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

Τεχνικές βελτιστοποίησης μετάφρασης διευθύνσεων στον RISC-V Rocket Chip Generator

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΤΕΡΖΗΣ

Επιβλέπων : Διονύσιος Πνευματικάτος
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

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

Τεχνικές βελτιστοποίησης μετάφρασης διευθύνσεων στον RISC-V Rocket Chip Generator

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΤΕΡΖΗΣ

Επιβλέπων : Διονύσιος Πνευματικάτος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Ιουλίου 2022.

.....	.....	.....
Διονύσιος Πνευματικάτος Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Γεώργιος Γκούμας Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

Κωνσταντίνος Τερζής

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

Περίληψη

Η αρχιτεκτονική συνόλου εντολών (Instruction Set Architecture) RISC-V αναπτύχθηκε αρχικά στο Πανεπιστήμιο του Berkeley για ερευνητικούς και εκπαιδευτικούς σκοπούς , ενώ πλέον χρησιμοποιείται και εμπορικά. Μία από τις σημαντικότερες υλοποιήσεις του RISC-V ISA αποτελεί ο Rocket Chip Generator, που αποτελεί μια συλλογή από παραμετροποιήσιμα εργαλεία και κομμάτια ενός επεξεργαστή, τα οποία μπορούν να παράξουν από μικροεπεξεργαστές για μικρά, ενσωματωμένα συστήματα μέχρι πολυπύρηνους επεξεργαστές γενικού ή ειδικού σκοπού. Είναι υλοποιημένος στην γλώσσα Chisel, μία -απλή και ευέλικτη- υψηλού επιπέδου γλώσσα περιγραφής υλικού, που επιτρέπει την δημιουργία πολύπλοκων λογικών κυκλωμάτων τόσο για FPGA όσο και για ASIC.

Στην παρούσα διπλωματική εργασία εξετάζουμε την επίδραση του συστήματος κρυφών μνημών και συγκεκριμένα του L1 Translation Lookaside Buffer (L1 TLB) στην επίδοση ενός μονοπύρηνου συστήματος Rocket. Το TLB αποτελεί μια μικρού μεγέθους κρυφή μνήμη που αποθηκεύει τις πιο πρόσφατα χρησιμοποιημένες μεταφράσεις εικονικών σελίδων σε φυσικές. Για κάθε αστοχία TLB (TLB Miss) πρέπει να γίνει διάσχιση του Πίνακα Σελίδων, με κόστος 40-100 επεξεργαστικούς κύκλους. Για την βελτίωση της επίδοσης του TLB έχουν γίνει πολλές προτάσεις τόσο ως προς τον τρόπο με τον οποίο θα αποθηκεύονται οι μεταφράσεις όσο και μέσω της τεχνικής προανάκλησης διευθύνσεων. Στην παρούσα εργασία θα ασχοληθούμε την οργάνωση του TLB, δηλαδή με το πώς αποθηκεύονται οι μεταφράσεις στο TLB και συγκεκριμένα με το Clustered TLB, έναντι του παρόντος Sectored TLB.

Κάθε καταχώρηση του Sectored TLB χαρτογραφεί μια ομάδα εικονικών σελίδων ευθυγραμμισμένων ως προς το sector factor, σε φυσικές σελίδες,παρέχοντας ευελιξία σε σχέση με την τοπικότητα των εικονικών σελίδων με αντάλλαγμα το μεγάλο κόστος σε επίπεδο υλικού. Από την άλλη το Clustered TLB χαρτογραφεί μια ομάδα εικονικών σελίδων σε μια ομάδα φυσικών σελίδων, ευθυγραμμισμένων και οι δύο ως προς το cluster factor. Με αυτόν τον τρόπο αποθηκεύονται αρκετά λιγότερες πληροφορίες, δίνοντας την δυνατότητα αύξησης του TLB Reach και παρά την μικρότερη ευελιξία εξαιτίας των περιορισμών ευθυγράμμισης που τίθενται, σε πολλές περιπτώσεις αποδίδει ακόμη καλύτερα, καθώς αξιοποιούνται μοτίβα χωρικής τοπικότητας που παράγονται από το Λειτουργικό Σύστημα.

Η τελική πρόταση που εξετάστηκε είναι η ταυτόχρονη ύπαρξη ενός Clustered TLB με ένα απλό TLB, στο οποίο θα αποθηκεύονται οι μεταφράσεις που δεν μπορούν να αποθηκευθούν στο πρώτο λόγω περιορισμών ευθυγράμμισης που τίθενται. Η σχεδίαση αυτή βασίζεται σε ήδη υπάρχουσα πρόταση, αν και διαφέρει ελαφρώς, καθώς αντί η αναζήτηση για σύμπτυξη μεταφράσεων να γίνεται εντός ενός cache line, γίνεται μεταξύ της ζητούμενης μετάφρασης και εκείνων που είναι ήδη αποθηκευμένες στο Clustered TLB.

Ο έλεγχος της απόδοσης της σχεδίασης μας στο Alveo U250 FPGA γίνεται μέσω της σουίτας μετροπρογραμμάτων SPEC2017, της χρήσης των hardware performance counters.

Λέξεις κλειδιά

RISC-V, Κρυφή Μνήμη Αναζήτησης Διευθύνσεων, Εικονική Μνήμη, FPGA, Σχεδίαση Υλικού, Μονάδα Διαχείρισης Μνήμης, FPGA

Abstract

The RISC-V Instruction Set Architecture was originally developed at the University of Berkeley for research and educational purposes, and is now used commercially. One of the major implementations of the RISC-V ISA is the Rocket Chip Generator, which is a collection of configurable processor tools and parts, that can produce everything from microprocessors for small, embedded systems to general/special-purpose multicore processors used in more complex computing systems. It is implemented in the Chisel language, a - simple and flexible - high-level hardware description language, which facilitates the creation of complex logic circuits for both FPGAs and ASICs.

In this thesis we examine the effect of the cache system and specifically the L1 Translation Lookaside Buffer (L1 TLB) on the performance of a single-core Rocket system. TLB is a small cache that stores the most recently used virtual-to-physical page translations. For each TLB miss, the Page Table must be traversed, at a cost of 40-100 processing cycles. To improve the performance of the TLB, many proposals have been made both in terms of how the translations will be stored and through the address prefetching technique. In this thesis we will deal with the organization of the TLB, i.e. how the translations are stored in the TLB and specifically with the Clustered TLB, versus the present Sectored TLB.

Each Sectored TLB entry maps a group of virtual pages, which are aligned by the sector factor, to physical pages, providing flexibility with respect to the locality of virtual pages at the expense of large hardware-level overhead. Clustered TLB on the other hand maps a group of virtual pages to a group of physical pages, both aligned with respect to the cluster factor. In this way, significantly less information is stored, giving the possibility to increase the TLB Reach, and despite less flexibility due to the alignment restrictions, it performs even better in many cases, as spatial locality patterns generated by the Operating System are exploited.

The final proposal considered is the simultaneous existence of a Clustered TLB with a conventional TLB, which stores the translations that cannot be stored in the former due to alignment constraints set. This design is based on an already existing proposal, although it differs slightly, as instead of the search for coalescable translations being done within a cache line, it is done between the requested translation and those already stored in the Clustered TLB.

Performance testing of our design is done by using the Alveo U250 FPGA, executing the SPEC2017 benchmarks suite and using the hardware performance counters.

Key words

RISC-V, Translation Lookaside Buffer (TLB), Virtual Memory, FPGA, Hardware Design, Memory Management Unit (MMU)

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Διονύση Πνευματικάτο που μου έδωσε την δυνατότητα εκπόνησης της διπλωματικής εργασίας. Ταυτόχρονα θα ήθελα να ευχαριστήσω θερμά τους ερευνητές Νικόλα-Χαράλαμπο Παπαδόπουλο και Βασίλη Καρακώστα για την εξαιρετική επικοινωνία και καθοδήγηση καθ' όλη την διάρκεια που χρειάστηκε για την ολοκλήρωση της διπλωματικής εργασίας.

Για τα τελευταία 8 χρόνια που πέρασα στην Αθήνα, θα ήθελα να ευχαριστήσω τους φίλους μου για τις υπέροχες στιγμές που έχουμε περάσει και θα περάσουμε, καθώς και την οικογένειά μου για την αμέριστη υποστήριξη που μου δείχνει εδώ και 26 χρόνια. Τέλος, θέλω να ευχαριστήσω την Μαρία για την αγάπη που μου δίνει στα δύσκολα και στα εύκολα.

Κωνσταντίνος Τερζής,

Αθήνα, 15η Ιουλίου 2022

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

Η Αρχιτεκτονική Συνόλου Εντολών (Instruction Set Architecture – ISA) RISC-V είναι μια επεκτάσιμη αρχιτεκτονική ανοιχτού κώδικα που ξεκίνησε να αναπτύσσεται στο Πανεπιστήμιο του Berkeley το 2010. Έχοντας αυτά τα χαρακτηριστικά αποτελεί μια χρήσιμη λύση στο πεδίο της έρευνας στον τομέα της Αρχιτεκτονικής Υπολογιστών.

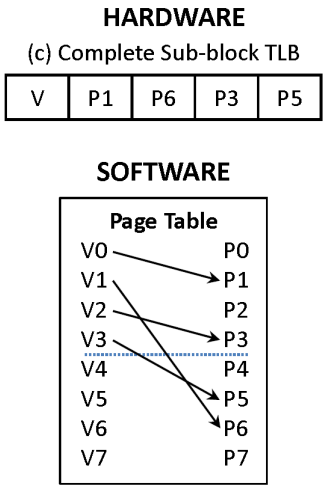
Αποτελεί εξέλιξη της ιδέας των επεξεργαστών RISC (Reduced Instruction Set Computer), που συλλήφθηκε αρχικά από τον David Patterson στο 1980 στο Πανεπιστήμιο του Berkeley, παρατηρώντας ότι το λειτουργικό σύστημα Unix όταν μεταφραζόταν για τον CISC (Complex Instruction Set Computer) μικροεπεξεργαστή Motorola 68000 χρησιμοποιούσε μονάχα το 30% του συνόλου των δυνατών εντολών.

Ο Rocket Chip Generator αποτελεί ένα παραμετροποιήσιμο μικροεπεξεργαστή που συμβαδίζει με τα επίσημα Specifications του RISC-V ISA και είναι μία από τις αρκετές υλοποιήσεις του RISC-V ISA σε επίπεδο μικροαρχιτεκτονικής. Αποτελεί μια ευέλικτη γεννήτρια συστημάτων System-on-Chip που έχει την δυνατότητα να παράγει συστήματα που ποικίλουν σε επίδοση, κατανάλωση ενέργειας ή σε ότι αφορά την απόδοσή τους. Η ανάπτυξη του Rocket ευνοείται από την high-level γλώσσα περιγραφής υλικού (Hardware Description Language – HDL), την Chisel, που είναι πρακτικά μια βιβλιοθήκη περιγραφής κυκλωμάτων ενσωματωμένη στην γλώσσα Scala.

1.1 Αντικείμενο της Διπλωματικής Εργασίας

Στην εργασία αυτή ασχολούμαστε με την μονάδα διαχείρισης μνήμης του Rocket (Memory Management Unit – MMU) και ειδικότερα με το Translation Lookaside Buffer (TLB). Το TLB αποτελεί μια μικρή κρυφή μνήμη στην οποία αποθηκεύονται οι πιο πρόσφατα χρησιμοποιημένες μεταφράσεις από εικονικές σε φυσικές διευθύνσεις μνήμης. Είναι πλήρως παραμετροποιήσιμο ως προς τον αριθμό των set και ways, με την προϋπόθεση να αποτελούν δυνάμεις του 2. [9]

Το πρόβλημα προκύπτει από τον τρόπο που αποθηκεύει τις μεταφράσεις, καθώς σύμφωνα με προηγούμενη έρευνα [11] μπορεί να επιτευχθεί η ίδια αλλά και ακόμη καλύτερη απόδοση με την αποθήκευση λιγότερης πληροφορίας στις καταχωρήσεις του (entries). Όταν μιλάμε για αποθήκευση πληροφορίας σε επίπεδο υλικού, αυτό μεταφράζεται σε μεταβολές όσον αφορά την κατανάλωση ενέργειας, το κόστος παραγωγής, την επίδοση και επιπλέον το υλικό που επιδιώκουμε να αποδεσμεύσουμε μπορεί να αξιοποιηθεί με διαφορετικό τρόπο. Η οργάνωση που υπάρχει αυτή την στιγμή (Complete Sub-block TLB ή Sectored TLB) [14] στον Rocket αποτελείται από καταχωρήσεις που χαρτογραφούν μια ομάδα γειτονικών εικονικών σελίδων σε φυσικές, χωρίς να απαιτείται οι φυσικές σελίδες να έχουν κάποια σχέση μεταξύ τους , όπως φαίνεται στην εικόνα 1.1.



Σχήμα 1.1: Οι μεταφράσεις που επιδιώκει να αξιοποιήσει το Sectored TLB

Στην πραγματικότητα όμως το Λειτουργικό Σύστημα, το οποίο είναι υπεύθυνο για την δέσμευση φυσικής μνήμης όταν αυτό ζητηθεί από κάποια εφαρμογή, συχνά παράγει από μόνο του μοτίβα κατά τα οποία ενδέχεται συνεχόμενες εικονικές σελίδες να αντιστοιχούν σε συνεχόμενες φυσικές σελίδες (continuous spatial locality) ή μια ομάδα κοντινών εικονικών σελίδων να αντιστοιχούν σε μια ομάδα κοντινών φυσικών σελίδων (clustered spatial locality). Με αυτόν τον τρόπο αντιμετωπίζεται αποτελεσματικά -σε μεγάλο βαθμό- το φαινόμενο του κατακερματισμού φυσικής μνήμης (memory fragmentation), δηλαδή η απουσία μεγάλου πλήθους συνεχόμενων, μη δεσμευμένων διευθύνσεων μνήμης. Η παρούσα οργάνωση του TLB όμως, αδυνατεί να εκμεταλλευτεί αυτά τα μοτίβα χωρικής τοπικότητας αποτελεσματικά.

Στην συγκεκριμένη διπλωματική εξετάζουμε διαφορετικές οργανώσεις του TLB που επιδιώκουν να αξιοποιήσουν τα συγκεκριμένα μοτίβα και τις συγκρίνουμε με βάση την χρήση πόρων, την επίδοσή τους ανάλογα με τις εφαρμογές που τρέχουν, καθώς και την κατανάλωση ενέργειας. [VK: explain more what you do in this thesis, 1-2 paragraphs]

1.2 Δομή Διπλωματικής Εργασίας

Στο **Κεφάλαιο 2** εξετάζεται το απαραίτητο θεωρητικό υπόβαθρο για την κατανόηση της διπλωματικής εργασίας, σχετικά με την αρχιτεκτονική συνόλου εντολών RISCV, τον Rocket Chip Generator, την εικονική μνήμη και το TLB.

Στο **Κεφάλαιο 3** αναλύονται ενδελεχώς οι διαφορετικές οργανώσεις του TLB, η οργάνωση που υποστηρίζεται ως τώρα στον Rocket καθώς και η οργάνωση που υλοποιήθηκε ως βήμα προς την βελτίωση της μετάφρασης διευθύνσεων.

Στο **Κεφάλαιο 4** παρατίθενται τα εργαλεία που χρησιμοποιήθηκαν για την εκπόνηση της διπλωματικής εργασίας.

Στο **Κεφάλαιο 5** αναλύονται τα αποτελέσματα της μελέτης μας με βάση βασικές μετρικές απόδοσης των κρυφών μνημών (caches)

Στο **Κεφάλαιο 6** προτείνονται μελλοντικές επεκτάσεις της διπλωματικής εργασίας

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό αναλύουμε την αρχιτεκτονική συνόλου εντολών RISC-V, την υλοποίησή της στον Rocket, εμβαθύνουμε θεωρητικά στις έννοιες που σχετίζονται με την Εικονική Μνήμη αλλά και στην σχετική υποστήριξη που αυτή απαιτεί σε επίπεδο υλικού. Θα πρέπει να τονιστεί πως η μελέτη αφορά την αρχιτεκτονική RISC-V, επομένως δεν μπορεί να μεταφερθεί γραμμικά σε άλλες αρχιτεκτονικές (π.χ. x86-64) και τα αποτελέσματα που θα λαμβάναμε αν μελετούσαμε άλλη αρχιτεκτονική θα ήταν πιθανώς διαφορετικά.

2.1 RISC-V Instruction Set Architecture

Η **αρχιτεκτονική συνόλου εντολών (Instruction Set Architecture – ISA)** [15] αποτελεί την διεπαφή μεταξύ του λειτουργικού συστήματος και του υλικού σε ένα υπολογιστικό σύστημα ή διαφορετικά ορίζει τον τρόπο με τον οποίο η Κεντρική Μονάδα Επεξεργασίας (Central Processing Unit – CPU) ελέγχεται από τα προγράμματα που τρέχουν σε έναν υπολογιστή. Αποτελείται από ένα σύνολο εντολών και συμβάσεων σχετικά με τον προγραμματισμό, τους υποστηριζόμενους τύπους δεδομένων, τους καταχωρητές, την διαχείριση της φυσικής μνήμης, την υποστήριξη ή όχι της εικονικής μνήμης.

Η **Αρχιτεκτονική Συνόλου Εντολών RISC-V** είναι μια επεκτάσιμη αρχιτεκτονική ανοιχτού κώδικα που ξεκίνησε να αναπτύσσεται στο Πανεπιστήμιο του Berkeley το 2010. Έχοντας αυτά τα χαρακτηριστικά αποτελεί μια χρήσιμη λύση στην έρευνα πάνω στην Αρχιτεκτονική Υπολογιστών, αλλά ταυτόχρονα τα τελευταία χρόνια έχει αρχίσει να τραβάει τα βλέμματα και για εμπορική χρήση. Το RISC-V ISA αποτελεί μια αρχιτεκτονική λίγων και απλών εντολών με το RV32I: RISC-V Base Integer ISA να περιέχει μόλις 48 εντολές. Είναι ανοιχτό υπό την έννοια ότι δεν υπόκειται σε κάποια άδεια χρήσης και δεν απαιτείται κάποιο αντίτιμο για την χρήση του. Το RISC-V Instruction Manual ορίζει 2 Specifications, **το Volume I: User-Level ISA** [6] στο οποίο περιέχονται τα σετ εντολών που επιτρέπεται να περιέχει ένα πρόγραμμα που τρέχει σε χώρο χρήστη, καθώς και το **Volume II: Privileged Architecture** [7]στο οποίο περιγράφεται πιο λεπτομερώς η αρχιτεκτονική συνόλου εντολών, επιπλέον επίπεδα εκτέλεσης με διαφορετικά δικαιώματα το καθένα, καθώς και τα χαρακτηριστικά τους για την διαχείριση ενός συστήματος.

Μια υλοποίηση της συγκεκριμένης Αρχιτεκτονικής Συνόλου Εντολών αποτελεί **ο Rocket Chip Generator**, μια γεννήτρια System-on-Chip (SoC) ανοιχτού κώδικα που παράγει παραμετροποιήσιμα RISC-V SoC ως προς μια πληθώρα λειτουργιών. Είναι γραμμένος πάνω στην γλώσσα Chisel, μία επέκταση της Scala, η οποία διευκολύνει την περιγραφή πολύπλοκων και παραμετροποιήσι-

μων ψηφιακών κυκλωμάτων, επεξεργαστικών πυρήνων, κρυφών μνημών και την διασύνδεση όλων των απαραίτητων μονάδων.

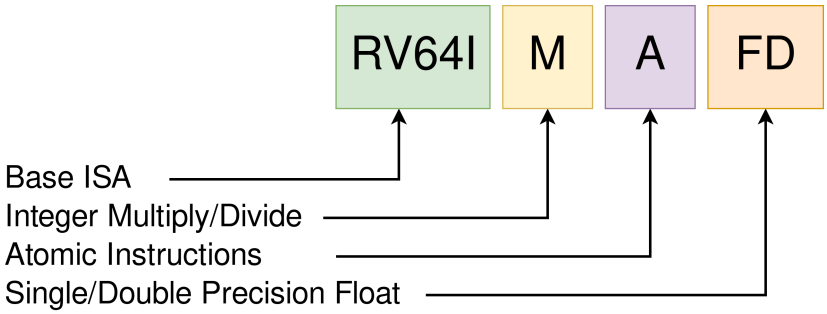
2.1.1 RISC-V User Level ISA

Το user level ISA αποτελείται από τα RV32/64/128 Integer Base ISA με εικονικές διευθύνσεις 32, 64 και 128 bit αντίστοιχα. Παράλληλα ορίζονται και επιπλέον Extensions τα οποία προσθέτουν εξειδικευμένες εντολές ή επιπλέον λειτουργικότητα, με σκοπό την συνολική βελτίωση της επίδοσης του συστήματος. Με αυτό τον τρόπο κατά την σχεδίαση ενός επεξεργαστή ειδικού σκοπού, όπως για παράδειγμα ενός επιταχυντή επεξεργασίας δεδομένων μηχανικής μάθησης για τα οποία απαιτούνται περίπλοκες πράξεις πινάκων, είναι δυνατή η προσθήκη εντολών που επιταχύνουν τις συγκεκριμένες πράξεις. Πέραν των επίσημων extension που παρουσιάζονται στον πίνακα 2.1 και ελέγχονται από το RISC-V Foundation[1], υπάρχει η δυνατότητα προσθήκης custom Extensions. Σε περίπτωση που τα extensions που χρησιμοποιούνται δεν επαρκούν για την εκτέλεση κάποια εφαρμογής χώρου χρήστη, εξαιτίας αδυναμίας υποστήριξης κάποιας εντολής, τότε προκύπτει ένα illegal instruction exception και η εντολή που το πυροδότησε προσομοιώνεται σε emulation routines.

Extension	Περιγραφή
M	Integer Multiplication and Division
A	Atomic Instructions
F	Single-Precision Floating Point Instructions
D	Double-Precision Floating Point Instructions
Q	Quad-Precision Floating-Point
C	Compressed Instructions
V	Vector Operations

Πίνακας 2.1: Βασικότερα Extensions

Τα Extensions που είναι απαραίτητα για την υποστήριξη ενός Λειτουργικού Συστήματος όπως για παράδειγμα το Linux είναι τα I (Integer), M (Integer Multiplication and Division), A(Atomic Instructions), F (Single-Precision Floating Point Instructions), D (Double-Precision Floating Point Instructions) με συμβολισμό **G (General)**. Το **RV64G ISA** είναι το βασικό ISA που θα χρησιμοποιηθεί κατά την διάρκεια της παρούσας εργασίας.



Σχήμα 2.1: Συμβολισμός RV64IMAFD Core

[1] https://www.riscv.org

2.1.2 RISC-V Privileged Architecture

Εκτός από την υποστήριξη εφαρμογών που τρέχουν στον χώρο χρήστη, υπάρχει ανάγκη υποστήριξης επιπλέον λειτουργικότητας για την διαχείριση του συστήματος. Οι σημαντικότεροι λόγοι ύπαρξης επιπρόσθετων επιπέδων εκτέλεσης με επιπλέον προνόμια είναι η διαχείριση και προστασία κοινών πόρων όπως η μνήμη και οι συσκευές, η υποστήριξη multitasking, καθώς και η απόκρυψη της υλοποίησης του υλικού, για μεγαλύτερη ευχέρεια ανάπτυξης λογισμικού

2.1.2.1 Privilege Levels

Το Privileged Architecture Specification ορίζει επιπλέον Privilege Levels, όπως βλέπουμε στον πίνακα 2.2:

Επίπεδο	Όνομα	Συντομογραφία
3	Machine	M-mode
2	για μελλοντική χρήση	
1	Supervisor	S-mode
0	User	U-mode

Πίνακας 2.2: Privilege Levels

Το κάθε επίπεδο προσθέτει επιπλέον **λειτουργικότητα, εντολές και καταχωρητές κατάστασης και ελέγχου, Control and Status Registers (CSRs)**. Κάθε ένα από αυτά έχει πλήρη πρόσβαση στα χαμηλότερα επίπεδα ενώ το αντίθετο ισχύει μονάχα υπό προϋποθέσεις: είτε για την προώθηση των διακοπών σε υψηλότερα επίπεδα, είτε την πρόσβαση στους performance counters. Με τον συνδυασμό των Privilege Levels, μπορούμε να κατασκευάσουμε συστήματα που ποικίλουν σε πολυπλοκότητα και λειτουργικότητα. Ο πίνακας 2.3 συνοψίζει αυτούς τους συνδυασμούς:

Supported Modes	Παράδειγμα Συστήματος
M	Απλά Ενσωματωμένα Συστήματα
M + U	Ασφαλή Ενσωματωμένα Συστήματα
M + U + S	Συστήματα με υποστήριξη λειτουργικού συστήματος
M + Virtual[U + S]	Συστήματα με υποστήριξη για πολλαπλα λειτουργικά συστήματα

Πίνακας 2.3: Πιθανά σενάρια στησίματος ενός RISC-V συστήματος

2.1.2.2 Machine-Mode

Το M-mode αποτελεί το ανώτερο επίπεδο δικαιωμάτων υπό την έννοια ότι είναι το μόνο επίπεδο το οποίο έχει πλήρη πρόσβαση στην αρχιτεκτονική και γι' αυτό πρέπει να υπάρχει σε κάθε υλοποίηση. Έτσι, κώδικας είτε λάθος είτε κακόβουλος που τρέχει σε M-mode καθιστά το σύστημα ανασφαλές και θα πρέπει να τρέχει σε U-Mode .

Παρέχει βασικούς καταχωρητές για τον έλεγχο του συστήματος, Control and Status Registers (CSRs), καθώς και επιπλέον καταχωρητές μέτρησης επίδοσης (Performance Counters). Οι Performance Counters μπορούν να μετράνε γεγονότα όπως ο αριθμός των interrupts, ο συνολικός αριθμός των κύκλων ρολογιού ή τα L1/L2 dTLB Misses. Για κάθε Performance Counter υπάρχει ένας Hardware

Performace Monitor Event Register, στον οποίο φορτώνεται ένας συγκεκριμένος κωδικός γεγονότος υλικού από μια πληθώρα επιλογών, καθεμία από τις οποίες αντιστοιχεί σε ένα διαφορετικό hardware performance event ενημερώνοντας τους performance counters.

Επιπλέον, το M-mode προσθέτει υποστήριξη για διακοπές/traps απαραίτητες για την ύπαρξη συσκευών και για την διαχείριση σφαλμάτων και γεγονότων όπως τα:

- **Access Faults**: Σφάλματα σελίδας ή ελλιπών δικαιωμάτων πρόσβασης στην μνήμη
- **Breakpoints**: Για την ευκολότερη αποσφαλμάτωση
- **Environment Calls**: Κλήσεις συστήματος για διαχείριση ενεργειών που απαιτούν περισσότερα δικαιώματα από ανώτερη βαθμίδα
- **Illegal Instructions**: Εντολές που προσπαθούν να εκτελεστούν με λιγότερα δικαιώματα, ανύπαρκτες εντολές καθώς και εντολές που δεν είναι υλοποιημένες στην μικροαρχιτεκτονική. Έγκυρες αλλά όχι υλοποιημένες στην μικροαρχιτεκτονική εντολές εξυπηρετούνται από ρουτίνες εξομοίωσης.
- **Misaligned address accesses**: Σφάλματα στοίχισης διευθύνσεων, στην αρχιτεκτονική RISC-V δεν επιτρέπονται misaligned διευθύνσεις για την απλοποίηση της μικροαρχιτεκτονικής.

2.1.2.3 Supervisor-Mode

Το S-mode προσθέτει τα βασικά χαρακτηριστικά που απαιτούνται για την υποστήριξη ενός λειτουργικού συστήματος. όπως είναι η διαχείριση εικονικής μνήμης ή οι κλήσεις συστήματος. Στην υποενότητα 2.6 αναλύουμε εκτενώς το σύστημα διαχείρισης εικονικής μνήμης στον Rocket.

