρχία μνήμης. Τα ποσοστά υπολογίστικαν σύμγωνα με τα στατιστικά που μας δίνει ο marss, και με την ακόλουθη φόρμουλα:



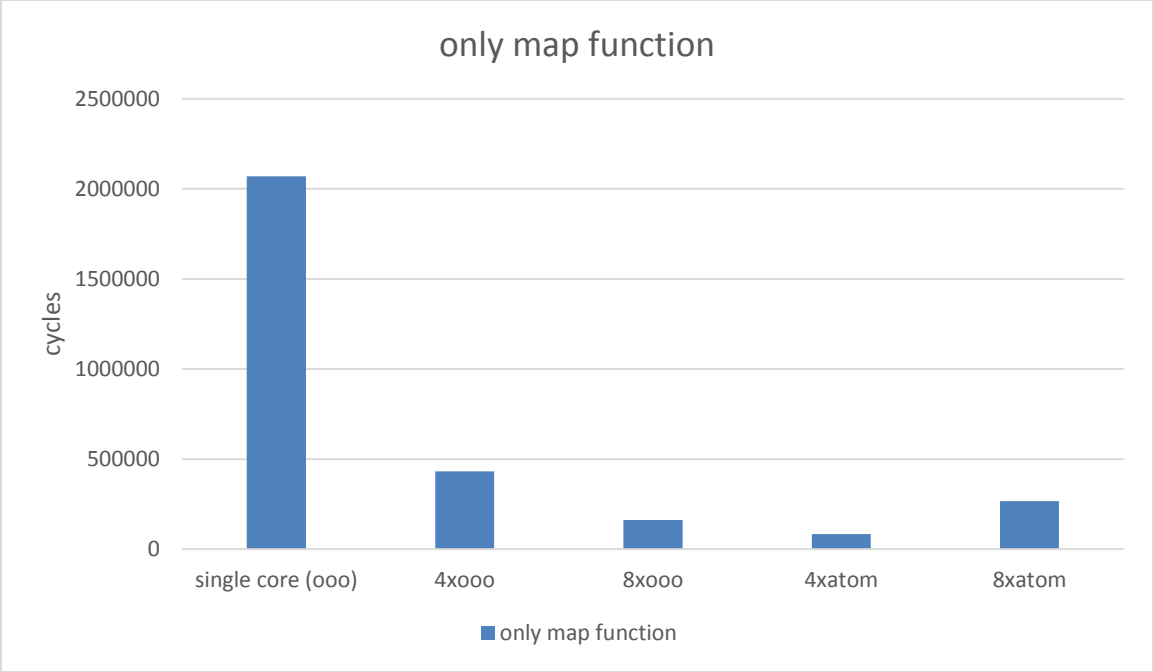
$$total_{memory_{access_{time}}} = \left((L1_{miss} + L1_{hit}) * L1_{latency}\right) + \left((L2_{miss} + L2_{hit}) * L2_{latency}\right)$$

Γράφημα 7: ποσοστό που ξοδεύεται σε κάθε μια από τις συναρτήσεις της εφαρμογής.

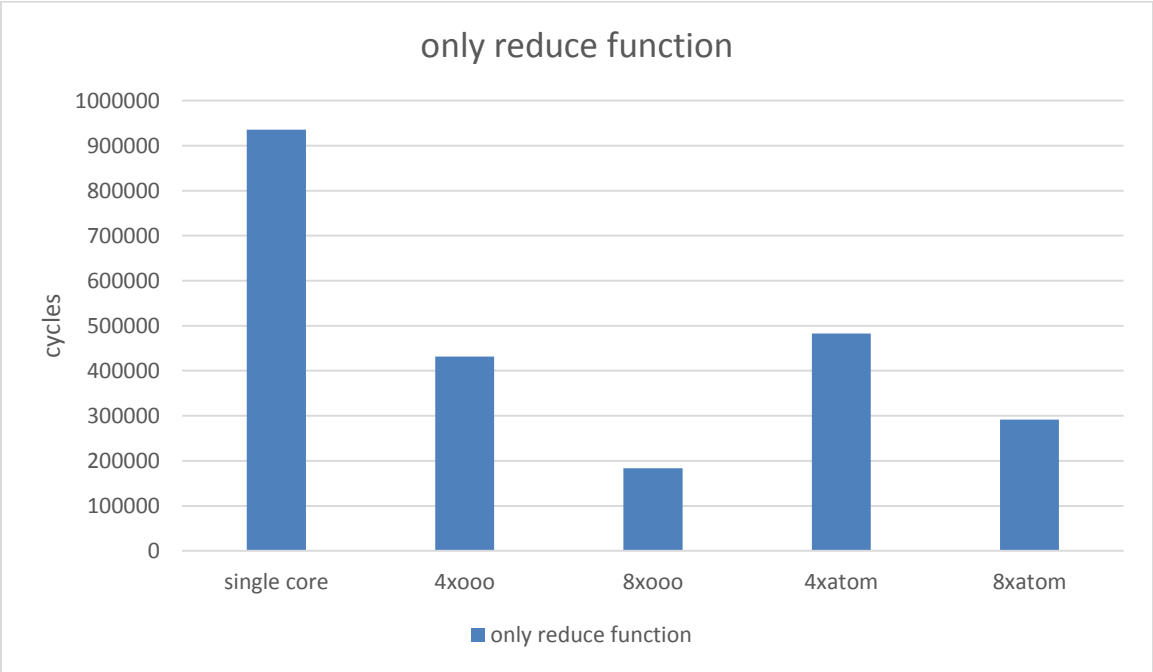
Όπως φαίνεται στο γράφημα 7, το μεγαλύτερο ποσοστό χρόνου ξοδεύεται στη συνάρτηση του map η οποία είναι και η συνάρτηση στην οποία περιμένουμε να δούμε και την μεγαλύτερη βελτίωση. Στα αποτελέσματα που θα παρουσιαστούν πιο κάτω δεν λαμβάνονται υπόψη οι ενέργειες του *host*.