2.1.2.4 User-mode

Το U-mode επιτρέπει την εκτέλεση του αναξιόπιστου κώδικα στο επίπεδο του χώρου χρήστη για την προστασία της λειτουργίας του συστήματος. Κώδικας που τρέχει σε U-mode δεν έχει πρόσβαση στους privileged CSRs καθώς και δεν επιτρέπεται να εκτελέσει privileged εντολές. Το U-mode προσθέτει τον μηχανισμό της φυσικής προστασίας μνήμης, **Physical Memory Protection (PMP)**. Το PMP χωρίζει την μνήμη του συστήματος σε περιοχές στις οποίες ο κώδικας του U-mode έχει συγκεκριμένα δικαιώματα πρόσβασης[13]. Ο μηχανισμός PMP επομένως προσθέτει επιπλέον βαθμίδες ασφαλείας για το σύστημα. Πέρα από το PMP ορίζεται και ο μηχανισμός Physical Memory Attributes (PMA) ο οποίος οριοθετεί περιοχές μνήμης και ορίζει τα χαρακτηριστικά τους[2].

2.2 Rocket Chip Generator

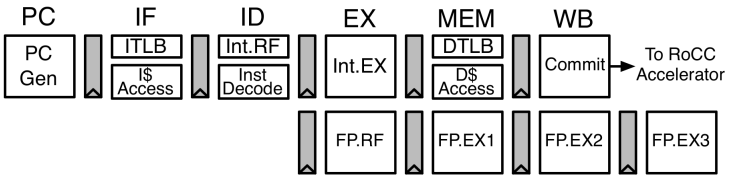
Ο Rocket Chip Generator [1] αποτελεί μια ανοιχτού κώδικα γεννήτρια System-on-Chip. Είναι υλοποιημένος πάνω στην Chisel και παράγει μικροεπεξεργαστές γενικού σκοπού οι οποίοι χρησιμοποιούν το RISC-V ISA. Μπορεί να παράξει τόσο in-order μικροεπεξεργαστές (Rocket) όσο και out-of-order (BOOM)[5]. Άλλες υλοποιήσεις του RISC-V ISA είναι το PULP Project - ETH

[2] Atomicity PMAs, Memory-Ordering PMAs, Coherency and Cacheability PMAs και λοιπά. Βλέπε RISC-V Instruction Set Manual, Volume II: Privileged Architecture version 1.10 παράγραφος 3.5

Zurich[12] και ο SHAKTI Processor Program - IIT Madras[8]. Τέλος, ο Rocket Chip Generator υποστηρίζει την ενσωμάτωση επιταχυντών (accelerators) στην μορφή επέκτασης εντολών και παρέχει μια σειρά παραμετροποιήσιμων components μικροεπεξεργαστών όπως Caches, TLBs, branch predictors, floating point units.

2.2.1 Rocket Core

Ο Rocket είναι μία 6-σταδίων βαθμωτή γεννήτρια επεξεργαστικών πυρήνων που υλοποιούν το RV32G και RV64G ISA. Το σύστημα διαχείρισης εικονικής μνήμης, Memory Management Unit (MMU), υποστηρίζει σελιδοποίηση εικονικής μνήμης (page-based virtual memory). Ο Rocket έχει παραμετροποιήσιμη Data Cache καθώς και παραμετροποιήσιμο σύστημα πρόβλεψης διακλάδωσης (branch predictor) στο πρώτο στάδιο (front-end). Για την επεξεργασία αριθμών κινητής υποδιαστολής ο Rocket χρησιμοποιεί τις υλοποιήσεις μονάδων κινητής υποδιαστολής σε Chisel (berkeley hardfloat) [3]. Ο Rocket επίσης υποστηρίζει τα RISC-V Machine, User και Supervisor privilege levels. Υπάρχει δυνατότητα παραμετροποίησης των περιεχόμενων extensions (M, A, F, D), των σταδίων σωλήνωσης των μονάδων κινητής υποδιαστολής, των χαρακτηριστικών (associativity, μέγεθος) των κρυφών μνημών και του TLB. Τέλος, ο Rocket μπορεί να παρουσιαστεί σαν βιβλιοθήκη εξαρτημάτων επεξεργαστών, καθώς αρκετά μέρη του όπως οι κρυφές μνήμες, το TLB, ο Page Table Walker και το Control and Status Register File χρησιμοποιούνται από άλλες υλοποιήσεις όπως ο BOOM.



Σχήμα 2.2: Στάδια διοχέτευσης στον Rocket Core

2.3 Η γλώσσα περιγραφής υλικού Chisel

2.3.1 Βασικά στοιχεία της Chisel

Η **Chisel (Constructing Hardware In a Scala Embedded Language)** [2] αποτελεί μια υψηλού επιπέδου γλώσσα περιγραφής υλικού βασισμένη στην γλώσσα προγραμματισμού Scala, που πρακτικά αποτελεί μια βιβλιοθήκη της. Αποτελεί μια προσπάθεια να ξεπεραστούν προβλήματα που παρουσιάζουν οι κλασσικές γλώσσες περιγραφής υλικού (Verilog, VHDL) οι οποίες παράγουν στατικά κυκλώματα. Αυτό επιτυγχάνεται αξιοποιώντας τεχνικές γλωσσών προγραμματισμού όπως ο αντικειμενοστραφής προγραμματισμός, ο συναρτητησιακός προγραμματισμός κ.ά.

Ορίζονται απλές και συγκεκριμένες δομές που χρησιμοποιούνται στην Ψηφιακή Σχεδίαση σχεδίασης, όπως είναι τα Wire(καλώδιο), Vec(διάνυστμα), Reg(καταχωρητής), Mem(μνήμη), Mux(πολυπλέκτες). Με την χρήση των παραπάνω και τον συνδυασμό τους διευκολύνεται και επιταχύνεται η περιγραφή κυκλωμάτων, καθώς επίσης καθίσταται δυνατή η δημιουργία γεννητριών κυκλωμάτων,

[3] https://github.com/ucb-bar/berkeley-hardfloat

ανάλογα τόσο με τις παραμέτρους που επιλέγονται, όσο και με την δυνατότητα της γλώσσας να αφαιρεί από το κύκλωμα δομές οι οποίες δεν χρησιμοποιούνται.

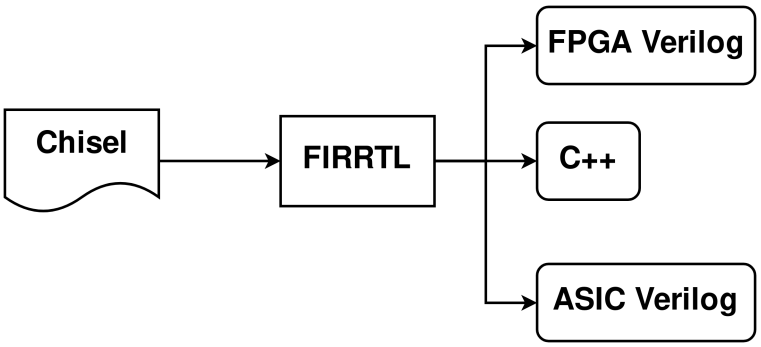
2.3.2 Ευέλικτη ενδιάμεση αναπαράσταση RTL

Η Chisel έπειτα από την μετάφρασή της μπορεί να παράξει είτε Verilog είτε C++. Η διαδικασία αυτή περνάει μέσα από τρία στάδια:

1. Η Chisel μεταφράζεται αρχικά σε κυκλωματική ενδιάμεση αναπαράσταση RTL, η οποία ονομάζεται Ευέλικτη ενδιάμεση αναπαράσταση RTL, Flexible Intermediate Representation for RTL (FIRRTL).

2. Γίνεται έλεγχος του FIRRTL και εφαρμόζονται πάνω του βελτιστοποιήσεις, όπως είναι η αφαίρεση πόρων ή δομών που δεν χρησιμοποιούνται.

3. Με βάση το βελτιστοποιημένο FIRRTL προκύπτει Verilog ή C++. Εάν ο έλεγχος του FIRRTL είναι επιτυχής τότε μπορεί να υλοποιηθεί σε FPGA ή ASIC.



Σχήμα 2.3: Παραγωγή κώδικα για διαφορετικές πλατφόρμες από την Chisel

2.4 Βασικοί όροι και έννοιες

- **Εικονική Μνήμη (Virtual Memory):** Η τεχνική της Εικονικής Μνήμης εισήχθη προκειμένου να δώσει παραπάνω ευελιξία στην εκάστοτε εφαρμογή ως προς το πόση μνήμη μπορεί να χρησιμοποιήσει, αυξάνοντας σε σημαντικό βαθμό τον χρόνο εκτέλεσής της σε συστήματα που εκτελούνται ταυτόχρονα παραπάνω από μία εφαρμογές. Κάθε εφαρμογή που εκτελείται έχει την δική της εικονική μνήμη που αποτελείται από συνεχόμενες διευθύνσεις, η οποία αντιστοιχεί σε κάποιο χώρο στην φυσική μνήμη που δεν είναι απαραίτητο να είναι συνεχόμενος.

- **Εικονικός Χώρος Διευθύνσεων (Virtual Address Space):** Είναι το σύνολο από εικονικές διευθύνσεις μνήμης και κάθε εφαρμογή έχει τον δικό της εικονικό χώρο διευθύνσεων. Το μέγεθός του εξαρτάται από το μήκος σε διευθύνσεων σε bit. Αν το μήκος των διευθύνσεων έχει μήκος n bit, τότε το μέγεθός του είναι $2^{n-3}byte$.

- **Φυσικός Χώρος Διευθύνσεων (Physical Address Space):** Είναι το σύνολο από φυσικές διευθύνσεις μνήμης που βρίσκονται στην Μνήμη Τυχαίας Προσπέλασης (Random Access

Memory – RAM). Το Λειτουργικό Σύστημα είναι υπεύθυνο για να την κατανείμει στις εφαρμογές ανάλογα με τις ανάγκες τους και ταυτόχρονα διαθέτει μηχανισμούς με τους οποίους δημιουργούνται μοτίβα στην αντιστοίχιση εικονικών σε φυσικές διευθύνσεις, τα οποία καλούμαστε και να αξιοποιήσουμε.

- **Σελιδοποίηση (Paging)**: Είναι η διαδικασία μέσω της οποίας ο εικονικός και ο φυσικός χώρος διευθύνσεων κατατμίζονται σε συνεχή τμήματα σταθερού μεγέθους, που ονομάζονται σελίδες (pages). Η εικονική μνήμη χωρίζεται σε εικονικές σελίδες (virtual pages), ενώ η φυσική μνήμη χωρίζεται σε φυσικές σελίδες (physical pages ή page frames). Όπως συμβαίνει στην πλειοψηφία των σύγχρονων αρχιτεκτονικών, έτσι και στην RISC-V, το ελάχιστο μέγεθος σελίδας που υποστηρίζεται είναι τα 4 KB (4096 byte).

2.5 Εικονική Μνήμη

Η χρήση της Εικονικής Μνήμης έρχεται να επιλύσει μια σειρά προβλημάτων ως προς το γεγονός ότι η φυσική μνήμη είναι πεπερασμένη, ότι μετά από κάποιο χρόνο λειτουργίες ο διαθέσιμος χώρος είναι ασυνεχής (εξωτερικός κατακερματισμός φυσικής μνήμης), όσο και ως προς την διαχείρισή της. Ο τρόπος που το κάνει αυτό είναι δίνοντας την εντύπωση στο κάθε πρόγραμμα που εκτελείται πως έχει διαθέσιμη ανά πάσα στιγμή ολόκληρη την μνήμη, κάνοντας πιο αποδοτική την εκτέλεσή του. Ταυτόχρονα, οι διάφορες εφαρμογές εκτελούνται απομονωμένα, δηλαδή δεν διαχειρίζονται κοινές περιοχές στην μνήμη, αυξάνοντας την ασφάλεια εκτέλεσης και βελτιώνοντας σημαντικά την χρήση της μνήμης.

Τόσο η φυσική όσο και εικονική μνήμη χωρίζονται σε **σελίδες**, δηλαδή σύνολα διευθύνσεων σταθερού μεγέθους τα οποία μπορούν να διευθυνσιοδοτηθούν από τα υψηλότερα bit μιας διεύθυνσης. Συνήθως το μικρότερο μέγεθος σελίδας είναι τα 4KB, αλλά υπάρχουν και σελίδες με μέγεθος 2MB ή 1GB. Σε κάθε περίπτωση το μέγεθος μιας σελίδας πρέπει να είναι πολλαπλάσιο του μικρότερου μεγέθους σελίδας που υποστηρίζεται από τον επεξεργαστή. Η διαδικασία με την οποία η φυσική ή η εικονική μνήμη χωρίζεται σε σελίδες ονομάζεται **σελιδοποίηση**. Με αυτό τον τρόπο δίνεται η δυνατότητα σε ένα πρόγραμμα να χρησιμοποιεί κομμάτια της φυσικής μνήμης τα οποία δεν είναι συνεχόμενα.

Παρόλα αυτά η χρήση της εικονικής μνήμης δημιουργεί και αρνητικές επιπτώσεις στην λειτουργία ενός συστήματος. Αυτές προκύπτουν από το γεγονός ότι οι εικονικές διευθύνσεις που χρησιμοποιεί κάθε πρόγραμμα πρέπει να μεταφραστούν σε φυσικές διευθύνσεις. Η διαδκασία αυτή είναι αρκετά επιβαρυντική τόσο ενεργειακά, όσο και για την απόδοση ενός υπολογιστή, καθώς ξοδεύονται πολλοί κύκλοι ρολογιού της Κεντρικής Μονάδας Επεξεργασίας (ΚΜΕ – CPU).

Η δομή που αναλαμβάνει να κάνει αυτή την μετάφραση είναι ο **Πίνακας Σελίδων (Page Table)**, η διαχείριση του οποίου γίνεται από το Λειτουργικό Σύστημα και βρίσκεται σε έναν ειδικό χώρο στην φυσική μνήμη. Κάθε εφαρμογή που εκτελείται έχει τον δικό της Πίνακα Σελίδων που μεταφράζει τις εικονικές διευθύνσεις της συγκεκριμένης εφαρμογής σε φυσικές διευθύνσεις. Όμως λόγω του μεγάλου μεγέθους του καθώς και επειδή βρίσκεται στην αργή (σε σχέση με τις caches που είναι πιο «κοντά» στην CPU) φυσική μνήμη, η αναζήτηση σε αυτόν είναι αρκετά κοστοβόρα σε κύκλους ρολογιού.

Έτσι μια νέα δομή, **το TLB (Translation Lookaside Buffer) αναλαμβάνει να αποθηκεύσει τις πιο πρόσφατες μεταφράσεις σελίδων από την εικονική στην φυσική μνήμη**. Το TLB βρίσκεται στις Level1 και Level2 Caches του επεξεργαστή. Η απόδοσή του είναι εξαιρετικά σημαντική καθώς καθορίζει το κατά πόσο θα επιταχυνθεί η διαδικασία της μετάφρασης.

2.5.1 Διαδικασία μετάφρασης

Προκειμένου να γίνει ορθή χρήση της εικονικής μνήμης απαιτείται η αρμονική συνύπαρξη και επικοινωνία μηχανισμών τόσο σε επίπεδο υλικού, όσο και σε επίπεδο λογισμικού. Κάθε δομή του συστήματος έχει σαφείς λειτουργίες και αρμοδιότητες.

Κάθε φορά που μια εφαρμογή επιχειρεί μια πρόσβαση στην μνήμη στέλνει ένα αίτημα στην Κεντρική Μονάδα Επεξεργασίας (CPU) μέσω του Λειτουργικού Συστήματος αναγράφοντας τον εικονικό αριθμό σελίδας (Virtual Page Number – VPN) που θέλει να προσπελάσει και περιμένει τον φυσικό αριθμό σελίδας (Physical Page Number – PPN) στον οποίο αντιστοιχεί αυτό το αίτημα. Έπειτα η ΚΜΕ προωθεί αυτό το αίτημα στο TLB, μαζί με έναν αναγνωριστικό κωδικό που αντιστοιχεί στην εφαρμογή η οποία έκανε το συγκεκριμένο αίτημα. Στην περίπτωση όπου η αναζήτηση στο TLB είναι επιτυχημένη (TLB hit) ο ζητούμενος αριθμός σελίδας επιστρέφεται άμεσα στην εφαρμογή και η εξυπηρέτηση του αιτήματος συνεχίζεται. Στην περίπτωση όπου η αναζήτηση στο TLB είναι άστοχη (TLB miss) πρέπει να γίνει διάσχιση του Πίνακα σελίδων (Page Table Walk) της εν λόγω εφαρμογής προκειμένου να βρεθεί η ζητούμενη μετάφραση. Η παραπάνω διαδικασία περιγράφεται αναλυτικότερα στην ενότητα 2.7.

Στις σύγχρονες αρχιτεκτονικές οι Πίνακες Σελίδων αποτελούνται από πολλά επίπεδα, καθένα από τα οποία απαιτεί μία πρόσβαση στην μνήμη, κάτι που αναδεικνύει την σημασία σωστής σχεδίασης του TLB για να περιοριστούν όσο το δυνατόν περισσότερο τα TLB misses.

2.6 Μονάδα Διαχείρισης Εικονικής Μνήμης στον Rocket

Σε αυτή την ενότητα ασχολούμαστε με την μονάδα διαχείρισης εικονικής μνήμης από την σκοπιά του RISC-V ISA, όπως ορίζεται από το Privileged Architecture Specification και την υλοποίηση του στον Rocket Chip.

2.6.1 Η διαχείριση μνήμης στο RISC-V ISA

Το S-mode ορίζει σύστημα εικονικής μνήμης το οποίο χωρίζει την μνήμη σε σταθερού μεγέθους σελίδες με σκοπό την διαχείριση και την προστασία της μνήμης. Όταν το σύστημα σελιδοποίησης είναι ενεργοποιημένο οι περισσότερες διευθύνσεις είναι εικονικές (virtual address) και πρέπει να μεταφραστούν σε φυσικές διευθύνσεις (physical address) έτσι ώστε να γίνει προσπέλαση στην φυσική μνήμη.

Ο Πίνακας Σελίδων αποτελείται από μια δενδρική δομή όπου οι τελευταίοι κόμβοι (leaf-nodes) του καθορίζουν κατά πόσον μία εικονική διεύθυνσή χαρτογραφείται σε μία φυσική διεύθυνσή. Αν ισχύει αυτό ορίζει ποια Privilege levels επιτρέπεται να έχουν πρόσβαση στην σελίδα αυτή. Προσπάθεια πρόσβασης σε μία σελίδα η οποία είτε δεν είναι χαρτογραφημένη (mapped), είτε δεν υπάρχουν αρκετά δικαιώματα για την προσπέλαση της οδηγεί σε σφάλμα σελίδας (Page Fault).

Ο RISC-V ορίζει διάφορα σχήματα σελιδοποίησης, όπως βλέπουμε:

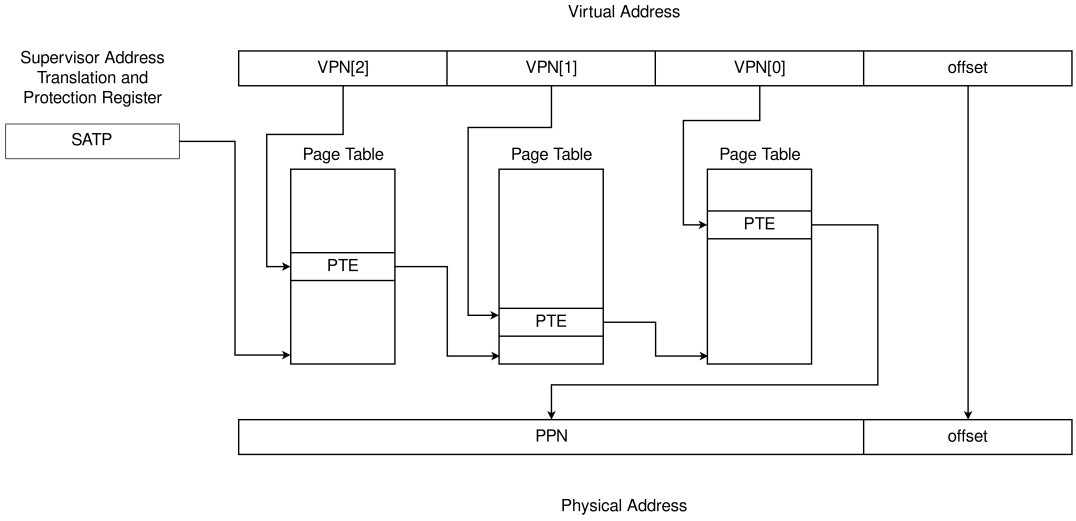
- **RV32 Paging Scheme**: 4KB base page
  - Sv32: 4GB virtual address space (2-level Page Table)
- **RV64 Paging Scheme**: 4KB base page, 2MB/1GB page support
  - Sv39: 512GB virtual address space (3-level Page Table)
  - Sv48: 256TB virtual address space (4-level Page Table)

Στην παρούσα εργασία ασχολούμαστε με το σχήμα Sv39 του RV64.

2.6.2 Σχήμα διαχείρισης μνήμης Sv39 (RV64)

Όπως αναφέρθηκε παραπάνω, το σχήμα sv39 του RV64 ορίζει μέγεθος σελίδας 4KB και Page Table 3-επιπέδων. Η δενδρική οργάνωση του Page Table του Sv39 φαίνεται στο σχήμα 2.4. Ο καταχωρητής συστήματος Supervisor Address Translation and Protection (satp) αποθηκεύει την φυσική διεύθυνση της ρίζας του Page Table. Κάθε καταχώρηση του Page Table μπορεί να είναι είτε τελικός κόμβος, είτε ένας δείκτης για την ρίζα του Page Table στο επόμενο επίπεδο. Κάθε τμήμα της εικονικής διεύθυνσης λειτουργεί σαν offset και προστίθεται στην ρίζα του Πίνακα Σελίδων για την εύρεση της επιθυμητής καταχώρησης. Η πρόσβαση σε κάθε επίπεδο κοστίζει ένα κύκλο ρολογιού με βάση την συχνότητα χρονισμού της φυσικής μνήμης, η οποία συνήθως είναι πολύ χαμηλότερη σε σχέση με εκείνη του επεξεργαστή.

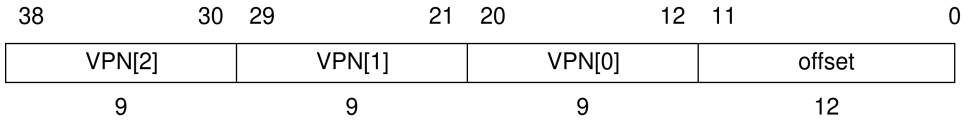
Τα Page Tables του σχήματος Sv39 περιέχουν 2^9 Page Table Entries. Επειδή όπως αναφέρθηκε οποιοδήποτε επίπεδο PTE μπορεί να είναι τελικός κόμβος (leaf node), πέρα από σελίδες μεγέθους 4KB το σχήμα Sv39 υποστηρίζει 2MB Megapages και 1GB Gigapages.



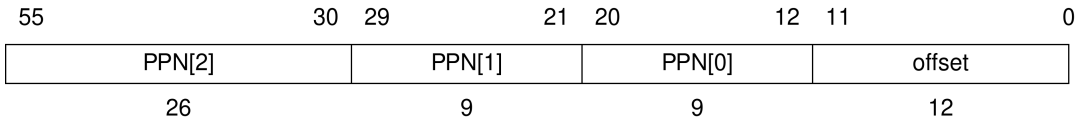
Σχήμα 2.4: Η οργάνωση του Page Table στο σχήμα sv39 (RV64)

Τα σχήματα 2.5, 2.6, 2.7 παρουσιάζουν την δομή μίας εικονικής σελίδας, μίας φυσικής σελίδας και ενός Page Table Entry αντίστοιχα. Τα 10 πρώτα bits ενός Page Table Entry είναι reserved για μελλοντική χρήση. Πέρα από τον αριθμό της φυσικής σελίδας τα υπόλοιπα πεδία του PTE είναι τα εξής:

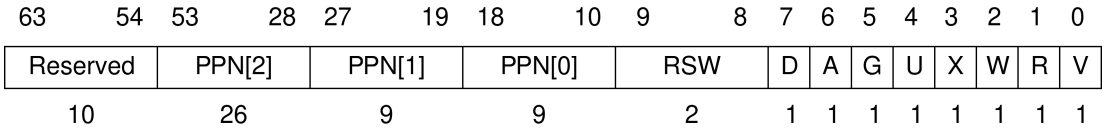
- RSW: Reserved πεδίο για μελλοντική χρήση από το S-mode
- D: Dirty bit, σημαίνει ότι έχει γίνει κάποια εγγραφή στην σελίδα
- A: Accessed bit, σημαίνει ότι η σελίδα έχει προσπελαστεί
- G: Global mapping
- U: Σελίδα που ανήκει στο U-mode
- X: Στην σελίδα επιτρέπεται εκτέλεση κώδικα
- W: Στην σελίδα επιτρέπεται εγγραφή
- R: Στην σελίδα επιτρέπεται διάβασμα δεδομένων
- V: Valid bit, η σελίδα είναι έγκυρη



Σχήμα 2.5: Εικονική σελίδα Sv39



Σχήμα 2.6: Φυσική σελίδα Sv39



Σχήμα 2.7: Sv39 Page Table Entry

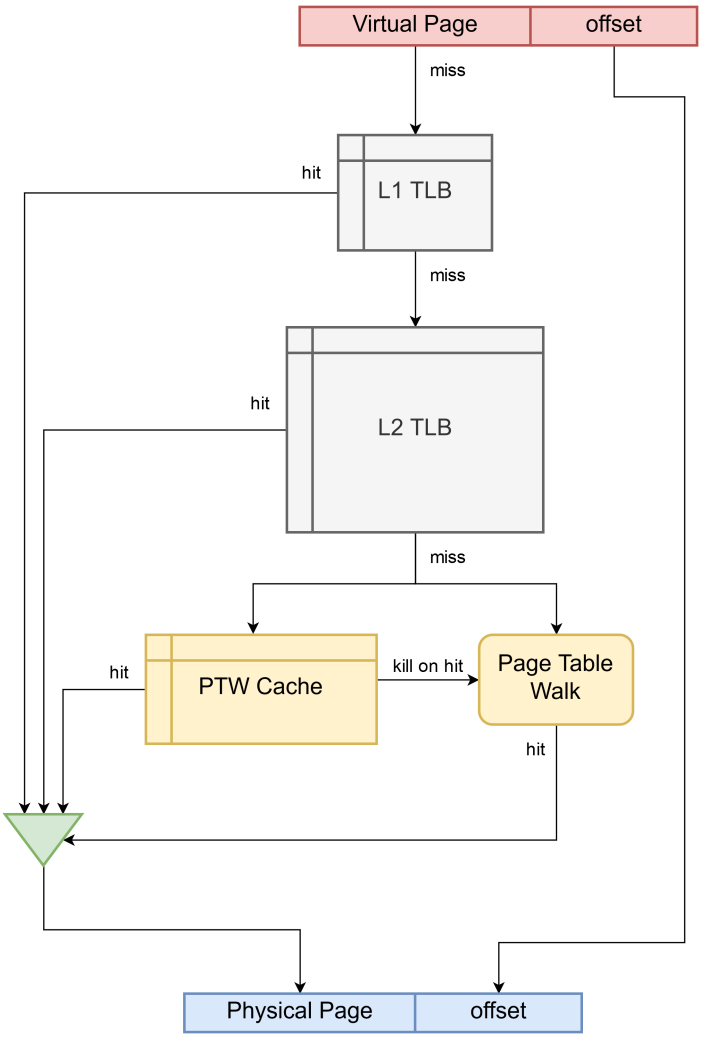
2.6.3 Η μονάδα διαχείρισης μνήμης στον Rocket Chip Generator

Το σύστημα διαχείρισης εικονικής μνήμης του Rocket Chip Generator έχει υλοποιημένη μονάδα Page Table Walk η οποία διασχίζει τον Πίνακα Σελίδων για την εύρεση μεταφράσεων από την εικονική στην φυσική μνήμη. Επειδή οι αναζητήσεις αυτές προσθέτουν μεγάλη καθυστέρηση είναι σκόπιμη η χρήση κρυφών μνημών του Πίνακα Σελίδων (Page Table Walk Cache) οι οποίες κρατάνε τις ενδιάμεσες μεταφράσεις, βελτιώνοντας την επίδοση. Η διαδικασία μετάφρασης εικονικών διευθύνσεων του Rocket Chip φαίνεται στο σχήμα 2.8. Με παραμετροποιήσιμο των αριθμό των Entries, set και ways υποστηρίζονται:

1. **Set associative L1 TLB**: Μικρό και γρήγορο πρώτου επιπέδου TLB με κόστος hit ένα κύκλο.