Η μηχανή που χρησιμοποιήθηκε για την ανάλυση του χρόνου είναι η 4xooo (επεξήγηση πιο κάτω στον πίνακα 2).

Αποτελέσματα:



Γράφημα 8: χρόνος εκτέλεσης (σε κύκλους) της συνάρτησης map



Γράφημα 9: χρόνος εκτέλεσης σε κύκλους της συνάρτησης reduce

Επεξήγηση γραφικών παραστάσεων αποτελεσμάτων:

Γράφημα 8: στο γράφημα 8 φαίνεται ο χρόνος εκτέλεσης της συνάρτησης map στις διαφορετικές μηχανές. Παρατηρώντας τα αποτελέσματα μπορούμε να πούμε πως για τις μηχανές 4xooo και 8xooo κερδίζουμε πολύ μόνο από το ότι εκτελούμε την συνάρτηση παράλληλα. Στη συνέχεια παρατηρώντας τις τιμές των μηχανών έξυπνης μνήμης και σε συνδυασμό με το γράφημα 6, που μας δείχνει πως η εφαρμογή μας είναι Memory-centric, βλέπουμε πως μειώνεται ακόμη περισσότερο ο χρόνος γιατί οι συγκεκριμένες μηχανές έχουν άμεση πρόσβαση στην ιεραρχία μνήμης χωρίς καθυστερήσεις.

Γράφημα 9: στο γράφημα 9 παρατηρούμαι τον χρόνο εκτέλεσης της συνάρτησης reduce στις διαφορετικές μηχανές. Σε αυτή την περίπτωση το κέρδος φαίνεται να προέρχεται κυρίως από τον παραλληλισμό της εκτέλεσης αφού οι μηχανές με τα «δυνατά» cores είναι πιο γρήγορες από αυτές που προσομοιώνουν την έξυπνη μνήμη. Αυτό είναι αναμενόμενο σε σχέση με το γράφημα 8 που δείχνει τους χρόνους για την map αφού η reduce είναι υπολογιστικά πιο περίπλοκη από την map.

7.Αποτελέσματα – Συμπεράσματα

Παρατηρώντας τις εφαρμογές που έχουν αναλυθεί για τις ανάγκες της διπλωματικής εργασίας μπορούμε να διακρίνουμε βασικές διαφορές.