2. **Set associative L2 TLB**: Μεγαλύτερο αλλά πιο αργό δευτέρου επιπέδου TLB με κόστος hit 2 κύκλους.

3. **Fully-associative Page Table Walk Cache**: Είναι ενσωματωμένη στην μονάδα του Page Table Walker και κρατάει non-leaf μεταφράσεις των τριών επιπέδων του Page Table με κόστος hit ένα κύκλο ανά επίπεδο.



Σχήμα 2.8: Διαδικασία μετάφρασης εικονικών διευθύνσεων στον Rocket Chip Generator

2.7 Οι βασικές λειτουργίες του TLB και οι διεπαφές του

Τα Data/Instruction TLB είναι δύο διαφορετικές μονάδες οι οποίες βασίζονται στο ίδιο template του Rocket Chip, και είναι αντίστοιχα συνδεδεμένα με την Data και την Instruction Cache. Επίσης τα Data/Instruction TLB έχουν ιδιωτικό Page Table Walker (PTW) σε περίπτωση αστοχίας TLB, για την εύρεση της φυσικής διεύθυνσης με την προσπέλαση του Page Table. Χωρίς βλάβη της γενικότητας, από εδώ και έπειτα όταν αναφερόμαστε στην έννοια του TLB αναφερόμαστε στο Data TLB.

Όπως έχει αναφερθεί, το TLB καλείται να αποθηκεύσει τις πιο πρόσφατες μεταφράσεις εικονικών σε φυσικές διευθύνσεις. Δέχεται στην είσοδό του την ζητούμενη εικονική σελίδα, αναζητά

με βάση αυτή αν υπάρχει αποθηκευμένη έγκυρη μετάφραση και σε αντίθετη περίπτωση στέλνει ένα αίτημα προκειμένου να γίνει διάσχιση του πίνακα σελίδων (Page Table Walk – PTW) του προγράμματος που αιτήθηκε την εικονική σελίδα.

Έτσι οι βασικές λειτουργίες που υλοποιούνται είναι οι εξής:

- **Lookup**: Αναζήτηση του ζητούμενου VPN στις αποθηκευμένες μεταφράσεις. Το VPN συγκρίνεται με τα αποθηκευμένα tag και σε περίπτωση που υπάρχει έγκυρη μετάφραση (TLB Hit) επιστρέφεται το αποθηκευμένο PPN.

- **Refill**: Σε περίπτωση άστοχης αναζήτησης (TLB Miss) κατά την διάρκεια του lookup, το TLB προωθεί το αίτημα στον PTW και εκείνος του απαντάει είτε με το ζητούμενο PPN, είτε με κάποιο Exception όπως είναι τα Page Fault ή τα Access Exception. Στην περίπτωση που η απάντηση είναι το ζητούμενο PPN, η νέα μετάφραση αποθηκεύεται στο TLB. Η επιλογή του way γίνεται με βάση την επιλεγμένη πολιτική αντικατάστασης. Ταυτόχρονα με την αποθήκευση της νέας μετάφρασης, η μονάδα του TLB επιστρέφει το ζητούμενο PPN στην DCache.

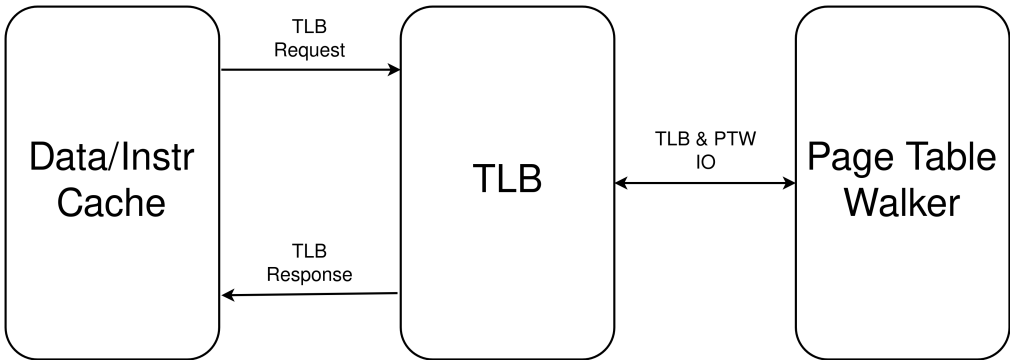
- **Invalidate**: Κατά την διάρκεια εκτέλεσης ενός προγράμματος είναι πιθανό να ζητηθεί η ακύρωση κάποιου entry στο TLB. Αυτό μπορεί να συμβεί εξαιτίας αλλαγών στα entries του Page Table. Όμως επειδή η προσπέλασή του για να γνωστοποιηθεί αυτή η αλλαγή στο TLB και να διατηρηθεί η συνάφεια των δεδομένων (data coherence) είναι αρκετά κοστοβόρα (40-100 κύκλους ρολογιού), η βέλτιστη επιλογή είναι η αφαίρεση του σχετικού entry από το TLB. Αυτό γίνεται μηδενίζοντας το valid bit που του αντιστοιχεί.

Προκειμένου το TLB να μπορεί να επικοινωνήσει με τις υπόλοιπες δομές, στον Rocket ορίζονται οι παρακάτω διεπαφές:

- **TLB Request (ως είσοδος)**: Η Data Cache στέλνει ένα TLB Request το οποίο περιλαμβάνει την εικονική διεύθυνση που πρέπει να αναζητηθεί.

- **Sfence Request (ως είσοδος)**: Σε περίπτωση που χρειαστεί να γίνει invalidate κάποια ή κάποιες καταχωρήσεις, η διεπαφή αυτή περιέχει πληροφορίες σχετικά με την εικονική σελίδα που πρέπει να ακυρωθεί ή τον χώρο διευθύνσεων. Συγκεκριμένα:

  - **rs1-rs2**: Στον rocket chip ορίζονται δύο δυαδικά σήματα, ο συνδυασμός των οποίων ορίζει με ποιο τρόπο γίνεται το invalidate. Συγκεκριμένα:
    - *rs1=0 και rs2=0:* Γίνεται ένα full TLB flush, δηλαδή ακυρώνονται όλες οι καταχωρήσεις του TLB, ανεξαρτήτως του επιπέδου τους στον πίνακα σελίδων.
    - *rs1=0 και rs2=1:* Ακυρώνονται όλες οι σελίδες που αφορούν τον χώρο διευθύνσεων που προσδιορίζεται στο ASID (βλέπε παρακάτω), ανεξαρτήτως του επιπέδου τους στον πίνακα σελίδων.
    - *rs1=1 και rs2=0:* Ακυρώνεται η σελίδα που βρίσκεται στο τελευταίο επίπεδο του πίνακα σελίδων και προσδιορίζεται από το vaddr (βλέπε παρακάτω), για όλους τους εικονικούς χώρους διευθύνσεων.
    - *rs1=1 και rs2=1:* Ακυρώνεται η σελίδα που βρίσκεται στο τελευταίο επίπεδο του πίνακα σελίδων και προσδιορίζεται από το vaddr (βλέπε παρακάτω), για τον χώρο διευθύνσεων που προσδιορίζεται στο ASID (βλέπε παρακάτω)



Σχήμα 2.9: Η διεπαφές επικοινωνίας του TLB με την Data/Instruction Cache και την μονάδα Page Table Walk

  - **Vaddr**: Προσδιορίζεται η εικονική διεύθυνση που πρέπει να ακυρωθεί.
  - **Asid**: Προσδιορίζεται ο εικονικός χώρος διευθύνσεων για τον οποίο γίνεται το invalidate.

- **TLB Response (ως έξοδος)**: Είναι η απάντηση του TLB στο request που δέχεται και περιέχει την ζητούμενη εικονική σελίδα, πληροφορίες για τον αν συνέβη TLB hit ή miss και αν η συγκεκριμένη πρόσβαση οδήγησε σε κάποιο exception.

- **TLB-PTW IO (αμφίδρομη)**: Έπειτα από ένα TLB Miss το αίτημα για την εικονική σελίδα προωθείται από το TLB στον Page Table Walker. Από την πλευρά της εξόδου του TLB, άρα της εισόδου στο PTW, προωθείται το TLB Request όπως ακριβώς περιγράφηκε παραπάνω. Ταυτόχρονα ο PTW δέχεται ως είσοδο πληροφορίες σχετικά με την κατάσταση των CSRs και την προστασία περιοχών της εικονικής μνήμης (ποιες λειτουργίες εκ των read/write/execute ορίζονται για κάθε περιοχή). Μετά την διάσχιση του πίνακα σελίδων, η δομή του Page Table Walker απαντάει στο TLB με την καταχώρηση του πίνακα σελίδων που βρέθηκε, το επίπεδό της και αν συνέβη κάποιο access exception.

Σε άλλες υλοποιήσεις ή άλλες αρχιτεκτονικές οι παραπάνω διεπαφές διαφέρουν.

2.8 Χωρική τοπικότητα των μεταφράσεων

Παρά τα πλεονεκτήματα που προσφέρει η χρήση της εικονικής μνήμης, εν τέλει καταλήγει στην επιβάρυνση της απόδοσης κατά 5-15%, εώς και 40-50% σε προγράμματα ανάλυσης μεγάλων δεδομένων (big data analysis) ή σε προγράμματα που χρησιμοποιούν την τεχνική της εικονικοποίησης (virtualization) [11]. Γι' αυτό το λόγο στην σχεδίαση Λειτουργικών Συστημάτων, που είναι υπεύθυνα για την διαχείριση του Πίνακα Σελίδων κάθε εφαρμογής, **έχουν αναπτυχθεί τεχνικές [10] (π.χ.: buddy allocation, memory compaction) μέσω των οποίων ένα πλήθος γειτονικών εικονικών σελίδων χαρογραφούνται σε γειτονικές φυσικές σελίδες, παράγοντας μια χωρική τοπικότητα (spatial locality) στις μεταφράσεις**. Έτσι καθεμία από τις διαφορετικές οργανώσεις του TLB που παρουσιάζονται στο κεφάλαιο 3 στοχεύουν στην αξιοποίηση κάποιας συγκεκριμένης μορφής χωρικής τοπικότητας στις καταχωρήσεις του Πίνακα Σελίδων.

Παρατηρούμε τρία επίπεδα σε σχέση με το πόσο έντονη είναι η παραγόμενη χωρική τοπικότητα. Στο πρώτο επίπεδο το Λειτουργικό Σύστημα δεν καταφέρνει να παράξει κάποιο μοτίβο. Αυτό

μπορεί να οφείλεται είτε στην μακροχρόνια συνεχόμενη χρήση του συστήματος από πολλαπλές εφαρμογές που οδηγεί στον κατακερματισμό της φυσικής μνήμης ή στην μη υποστήριξη τεχνικών που μπορούν να παράξουν αποτελεσματικά μορφές χωρικής τοπικότητας. Στο δεύτερο το Λειτουργικό Σύστημα αντιστοιχεί ένα μεγάλο αριθμό συνεχόμενων εικονικών σελίδων σε συνεχόμενες ευθυγραμμισμένες φυσικές σελίδες. Σε αυτή την περίπτωση αποφασίζει ότι είναι προτιμότερο να γίνει χρήση των superpages παρά το παραπάνω κόστος που έχουν, οι οποίες αποθηκεύονται σε μια ξεχωριστή μονάδα TLB. Όμως το πιο σύνηθες είναι το τρίτο επίπεδο που αποτελεί την ενδιάμεση περίπτωση από τα δύο πρώτα. Σε αυτό το επίπεδο ένας ικανός αριθμός δεκάδων εώς και εκατοντάδων γειτονικών εικονικών σελίδων αντιστοιχίζεται σε αντίστοιχο αριθμό γειτονικών φυσικών σελίδων.

Εντός του τρίτου επιπέδου παρατηρούμε δύο μορφές χωρικής τοπικότητας:

- **Συνεχόμενη χωρική τοπικότητα (contiguous spatial locality)**: Σε αυτή την μορφή τοπικότητας συνεχόμενες εικονικές σελίδες αντιστοιχούν σε συνεχόμενες φυσικές σελίδες. Διαφέρουν από τα superpages με δύο τρόπους. Πρώτον για την παραγωγή των superpages χρειάζεται συγκεκριμένος αριθμός συνεχόμενων σελίδων (π.χ. στην αρχιτεκτονική RISC-V χρειάζονται 512 συνεχόμενες σελίδες μεγέθους 4KB για την παραγωγή ενός superpage μεγέθους 2MB), ενώ στην συγκεκριμένη μορφή μπορούμε να έχουμε έναν οποιοδήποτε αριθμό συνεχόμενων σελίδων. Δεύτερον αίρεται ο περιορισμός της ευθυγράμμισης που υπάρχει στα superpages, δηλαδή η διεύθυνση των (εικονικών ή φυσικών) σελίδων δεν απαιτείται να είναι πολλαπλάσιο του μεγέθους των σελίδων.

- **Ομαδοποιημένη χωρική τοπικότητα (clustered spatial locality)**: Σε αυτή την μορφή τοπικότητας μια ομάδα (cluster) Χ ευθυγραμμισμένων εικονικών σελίδων αντιστοιχίζονται σε μια ομάδα (cluster) Χ ευθυγραμμισμένων φυσικών σελίδων, χωρίς αυτές να απαιτούνται να είναι συνεχόμενες. Η απαίτηση για ευθυγράμμιση σχετίζεται με τον πλήθος σελίδων Χ που ζητάμε να ανήκουν στις δύο ομάδες, δηλαδή ζητάμε η πρώτη σελίδα αυτής της ομάδας να είναι πολλαπλάσιο του Χ.

2.9 Διαφορετικές οργανώσεις του TLB

Η σημασία του TLB στην επίδοση ενός επεξεργαστή έχει οδηγήσει σε πολλές διαφορετικές προτάσεις όσον αφορά την εσωτερική του οργάνωση. Αυτή αφορά κυρίαρχα τον τρόπο με τον οποίο αποθηκεύονται τα entries και κατά συνέπεια το πώς υλοποιούνται οι λειτουργίες που προαναφέρθηκαν. Ταυτόχρονα όλες διαφορετικές προτάσεις που έχουν γίνει επιχειρούν να εκμεταλλευθούν την τοπικότητα των αντιστοιχίσεων που παράγεται από το λειτουργικό σύστημα, αλλά κυρίαρχα να αυξήσουν το TLB Reach χρησιμοποιώντας όσο λιγότερο χώρο γίνεται. Ως TLB Reach ορίζουμε τον μέγιστο αριθμό μεταφράσεων που μπορεί να αποθηκευθεί μέσα στο TLB. Στις παρακάτω υποενότητες θεωρούμε ότι ασχολούμαστε μόνο με το σχήμα εικονικής μνήμης Sv39 του RISC-V ISA.

2.9.1 Απλό TLB

Στην πιο απλή μορφή του TLB, τα Entries αποτελούνται από:

- **VPN tag (27 bits)**: Το αποθηκευμένο VPN
- **1 Valid Bit**: Ενημερώνει αν η μετάφραση είναι έγκυρη.
- **PPN (44 bits)**: Το PPN στο οποίο αντιστοιχεί το VPN tag
- **Attribute bits (15 bits)**: Περιέχουν πληροφορίες σχετικά με τα δικαιώματα πρόσβασης της σελίδας σε user/supervisor/privileged level, δηλαδή αν μπορεί να διαβαστεί (read), να γραφτεί (write) ή να εκτελεστεί (execute)

VPN [27 bits]	Valid [1 bit]	Entry Data	
		PPN [44 bits]	Attributes [15 bits]

Σχήμα 2.10: Η δομή της καταχώρησης ενός απλού TLB

Το απλό TLB πρακτικά κάνει αντιστοίχιση μία προς μία τις εικονικές στις φυσικές σελίδες δίχως να αξιοποιεί κάποια από τις μορφές χωρικής τοπικότητας που αναφέρθηκαν στην υποενότητα 2.8. Χρησιμοποιείται κυρίαρχα για την αποθήκευση των superpages και ενδέχεται να αποδίδει καλύτερα σε προγράμματα με μικρό memory footprint, καθώς αποθηκεύει πολλές πληροφορίες και δεν έχει περαιτέρω περιορισμούς όπως οι οργανώσεις που ακολουθούν.

Παρακάτω παρουσιάζουμε το πώς υλοποιείται η κάθε λειτουργία του απλού TLB:

- **Lookup**: Αρχικά επιλέγεται το κατάλληλο set με βάση την εικονική σελίδα που έχει ζητηθεί, ελέγχοντας τα τελευταία log2(nsets) bits. Στην συνέχεια ελέγχουμε κάθε VPN που είναι αποθηκευμένο σε αυτό το set και στην περίπτωση που είναι ίδιο με το ζητούμενο έχουμε TLB Hit. Σε αντίθετη περίπτωση έχουμε TLB Miss και το αίτημα προωθείται στον Page Table Walker.
- **Refill**: Μετά την απάντηση του Page Table Walker, αρχικά επιλέγεται το κατάλληλο set στο οποίο θα αποθηκευτεί η νέα μετάφραση με βάση την εικονική σελίδα που έχει ζητηθεί, ελέγχοντας τα τελευταία log2(nsets) bits του ζητούμενου VPN. Το way στο οποίο θα αποθηκευτεί επιλέγεται με βάση την επιλεγμένη πολιτική αντικατάστασης, συνήθως LRU (Least Recently Used).
- **Invalidate**: Όταν ζητηθεί η ακύρωση κάποιας καταχώρησης, το πρώτο βήμα είναι να την εντοπίσουμε με τον τρόπο που περιγράφεται στο lookup και στην συνέχεια απλά μηδενίζουμε το valid bit της καταχώρησης.

2.9.2 Coalesced TLB – Fully associative

Τα Coalesced TLB [10] επιχειρούν να αντιστοιχίσουν έναν αριθμό συνεχόμενων εικονικών σελίδων σε συνεχόμενες φυσικές σελίδες, αξιοποιώντας μορφές συνεχόμενης χωρικής τοπικότητας. Ορίζεται ένας αριθμός (coalescing factor) που ορίζει τον μέγιστο αριθμό μεταφράσεων που μπορούν να συγχωνευθούν σε μία καταχώρηση. Όπως φαίνεται και στο σχήμα 2.11 σε κάθε καταχώρηση αποθηκεύεται 1 VPN, 1 PPN, 1 valid bit και ένας αριθμός (coalescing length) μεγέθους log2(coalescing factor) bits που υποδεικνύει πόσες συνεχόμενες μεταφράσεις έχουν αποθηκευτεί μέσα στην καταχώρηση, καθώς και τα κοινά attribute bits των σελίδων.

Base VPN	Valid Bit	Base PPN	Coalescing Length	Attributes
27 bits	1 bit	54 bits	$\log_2$(coalescing factor) bits	15 bits

Σχήμα 2.11: Καταχώρηση ενός Coalesced TLB

Παρακάτω παρουσιάζουμε το πώς υλοποιείται η κάθε λειτουργία του Coalesced TLB – Fully associative:

- **Lookup**:
  - **Έλεγχος εύρους**: Για κάθε καταχώρηση του TLB γίνεται η παρακάτω σύγκριση:

    Base VPN <= Requested VPN <= Base VPN + Coalescing Length

    Αν αυτή η σύγκριση είναι επιτυχημένη για κάποια καταχώρηση, τότε έχουμε TLB Hit. Σε αντίθετη περίπτωση έχουμε TLB Miss και το αίτημα προωθείται στον Page Table Walker.
  - **Επιστροφή του ζητούμενου PPN**: Σε περίπτωση επιτυμένου ελέγχου εύρους, το ζητούμενο PPN προκύπτει προσθέτοντας στο Base PPN την διαφορά μεταξύ του ζητούμενου VPN και του αποθηκευμένου VPN, δηλαδή:

    Requested PPN = Base PPN + Requested VPN - Base VPN
- **Refill**: Μετά την επιστροφή της ζητούμενης μετάφρασης από τον Page Table Walker πρέπει να ελέγξουμε αν αυτή μπορεί να συγχωνευθεί με κάποια ήδη υπάρχουσα ή αν πρέπει να δημιουργηθεί κάποια καινούρια.
  - **Γειτονικά Hit**: Για να μπορεί η νέα μετάφραση να συγχωνευθεί πρέπει τόσο το ζητούμενο VPN όσο και το νέο PPN να είναι «γειτονικά» ως προς κάποια ήδη υπάρχουσα μετάφραση. Αυτό σημαίνει ότι πρέπει να ισχύει κάποια από τις δύο παρακάτω συνθήκες (σημειώνουμε πως η χρήση του «/» υποδηλώνει πως η συνθήκη πρέπει να ισχύει και για το VPN και για το PPN):
    1. *Requested VPN/PPN = Base VPN/PPN + Coalescing Length + 1:* Σε αυτή την περίπτωση απλά αυξάνουμε κατά ένα την τιμή του Coalescing Length. Στην περίπτωση που έχουμε φτάσει ήδη στην μέγιστη τιμή του η πιο απλή λύση είναι να ακυρώσουμε την ήδη υπάρχουσα καταχώρηση και να βάλουμε την καινούρια, κάτι που μας κοστίζει χρήσιμες μεταφράσεις οι οποίες θα χρησιμοποιηθούν ξανά μελλοντικά προσθέτοντας παραπάνω κύκλους ρολογιού για την ολοκλήρωση ενός προγράμματος. Η δεύτερη λύση είναι να μην ακυρώνουμε την ήδη υπάρχουσα καταχώρηση αλλά απλά να προσθέτουμε μια καινούρια. Αυτή η επιλογή δημιουργεί την ανάγκη χρήσης μεγαλύτερου TLB καθώς αν διατηρούσαμε το ίδιο μέγεθος θα διώχνονταν και πάλι μεταφράσεις με μεγάλο Coalescing Length, εξαιτίας της πολιτικής αντικατάστασης LRU. Η τρίτη λύση είναι η τροποποίηση της πολιτικής αντικατάστασης ούτως ώστε να λαμβάνει υπ' όψιν της και το Coalescing Length κάθε καταχώρησης. Η τέταρτη και τελευταία λύση είναι η χρήση ενός δεύτερου απλού

TLB μικρού μεγέθους στο οποίο θα αποθηκεύονται οι μεταφράσεις που δεν μπορούν να συγχωνευθούν με κάποια άλλη, το οποίο θα ελέγχεται ταυτόχρονα με το Coalesced TLB κατά την διαδικασία του Lookup.

2. *Requested VPN/PPN = Base VPN/PPN -1:* Στην περίπτωση αυτή αλλάζουμε τα Base VPN/PPN βάζοντας τα καινούρια και ταυτόχρονα αυξάνουμε το Coalescing Length κατά ένα. Η περίπτωση όπου το Coalescing Length έχει φτάσει ήδη την μέγιστη τιμή του αντιμετωπίζεται όπως προηγουμένως.

– **Έλεγχος για attribute bits:** Οι σελίδες που συγχωνεύονται σε μια καταχώρηση πρέπει να μοιράζονται τις ίδιες ιδιότητες, δηλαδή τα ίδια attribute bits. Επομένως πρέπει να γίνει σύγκριση μεταξύ των attribute bits της νέας μετάφρασης με εκείνα που βρίσκονται στην καταχώρηση-«γείτονα». Αν είναι ίδια τότε προχωράμε κανονικά, ενώ αν είναι διαφορετικά πρέπει να δημιουργηθεί μια καινούρια καταχώρηση με το σκεπτικό που περιεγράφηκε προηγουμένως.

- **Invalidate:** Ο πιο απλός τρόπος να ακυρώσουμε μια εικονική σελίδα είναι να εντοπίσουμε σε πιο way βρίσκεται, με την διαδικασία που περιγράφεται στο Lookup και έπειτα να μηδενίσουμε το valid bit ακυρώνοντας ολόκληρη την καταχώρηση. Παρά την απλότητα της διαδικασίας, καταλήγουμε να χάνουμε αρκετές πιθανά χρήσιμες μεταφράσεις. Μια άλλη λύση που προτείνουμε στην διπλωματική είναι να μειώσουμε την τιμή του coalescing length αντί να μηδενίσουμε το valid bit. Έτσι η νέα τιμή του coalescing length θα είναι η εξής:

$$\text{New coalescing length} = \text{Requested VPN for invalidation} - \text{Base VPN}$$

2.9.3 Sub-block TLBs

Τα sub-blocking TLB[14] επιχειρούν να αντιστοιχίσουν μία ομάδα συνεχόμενων εικονικών σελίδων σε εικονικές. Υπάρχουν δύο υποκατηγορίες, τα Complete Sub-block TLB και τα Partial Subblock TLB. Και για τις δύο υποκατηγορίες ορίζεται ο subblock factor (ή nSectors), που εκφράζει πόσες μεταφράσεις μπορούν να συγχωνευθούν μέσα σε μια μοναδική καταχώρηση. Για να μπορεί να υλοποιηθεί απρόσκοπτα σε επίπεδο υλικού, ο subblock factor πρέπει να είναι δύναμη του 2. Το VPN χωρίζεται σε τρία πεδία, όπως φαίνεται στο σχήμα 2.12, και η χρήση του καθενός θα επεξηγηθεί παρακάτω.

VPN tag	**Set selection bits**	**Sector Offset**
[VPN Bits - $\log_2$(n_sets) - $\log_2$(subblock factor)]	[$\log_2$(n_sets)]	[$\log_2$(subblock factor)]

(α) Η κατάτμηση του VPN για οποιοδήποτε αριθμό set και subblock factor

VPN tag	**Set selection bits**	**Sector Offset**
[23 bits]	[2 bits]	[2 bits]

(β) Η διάτμηση του VPN για 4 set και subblock factor=4

Σχήμα 2.12: Η διάτμηση του VPN στα Sub-block TLB

2.9.3.1 Complete Sub-block TLB

Αποτελεί την τωρινή οργάνωση του TLB στον Rocket. Τα Complete Sub-block TLB αναζητούν συνεχόμενες εικονικές σελίδες και για καθεμία από αυτή αποθηκεύουν την αντίστοιχη φυσική σελίδα, μαζί με ένα valid bit ανά sub-entry και τα attribute bits κάθε σελίδας. Έτσι, οι φυσικές σελίδες δεν είναι απαραίτητο να έχουν κάποιο μοτίβο τοπικότητας και μπορούν να είναι σκόρπιες στον πίνακα σελίδων. Η βελτίωση σε σχέση με το απλό TLB οφείλεται στο ότι αντί να αποθηκεύουμε όλα τα subblock factor συνεχόμενα VPN, αποθηκεύουμε ως tag μονάχα ένα, όπως φαίνεται στο σχήμα 2.13.