Η εφαρμογή του *wordcount* είναι η εφαρμογή που επωφελείται σε μεγάλο βαθμό από μια οργάνωση μηχανής στο μοντέλο της έξυπνης μνήμης. Αυτό γιατί όπως φαίνεται στο γράφημα 6, είναι μια εφαρμογή η οποία σχεδόν το 75% του χρόνου εκτέλεσης της το καταναλώνει σε διαδικασίες διαβάσματος και γραψίματος στη μνήμη. Αυτός ακριβώς είναι και ο σκοπός των αρχιτεκτονικών έξυπνης μνήμης. Όπως αναφαίρετα και σε προηγούμενα κεφάλαια, και όπως φαίνεται και από το κεφάλαιο των σχετικών εργασιών, όλες οι αρχιτεκτονικές αυτού του τύπου σαν κύριο στόχο έχουν να μειώσουν στο ελάχιστο αυτό τον χρόνο. Μια υλοποίηση όπως αυτή που εξετάστηκε, ήταν αναμενόμενο να επωφεληθεί στο μέγιστο γιατί θεωρητικά το κάθε *processing element*, στη περίπτωση των πειραμάτων που έγιναν, ο κάθε πυρήνας τύπου atom, έχει άμεση πρόσβαση σε ένα κομμάτι του αρχείου και μπορεί να εκτελέσει την διαδικασία του map σαν να είναι όλο το partition του αρχείου που εξετάζει στην *cache* του. Ακόμη το *wordcount* σε αντίθεση με το *kmeans* δεν έχει υπολογιστικά περίπλοκες πράξεις.

Από την άλλη, η εφαρμογή του *kmeans*, όπως φαίνεται και από το γράφημα 2, είναι εντελώς αντίθετη εφαρμογή από το *wordcount*. Δηλαδή το μεγαλύτερο μέρος της εκτέλεσης της (σχεδόν 75%) είναι καθαρά υπολογιστικό και όχι I/O. Άρα για να βελτιωθεί η απόδοση μιας εφαρμογής με αυτά τα χαρακτηριστικά, (computational-bound application) χρειάζονται πιο ισχυροί επεξεργαστές.

Όπως αναφέρεται και την εργασία του *FlexRAM*, που ήταν η βάση για τα πειράματα, τα processing elements, είναι μικρών δυνατοτήτων πυρήνες. Άρα τα αποτελέσματα που παίρνουμε είναι αναμενόμενα, αφού τα *atom cores* είναι κατά πολύ πιο «αδύνατα» από τα *out-of-order*.

Μέσα από την πειραματική διαδικασία που ακολουθήθηκε μπορούμε να πούμε πως με τις αρχιτεκτονικές τύπου έξυπνης μνήμης (PIM-architectures), μπορούμε να παρατηρήσουμε βελτίωση στην επίδοση εφαρμογών με συγκεκριμένα χαρακτηριστικά. Μια εφαρμογή για να επωφεληθεί σε βαθμό που να μπορούμε να πούμε πως αξίζει να τρέξει σε μια τέτοια αρχιτεκτονική πρέπει να είναι memory-centric εφαρμογή, δηλαδή το μεγαλύτερο κομμάτι της να είναι επικοινωνία με την ιεραρχία μνήμης και οι υπολογισμοί που χρειάζεται να κάνει η υπόλοιπη εφαρμογή να μην είναι υπολογιστικά πολύ περίπλοκοι.

8.Μελλοντική Εργασία

Τα πειράματα που παρουσιάστηκαν στην παρούσα διπλωματική εργασία είναι μόνο μια αρχή, μια πρώτη προσέγγιση για κατανόηση των PIM αρχιτεκτονικών. Από τα συμπεράσματα φαίνεται πως αυτό το μοντέλο οργάνωσης μηχανών μπορεί να επιτύχει σημαντική βελτίωση στην απόδοση (σε σχέση με τον χρόνο) σε μεγάλο φάσμα εφαρμογών.

Για να μπορούμε όμως να καταλήξουμε σε πιο συγκεκριμένα συμπεράσματα θα πρέπει να γίνουν περισσότερες δοκιμές. Από τα προβλήματα που συναντήσαμε κατά την πειραματική διάταξη μπορούμε να πούμε πως ο συγκεκριμένος εξομοιωτής (Marss x86) δεν είναι και ο πιο κατάλληλος για πειραματική μελέτη PIM αρχιτεκτονικών γιατί δεν υποστηρίζει αυτό το είδος οργάνωσης. Εκμεταλλευόμενοι όμως την δυνατότητα του για εξομοίωση μηχανών πολυπύρηνων και ετερογενών μηχανών μπορέσαμε να προσομοιώσουμε σε κάποιο βαθμό την οργάνωση της *FlexRAM*. Καλό θα ήταν τα ίδια ή παρόμοια πειράματα να γίνουν με την βοήθεια κάποιου εργαλείου που να υποστηρίζει αυτά τα μοντέλα. Αυτό γιατί θα ήταν πολύ σημαντικό να εξεταστεί η συμπεριφορά των εφαρμογών για πολύ περισσότερα *cores/processing elements*. Με αυτό τον τρόπο θα μπορούσαμε να καταλήξουμε και σε συμπεράσματα για το πόσο εύκολα επεκτάσιμες είναι οι μηχανές αυτές.