		Entry data	
VPN tag [27 bits]	Valid 0 [1 bit]	PPN 0 [54 bits]	Attributes 0 [15 bits]
	Valid 1 [1 bit]	PPN 1 [54 bits]	Attributes 1 [15 bits]
	Valid 2 [1 bit]	PPN 2 [54 bits]	Attributes 2 [15 bits]
	Valid 3 [1 bit]	PPN 3 [54 bits]	Attributes 3 [15 bits]

Σχήμα 2.13: Καταχώρηση ενός Sectored/Complete Sub-Block TLB

Παρακάτω παρουσιάζουμε το πώς υλοποιείται η κάθε λειτουργία του Complete Sub-block TLB:

- **Lookup**:
  - **Επιλογή του κατάλληλου set**: Από την ζητούμενη εικονική σελίδα αποκόπτουμε τα κατάλληλα bit για την επιλογή του set
  - **Αναζήτηση για sector hits στα ways του set**: Αποκόπτουμε τα bits του VPN tag από την ζητούμενη εικονική σελίδα. Αν υπάρχει κάποιο αποθηκευμένο entry με το ίδιο tag και τουλάχιστον ένα valid sub-entry τότε έχω sector hit. Σε αντίθετη περίπτωση υπάρχει TLB Miss, το αίτημα για την εικονική σελίδα προωθείται στον Page Table Walker και όταν υπάρχει έγκυρη απάντηση προχωράω στο Refill.
  - **Έλεγχος για TLB Hit**: Αν έχω sector hit τότε ελέγχω το valid bit του sub-entry που υποδεικνύεται από τα τελευταία log2(subblock factor) bits της ζητούμενης εικονικής σελίδας. Αν το valid bit είναι ίσο με 1 τότε έχω TLB Hit και το αποθηκευμένο PPN επιστρέφεται στην DCache, αλλιώς έχω TLB Miss.
- **Refill**:
  - **Επιλογή του κατάλληλου set**: Το set που επιλέξαμε κατά την διαδικασία το lookup αποθηκεύεται σε έναν καταχωρητή.
  - **Επιλογή του κατάλληλου way**: Αν κατά την διαδικασία του lookup είχαμε sector hit για την ζητούμενη εικονική σελίδα, τότε η νέα μετάφραση αποθηκεύεται υποχρεωτικά στο συγκεκριμένο way. Συγκεκριμένα επιλέγεται το sub-entry με βάση το sector Offset, αποθηκεύεται το PPN με το οποίο απάντησε ο PTW μαζί με τα attribute bits και το valid bit γίνεται 1. Σε περίπτωση που δεν υπάρχει sector hit, το way επιλέγεται με βάση την επιλεγμένη πολιτική αντικατάστασης. Αν το TLB είναι γεμάτο, επομένως η νέα μετάφραση αντικαθιστά κάποια παλιά, πρέπει να ακυρώσουμε την παλιά καταχώρηση

πριν την εγκατάσταση της καινούριας. Αυτό περιλαμβάνει τον μηδενισμό όλων των valid bit, την αντικατάσταση του VPN tag, την προσθήκη του νέου PPN και την αλλαγή του κατάλληλου valid bit σε 1.

- **Invalidate**:
  - Αν το invalidate αφορά μια συγκεκριμένη εικονική σελίδα τότε ο εντοπισμός της γίνεται όπως στο lookup, μηδενίζοντας το valid bit που αφορά το συγκεκριμένο sector Offset
  - Αν το invalidate αφορά ένα ολόκληρο sector, τότε πρέπει να το εντοπίσουμε και να μηδενιστούν τα valid bits όλων των subentries

2.9.3.2 Partial Sub-block TLB

Το partial sub-block TLB αποτελεί μια βελτιωμένη εκδοχή του complete sub-block όσον αφορά την αναλογία bits/entry. Σε αυτή την περίπτωση μια ομάδα συνεχόμενων ευθυγραμμισμένων εικονικών σελίδων αντιστοιχείται σε μια ομάδα συνεχόμενων ευθυγραμμισμένων φυσικών σελίδων. Αποτελεί μια οργάνωση TLB που επιχειρεί να αξιοποιήσει και τις δύο μορφές χωρικής τοπικότητας.

Για να γίνει αυτό υπάρχει η απαίτηση τόσο οι εικονικές όσο και οι φυσικές σελίδες να είναι ευθυγραμμισμένες (aligned). Αυτό σημαίνει ότι οι ομάδες σελίδων δεν επιλέγονται τυχαία, αλλά πρέπει το sector offset της πρώτης σελίδας της ομάδας να είναι μηδέν και το sector offset της τελευταίας να είναι 2sub-block factor-1. Επιπλέον πρέπει τα sector offset τόσο της εικονικής όσο και της φυσικής σελίδας να είναι ίδια. Οι μεταφράσεις που δεν πληρούν την τελευταία προϋπόθεση αποθηκεύονται σε μια ξεχωριστή μονάδα TLB.

Το Partial Sub-block TLB διαφέρει από το Coalesced TLB σε δύο χαρακτηριστικά. Αφ' ενός στο Partial Sub-block αποθηκεύεται ένα PPN Mask κα απαιτείται τα offset των VPN και PPN να είναι ίσα, ενώ στο Set-Associative Coalesced TLB αποθηκεύεται ολόκληρο το PPN δίχως να υπάρχει κάποια απαίτηση για τα offset. Αφ' ετέρου συγκριτικά με το Fully- associative Coalesced TLB, το Partial Sub-block δεν απαιτεί οι μεταφράσεις που αποθηκεύονται να είναι συνεχόμενες αρκεί να ικανοποιούνται τα κριτήρια που προαναφέρθηκαν.

Οι καταχωρήσεις του Partial Sub-block TLB αποτελούνται από ένα VPN Mask, ένα PPN Mask, 1 valid bit ανά μετάφραση και attribute bits ανά μετάφραση, όπως φαίνεται στο σχήμα 2.14.

<table>
<tr><td rowspan="4">VPN Mask
[27 bits]</td><td rowspan="4">PPN Mask
[54 bits]</td><td>Valid 0 [1 bit]</td><td>Attributes 0 [15 bits]</td></tr>
<tr><td>Valid 1 [1 bit]</td><td>Attributes 1 [15 bits]</td></tr>
<tr><td>Valid 2 [1 bit]</td><td>Attributes 2 [15 bits]</td></tr>
<tr><td>Valid 3 [1 bit]</td><td>Attributes 3 [15 bits]</td></tr>
</table>

Σχήμα 2.14: Καταχώρηση ενός Partial Sub-Block TLB

Παρακάτω παρουσιάζουμε το πώς υλοποιείται η κάθε λειτουργία του Partial Sub-block TLB:

- **Lookup**: Αγνοούμε τα τελευταία log2(subblock factor) bits του VPN (VPN Offset) και τα υπόλοιπα αποτελούν το lookup tag. Αν το lookup tag είναι αποθηκευμένο εντός του TLB (sector hit) τότε ελέγχουμε το valid bit που υποδεικνύεται από το VPN Offset. Αν αυτό είναι έγκυρο τότε έχω TLB Hit και το PPN που επιστρέφεται προκύπτει συνενώνοντας το PPN Mask με τα

τελευταία log2(subblock factor) bits του VPN. Σε αντίθετη περίπτωση υπάρχει TLB Miss, το αίτημα για την εικονική σελίδα προωθείται στον Page Table Walker και όταν υπάρχει έγκυρη απάντηση προχωράω στο Refill.

- **Refill**: Αν κατά την διαδικασία του lookup είχαμε sector hit για την ζητούμενη εικονική σελίδα, τότε η νέα μετάφραση αποθηκεύεται υποχρεωτικά στο συγκεκριμένο way. Συγκεκριμένα πρέπει τα τελευταία log2(subblock factor) bits τόσο του VPN όσο και του VPN να είναι τα ίδια. Αν αυτό ισχύει τότε το valid bit που υποδεικνύεται το VPN Offset γίνεται έγκυρο. Στην περίπτωση που τα δύο Offset είναι διαφορετικά η νέα μετάφραση αποθηκεύεται σε μια ξεχωριστή μονάδα TLB.

  Σε περίπτωση που δεν υπάρχει sector hit, το way επιλέγεται με βάση την επιλεγμένη πολιτική αντικατάστασης ακολουθώντας ακριβώς την ίδια λογική με προηγουμένως για τα VPN και PPN Offset. Αν το TLB είναι γεμάτο, επομένως η νέα μετάφραση αντικαθιστά κάποια παλιά, πρέπει να ακυρώσουμε την παλιά καταχώρηση πριν την εγκατάσταση της καινούριας. Αυτό περιλαμβάνει τον μηδενισμό όλων των valid bit και την αντικατάσταση του VPN tag.

- **Invalidate**:
  - Αν το invalidate αφορά μια συγκεκριμένη εικονική σελίδα τότε ο εντοπισμός της γίνεται όπως στο lookup, μηδενίζοντας το valid bit που αφορά το συγκεκριμένο sector Offset
  - Αν το invalidate αφορά ένα ολόκληρο sector, τότε πρέπει να το εντοπίσουμε και να μηδενιστούν τα valid bits όλων των subentries

2.9.4 Clustered TLB

Το Clustered TLB [11] αποτελεί την βέλτιστη πρόταση για την οργάνωση του TLB όσον αφορά την αναλογία bits/entry σε σχέση με όσες εξετάζουμε στην παρούσα διπλωματική εργασία και είναι εκείνη που υλοποιήσαμε στον Rocket. Σε αυτή την περίπτωση μία ομάδα ευθυγραμμισμένων εικονικών σελίδων αντιστοιχίζεται σε μία ομάδα ευθυγραμμισμένων φυσικών σελίδων, αξιοποιώντας κυρίαρχα μορφές ομαδοποιημένης χωρικής τοπικότητας. Ορίζεται ο cluster factor, σε αντιστοιχία με τον subblock factor που είδαμε στα Sub-block TLB, που ορίζει τον μέγιστο αριθμό μεταφράσεων που μπορούν να εκφραστούν μέσα από ένα μοναδικό entry.

Διαφέρει από το Complete Sub-block TLB ως προς το ότι αντί να αποθηκεύουμε ολόκληρα τα PPN για κάθε subentry, αποθηκεύουμε μια μόνο φορά τα bit του PPN εκτός από τα τελευταία log2(cluster factor) bit σε ένα PPN Mask και τόσα cluster offset όσα υποδεικνύει ο cluster factor, μεγέθους log2(cluster factor) bit το καθένα. Ταυτόχρονα υπάρχει η απαίτηση και για τις φυσικές σελίδες να είναι ευθυγραμμισμένες στον πίνακα σελίδων ως προς τον cluster factor. Διαφέρει από το Partial Sub-block καθώς στο clustered TLB δεν είναι ανάγκη να είναι ίδια τα cluster offset της εικονικής και της φυσικής σελίδας.

Οι καταχωρήσεις του Clustered TLB αποθηκεύουν ως tag ένα VPN mask και ένα PPN mask, ενώ τα δεδομένα τους αποτελούνται από 3 διανύσματα μεγέθους cluster offset στα οποία αποθηκεύονται τα PPN Offset, τα attribute bits και τα valid bit.

Παρακάτω παρουσιάζουμε το πώς υλοποιείται η κάθε λειτουργία του Clustered TLB:

- **Lookup**:

VPN tag [VPN Bits - $\log_2$(n_sets) - $\log_2$(cluster factor)]	Set selection bits [$\log_2$(n_sets)]	Sector Offset [$\log_2$(cluster factor)]

Σχήμα 2.15: Η κατάτμηση ενός VPN στο Clustered TLB

VPN Mask [27 bits]	PPN Mask [52 bits]	Valid 0 [1 bit]	Attributes 0 [15 bits]
		Valid 1 [1 bit]	Attributes 1 [15 bits]
		Valid 2 [1 bit]	Attributes 2 [15 bits]
		Valid 3 [1 bit]	Attributes 3 [15 bits]

Σχήμα 2.16: Καταχώρηση ενός Clustered TLB

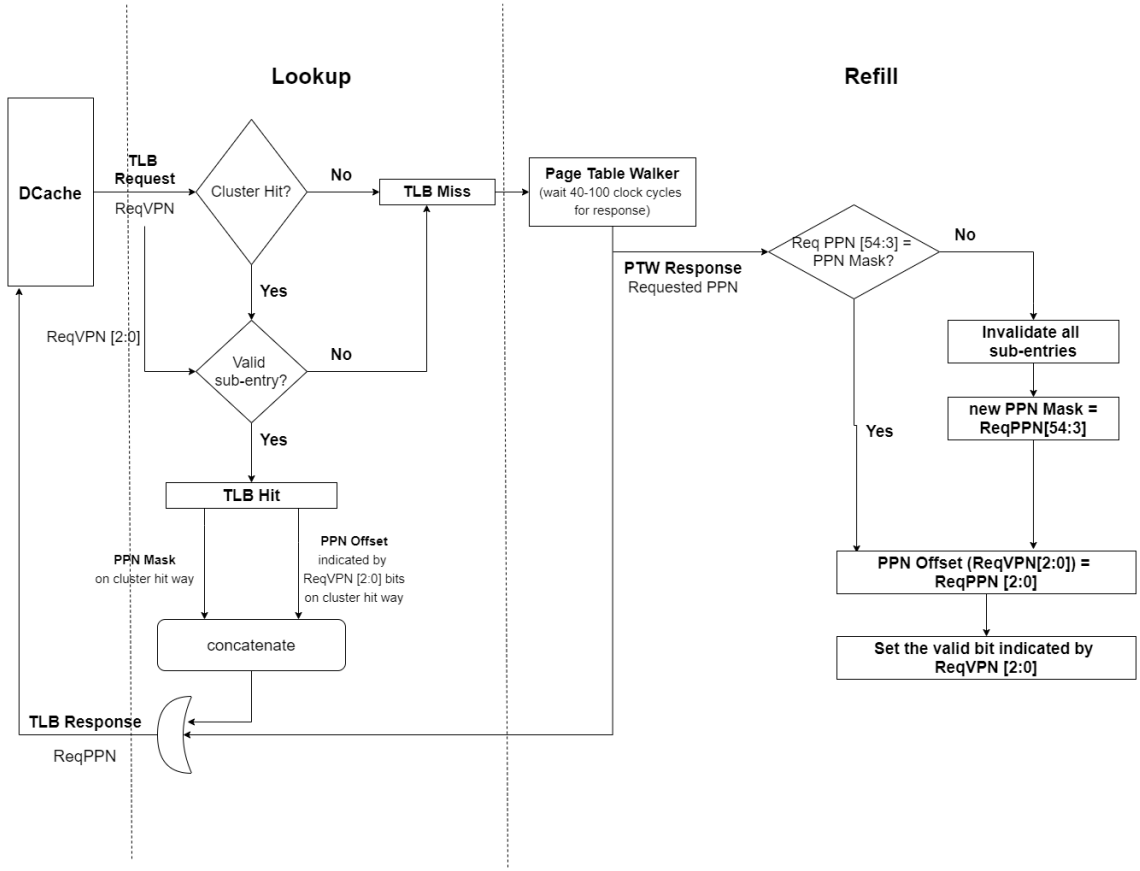
  - **Επιλογή του κατάλληλου set**: Από την ζητούμενη εικονική σελίδα αποκόπτουμε τα κατάλληλα bit για την επιλογή του set
  - **Αναζήτηση για cluster hits στα ways του set**: Από τα bit της ζητούμενης εικονικής σελίδας αγνοούμε τα τελευταία log2(cluster factor) bits, για να πάρουμε το ζητούμενο VPN Mask. Αν υπάρχει κάποιο αποθηκευμένο entry με το ίδιο VPN Mask και τουλάχιστον ένα valid sub-entry τότε έχω cluster hit. Σε αντίθετη περίπτωση υπάρχει TLB Miss, το αίτημα για την εικονική σελίδα προωθείται στον Page Table Walker και όταν υπάρχει έγκυρη απάντηση προχωράω στο Refill.
  - **Έλεγχος για TLB hit**: Αν έχω cluster hit τότε ελέγχω το valid bit του sub-entry που υποδεικνύεται από τα τελευταία log2(cluster factor) bits της ζητούμενης εικονικής σελίδας. Αν το valid bit είναι ίσο με 1 τότε έχω TLB Hit και το κατάλληλο PPN επιστρέφεται στην DCache, αλλιώς έχω TLB Miss. Η φυσική σελίδα που επιστρέφεται, προκύπτει από την συνένωση του αποθηκευμένου PPN Mask και του PPN Offset του sub-entry που επιλέξαμε.

- **Refill**:

  - **Επιλογή του κατάλληλου set**: Το set που επιλέξαμε κατά την διαδικασία το lookup αποθηκεύεται σε έναν καταχωρητή.
  - **Επιλογή του κατάλληλου way**: Αν κατά την διαδικασία του lookup είχαμε cluster hit για την ζητούμενη εικονική σελίδα, τότε η νέα μετάφραση αποθηκεύεται υποχρεωτικά στο συγκεκριμένο way. Αν παρά το cluster hit, το αποθηκευμένο PPN Mask είναι διαφορετικό από εκείνο που αντιστοιχεί στην απάντηση που λάβαμε από το PTW, δεν υπάρχει δηλαδή PPN Mask Match, τότε το υπάρχων entry ακυρώνεται προκειμένου να εγκατασταθεί το καινούριο. Συγκεκριμένα επιλέγεται το sub-entry με βάση το cluster Offset, αποθηκεύεται το PPN Offset με βάση την απάντηση του PTW μαζί με τα attribute bits και το valid bit γίνεται 1. Σε περίπτωση που δεν υπάρχει sector hit, το way επιλέγεται με βάση την επιλεγμένη πολιτική αντικατάστασης. Αν το TLB είναι γεμάτο, επομένως η νέα μετάφραση αντικαθιστά κάποια παλιά, πρέπει να ακυρώσουμε την παλιά καταχώρηση πριν την εγκατάσταση της καινούριας. Αυτό περιλαμβάνει τον μηδενισμό όλων των valid bit και την αντικατάσταση των VPN mask και PPN mask.

- **Invalidate**:

– Αν το invalidate αφορά μια συγκεκριμένη εικονική σελίδα τότε ο εντοπισμός της γίνεται όπως στο lookup, μηδενίζοντας το valid bit που αφορά το συγκεκριμένο cluster Offset
– Αν το invalidate αφορά ένα ολόκληρο cluster, τότε μηδενίζουμε τα valid bits όλων των subentries που αφορούν αυτό το cluster.



Σχήμα 2.17: Το lookup και το refill στο Clustered TLB

Κεφάλαιο 3

Σχεδιασμός και Υλοποίηση

Στο κεφάλαιο αυτό παρουσιάζεται η μελέτη λειτουργίας που έγινε πάνω στις διαφορετικές μορφές που έχουν προταθεί για το L1 TLB, η ανάλυση του τωρινού TLB του Rocket Chip Generator και έπειτα η σχεδίαση και υλοποίηση παραμετροποιήσιμου Multi-granular Clustered TLB.

3.1 Multi-granular Clustered TLB

Υπάρχουν αρκετές περιπτώσεις όπου μία νέα μετάφραση η οποία επιστρέφεται από τον PTW αντικαθιστά μια ήδη υπάρχουσα που έχει υψηλό βαθμό clustering. Τέτοιες περιπτώσεις δεν μας αφήνουν να εκμεταλλευτούμε πλήρως τα πλεονεκτήματα του clustered TLB, καθώς αφ' ενός ακυρώνονται χρήσιμες μεταφράσεις αυξάνοντας τα TLB Misses, αφ' ετέρου χρησιμοποιούνται καταχωρήσεις με μία μόνο μετάφραση αντί για όσες ορίζεται από τον cluster factor.

Για την επίλυση αυτού του προβλήματος επιλέγουμε να προσθέσουμε ένα απλό TLB στο οποίο θα αποθηκεύονται οι μεταφράσεις που δεν μπορούν να συγχωνευθούν με άλλες. Τα κριτήρια για την εγγραφή μιας μετάφρασης σε αυτό το TLB είναι δύο: πρώτον, η μετάφραση να κάνει VPN Mask Match με κάποια ήδη υπάρχουσα καταχώρηση, αλλά όχι PPN Mask Match, και δεύτερον ο αριθμός των έγκυρων μεταφράσεων που υπάρχουν σε αυτή την καταχώρηση να υπερβαίνει κάποιο όριο (cluster threshold).

Έτσι οι λειτουργίες του Multi-granular TLB μεταβάλλονται ως εξής:

- **Lookup**: Τα δύο TLB ελέγχονται ταυτόχρονα και τυχόν επιτυχής αναζήτηση σε κάποιο από τα δύο σημαίνει TLB Hit. Η αναζήτηση και η επιστροφή του PPN (στην περίπτωση TLB Hit) για το απλό TLB και το Clustered TLB πραγματοποιείται όπως ήδη έχει περιγραφεί. Η διαδικασία αυτή φαίνεται από μια high-level οπτική στο σχήμα 3.1.

- **Refill**: Διαφοροποιείται από την διαδικασία του refill στο Clustered TLB ως εξής: πλέον στην περίπτωση που έχω VPN Mask Match με κάποια ήδη υπάρχουσα καταχώρηση, αλλά όχι PPN Mask Match, δεν ακυρώνω αυτή την καταχώρηση. Αντί αυτού ελέγχω πόσες έγκυρες μεταφράσεις υπάρχουν σε αυτή την καταχώρηση και αν αυτές υπερβαίνουν το cluster threshold τότε η νέα μετάφραση αποθηκεύεται στο απλό TLB. Σε κάθε άλλη περίπτωση η νέα μετάφραση αποθηκεύεται στο clustered TLB με τον τρόπο που περιγράφηκε στο προηγούμενο κεφάλαιο. Συνοπτικά η επιλογή για το ποιο TLB θα γίνει refill παρουσιάζεται στον πίνακα 3.1

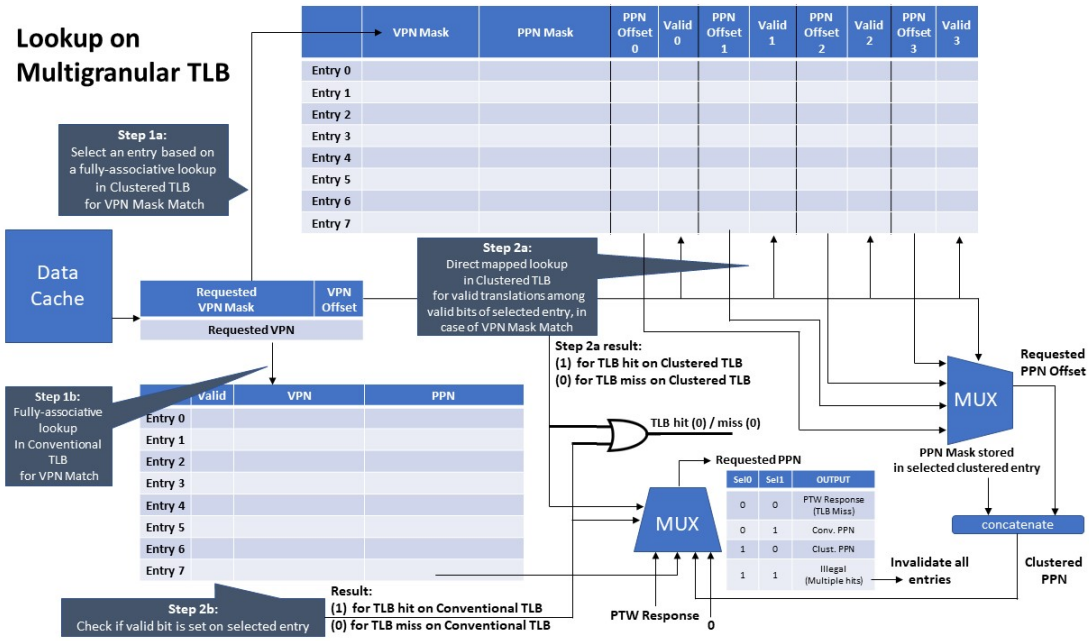
- **Invalidate**:

– Αν το invalidate αφορά μια συγκεκριμένη εικονική σελίδα τότε ο εντοπισμός της γίνεται όπως στο lookup, μηδενίζοντας είτε το valid bit που αφορά το συγκεκριμένο cluster Offset ή το valid bit της καταχώρησης στο απλό TLB, ανάλογα με το που είναι αποθηκευμένη η εικονική σελίδα

– Αν το invalidate αφορά ένα ολόκληρο cluster, τότε μηδενίζουμε τα valid bits όλων των subentries που αφορούν αυτό το cluster.

VPN Mask Match	PPN Mask Match	Valid Sub-entries	Refill TLB
0	x	x	Clustered
1	1	x	Clustered
1	0	< Cluster threshold	Clustered
1	0	>= Cluster threshold	Conventional

Πίνακας 3.1: Επιλογή του κατάλληλου TLB που θα γίνει refill στο Multi-granular TLB



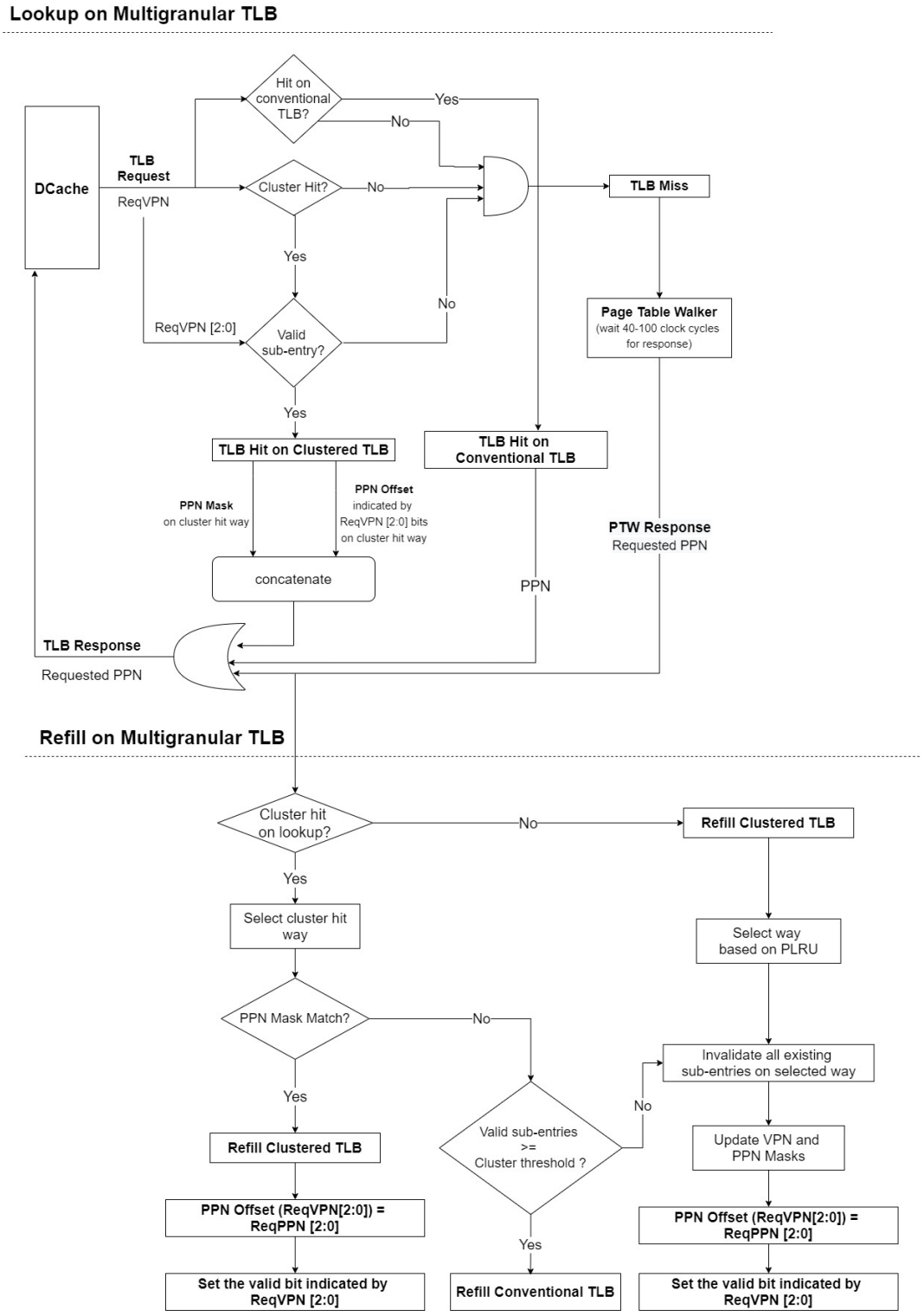
Σχήμα 3.1: Μια high-level οπτική του refill στο Multigranular TLB

3.2 Ενσωμάτωση αλλαγών στον κώδικα του Rocket

Το πρώτο βήμα ήταν να καταγραφούν οι βασικές **διαφορές μεταξύ του Sectored TLB και του Clustered TLB**, τόσο ως προς τα περιεχόμενα των καταχωρήσεων, όσο και ως προς την συνολική τους λειτουργία. Στοιχειοθετούμε τις εξής διαφορές οι οποίες ενσωματώθηκαν στον κώδικα του TLB του Rocket [1]:

1. **Το Clustered TLB χρησιμοποιεί δύο διαφορετικά πεδία PPN Mask (ένα για κάθε καταχώρηση) και PPN offset (ένα για κάθε υποκαταχώρηση) σε αντίθεση με το Sectored TLB το οποίο αποθηκεύει ολόκληρο το PPN για κάθε υποκαταχώρηση.**

[1] Εδώ στην αρχική του μορφή: https://bit.ly/31dmp6r



Σχήμα 3.2: Το lookup και το refill στο Multi-granular TLB

Για αυτό το λόγο αντικαταστάθηκε το πεδίο ppn στην κλάση TLB Entry Data με ένα διάνυσμα καταχωρητών μεγέθους nClusters που έχει στοιχεία μεγέθους log2(nClusters) bits τα οποία αντιστοιχούν στα ppnOffset. Ταυτόχρονα προστέθηκε στην κλάση TLB Entry ένα νέο πεδίο το οποίο κρατάει το PPN Mask. Τέλος, για την αποθήκευση του PPN Mask ως πεδίο

της κλάσης TLB Entry Data προστίθεται ακόμη μία παράμετρος -που αντιστοιχεί στο νέο PPN που θέλω να αποθηκεύσω- στην συνάρτηση ”insert” της κλάσης TLB Entry.

2. **Για την προσθήκη μιας νέας υποκαταχώρησης σε μία ήδη έγκυρη καταχώρηση στο Clustered TLB απαιτείται να υπάρχουν ταυτόχρονα VPN Mask Match και PPN Mask Match, ενώ στο Sectored TLB απαιτείται μονάχα το πρώτο.**

   Έτσι δημιουργούμε μια νέα συνάρτηση ”ppnMaskMatch” η οποία δίνει αληθές αποτέλεσμα αν όλα τα bits εκτος τα τελευταία log2(nClusters) bits του PPN της νέας μετάφρασης και του PPN Mask της ελεγχόμενης καταχώρησης είναι ίσα και επίσης υπάρχει τουλάχιστον μία έγκυρη μετάφραση στην ελεγχόμενη καταχώρηση. Η νέα συνάρτηση χρησιμοποιείται κατά την διαδικασία του refill όπου και πρέπει να ελεγχθεί αν έχω ταυτόχρονα VPN Mask Match και PPN Mask Match. Συγκεκριμένα αποθηκεύεται στον καταχωρητή ”waddr” το Way στο οποίο θα αποθηκευθεί η νέα μετάφραση. Αυτό γίνεται μέσω ενός πολυπλέκτη που επιλέγει μεταξύ δύο διευθύνσεων με βάση το αν έχω cluster Hit ή όχι. Στην πρώτη περίπτωση επιλέγεται το way στο οποίο υπάρχει cluster hit ενώ στην δεύτερη επιλέγεται εκείνο που προκύπτει από την πολιτική αντικατάσης. Σε κάθε περίπτωση αν δεν έχω ταυτόχρονα VPN Mask Match και PPN Mask Match οι υποκαταχωρήσεις του way που έχει φορτωθεί στο waddr πρέπει να γίνουν invalidate.

3. **Στο Clustered TLB το PPN που επιστρέφεται έπειτα από ένα TLB Hit προκύπτει από την συνένωση του PPN Mask με το PPN offset που υποδεικνύεται από τελευταία log2(nClusters) bits του VPN. Αντίθετα στο Sectored TLB το οποίο αποθηκεύει ένα PPN για κάθε μετάφραση, επιστρέφεται απευθείας το PPN που υποδεικνύεται από τελευταία log2(nSectors) bits του VPN.**

   Για τον λόγο αυτό τροποποιούμε την συνάρτηση ”ppn” ώστε να λειτουργεί με τον παραπάνω τρόπο.

Με βάση αυτές τις αλλαγές αναπτύχθηκε στην συνέχεια η δομή του **Multi-granular Clustered TLB**. Για να φτάσουμε σε αυτή την οργάνωση έγιναν οι εξής επιπλέον αλλαγές:

1. **Προσθήκη διανύσματος καταχωρητών ”singleton entries” και προσθήκη του στα ”ordinary entries” και ”all real entries”.**

   Για την δημιουργία του διανύσματος καταχωρητών αξιοποιούμε την δυνατότητα της Chisel να αποτελεί γεννήτρια υλικού, καθώς ”πατάμε” πάνω στο ίδιο template που χρησιμοποιείται για την δημιουργία του Clustered TLB για να δημιουργήσουμε ένα δεύτερο απλό TLB, απλά δίνοντας την τιμή 1 στην παράμετρο nClusters. Το νέο διάνυσμα προστίθεται στις συναρτήσεις ”ordinary entries” και ”all real entries”.

   Η πρώτη πλέον επιστρέφει όλα τα entries του Clustered TLB που ανήκουν στο ίδιο set με το VPN που ζητείται, τα entries του απλού TLB, τα superpage entries. Χρησιμοποιείται κατά το lookup, το οποίο με αυτό τον τρόπο μπορεί να γίνεται ταυτόχρονα σε όλα τα TLB, καθώς και για το refill.

   Η συνάρτηση ”all real entries” χρησιμοποιείται όταν έρθει ένα αίτημα για sfence ή για να γίνει έλεγχος για πολλαπλά hit. Στην περίπτωση που δύο ή περισσότερες αποθηκευμένες

μεταφράσεις ταιριάζουν με την ζητούμενη εικονική διεύθυνση σημαίνει ότι έχω **multiple hit** κάτι που αποτελεί αστοχία σχεδίασης του Λειτουργικού Συστήματος. Όταν αυτό συμβεί ακυρώνονται όλες οι καταχωρήσεις του TLB και η απάντηση στην Data Cache είναι ότι συνέβη ένα TLB Miss.

2. **Τροποποίηση πολιτικής αντικατάστασης του Clustered TLB**

   Όταν το Clustered TLB λειτουργεί μόνο του, αρκεί να υπάρχει cluster hit σε κάποιο Way για να γίνει πρόσβαση σε αυτό και άρα να ενημερωθούν εκείνοι οι παράγοντες που καθορίζουν ποιο way θα αντικατασταθεί έπειτα. Στο Multi-granular Clustered TLB στην περίπτωση που έχω cluster hit σε κάποιο way αλλά όχι PPN Mask Match σε αυτό, τότε αυτή η μετάφραση αποθηκεύεται στο απλό TLB. Επομένως τροποποιούμε την πολιτική αντικατάστασης του Clustered TLB προκειμένου να λαμβάνει υπ' όψιν της τα πραγματικά hit στο Clustered TLB. Ταυτόχρονα επιλέγουμε πολιτική αντικατάστασης plru για το απλό TLB.

3. **Τροποποίηση της λειτουργίας του refill**

   Πλέον στην περίπτωση που έχω VPN Mask Match με κάποια ήδη υπάρχουσα καταχώρηση, αλλά όχι PPN Mask Match, δεν ακυρώνω αυτή την καταχώρηση. Αντί αυτού ελέγχω πόσες έγκυρες μεταφράσεις υπάρχουν σε αυτή την καταχώρηση και αν αυτές υπερβαίνουν το cluster threshold τότε η νέα μετάφραση αποθηκεύεται στο απλό TLB.

3.3 Μελέτη του critical path

Σε ότι αφορά το critical path, ένα Fully-associative Clustered TLB προσφέρει βελτίωση σε σχέση με ένα Fully-associative απλό TLB, κάτι εξαιρετικά σημαντικό από την στιγμή που μελετάμε το L1 TLB. Το critical path αποτελεί την χειρότερη δυνατή διαδρομή ροής δεδομένων η οποία επηρεάζει άμεσα τον χρονισμό του επεξεργαστή. Η βελτίωση στο critical path προκύπτει από:

1. **Την μείωση του αριθμού των καταχωρήσεων λόγω του clustering**

   Στο απλό TLB, κάθε καταχώρηση απεικονίζει μεταφράσεις μία προς μία, συνεπώς το πλήθος των καταχωρήσεων είναι ίσο με το πλήθος των μεταφράσεων. Αντίθετα στο Clustered TLB, πολλαπλές μεταφράσεις συγχωνεύονται μέσα σε μία καταχώρηση. Έτσι το πλήθος των καταχωρήσεων είναι ίσο με

   $$ClusteredEntries = \frac{TotalTranslations}{nClusters}$$

   Αυτή η μεταβολή σε επίπεδο υλικού, αν για παράδειγμα συγκρίνουμε ένα απλό TLB 32 καταχωρήσεων με ένα Clustered TLB 32 μεταφράσεων και cluster factor ίσο με 8, σημαίνει ότι **στο Clustered TLB απαιτούνται 8 φορές (ή αλλιώς όση είναι η τιμή του cluster factor) λιγότεροι συγκριτές για την επιλογή του κατάλληλου entry**. Αυτή η μείωση του πλήθους των καταχωρήσεων, είναι αποδεδειγμένο [9] πως βελτιώνει το critical path, μεταφράζεται όμως και σε μικρότερο κόστος παραγωγής, μικρότερο μέγεθος και λιγότερη κατανάλωση ενέργειας, καθώς απαιτούνται πολύ λιγότερες λογικές πύλες.

2. **Την χρήση λιγότερων bit ως VPN tag**

   Ως απόρροια του clustering, χρησιμοποιούνται log2(nClusters) λιγότερα bit για την (Fully-associative) αναζήτηση του VPN tag. Έτσι, σε ένα Clustered TLB 32 μεταφράσεων και cluster factor ίσο με 8, χρησιμοποιούνται 24 bits αντί για 27, μικραίνοντας το "βάθος" του κυκλώματος σύγκρισης και συνεπώς μειώνοντας περαιτέρω το critical path. Έπειτα από αυτή την αναζήτηση, ακολουθεί μια Direct-mapped αναζήτηση με βάση το VPN Offset για την επιλογή του κατάλληλου ζεύγους PPN Offset/Valid bit, η οποία είναι φθηνή και γρήγορη σε επίπεδο υλικού, επομένως έχει μηδαμινή επίδραση στο critical path.

Μελετώντας, όμως, το Multigranular TLB, οι παράγοντες που πρέπει να ληφθούν υπ' όψιν είναι αρκετά περισσότεροι και αλληλοδιαπλεκόμενοι, με την βασικότερη παράμετρο να είναι η αναλογία μεγέθους του απλού TLB σε σχέση με το Clustered. Έτσι διακρίνουμε δύο οριακές περιπτώσεις:

1. **Όταν ο αριθμός των καταχωρήσεων του απλού TLB είναι ίσος με τον αριθμό των *καταχωρήσεων* στο Clustered TLB** [2]

   Δηλαδή όταν

   $$Conv.Entries = ClusteredEntries = \frac{TotalClusteredTranslations}{nClusters}$$

   Στην περίπτωση αυτή το κύκλωμα σύγκρισης του VPN tag, περιλαμβάνει το ίδιο πλήθος συγκριτών και στα δύο TLB, όμως στο Clustered TLB ο καθένας έχει μέγεθος 24 bits αντί για 27 bits, με το μικρό επιπλέον κόστος της Direct-mapped αναζήτησης. Επομένως το critical path στο Multigranular TLB προκύπτει με βάση το απλό TLB.

2. **Όταν ο αριθμός των καταχωρήσεων του απλού TLB είναι ίσος με τον αριθμό των *μεταφράσεων* στο Clustered TLB** [3]. Δηλαδή όταν

   $$Conv.Entries = ClusteredEntries * nClusters = TotalClusteredTranslations$$

   Στην περίπτωση αυτή το απλό TLB χρησιμοποιεί πολλαπλάσιο πλήθος συγκριτών, οι οποίοι μάλιστα είναι και μεγαλύτερου μεγέθους. Συνεπώς και σε αυτή την περίπτωση το critical path στο Multigranular TLB προκύπτει με βάση το απλό TLB.

Βλέπουμε, λοιπόν, πως σε κάθε περίπτωση το critical path καθορίζεται από το μέγεθος του απλού TLB. Στην καλύτερη περίπτωση, όταν ο αριθμός των καταχωρήσεων του απλού TLB είναι ίσος με τον αριθμό των καταχωρήσεων στο Clustered TLB, είναι αρκετά μικρό, ενώ όσο μεγαλώνει το απλό TLB μεγαλώνει και το critical path (περίπτωση 1). Στην χειρότερη περίπτωση όμως, αν θέλουμε να έχουμε ίσο TLB Reach με ένα απλό TLB, το critical path του Multigranular TLB καθορίζεται από το απλό TLB που βρίσκεται σε αυτό και θα περιέχει τον μισό αριθμό καταχωρήσεων σε σχέση

[2] Δηλαδή για παράδειγμα, αν έχω ένα Clustered TLB 32 **μεταφράσεων** και cluster factor ίσο με 8, τότε ο αριθμός των **καταχωρήσεων** του απλού TLB είναι 32/8=4 καταχωρήσεις

[3] Δηλαδή για παράδειγμα, αν έχω ένα Clustered TLB 32 **μεταφράσεων** και cluster factor ίσο με 8, τότε ο αριθμός των **καταχωρήσεων** του απλού TLB είναι 32 καταχωρήσεις

με ένα απλό TLB που λειτουργεί μόνο του, χωρίς δηλαδή την ύπαρξη του Clustered. Αυτό συμβαίνει όταν ο αριθμός των καταχωρήσεων του απλού TLB είναι ίσος με τον αριθμό των μεταφράσεων στο Clustered TLB (περίπτωση 2). Τέλος, στο Multigranular TLB, πέραν της καθυστέρησης σε επίπεδο πυλών που περιγράφηκε προηγουμένως, προστίθεται μια επιπλέον μικρή καθυστέρηση εξαιτίας του πολυπλέκτη επιλογής PPN.

3.4 Εξοικονόμηση χώρου στο MG-TLB έναντι του Sectored TLB

Προκειμένου να ελέγξουμε το συνολικό μέγεθος κάθε οργάνωσης σε bit, αρχικά πρέπει να δούμε πόσα bit καταναλώνονται για κάθε πεδίο. Στην συνέχεια τα προσθέτουμε για να βρούμε το συνολικό αριθμό bit ανά καταχώρηση αλλά και το πόσα bit χρησιμοποιούνται για κάθε μετάφραση στο Sectored (πίνακας 3.2) και το Clustered TLB (πίνακας 3.3). Αυτό γίνεται μέσω της φόρμουλας

$$TotalBitsPerTranslation = \frac{TotalBitsPerEntry}{nSectors/nClusters}$$

Παρακάτω οι συμβολισμοί SectorX ή ClusterX αναφέρονται σε Sectored ή Clustered TLB με παράγοντες nSectors=X ή nClusters=X αντίστοιχα[4]. Τέλος εξετάζουμε την συνολικό μέγεθος κάθε οργάνωσης του TLB για διαφορετικό αριθμό Ways (πίνακας 3.4) μέσω της φόρμουλας

$$TotalBitsInTLB = nWays * TotalBitsPerTranslation$$

Πεδίο	Sector4	Sector8	Sector16
Tag (VPN)	27	27	27
Valid	4	8	16
PPN Mask	0	0	0
PPN	216	432	864
PPN offset	0	0	0
Attribute bits	60	120	240
Total bits per entry	307	587	1147
Total bits per translation	76.75	73.375	71.6875

Πίνακας 3.2: Ανάλυση των bit που χρειάζονται για κάθε πεδίο στο Sectored TLB

Παρατηρούμε από τους πίνακες 3.2, 3.3 ότι **το Sectored TLB χρησιμοποιεί τον τριπλάσιο χώρο για την αποθήκευση μιας μετάφρασης σε σύγκριση με το Clustered TLB**, το οποίο προσφέρει πολύ μεγαλύτερη πυκνότητα κωδικοποίησης. Αυτό οφείλεται στο γεγονός ότι στο Sectored TLB αποθηκεύονται ολόκληρα τα PPN ενώ στο Clustered TLB αποθηκεύεται μονάχα ένα PPN Mask με πολλαπλά (nSectors) PPN Offset. **Η βελτίωση αυτή έρχεται με αντάλλαγμα τους μεγαλύτερους περιορισμούς που θέτει το Clustered TLB σε σχέση με την ευθυγράμμιση των εικονικών και των φυσικών σελίδων.**

Ο χώρος που εξοικονομείται από την μεγαλύτερη πυκνότητα κωδικοποίησης του Clustered TLB αφιερώνεται τόσο στην αύξηση του TLB Reach, δηλαδή του συνολικού αριθμού μεταφράσεων στο TLB, όσο και στην προσθήκη του δεύτερου απλού TLB το οποίο συμβάλλει στην αποθήκευση μετα-

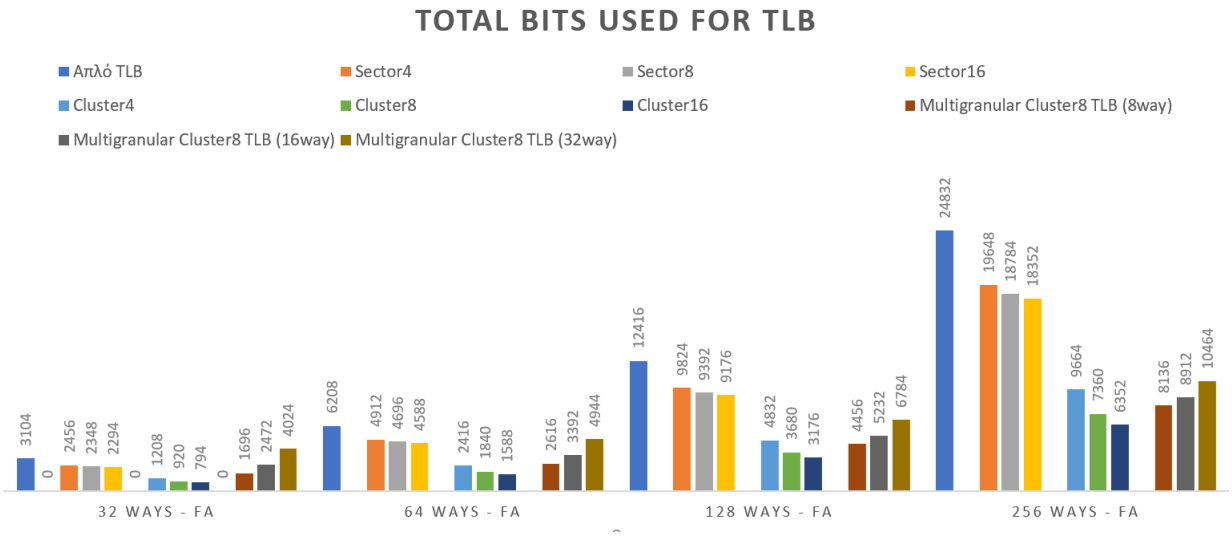
[4] Σε άλλη βιβλιογραφία ο συμβολισμός SectorX ή ClusterX αναφέρεται στον αριθμό των bit που απαιτούνται για την κωδικοποίηση του Offset, δηλαδή X=log2(nSectors ή nClusters)

Πεδίο	Cluster4	Cluster8	Cluster 16	Cluster1
Tag (VPN)	27	27	27	27
Valid	4	8	16	1
PPN Mask	52	51	50	0
PPN	0	0	0	54
PPN offset	4	8	16	0
Attribute bits	60	120	240	15
Total bits per entry	151	230	397	97
Total bits per translation	37.75	28.75	24.8125	97

Πίνακας 3.3: Ανάλυση των bit που χρειάζονται για κάθε πεδίο στο Clustered και το απλό TLB (Cluster1)

	32 Ways	64 Ways	128 Ways	256 Ways
Απλό TLB	3104	6208	12416	24832
Sector4	2456	4912	9824	19648
Sector8	2348	4696	9392	18784
Sector16	2294	4588	9176	18352
Cluster4	1208	2416	4830	9664
Cluster8	920	1840	3680	7360
Cluster16	794	1588	3176	6352
MG Cluster8 (8way)	1696	2616	4456	8136
MG Cluster8 (16way)	2472	3392	5232	8912
MG Cluster8 (32way)	4024	4944	6784	10464

Πίνακας 3.4: Συνολικό μέγεθος οργανώσεων TLB για διαφορετικό αριθμό ways
Σημείωση: Ο συμβολισμός "MG Cluster 8 (8/16/32 way) υποδηλώνει την ταυτόχρονη ύπαρξη ενός Cluster8 TLB με τόσα Ways όσα ορίζονται στην πρώτη σειρά κάθε στήλης και ενός δεύτερου απλού TLB με 8/16/32 ways αντίστοιχα



Σχήμα 3.3: Συνολικά χρησιμοποιημένα bits για κάθε οργάνωση TLB για 32/64/128/256 ways

φράσεων που δεν μπορούν να καλυφθούν εξαιτίας των περιορισμών που τίθενται από το Clustered TLB. Χαρακτηριστικά ο συνδυασμός ενός 256 Ways Cluster8 TLB με ένα 32 Way απλό TLB μπορεί να αποθηκεύσει 288 μεταφράσεις με την χρήση 10464 bit έναντί ενός 128 Ways Sector8 TLB που

χρησιμοποιεί 9392 bits. **Δηλαδή με αύξηση της τάξεως του μόλις 10,24% στο μέγεθος του TLB επιτυγχάνεται αύξηση της τάξεως του 55,5% στο TLB Reach**.

3.5 Διαφορές με την ήδη υπάρχουσα βιβλιογραφία

Η σχεδίαση του Multigranular TLB που περιγράφηκε προηγουμένως, αν και βασίζεται σε ήδη υπάρχουσα βιβλιογραφία [11], διαφέρει από αυτή σε ορισμένα σημεία. Η υπάρχουσα έρευνα μελετά την δυνατότητα συγχώνευσης (coalescing) μεταφράσεων και τα αποτελέσματα αυτής στο L2 TLB. Στην παρούσα διπλωματική εργασία δείχνουμε πως αντίστοιχη λογική είναι δυνατόν να υλοποιηθεί και για το L1 TLB.

Συγκεκριμένα στην **υπάρχουσα έρευνα**, έπειτα από ένα TLB Miss, **το PTW επιστρέφει στο L2 TLB ένα ολόκληρο cache line 64 byte**, που περιέχει την ζητούμενη μετάφραση μαζί με τις γειτονικές της στον πίνακα σελίδων. Έτσι, **ο έλεγχος της δυνατότητας συγχώνευσης (coalescing logic) γίνεται μεταξύ της ζητούμενης μετάφρασης και των μεταφράσεων του cache line** στο οποίο ανήκει η ζητούμενη μετάφραση. Ο έλεγχος αυτός πραγματοποιείται στο L2 TLB, δηλαδή **εκτός του critical path**. Αν το πλήθος των μεταφράσεων που μπορούν να συγχωνευθούν ξεπερνάει το cluster threshold, τότε αυτές αποθηκεύονται στο Clustered L2 TLB, αλλίως αποθηκεύεται στο απλό L2 TLB μόνο η ζητούμενη μετάφραση.

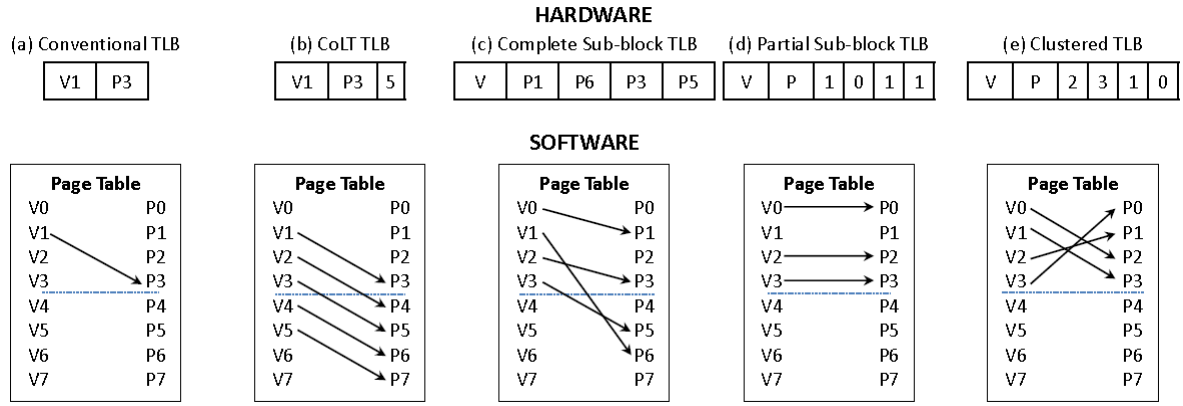
Στην σχεδίαση που προτείνεται στην διπλωματική εργασία, η αναζήτηση για συγχώνευση μεταφράσεων δεν γίνεται με τον ίδιο τρόπο, καθώς έπειτα από ένα TLB Miss, **το PTW επιστρέφει μόνο την ζητούμενη μετάφραση** και όχι τις γειτονικές της. Η μετάφραση αποθηκεύεται ως έχει στο απλό L2 TLB και **το coalescing logic υπάρχει μόνο στο L1 TLB, το οποίο βρίσκεται στο critical path**. Συγκεκριμένα, το coalescing logic **αναζητά αν υπάρχει κάποια ήδη υπάρχουσα καταχώρηση με την οποία μπορεί να συγχωνευθεί η νέα μετάφραση**, δηλαδή κάποια καταχώρηση με την οποία να μοιράζονται κοινά VPN/PPN Masks. Σε περίπτωση έχουν μόνο κοινό VPN Mask με κάποια καταχώρηση, πρέπει να ληθεί μια απόφαση σχετικά με το αν αυτή η καταχώρηση θα ακυρωθεί. Στην περίπτωση όπου το πλήθος των έγκυρων αποθηκευμένων μεταφράσεων της υπό έλεγχο καταχώρησης είναι μεγαλύτερο ή ίσο από το cluster threshold, τότε η νέα μετάφραση θα αποθηκευθεί στο απλό TLB, αλλιώς θα την αντικαταστήσει.

3.6 Σύνοψη

Συνοψίζοντας, βλέπουμε πως το ζήτημα διαχείρισης της εικονικής μνήμης έχει απασχολήσει σε μεγάλο βαθμό την ερευνητική κοινότητα της αρχιτεκτονικής υπολογιστών, κάτι που αποτυπώνεται και στην πληθώρα των προτάσεων που έχουν διατυπωθεί για την οργάνωση του TLB. Βλέπουμε πως αυτές μπορούν να συγκριθούν και να αξιολογηθούν με βάση το μέγεθός τους σε bits, τους περιορισμούς ευθυγράμμισης για τις εικονικές και τις φυσικές σελίδες (πίνακας 3.5), το αν καταφέρνουν να συγχωνεύσουν μεταφράσεις σε μια καταχώρηση ή όχι, το αν αξιοποιούν τα μοτίβα χωρικής τοπικότητας που παράγονται από το Λειτουργικό Σύστημα ή όχι (σχήμα 3.4 και πίνακας 3.5), και σε περίπτωση που τα αξιοποιούν ποια είναι αυτά (συνεχόμενη, ομαδοποιημένη ή και τα δύο). Τέλος μπορούν να συγκριθούν και να αξιολογηθούν με βάση τις επιδόσεις τους σε διαφορετικά workloads, κάτι που θα αναλύσουμε στο Κεφάλαιο 5.