Στην παρούσα διπλωματική εργασία εξετάστηκαν μόνο 2 εφαρμογές οι οποίες έδωσαν μια καλή εικόνα για τα χαρακτηριστικά που πρέπει να έχει μια εφαρμογή για να μπορέσει να εκμεταλλευτεί τα πλεονεκτήματα μιας PIM αρχιτεκτονικής. Πρέπει όμως να δοκιμαστεί ένα μεγαλύτερος εύρος εφαρμογών για να μπορούμε με περισσότερα στοιχεία να καταλήξουμε σε σαφή συμπεράσματα για τα επιθημιτά χαρακτηριστικά των εφαρμογών. Ακόμα θα πρέπει να δοκιμαστεί και να εξεταστεί η

συμπεριφορά των εφαρμογών για πολύ μεγαλύτερα δεδομένα εισόδου για να μπορούμε να πούμε αν αλλάζει κάτι στα αποτελέσματα, αν έχουμε λογαριθμική μείωση του χρόνου σε σχέση με απλές Multicore μηχανές ή αν υπάρχει κάποιο όριο.

Σε μια μελλοντική εργασία θα ήταν επιθυμητό να δοκιμαστούν οι αρχιτεκτονικές αυτές όχι μόνο από πλευράς επίδοσης (χρόνοι εκτέλεσης εφαρμογών), αλλά και από την οπτική γωνία της κατανάλωσης ενέργειας και της ενέργειας που χρειάζεται για να μην υπερθερμάνετε μια τέτοια μηχανή. Άλλωστε αυτό είναι ένα άλλο μεγάλο ζήτημα στην ενότητα της αρχιτεκτονικής υπολογιστών.

9.Βιβλιογραφία

1. Patterson, David, et al. "A case for intelligent RAM." *Micro, IEEE* 17.2 (1997): 34-44.

2. Fraguela, Basilio B., et al. "Programming the FlexRAM parallel intelligent memory system." *ACM SIGPLAN Notices*. Vol. 38. No. 10. ACM, 2003.

3. Yoo, Seung-Moon, et al. "FlexRAM architecture design parameters." *Center for Supercomputing Research and Development (CSRD), Tech. Rep* 1584 (2000).

4. La Coss, Jeff. "The DIVA emulator: Accelerating architecture studies for PIM-based systems." *Intelligent Memory Systems*. Springer Berlin Heidelberg, 2001. 179-182.

5. Hall, Mary, and Craig Steele. "Memory management in a PIM-based architecture." *Intelligent Memory Systems*. Springer Berlin Heidelberg, 2001. 104-121.

6. Kirsch, Graham. "Active Memory: Micron? s Yukon." *Parallel and Distributed Processing Symposium, International*. IEEE Computer Society, 2003.

7. Upchurch, Ed, Thomas Sterling, and Jay Brockman. "Analysis and modeling of advanced pim architecture design tradeoffs." *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*. IEEE, 2004.

8. Huang, Tsung-Chuan, and Slo-Li Chu. "SAGE: a new analysis and optimization system for FlexRAM architecture." *Intelligent Memory Systems*. Springer Berlin Heidelberg, 2001. 160-168.

9. Mai, Ken, et al. *Smart memories: A modular reconfigurable architecture*. Vol. 28. No. 2. ACM, 2000.

10. Trancoso, Pedro. "In-memory parallelism for database workloads." *Euro-Par 2002 Parallel Processing*. Springer Berlin Heidelberg, 2002. 532-542.

11. Oskin, Mark, Frederic T. Chong, and Timothy Sherwood. *Active pages: a computation model for intelligent memory*. Vol. 26. No. 3. IEEE Computer Society, 1998.

12. Teller, Justin, Charles B. Silio Jr, and Bruce Jacob. "Performance characteristics of MAUI: an intelligent memory system architecture."*Proceedings of the 2005 workshop on Memory system performance*. ACM, 2005.

13. Alexander, Dong Ping Zhang1 Nuwan Jayasena, et al. "TOP-PIM: Throughput-Oriented Programmable Processing in Memory."

14. Dlugosch, Paul, et al. "An efficient and scalable semiconductor architecture for parallel automata processing." (2014): 1-1.

15. **http://www.micron.com/about/innovations/automata-processing**

16. **http://www.micronautomata.com**

17. Patel, Avadh, Furat Afram, and Kanad Ghose. "Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors." *1st International Qemu Users' Forum*. 2011.

18. Jeddeloh, Joe, and Brent Keeth. "Hybrid memory cube new DRAM architecture increases density and performance." *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 2012.