	Περιορισμοί ευθυγράμμισης	Χωρική τοπικότητα
Απλό TLB	Κανένας	Καμία
Coalesced TLB (Fully-associative)	Κανένας	Συνεχόμενη
Coalesced TLB (Set-associative)	VPN και PPN	Συνεχόμενη
Partial Sub-block	VPN και PPN	Συνεχόμενη και Ομαδοποιημένη
Complete Sub-block	VPN	Κυρίως Ομαδοποιημένη
Clustered	VPN και PPN	Ομαδοποιημένη (Μπορεί να εκμεταλλευθεί και την συνεχόμενη)

Πίνακας 3.5: Περιορισμοί ευθυγράμμισης και αξιοποίηση χωρικής τοπικότητας για κάθε οργάνωση του TLB



Σχήμα 3.4: Η αξιοποίηση της χωρικής τοπικότητας των μεταφράσεων στον Πίνακα Σελίδων από κάθε οργάνωση του TLB και οι καταχωρήσεις τους [11]

Το Multigranular TLB, δηλαδή η ταυτόχρονη ύπαρξη και λειτουργία ενός Clustered TLB με ένα απλό TLB, **καταφέρνει να διατηρήσει ένα σχετικά μικρό μέγεθος για τις καταχωρήσεις του**. Αυτό προκύπτει από το γεγονός ότι στο Clustered TLB αποθηκεύονται PPN Offset μεγέθους nClusters bits, αντί για ολόκληρα τα PPN όπως γίνεται στο Sectored. Ταυτόχρονα **αξιοποιεί μοτίβα χωρικής τοπικότητας**, τόσο ομαδοποιημένης (κυρίαρχα) όσο και συνεχόμενης (δευτερευόντως), καταφέρνοντας όμως παράλληλα να καλύψει και μεταφράσεις που δεν εμπίπτουν σε αυτά τα μοτίβα. **Η ύπαρξη του απλού TLB που καλύπτει τις μεταφράσεις της τελευταίας κατηγορίας, βελτιώνει τις επιδόσεις του TLB σε συνθήκες υψηλού memory fragmentation, όπου τα μοτίβα χωρικής τοπικότητας είναι πιο δυσεύρετα, ενώ την ίδια στιγμή επιτρέπει στις καταχωρήσεις του Clustered TLB να συγχωνεύσουν περισσότερες μεταφράσεις**, από την στιγμή που οι ήδη υπάρχουσες δεν θα αντικατασταθούν με άλλες οι οποίες δεν μπορούν να συγχωνευθούν. Τέλος, **βελτιώνει την σχεδίαση σε επίπεδο critical path και κατανάλωσης πόρων** αν συγκριθεί με ένα απλό TLB με ίσο TLB Reach. Το critical path του Multigranular TLB καθορίζεται από το μέγεθος του απλού TLB και ειδικά από την αναλογία μεγέθους που αυτό έχει σε σχέση με το Clustered.

Κεφάλαιο 4

Μεθοδολογία

Στο κεφάλαιο αυτό θα αναλύσουμε τις πλατφόρμες υλικού/λογισμικού που θα χρησιμοποιήσουμε στην παρούσα εργασία. Αρχικά θα περιγράψουμε τα βήματα που ακολουθούμε κατά την διάρκεια σχεδίασης του υλικού, τα απαραίτητα εργαλεία, και έπειτα το λογισμικό με το οποίο ελέγχουμε την ορθότητα και τις επιδόσεις του υλικού.

Η ανάπτυξη του υλικού χωρίζεται σε δύο βήματα: Αρχικά μέσω του **ακριβή ανά κύκλο επεξεργασίας προσομοιωτή Verilator (cycle-accurate hardware emulator)** ελέγχουμε την ορθότητα της σχεδίασης, ενώ ταυτόχρονα λαμβάνουμε κάποιες πρώτες εκτιμήσεις για την επίδοση της σχεδίασης τρέχοντας microbenchmarks μέσω του Spike. Έπειτα χρησιμοποιώντας το Alveo U250 FPGA ελέγχουμε την πραγματική επίδοση της σχεδίασης τρέχοντας πιο απαιτητικά Workloads της σουίτας μετροπρογραμμάτων SPEC2017. Οι διαφορές μεταξύ των προσεγγίσεων συνοψίζονται στον πίνακα 4.1.

Feature	Software emulation	FPGA
Compilation Time	Fast	Slow
Simulation Time	Very Slow	Very Fast
Debugging	Easy	Very Hard

Πίνακας 4.1: Hardware emulation vs FPGA

Αφού μελετήσουμε τα εργαλεία και τις μεθόδους ελέγχου του υλικού στο λογισμικό θα επιστρέψουμε στην διαδικασία στησίματος του bitstream για το FPGA.

4.1 Έλεγχος του υλικού στο λογισμικό

Το πρώτο βήμα της διπλωματικής είναι η ενσωμάτωση των επιθυμητών στον κώδικα Chisel που περιγράφει τον Rocket. Με βάση την διαδικασία μετάφρασης σε Verilog/C++ που περιγράφηκε στην ενότητα 2.3.2, ο Rocket μπορεί να προσομειωθεί μέσω του Verilator και να γίνει έλεγχος της συμπεριφοράς του σε δικά μας ή ήδη υπάρχοντα benchmarks. Τα microbenchmarks που δημιουργήσαμε είναι γραμμένα σε C, μεταφράζονται σε RISC-V binaries μέσω compiler που υπάρχει στα riscv-tools και προσομοιώνονται μέσω του Spike. Τα riscv-tools[1] περιέχουν απαραίτητα εργαλεία για την λειτουργία (compilers, riscv-opcodes, bootloader, kernel), τον έλεγχο (riscv-tests) και την αποσφαλμάτωση (Spike) της σχεδίασής μας. Το Spike είναι ένας γρήγορος προσομοιωτής μέσω του οποίου μπορούμε να εκτελέσουμε RISC-V binaries ώστε να ελέγξουμε την ορθότητα τους πριν τα φορτώσουμε στο FPGA.Η διαφορά των δύο προσομοιωτών έγκειται στο γεγονός ότι το

[1] https://github.com/riscv-software-src/riscv-tools

Spike χρησιμεύει στον γρήγορο έλεγχο και αποσφαλμάτωση του λογισμικού ενώ ο Verilator στον έλεγχο και την αποσφαλμάτωση του υλικού. Η παραπάνω διαδικασία προϋποθέτει την ύπαρξη ενός ελάχιστου περιβάλλοντος linux πάνω στο οποίο θα τρέξουν τα προγράμματα αυτά, το οποίο προσφέρεται μέσω του linux proxy kernel και φορτώνεται μέσω του Berkeley Boot Loader (BBL).

4.1.1 Verilator

Ο Verilator [16] είναι ένα ελεύθερο και ανοιχτού κώδικα εργαλείο το οποίο μετατρέπει την Verilog που παράγει η Chisel σε ένα συμπεριφορικό cycle-accurate C++/SystemC μοντέλο και υποστηρίζει μηνύματα debug ανά κύκλο μηχανής. Έτσι, χρησιμοποιώντας δηλώσεις assert - printf εντός του κώδικα Chisel διευκολύνεται η αποσφαλμάτωση του σχεδιασμού.

Με βάση το εκτελέσιμο που παράγει ο Verilator μπορούμε να τρέξουμε τα official riscv-tests, καθώς και να στήσουμε δικά μας microbenchmarks προκειμένου να λάβουμε κάποια πρώτα αποτελέσματα για την ορθότητα και την επίδοση της σχεδίασης. Χρησιμοποιώντας μηνύματα αποσφαλμάτωσης (printf) μπορούμε σε κάθε κύκλο μηχανής να ελέγξουμε τα περιεχόμενα των κρυφών μνημών, τα εσωτερικά σήματα και να έχουμε εν γένει πλήρη εποπτεία του τι συμβαίνει ανά πάσα στιγμή σε επίπεδο υλικού. Έτσι, η διαδικασία διόρθωσης του υλικού γίνεται αρκετά γρήγορη και παραγωγική. Βέβαια παρά το γεγονός ότι ο Verilator διευκολύνει πολύ το debugging, είναι εξαιρετικά αργός[2].

4.1.2 Linux-pk και Berkeley Boot Loader (BBL)

Ο BBL είναι ο bootloader του RISC-V Linux, τρέχει σε M-mode οπότε έχει πλήρη διαφάνεια στον RISC-V Core, προετοιμάζει το σύστημα, δηλαδή τους απαραίτητους CSRs και το μηχανισμό Physical Memory Protection (PMP) και έπειτα παραδίδει την εκτέλεση στο Linux. Ο BBL περιέχεται στο πακέτο riscv-pk το οποίο μπορούμε να εντοπίσουμε εντός των riscv-tools. Το πακέτο riscv-pk πέρα από τον BBL περιέχει τον RISC-V Proxy Kernel (PK) ο οποίος είναι ένας minimal kernel που χρησιμοποιείται από το Spike για την εκτέλεση προγραμμάτων χώρου χρήστη.

Τροποποιούμε τον BBL ούτως ώστε να μπορούμε να διαβάζουμε και να ενημερώνουμε τους performace counters για τα γεγονότα που μας ενδιαφέρουν, όπως είναι οι αστοχίες στο Data TLB και οι κύκλοι ρολογιού. Τέλος, ελέγχουμε ότι είναι ενεργοποιημένη η προώθηση (delegation[3]) των performance counters στο U-mode, ώστε να έχουμε πρόσβαση από τον χώρο χρήστη (όπου τρέχουν τα benchmarks) στους μετρητές.

4.1.3 Έλεγχος του τελικού εκτελέσιμου με το Spike

Το Spike είναι ένας γρήγορος προσομοιωτής εκτελέσιμων της αρχιτεκτονικής RISC-V. Με το Spike ελέγχουμε την ορθότητα του λογισμικού χωρίς να μπορούμε όμως να δούμε την εσωτερική κατάσταση του επεξεργαστή όπως κάνουμε με τον Verilator. Δεν μπορούμε να δούμε τις τιμές των CSRs, τα σήματα ελέγχου καθώς και το εσωτερικό των κρυφών μνημών όπως για παράδειγμα το TLB.

[2] Μία ημέρα για να εκκινήσει linux σε σύγχρονο επεξεργαστή x86-64

[3] Βλέπε καταχωρητές `mcounteren`, `scounteren`, RISC-V ISA Spec Volume II: Privileged Architecture

4.2 Δημιουργία bitstream

Αφού ολοκληρωθεί ο έλεγχος της σχεδίασης του υλικού στον Verilator, μπορούμε να προχωρήσουμε στην διαδικασία παραγωγής του bitstream. **Το bitstream περιέχει την περιγραφή της λογικής του hardware, τον προγραμματισμό των Look-up Tables (LUTs), Flip-Flops (FFs) του PL καθώς και την δρομολόγηση (routing) μεταξύ των στοιχείων συνδυαστικής λογικής και μνήμης**.

Για την παραγωγή του bitstream ακολουθήσαμε της οδηγίες μηχανικού της Xilinx όπως αναγράφονται στο **repository** `vivado-risc-v`[4] το οποίο περιέχει:

1. **Αυτοματοποιημένα script για την παραγωγή του bitstream** που στοχεύουν σε μια πληθώρα FPGA για διαφορετικές εκδόσεις του Vivado (2020.x, 2021.x).

2. **RISC-V Open Source Supervisor Binary Interface (OpenSBI)**:[5] Στην παρούσα υλοποίηση αποτελεί την διεπαφή μεταξύ ενός εξειδικευμένου - ανάλογα το υλικό - κώδικα που τρέχει σε M-Mode και του bootloader, αλλά μπορεί να χρησιμοποιηθεί και ως διαφορετική διεπαφή.

3. **U-Boot**[6]: Αποτελεί τον Last Stage Bootloader του συστήματος, αλλά μπορεί να χρησιμοποιηθεί και ως First Stage Boot Loader (FSBL)

4. **Linux Kernel 5.15.4 και Debian root file system** Για την υποστήριξη λειτουργικού συστήματος και την χρήση benchmarks που τρέχουν σε χώρο χρήστη.

Το πρώτο βήμα είναι να κάνουμε clone το repository, να εγκαταστήσουμε και να ενημερώσουμε τα submodules που χρησιμοποιούνται. Έπειτα αντικαθιστούμε το αρχείο TLB.scala που περιέχει την περιγραφή του παλιού TLB με το αρχείο TLB.scala που αντιστοιχεί στην περιγραφή της νέας οργάνωσης που υλοποιήθηκε.

Το επιθυμητό bitstream δημιουργείται εκτελόντας την εντολή `make CONFIG=xxx BOARD=yyy bitstream` όπου αντί για xxx επιλέγουμε την επιθυμητή οργάνωση του επεξεργαστή όσον αφορά τον χώρο διευθύνσεων (32/64 bit), τον αριθμό και το μέγεθος των επεργαστικών πυρήνων, το μέγεθος της L2 cache, ενώ μπορούμε επίσης να δημιουργήσουμε επεξεργαστές BOOM. Υπάρχει η δυνατότητα δημιουργίας επιπλέον οργανώσεων, πέραν των ήδη υπάρχοντων, μέσω της επεξεργασίας του αρχείου που βρίσκεται στο path vivado-risc-v/src/main/scala/rocket.scala. Η παράμετρος yyy καθορίζει την πλατφόρμα για την οποία δημιουργείται το bitstream.

Για κάθε αλλαγή που γίνεται στο TLB, όπως το μέγεθός του, το associativity και ο βαθμός clustering, είναι απαραίτητη η εκ νέου παραγωγή του bitstream. Προκειμένου να αυτοματοποιηθεί αυτή η διαδικασία παραμετροποιήθηκε το repository rocket-panel[7] το οποίο μας δίνει την δυνατότητα μαζικής παραγωγής bitstream. Στο script `generate-bitstream.sh` "πακετάρονται" όλες οι εντολές που είναι απαραίτητες για την παραγωγή ενός μοναδικού bitstream, το στήσιμο και την εκκίνηση του περιβάλλοντος linux, όπως περιγράφεται στην επόμενη ενότητα. Στο script `rocket-panel.sh` αυτοματοποιούνται οι αλλαγές που πρέπει να γίνουν στον κώδικα Chisel του Rocket

[4] https://github.com/eugene-tarassov/vivado-risc-v
[5] https://github.com/riscv-software-src/opensbi
[6] https://github.com/u-boot/u-boot
[7] https://github.com/ncppd/rocket-panel

προκειμένου να γίνει εξερεύνηση του χώρου σχεδίασης βάσει των παραγόντων που προαναφέρθηκαν.

Στην παρούσα εργασία χρησιμοποιήσαμε το **FPGA Xilinx U250,** προκειμένου να ελέγξουμε την σχεδίαση. Μέσω του vivado-risc-v δημιουργούμε bitstream τα οποία στοχεύουν στην δημιουργία μονοπύρηνων επεξεργαστικών πυρήνων (`CONFIG=rocket64b1`). Λόγω έλλειψης υποστήριξης, η σχεδίαση μπορεί να "δει" μόνο 2GB μνήμης RAM.

4.3 Στήσιμο και εκκίνηση περιβάλλοντος Debian Linux στο Xilinx U250 FPGA

Μετά την παραγωγή του bitstream και την ολοκλήρωση των σταδίων Synthesis και Implementation, έπεται το στησίμο και η εκκίνηση ενός περιβάλλοντος Linux. Η διαδικασία αυτή περνάει μέσα από διάφορα στάδια:

1. Αρχικά εκτελείται κώδικας σε M-Mode που αρχικοποιεί τον επεξεργαστή, τον memory controller, δημιουργεί τον χάρτη μνήμης και έπειτα παραδίδει την εκτέλεση στον FSBL που βρίσκεται σε μια διεύθυνση προκαθορισμένη από τον κατασκευαστή του FPGA.

2. Ο **First Stage Boot Loader (FSBL)** προετοιμάζει το σύστημα, δηλαδή τους απαραίτητους CSRs και το μηχανισμό Physical Memory Protection (PMP) και έπειτα παραδίδει την εκτέλεση στον Second Stage Bootloader. Ο λόγος ύπαρξης δύο διαφορετικών σταδίων boot loader έγκειται στο μικρό επιτρεπόμενο μέγεθος του πρώτου. Στην παρούσα εργασία τον ρόλο αυτό τον επιτελεί το **OpenSBI**.

3. Ο **Second Stage Boot Loader (SSBL)** εκτελεί τον απαραίτητο κώδικα για την εκκίνηση του kernel και έπειτα παραδίδει την εκτέλεση σε αυτόν. Με την φράση "παραδίδει την εκτέλεση" εννοείται πως ο SSBL παύει την εκτέλεση του και έπειτα εκτελείται ο κώδικας του kernel, δίχως να επιστρέφει η εκτέλεση στον SSBL όταν τρέξει ο κώδικας του kernel. Στην παρούσα υλοποίηση τον ρόλο αυτό τον επιτελεί το U-Boot, το οποίο διαθέτει υποστήριξη για την αναγνώριση παραπάνω από ενός kernel προσφέροντας την δυνατότητα στησίματος multi-boot συστημάτων.

4. Εκκινείται ο **Linux kernel**, ο οποίος είναι υπεύθυνος για τον εντοπισμό και την διαχείριση του συστήματος αρχείων (file system), την διαχείριση της μνήμης, των συσκευών, του δικτύου και τον συντονισμό μεταξύ των διαφορετικών προγραμμάτων που τρέχουν ταυτόχρονα. Ως σύστημα αρχείων ορίζεται η δομή που περιγράφει την οργάνωση των αρχείων στα partition του δίσκου, την διασύνδεσή τους, τα δικαιώματα πρόσβασης σε αυτά και τον τρόπο με τον οποίο αποθηκεύονται και προσπελάσονται. Κατά την εκκίνησή του ο πυρήνας του linux αναζητά το root filesystem (rootfs) το οποίο αφ' ενός είναι η ρίζα (root) για τα υπόλοιπα συστήματα αρχείων και αφ' ετέρου περιέχει απαραίτητο κώδικα για την εκκίνηση του συστήματος.

Λόγω τεχνικών προβλημάτων που παρουσιάστηκαν, η διαδικασία που ακολουθήθηκε διαφέρει ελαφρώς από εκείνη που περιγράφεται στο repository. Λόγω απουσίας υποστήριξης κάρτας SD από το U250 FPGA είχαμε **δύο επιλογές σχετικά με την τοποθέτηση του root filesystem** και

των αρχείων της σουίτας μετροπρογραμμάτων (benchmarks suite) SPEC: είτε θα τα φορτώναμε στην RAM, είτε μέσω της τεχνολογίας NFS (Network File System). Η πρώτη επιλογή αφ' ενός θα απαιτούσε επιπλέον τροποποιήσεις στους bootloaders και στον linux kernel, αφ' ετέρου θα έπαιρνε πάρα πολλές μέρες καθώς θα έπρεπε να γίνει μέσω του πρωτοκόλλου JTAG. Έτσι, το FPGA "βλέπει" το root filesystem μέσω NFS, δηλαδή μέσω δικτύου, και στην πραγματικότητα βρίσκεται σε υπολογιστή του εργαστηρίου, του οποίου η διεύθυνση IP είναι γνωστή και στατική.

4.4 SPEC 2017 [4]

Για την αξιολόγηση των αποτελεσμάτων της σχεδίασής μας χρησιμοποιήθηκε η σουίτα μετροπρογραμμάτων (benchmark suite) SPEC2017, που παρέχει εφαρμογές μεγάλου μεγέθους οι οποίες προκαλούν μεγάλο αριθμό από TLB Misses. Τα συνολικά 43 benchmarks χωρίζονται σε 4 κατηγορίες: SPECrate Integer, SPECspeed Integer, SPECrate Floating Point, SPECspeed Floating Point. Οι εφαρμογές της κατηγορίας SPECspeed χρησιμοποιούν ως μετρική αξιολόγησης τον χρόνο που απαιτείται για να ολοκληρωθούν, ενώ οι εφαρμογές της κατηγορίας SPECraate χρησιμοποιούν ως μετρική αξιολόγησης πόσες φορές μπορεί να τρέξει μια εφαρμογή σε ένα δεδομένο χρονικό διάστημα (throughput). Σχεδόν κάθε benchmark διαθέτει και τις δύο εκδόσεις. Από την άλλη τα benchmarks της κατηγορίας integer περιέχουν πράξεις σχεδόν αποκλειστικά μεταξύ ακέραιων αριθμών, ενώ εκείνα της κατηγορίας Floating Points περιέχουν δεκαδικούς αριθμούς. Για την αξιολόγηση της σχεδίασης τρέξαμε τα benchmarks των κατηγοριών intspeed, intrate με test input. Από αυτά υπήρχε αδυναμία στο να τρέξουμε τα 502/602.gcc_r/s, 520/620.omnetpp_r/s καθώς και το 631.deepsjeng_r/s εξαιτίας ελλιπών βιβλιοθηκών.

4.5 Σύνοψη μεθοδολογίας

Αφού ολοκληρώσουμε τα παραπάνω βήματα θα έχουμε έτοιμα τα εργαλεία ανάπτυξης υλικού καθώς και το λογισμικό για τον έλεγχο ορθότητας και επίδοσης του υλικού. Ανοίγοντας το Vivado μπορούμε να πάρουμε πληροφορίες σχετικά με την σχεδίαση μας και πόσους πόρους δεσμεύει όπως LUTs, FFs και Block RAM. Στα επόμενα κεφάλαια θα χρησιμοποιήσουμε τα εργαλεία που ετοιμάσαμε για την υλοποίηση και την εκτίμηση της επίδοσης της σχεδίασής μας.

Κεφάλαιο 5

Αξιολόγηση αποτελεσμάτων, συγκρίσεις και συμπεράσματα

Στο κεφάλαιο αυτό θα αναλύσουμε την επίδοση του αρχικού Sectored TLB σε σύγκριση με τα Clustered TLB και Multigranular (MG) Clustered TLB με βάση μια σειρά μετρικών, οι τιμές των οποίων προκύπτουν έπειτα από την εκτέλεση της σουίτας μετροπρογραμμάτων SPEC2017.

5.1 Μετρικές Ανάλυσης

Οι μετρικές που θα χρησιμοποιηθούν στην παρούσα εργασία χωρίζονται σε τρεις ομάδες: στις μετρικές επιδόσεων, στις μετρικές χώρου που προκύπτουν από την υλοποίηση στο FPGA και τον υπολογισμό των συνολικών bit που χρειάζονται για κάθε υλοποίηση καθώς και στον συνδυασμό τους.

- **Μετρικές πόρων**: Προκύπτουν από την ανάλυση των bit που χρειάζεται για κάθε πεδίο κάθε οργάνωσης του TLB.
- **Μετρικές επίδοσης**: Εξέταση της σουίτας μετροπρογραμμάτων SPEC2017
  - **L1 dTLB misses**: Οι συνολικές αστοχίες στο L1 Data TLB κατά την διάρκεια εκτέλεσης ενός προγράμματος.
  - **Total cycles**: Οι συνολικοί κύκλοι επεξεργασίας κατά την εκτέλεση ενός προγράμματος.
  - **IPC (Instructions per Cycle)**: Αποτελεί το πλήθος των εντολών που ολοκληρώνονται κατά μέσο όρο σε κάθε κύκλο επεξεργασίας. Η αύξησή του υποδεικνύει την καλύτερη επίδοση της σχεδίασης.
  - **MPKI (Misses Per Kilo Instructions)**: Αποτελεί το πλήθος των αστοχιών στο L1 Data TLB που συμβαίνουν κατά μέσο όρο ανά 1000 εντολές. Η αύξησή του υποδεικνύει χειροτέρευση της επίδοσης της σχεδίασης.

Την ποσοτική διαφορά των τιμών που εξετάζουμε θα την υπολογίζουμε με την φόρμουλα:

$$PercentIncrease = \frac{NewValue - OriginalValue}{OriginalValue} * 100$$

Η φόρμουλα αυτή μπορεί να πάρει αρνητικές ή θετικές τιμές τις οποίες θα τις αναφέρουμε ως μείωση ή αντίστοιχα αύξηση.

5.2 Αναλυτική παρουσίαση των αποτελεσμάτων

Στις επόμενες υποενότητες παρουσιάζονται τα αποτελέσματα των μετρήσεων που ελήφθησαν με βάση τις μετρικές πόρων και επίδοσης οι οποίες προαναφέρθηκαν. Η πληθώρα των μετρικών αλλά κυριάρχα των παραμέτρων που πρέπει να εξεταστούν έκανε αναγκαία την συγκεκριμενοποίηση του σκεπτικού βάση του οποίου θα συγκρίνουμε διαφορετικές οργανώσεις TLB μεταξύ τους, ούτως ώστε να συγκριθούν όσο το δυνατόν περισσότερο οργανώσεις με κοντινά χαρακτηριστικά. Προκειμένου τα αποτελέσματα να είναι πιο ακριβή ως προς την επίδραση των αλλαγών που έγιναν, **απενεργοποιήθηκαν το L2 TLB καθώς και η Page Table Walk Cache**. Βάση αυτού του σκεπτικού επιλέχθηκαν συγκεκριμένοι συνδυασμοί μεγεθών (πίνακας 5.1) προκειμένου να συγκρίνουμε το Sectored TLB με το Multigranular TLB με βάση αφ' ενός το μέγεθος κάθε οργάνωσης TLB σε bits (υποενότητα 5.2.1) και αφ' ετέρου το TLB Reach (υποενότητα 5.2.2).

Config Name	L1 TLB	L1 Conventional TLB	TLB Reach	L1 TLB Size in bits
C8_32FA_0	32 Entries - FA	0	32	920
C8_32FA_16	»	16	48	2472
C8_32FA_32	»	32	64	4024
C8_64FA_0	64 Entries - FA	0	64	1840
C8_64FA_32	»	32	96	4944
C8_64FA_64	»	64	128	8048
S8_32FA	32 Entries - FA	-	32	2348
S8_64FA	64 Entries - FA	-	64	4696
S8_128FA	128 Entries - FA	-	128	9392

Πίνακας 5.1: Οργανώσεις του Multigranular TLB και Sectored TLB για τις οποίες λήφθηκαν οι μετρήσεις

Η ονομασία των οργανώσεων ακολουθεί το εξής σκεπτικό: Αρχικά αναγράφεται το είδος του TLB στο οποίο αναφερόμαστε (C για το Multigranular και S για το Sectored) μαζί με τον cluster/sector factor της οργάνωσης. Έπειτα ακολουθεί το μέγεθος του L1 TLB μαζί με το associativity και τέλος (μόνο για το multigranular TLB) το μέγεθος του απλού TLB.

Σε αυτό το σημείο θα πρέπει να αναφερθεί ότι **κατά την ανάλυση των μετρήσεων εμφανίζονται ορισμένες απρόβλεπτες συμπεριφορές, τόσο στο Sectored TLB όσο και στο Multigranular TLB**. Χαρακτηριστικό παράδειγμα αυτών είναι το γεγονός ότι η αύξηση των καταχωρήσεων, τόσο του Sectored όσο και του Clustered TLB, από 32 σε 128, δεν συνοδεύεται από μείωση των Data TLB Misses όπως αναμενόταν, αλλά αντίθετα από την αύξησή τους. Αυτή η συμπεριφορά μπορεί να οφείλεται σε κάποιο ήδη υπάρχων bug στο TLB ή ακόμα και στο access pattern ορισμένων εφαρμογών ή και στα δύο. Καθώς ο Rocket Chip Generator είναι μεγάλο project, συχνά εμφανίζονται bugs τα οποία γίνονται trigger από συγκεκριμένα access patterns [1]. Αυτό δεν σημαίνει ότι το TLB δεν λειτουργεί σωστά, αλλά ότι η συμπεριφορά του δεν είναι η προβλεπόμενη.

5.2.1 Σύγκριση Sectored και Multigranular TLB με βάση το μέγεθος της κάθε οργάνωσης σε bit

Όπως έχει ήδη αναφερθεί, το Clustered TLB αποθηκεύει πολύ λιγότερες πληροφορίες σε σχέση με το Sectored, κάτι που μικραίνει σε μεγάλο βαθμό το μέγεθος του σε bits. Ταυτόχρονα απελευθε-

[1] https://github.com/chipsalliance/rocket-chip/pull/2601

ρώνει χώρο για την προσθήκη του δεύτερου απλού TLB, ο συνδυασμός των οποίων οδηγεί στην οργάνωση του Multigranular TLB, ισοσκελίζοντας με αυτό τον τρόπο τους επιπλέον περιορισμούς -και άρα την ενδεχόμενη μείωση των επιδόσεων- που τίθενται από το Clustered TLB σε σχέση με την φυσική μάσκα.

Στην ενότητα αυτή εξετάζουμε ποια από τις δύο προσεγγίσεις για την οργάνωση του TLB αξιοποιεί καλύτερα τον ίδιο -προσεγγιστικά- χώρο.

5.2.1.1 Σύγκριση S8_32FA - C8_32FA_16 - C8_64FA_0 - C8_16FA_16

	S8_32FA	C8_16FA_16	C8_32FA_16	C8_64FA_0
Size in bits	2348	-14.31%	5.28%	-21.64%
CPU Cycles (B)	3,160	1.31%	6.07%	1.27%
L1 dTLB Misses (M)	22,108	10.02%	12.80%	11.05%
Average IPC	0.5026	-1.14%	-1.87%	-1.23%
Average MPKI	13.919	9.86%	8.37%	11.01%

Πίνακας 5.2: Συγκεντρωτικός πίνακας συγκρίσεων S8_32FA - C8_16FA_16 - C8_32FA_16 - C8_64FA_0

Σε TLB μικρού μεγέθους η συνολική επίδοση όλων των οργανώσεων που φαίνεται στον πινακα 5.2 είναι αρκετά κοντινή, με εξαίρεση το C8_32FA_16 που παρουσιάζει αύξηση 6.07% στους συνολικούς κύκλους εκτέλεσης, ενώ το S8_32FA βέβαια έχει ένα ελαφρύ προβάδισμα σε σχέση με τις υπόλοιπες οργανώσεις όσον αφορά αυτόν τον τομέα. Παρ' όλα αυτά η μείωση του απαιτούμενου χώρου κατά 21.64% στο C8_64FA_0 είναι αξιοσημείωτη, αντισταθμίζονας την μικρή αύξηση στους κύκλους εκτέλεσης που προκύπτει από τα σαφώς περισσότερα Data TLB Misses.

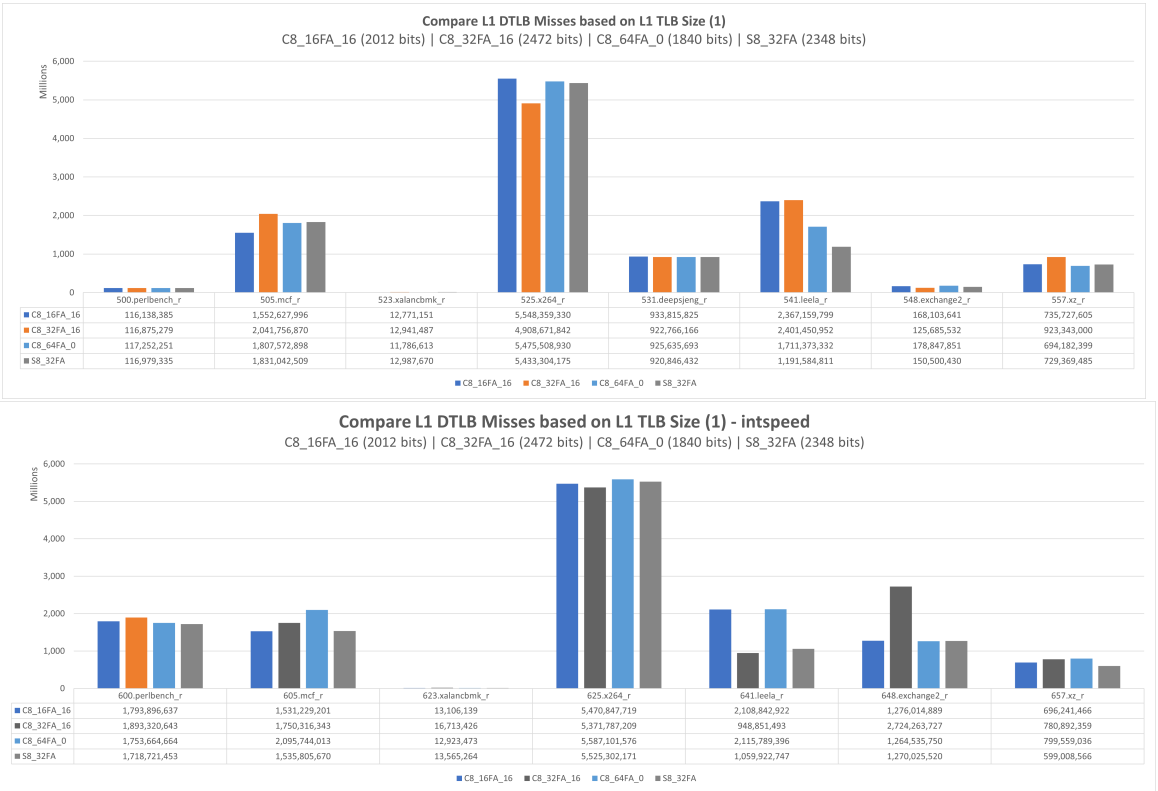
Σε ότι αφορά τα επιμέρους benchmarks, οι βασικές διαφορές μεταξύ του S8_32FA και του C8_64FA_0 όσον αφορά τα Data TLB Misses εντοπίζονται στα 541.leela_r (+43,62%), 605.mcf_s(+36.46%), 641.leela_s(+99,62%), 657.xz_s (+33.48%) [2]. Παρατηρούμε ότι κατά κύριο λόγο η αύξηση στα L1 Data TLB Misses οφείλεται στα benchmarks της κατηγορίας intspeed, όπου εκτελούνται ταυτόχρονα πολλά instances του ίδιου benchmark. Επίσης τα intspeed benchmarks εκτελούνται υπό συνθήκες υψηλού memory fragmentation, καθώς εκτελούνται έπειτα από εκείνα της κατηγορίας intrate. Επομένως είναι λογικό το Sectored TLB να αποδίδει καλύτερα από την στιγμή που αποθηκεύει ολόκληρο τον αριθμό φυσικής σελίδας (PPN) για κάθε εικονική.

5.2.1.2 Σύγκριση S8_64FA - C8_32FA_32 - C8_64FA_32 - C8_128FA_0 - C8_128FA_16

Με την αύξηση του μεγέθους του L1 TLB παρατηρούμε ότι οι οργανώσεις του Multigranular TLB αρχίζουν να αποδίδουν καλύτερα σε σημαντικό βαθμό σε σύγκριση με το Sectored. Αρχικά το C8_128FA_0 επιτυγχάνει αποτελέσματα με μηδαμινές διαφορές σε σχέση με το S8_64FA χρησιμοποιώντας 21,64% λιγότερο χώρο, κάτι που μεταφράζεται μεταξύ άλλων σε χρήση λίγοτερης ενέργειας. Η προσθήκη του δεύτερου απλού TLB 16 καταχωρήσεων (C8_128FA_16) φαίνεται ότι έχει μια μικρή επίδραση στην μείωση των Data TLB Misses, του MPKI και την αύξηση του IPC,

[2] Τα ποσοστά που αναγράφονται στις παρενθέσεις αποτελούν την αύξηση(+)/μείωση(-) των DTLB Misses στα Multigranular TLBs σε σχέση με τα Sectored

Σχήμα 5.1: Πλήθος κύκλων επεξεργαστή για την περάτωση των benchmarks - S8_32FA - C8_32FA_16 - C8_64FA_0 - C8_16FA_16



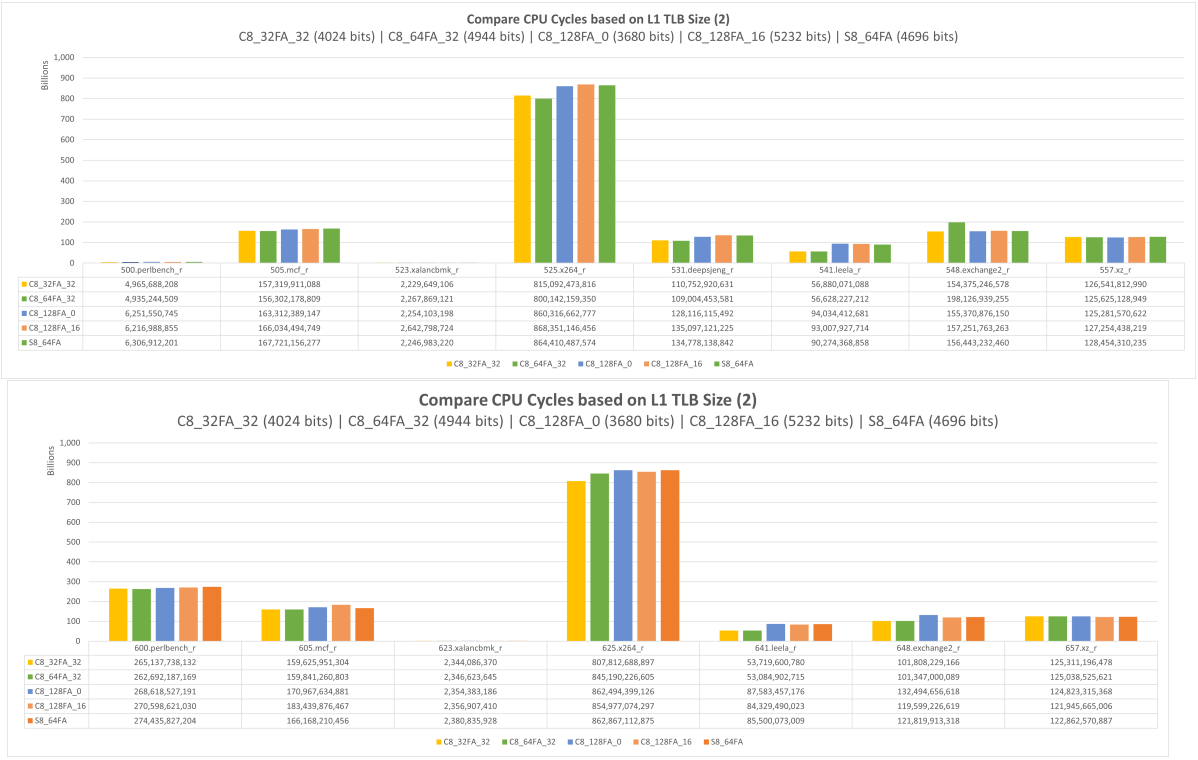
Σχήμα 5.2: Πλήθος αστοχιών L1 TLB - S8_32FA - C8_32FA_16 - C8_64FA_0 - C8_16FA_16

	S8_64FA	C8_32FA_32	C8_64FA_32	C8_128FA_0	C8_128FA_16
Size in bits	4696	-14.31%	5.28%	-21.64%	11.41%
CPU Cycles (B)	3,186	-7.62%	-5.78%	-0.08%	0.20%
L1 dTLB Misses (M)	24,756	-63.60%	-62.92%	0.12%	-1.82%
Average IPC	0.4987	7.97%	7.23%	0.17%	6.80%
Average MPKI	15.578	-63.50%	-63.31%	0.02%	-8.26%

Πίνακας 5.3: Συγκεντρωτικός πίνακας συγκρίσεων S8_64FA - C8_32FA_32 - C8_64FA_32 - C8_128FA_0 - C8_128FA_16

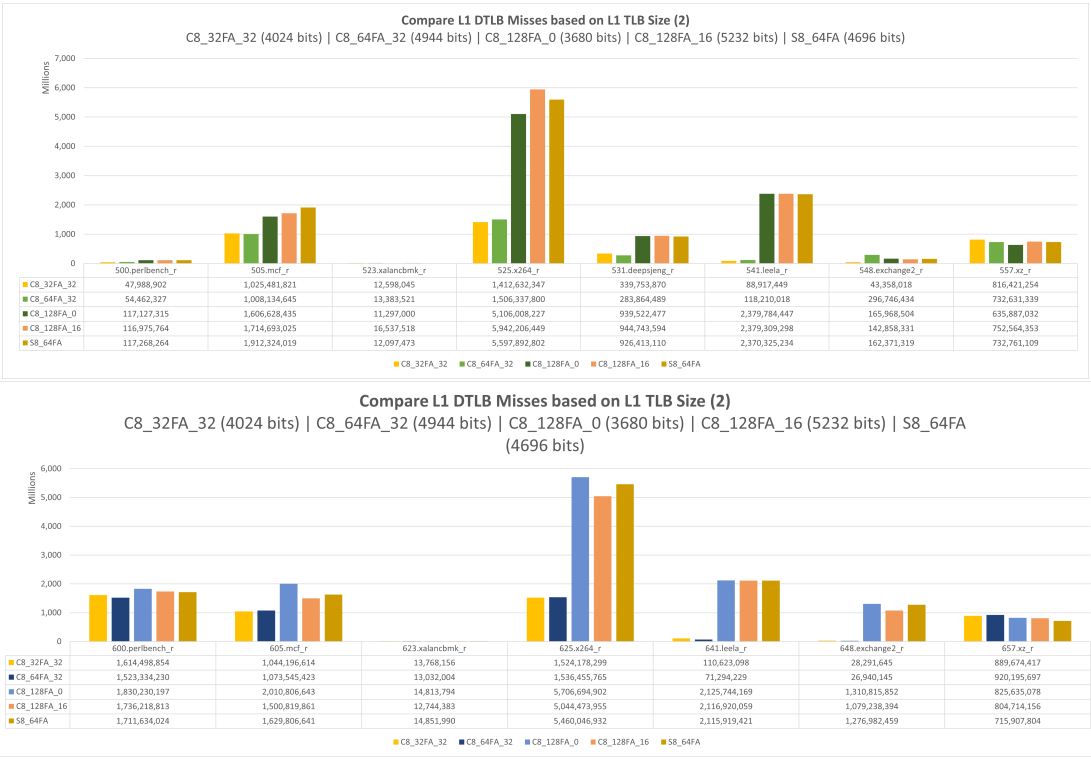
αλλά με το κόστος του επιπλέον χώρου κατά 11,41% που καθιστά την επιλογή του συγκεκριμένου configuration ασύμφορη.

Τα πιο εντυπωσιακά αποτελέσματα καταγράφονται από τα C8_32FA_32 και C8_64FA_32 με μείωση των συνολικών κύκλων κατα 7,62% και 5,78% αντίστοιχα, με το C8_64FA_32 όμως να έχει το μειονέκτημα του αυξημένου μεγέθους του. Από την άλλη **το C8_32FA_32 καταγράφει τον μικρότερο αριθμό κύκλων και τον δεύτερο λιγότερο αριθμό L1 Data TLB Misses, αποτελώντας την βέλτιστη πρόταση που προκύπτει από τις μετρήσεις**, καθώς συνδυάζει την εξοικονόμηση χώρου με πολύ καλές επιδόσεις. Συγκεκριμένα επιτυγχάνεται μείωση κατά 63,6% στα L1 Data TLB Misses, μείωση κατά 7,62% στους κύκλους και μείωση κατά 14,31% (ή 1672 bits) σε σχέση με το S8_64FA.



Σχήμα 5.3: Πλήθος κύκλων επεξεργαστή για την περάτωση των benchmarks - S8_64FA - C8_32FA_32 - C8_64FA_32 - C8_128FA_0 - C8_128FA_16

Σε ότι αφορά τα επιμέρους benchmarks το C8_32FA_32 αποδίδει καλύτερα τόσο σε εκείνα της κατηγορίας intrate (-7,9% στους κύκλους και -68% στα L1 Data TLB Misses) όσο και σε εκείνα της κατηγορίας intspeed (-7,35% στους κύκλους και -59,57% στα L1 Data TLB Misses). Τα μόνα



Σχήμα 5.4: Πλήθος αστοχιών L1 TLB - S8_64FA - C8_32FA_32 - C8_64FA_32 - C8_128FA_0 - C8_128FA_16

benchmarks στα οποία δεν υπήρχε βελτίωση ήταν τα 523.xalancbmk_r (-0,77% στους κύκλους αλλά +4,13% στα L1 Data TLB Misses), 557.xz_r (-1,48% στους κύκλους αλλά +11,41% στα L1 Data TLB Misses) και 657.xz_s (+2% στους κύκλους και -24,27% στα L1 Data TLB Misses). Όμως ακόμα και στα πιο ”βαριά” benchmarks, εκείνα δηλαδή στα οποία εκτελείται μεγάλος αριθμός εντολων το C8_32FA_32 παρουσιάζει εξαιρετικά αποτελέσματα. Χαρακτηριστικά παραδείγματα είναι τα 505.mcf_r (-6,2% στους κύκλους και -46,37% στα L1 Data TLB Misses), 525.x264_r (-5,7% στους κύκλους και -74,76% στα L1 Data TLB Misses), 541.leela_r (-36,92% στους κύκλους και -96,25% στα L1 Data TLB Misses).

5.2.1.3 Σύγκριση S8_128FA - C8_64FA_64 - C8_128FA_64

	S8_128FA	C8_64FA_64	C8_128FA_64
Size in bits	9392	-14.31%	5.28%
CPU Cycles (B)	3,297	-1.14%	-10.63%
L1 dTLB Misses (M)	24,654	-19.95%	-64.02%
Average IPC	0.4949	-1.65%	9.18%
Average MPKI	15.106	-17.67%	-63.13%

Πίνακας 5.4: Συγκεντρωτικός πίνακας συγκρίσεων S8_128FA - C8_64FA_64 - C8_128FA_64

Και σε αυτή την περίπτωση το Multigranular TLB φαίνεται να υπερτερεί έναντι του Sectored. Για αρχή το C8_64FA_64 συνδυάζει το μειωμένο μέγεθος με βελτίωση στο πλήθος των κύκλων και των L1 Data TLB Misses, ακόμα και αν δεν είναι στον αναμενόμενο βαθμό. Το C8_64FA_32 που παρουσιάστηκε στην προηγούμενη υποενότητα απέδιδε ακόμα καλύτερα σε ακόμη μικρότερο χώρο.

Έτσι φαίνεται πως το μέγεθος του απλού TLB κατέχει σημαντικό ρόλο στην συνολική επίδοση της σχεδίασης. Την ίδια στιγμή το C8_128FA_64 παρουσιάζει σημαντικά καλύτερα αποτελέσματα (-10,63% στους κύκλους και -64,02% στα L1 Data TLB Misses) έχοντας το μειονέκτημα της χρήσης ελαφρώς παραπάνω χώρου κατά 5,28% (ή 496 bits). Η συγκεκριμένη σχεδίαση επιτυγχάνει τα λιγότερα L1 Data TLB Misses και το δεύτερο μικρότερο πλήθος κύκλων επεξεργασίας σε σχέση με όσες σχεδιάσεις μελετήθηκαν. Σε ότι αφορά τα επιμέρους benchmarks το C8_128FA_64 υπερτερεί του S8_128FA σε όλα τα benchmarks πλην των 523.xalancbmk_r (+1,25% στα L1 Data TLB Misses) και 557.xz_r (+17,4% στους κύκλους).

Σχήμα 5.5: Πλήθος κύκλων επεξεργαστή για την περάτωση των benchmarks - S8_128FA - C8_64FA_64 - C8_128FA_64

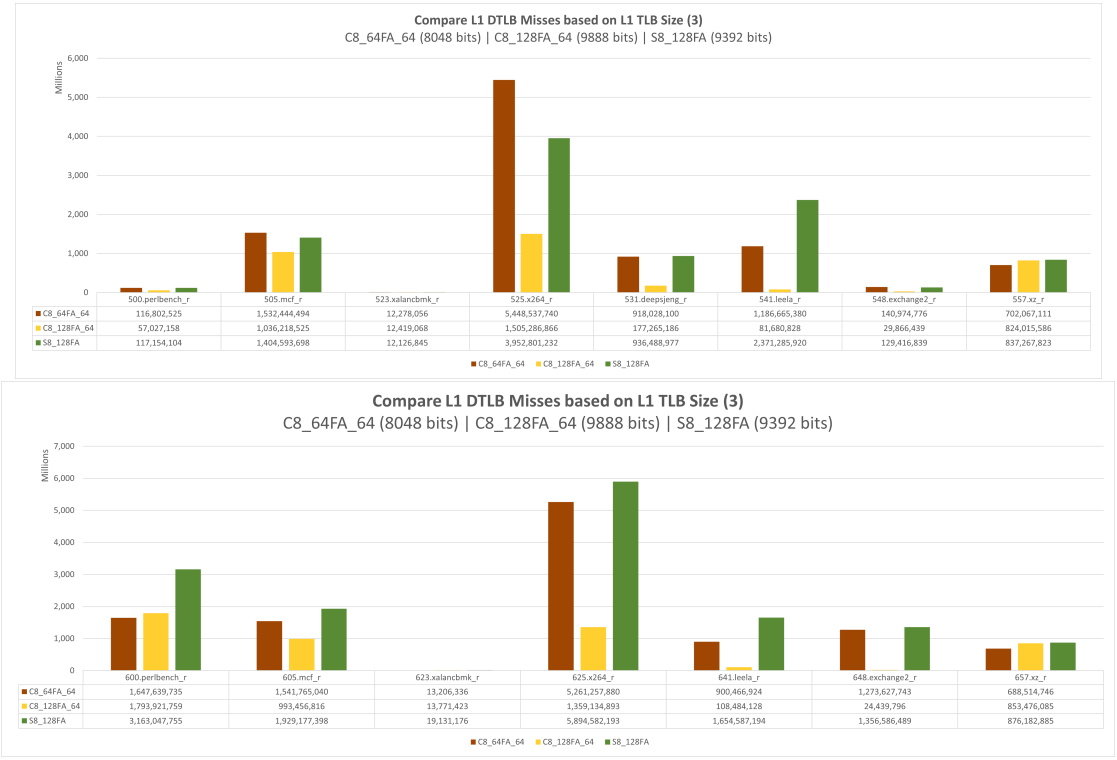
5.2.2 Σύγκριση Sectored και Multigranular TLB με βάση το TLB Reach

5.2.2.1 Σύγκριση των S8_32FA C8_16FA_16 C8_32FA_0

	S8_32FA	C8_16FA_16	C8_32FA_0
Size in bits	2348	-14.31%	-60.82%
CPU Cycles (B)	3,160	1.31%	0.61%
L1 dTLB Misses (M)	22,108	10.02%	9.74%
Average IPC	0.5026	-1.14%	-0.80%
Average MPKI	13.919	9.86%	9.95%

Πίνακας 5.5: Συγκεντρωτικός πίνακας συγκρίσεων S8_32FA C8_16FA_16 C8_32FA_0

Όπως φαίνεται από τον πίνακα 5.5, τα Multigranular TLBs μπορούν να επιτύχουν εξαιρετικά κοντινά αποτελέσματα με το Sectored εξοικονωμόντας ταυτόχρονα χώρο, δηλαδή χρησιμοποώ-



Σχήμα 5.6: Πλήθος αστοχιών L1 TLB - S8_128FA - C8_64FA_64 - C8_128FA_64

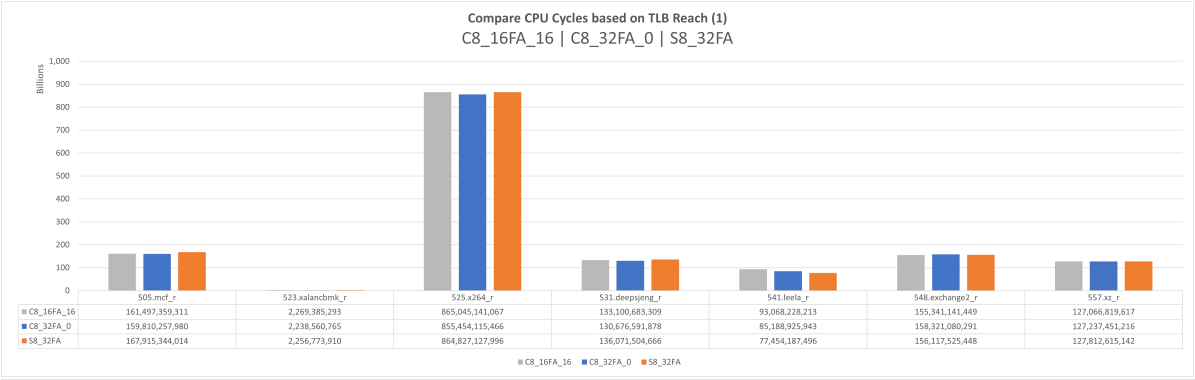
ντας μικρότερο πλήθος καταχωρητών και Flip-Flops. Συγκεκριμένα το C8_32FA_0 παρά το γεγονός ότι παρουσιάζει αύξηση στα L1 Data TLB Misses κατά 9,74%, τα benchmarks ολοκληρώνονται σε αντίστοιχους επεξεργαστικούς κύκλους. Αν συνυπολογίσουμε όμως την χρήση λιγότερων καταχωρητών κατά 60,82%, συμπεραίνουμε ότι το C8_32FA_0 αποτελεί μια πολύ καλύτερη εναλλακτική έναντι του S8_32FA. Σε ότι αφορά τα επιμέρους benchmarks, όπως φαίνεται στον πίνακα 5.6 οι επιδόσεις στα περισσότερα είναι αρκετά κοντινές με εξαίρεση τα 541.leela_r, 605.mcf_r και 641.leela_r.

5.2.2.2 Σύγκριση των S8_64FA C8_32FA_32 C8_64FA_0

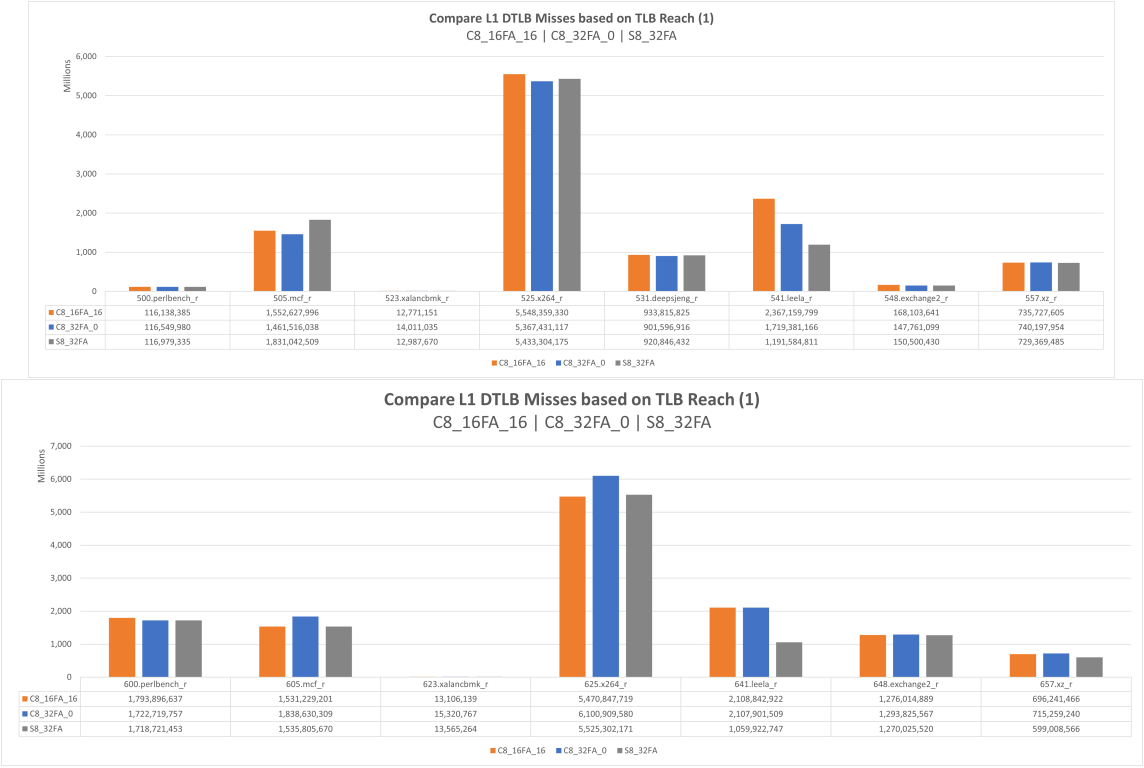
Όπως βλέπουμε στον πίνακα 5.7 μπορεί να επιτευχθεί τεράστια βελτίωση μέσω της αντικατάστασης του Sectored TLB είτε με ένα Clustered (C8_64FA_0) είτε με ένα Multigranular TLB (C8_32FA_32). Αφ' ενός το C8_64FA_0 πετυχαίνει εξαιρετικά κοντινά αποτελέσματα με το S8_64FA χρησιμοποιώντας λιγότερο από τον μισό χώρο (-60,82% στο μέγεθος σε bits). Αφ' ετέρου το C8_32FA_32, το οποίο όπως έχει αναφερθεί καταγράφει τις καλύτερες επιδόσεις σε σχέση με όσα configurations αναλύθηκαν, συνδυάζει την μείωση των απαιτούμενων επεξεργαστικών κύκλων, την μείωση των L1 Data TLB Misses, με την μείωση του απαιτούμενου χώρου σε bits κατά 14,31%.

Σε ότι αφορά τα επιμέρους benchmarks, το C8_32FA_32 υπερτερεί του S8_64FA σε όλα πλην των 557.xz_r (+11,47% στα L1 Data TLB Misses) και 657.xz_s (+24,27% στα L1 Data TLB Misses και +2% στους κύκλους).

Από την άλλη το C8_64FA_0 υπερτερεί σε όλα τα benchmarks της κατηγορίας intrate (-7,69% στα L1 Data TLB Misses και -0,62% στους κύκλους) εκτός από το 548.exchange2_r (+10,15% στα L1 Data TLB Misses και -0,1% στους κύκλους). Η εικόνα όμως στα benchmarks της κατηγορίας



Σχήμα 5.7: Πλήθος κύκλων επεξεργαστή για την περάτωση των benchmarks - S8_32FA C8_16FA_16 C8_32FA_0



Σχήμα 5.8: Πλήθος αστοχιών L1 TLB - S8_32FA C8_16FA_16 C8_32FA_0

intspeed είναι διαφορετική, καθώς παρουσιάζει +5,44% στα L1 Data TLB Misses και +1,46% στους

Benchmark	S8_32FA	C8_32FA_0
500.perlbench_r	6,065,065,281	1.24%
505.mcf_r	167,915,344,014	**-4.83%**
523.xalancbmk_r	2,256,773,910	**-0.81%**
525.x264_r	864,827,127,996	**-1.08%**
531.deepsjeng_r	136,071,504,666	**-3.97%**
541.leela_r	77,454,187,496	9.99%
548.exchange2_r	156,117,525,448	1.41%
557.xz_r	127,812,615,142	**-0.45%**
Intrate Sum	1,538,520,143,953	-0.87%
600.perlbench_r	270,743,305,668	**-0.39%**
605.mcf_r	163,624,185,125	14.77%
623.xalancbmk_r	2,363,076,595	3.93%
625.x264_r	862,263,240,526	0.28%
641.leela_r	71,404,084,770	19.62%
648.exchange2_r	131,136,650,168	**-5.57%**
657.xz_r	120,604,386,728	0.37%
Intspeed Sum	1,622,138,929,580	2.02%

Πίνακας 5.6: Σύγκριση S8_32FA και C8_32FA_0 στα επιμέρους benchmarks με βάση τους επεξεργαστικούς κύκλους

	S8_64FA	C8_32FA_32	C8_64FA_0
Size in bits	4696	-14.31%	-60.82%
CPU Cycles (B)	3,186	-7.62%	.44%
L1 dTLB Misses (M)	24,756	-63.60%	-0.83%
Average IPC	0.4987	7.97%	-0.46%
Average MPKI	15.578	-63.50%	-0.81%

Πίνακας 5.7: Συγκεντρωτικός πίνακας συγκρίσεων S8_64FA C8_32FA_32 C8_64FA_0

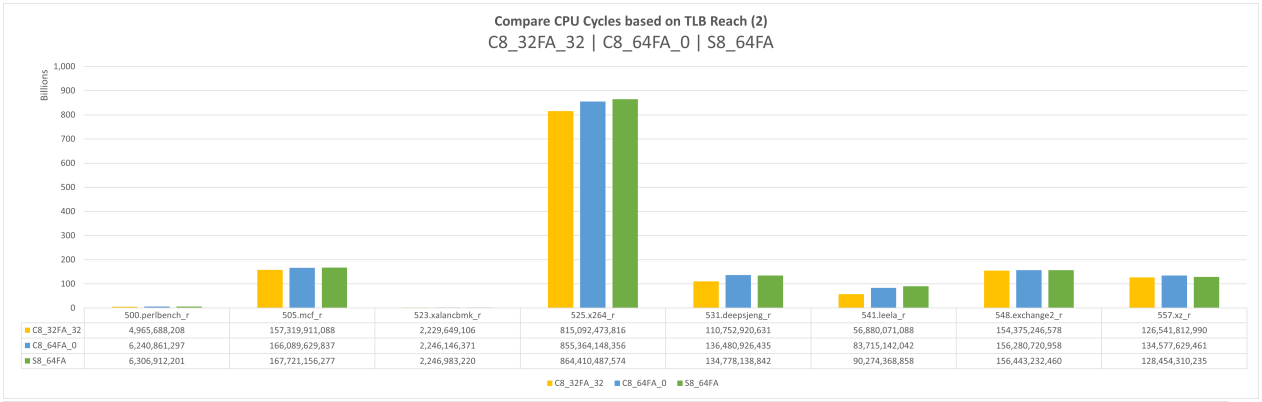
κύκλους με βασικότερες διαφορές στα 605.mcf_r (+28,6% στα L1 Data TLB Misses και +17,86% στους κύκλους) και 657.xz_r (+11,68% στα L1 Data TLB Misses και +0,76% στους κύκλους).

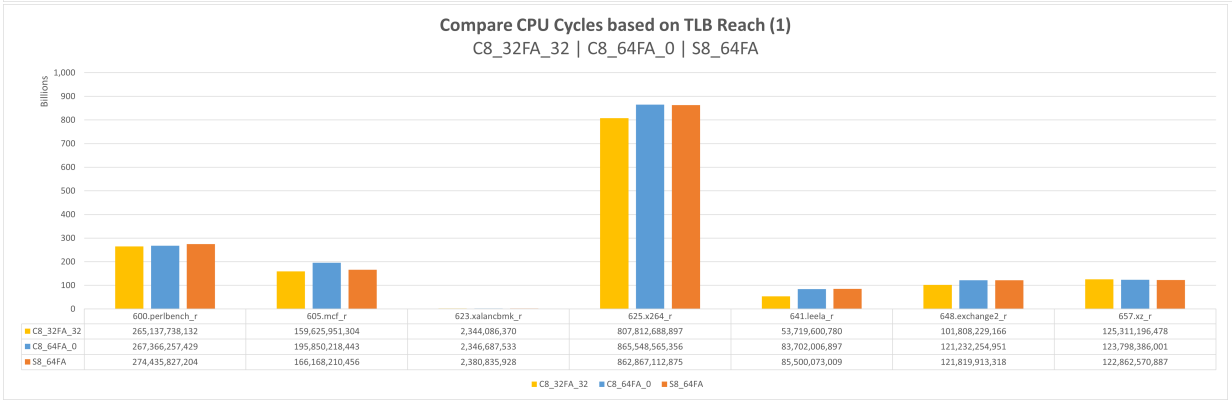
5.2.2.3 Σύγκριση των S8_128FA C8_64FA_64 C8_128FA_0

	S8_128FA	C8_64FA_64	C8_128FA_0
Size in bits	9392	-14.31%	-60.82%
CPU Cycles (B)	3,297	-1.14%	-3.44%
L1 dTLB Misses (M)	24,654	-19.95%	0.54%
Average IPC	0.4949	-1.65%	0.94%
Average MPKI	15.106	-17.67%	3.15%

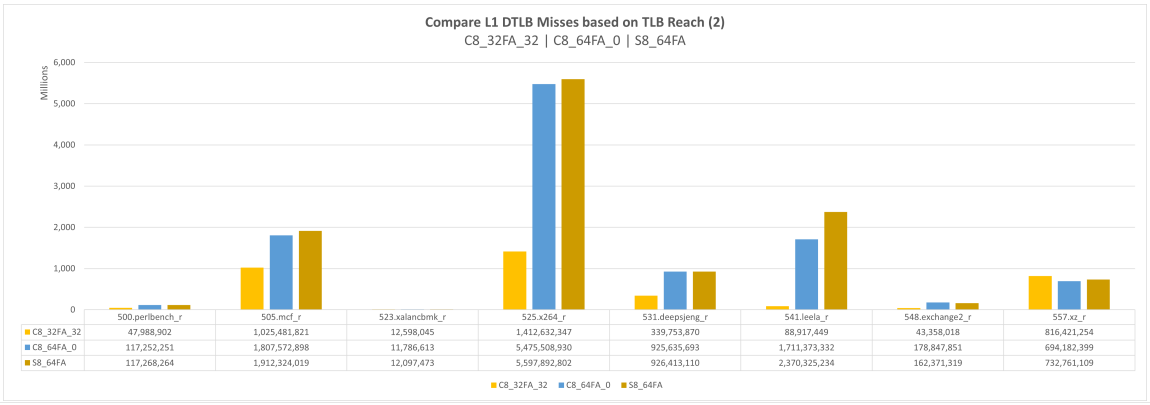
Πίνακας 5.8: Συγκεντρωτικός πίνακας συγκρίσεων S8_128FA C8_64FA_64 C8_128FA_0

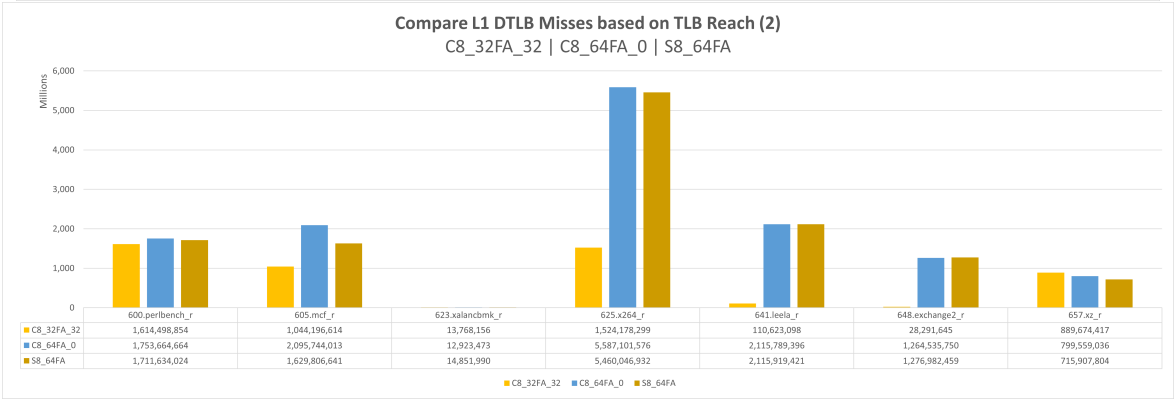
Αντίστοιχα με προηγουμένως, όπως φαίνεται στον πίνακα 5.8 μπορεί να επιτευχθεί σημαντική βελτίωση της επίδοσης του Sectored TLB. Αφ' ενός η αντικατάστασή του από το C8_128FA_0 πετυχαίνει εξαιρετικά μεγάλη εξοικονόμηση χώρου (-60,82%) μειώνοντας ταυτόχρονα τους συνολικούς κύκλους ολοκληρωσης που απαιτούνται κατά 3,44%. Ταυτόχρονα το C8_64FA_64 συνδυάζει την





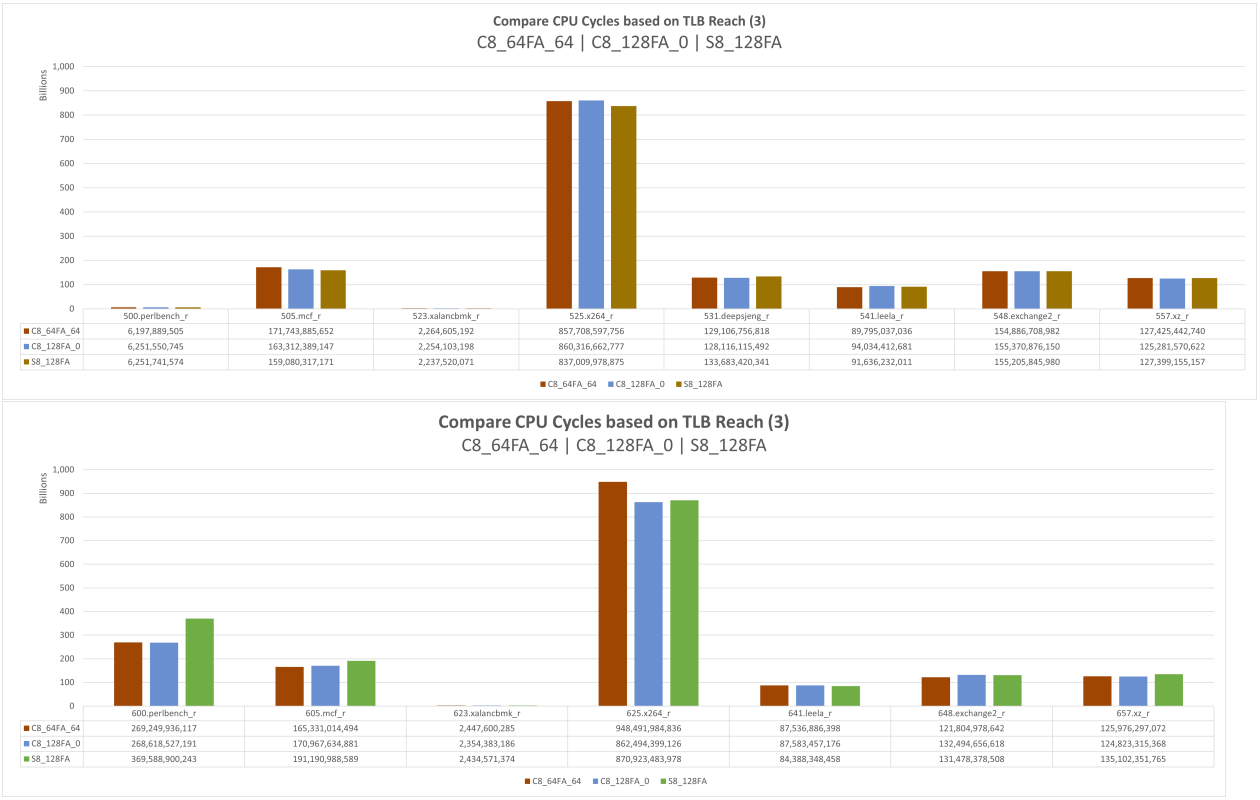
Σχήμα 5.9: Πλήθος κύκλων επεξεργαστή για την περάτωση των benchmarks - S8_64FA C8_32FA_32 C8_64FA_0





Σχήμα 5.10: Πλήθος αστοχιών L1 TLB - S8_64FA C8_32FA_32 C8_64FA_0

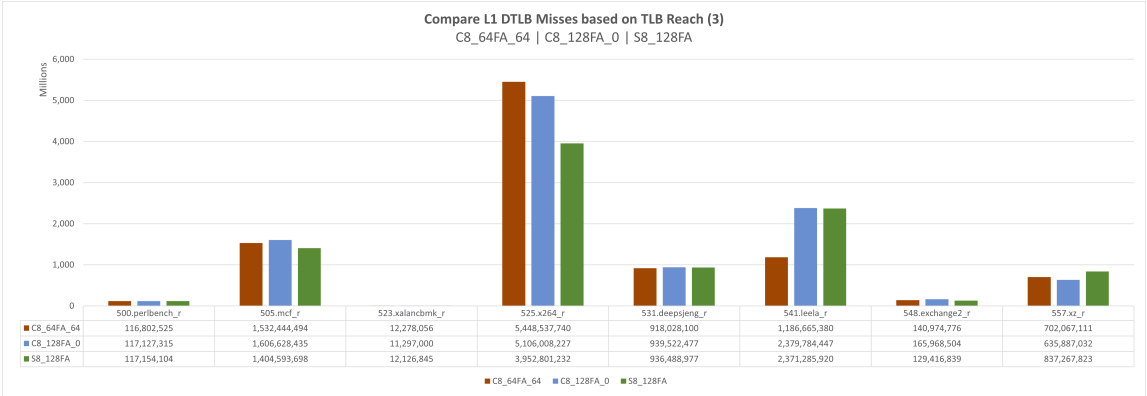
μείωση του μεγέθους κατά 14,31% με την μείωση των L1 DTLB Misses κατά 19,95% και των συνολικών κύκλων κατά 1,14%.

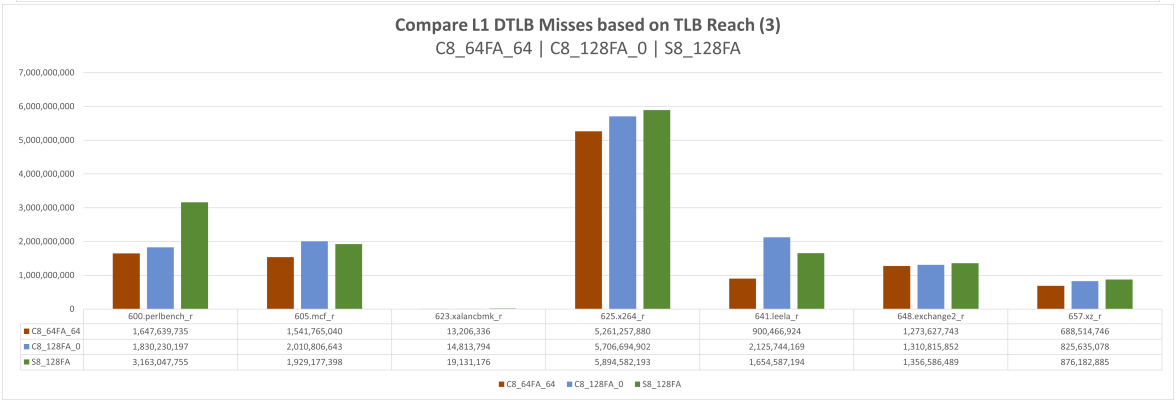


Σχήμα 5.11: Πλήθος κύκλων επεξεργαστή για την περάτωση των benchmarks - S8_128FA C8_64FA_64 C8_128FA_0

Σε ότι αφορά τα επιμέρους benchmarks, σε εκείνα της κατηγορίας intrate το C8_64FA_64 φαίνεται να αποδίδει καλύτερα καθώς παρουσιάζεται μείωση κατά 13,86% στα L1 DTLB Misses η οποία όμως συνοδεύεται με αύξηση στους κύκλους κατά 1,76%. Μια τέτοια συμπεριφορά δεν είναι φυσιολογική και πιθανότατα οφείλεται σε κάποιο ήδη υπάρχων bug στον Rocket, σε λάθη που μπορεί να έγιναν κατά την διάρκεια των μετρήσεων και ιδιαιτερότητες του περιβάλλοντος στο οποίο αυτές εκτελέστηκαν, οι οποίες ενδεχομένως επηρέασαν την λειτουργία του FPGA ή και σε όλα τα προηγούμενα. Οι μεγαλύτερες διαφορές εντοπίζονται στο 541.leela_r (-50% στα L1 Data TLB Misses και -2% στους κύκλους) και στο 525.x264_r (+37,84% στα L1 Data TLB Misses και -2,47% στους κύκλους). Στην ίδια κατηγορία το C8_128FA_0 αποδίδει χειρότερα σε σημαντικό βαθμό (+12,3% στα L1 Data TLB Misses και +1,48% στους κύκλους) σε σχέση με το S8_128FA, καταγράφοντας καλύτερα αποτελέσματα μονάχα στο 557.xz_r (-24% στα L1 Data TLB Misses και -1,66% στους κύκλους).

Από την άλλη, όμως, στην κατηγορία intspeed υπάρχει σημαντική βελτίωση των αποτελεσμάτων τόσο από το C8_64FA_64 με -24% στα L1 Data TLB Misses και -3,6% στους κύκλους, όσο και από το C8_128FA_0 με -7,18% στα L1 Data TLB Misses και -7,6% στους κύκλους. Το C8_64FA_64 αποδίδει καλύτερα σε όλα τα επιμέρους benchmarks δείχνοντας την μεγαλύτερη βελτίωση στα 600.perlbench_s (-47.91% στα L1 Data TLB Misses και -27.15% στους κύκλους), 605.mcf_s (-20% στα L1 Data TLB Misses και -13.53% στους κύκλους), 641.leela_s (-45,58% στα L1 Data TLB Misses και +3.731% στους κύκλους) και 657.xz_s (-21,42% στα L1 Data TLB Misses και -6,755% στους κύκλους).





Σχήμα 5.12: Πλήθος αστοχιών L1 TLB - S8_128FA C8_64FA_64 C8_128FA_0

5.2.3 Επίδραση της προσθήκης του απλού TLB

Ταυτόχρονα, με βάση τα αποτελέσματα φαίνεται η **σημασία της προσθήκης του απλού TLB στην βελτίωση των αποτελεσμάτων**. Συγκεκριμένα, όπως φαίνεται στους πίνακες 5.9, 5.10, 5.11, η προσθήκη του δεύτερου TLB μειώνει τα L1 Data TLB Misses εώς και 64,21% και τους επεξεργαστικούς κύκλους εώς και 7,45% στο C8_128FA_64. Βέβαια, το αν θα υπάρχει βελτίωση ή όχι και σε τί βαθμό, είναι άμεσα εξαρτημένο από το μέγεθος του απλού TLB, τόσο σε ότι αφορά τον αριθμό των μεταφράσεων, όσο και σε σχέση με το Clustered TLB. Ένα απλό TLB 16 θέσεων φαίνεται πως δεν είναι σε θέση να βελτιώσει τις επιδόσεις, ενώ παράλληλα αυξάνει το μέγεθος της σχεδίασης σε σημαντικό βαθμό. Αυτό συμβαίνει εξαιτίας του μικρού του μεγέθους, το οποίο συνεπάγεται ότι οι μετάφρασεις που αποθηκεύονται αντικαθιστώνται από καινούριες πρωτού προλάβουν να χρησιμοποιηθούν ξανά. Έτσι, προκειμένου να υπάρχει όντως βελτίωση στις επιδόσεις, **το ελάχιστο μέγεθος του απλού TLB πρέπει να είναι οι 32 μεταφράσεις**.

	C8_32FA_0	C8_32FA_16	C8_32FA_32
Size in bits	920	168.70%	337.39%
CPU Cycles (B)	3,179	5.43%	-7.42%
L1 dTLB Misses (M)	24,263	2.79%	-62.86%
Average IPC	0.499	-1.08%	8.00%
Average MPKI	15.304	-1.44%	-62.85%

Πίνακας 5.9: Επίδραση του απλού TLB στην επίδοση του Multigranular TLB για 32 Clustered entries

	C8_64FA_0	C8_64FA_32	C8_64FA_64
Size in bits	1840	168.70%	337.39%
CPU Cycles (B)	3,200	-6.19%	1.85%
L1 dTLB Misses (M)	24,551	-62.62%	-19.62%
Average IPC	0.496	7.73%	-1.94%
Average MPKI	15.452	-63.01%	-19.51%

Πίνακας 5.10: Επίδραση του απλού TLB στην επίδοση του Multigranular TLB για 64 Clustered entries

	C8_128FA_0	C8_128FA_16	C8_128FA_64
Size in bits	3680	42.17%	168.70%
CPU Cycles (B)	3,184	0.28%	-7.45%
L1 dTLB Misses (M)	24,786	-1.94%	-64.21%
Average IPC	0.500	6.62%	8.17%
Average MPKI	15.582	-8.29%	-64.25%

Πίνακας 5.11: Επίδραση του απλού TLB στην επίδοση του Multigranular TLB για 128 Clustered entries

5.3 Συμπεράσματα

Αρχικά συμπεραίνουμε ότι **η αντικατάσταση του Sectored TLB ακόμα και από ένα Clustered με ίδιο αριθμό μεταφράσεων**, δίχως δηλαδή το απλό που θα το έκανε Multigranular, **προσφέρει σημαντικά πλεονεκτήματα**. Το σημαντικότερο από αυτά είναι η **μείωση του χρησιμοποιούμενου χώρου κατά 60,82%**, η οποία μεταφράζεται σε χρήση λιγότερων Flip-Flops (με διαφορετική ποσοστιαία μείωση αν συγκρίνουμε το σύνολο των απαιτούμενων FFs) και άρα **μειωμένη κατανάλωση ενέργειας του SoC**. Ακόμη, αυτή η αλλαγή συνεπάγεται **μείωση του μεγέθους του ολοκληρωμένου κυκλώματος (die size)**. Τέλος, **οι πόροι που απελευθερώνονται μπορούν να αξιοποιηθούν για την βελτιστοποίηση ή την δημιουργία άλλων κομματιών του επεξεργαστικού πυρήνα**.

Τα περιθώρια βελτίωσης των αποτελεσμάτων διαφέρουν ανάλογα με το TLB Reach και συνεπώς το μέγεθος του TLB σε bits. Σε TLB 32 μεταφράσεων, αν λάβουμε υπ' όψιν μας μονάχα τα L1 Data TLB Misses και τον αριθμό των επεξεργαστικών κύκλων, το Sectored έχει ένα ελαφρύ προβάδισμα σε σχέση τόσο με το Clustered όσο και με το Multigranular TLB. Αυτό οφείλεται στο γεγονός ότι το Sectored TLB αποθηκεύει ολόκληρα τα PPN χωρίς την απαίτηση αυτά να ανήκουν στην ίδια περιοχή στην φυσική μνήμη, εν αντιθέσει με το Multigranular TLB. Σε αυτές τις περιπτώσεις θα μπορούσε να εξεταστεί η μείωση του cluster factor από 8 σε 4, προκειμένου να μπορούν να καλυφθούν περισσότερες περιοχές όπου η εικονική μνήμη χαρτογραφείται στην φυσική.

Όσο όμως το πλήθος των μεταφράσεων μεγαλώνει τα πλεονεκτήματα του Multigranular TLB φαίνονται ξεκάθαρα. Χαρακτηριστικά όταν έχουμε 64 μεταφράσεις, επιτυγχάνεται από το C8_32FA_32 σε σύγκριση με το S8_32FA **μείωση των συνολικών κύκλων κατα 7,62% και των L1 Data TLB Misses κατά 63,6%, παράλληλα με την μείωση του απαιτούμενου χώρου κατά 14,31%**. Η συγκεκριμένη οργάνωση αποτελεί εναλλακτική αντικατάστασης ακόμα κα ενός Sectored TLB 128 μεταφράσεων.

Στις περισσότερες περιπτώσεις, ένα Sectored TLB είναι δυνατόν να αντικατασταθεί από ένα Multigranular TLB προσφέροντας σημαντικά πλεονεκτήματα στα περισσότερα πεδία που αναλύσαμε. Συγκεκριμένα ένα S8_32FA μπορεί να αντικατασταθεί από ένα C8_32FA_0 προσφέροντας μείωση κατά 60,82% στο μέγεθος του L1 TLB, αλλά και μία μικρή επιβάρυνση της τάξεως του 0,61% στους επεξεργαστικούς κύκλους που προκύπτει από τα επιπλέον 9,74% L1 Data TLB Misses. Επίσης, ένα S8_64FA μπορεί να αντικατασταθεί από ένα C8_32FA_32 προσφέροντας μείωση κατά 14,16% στο μέγεθος του L1 TLB, μειώνοντας παράλληλα κατά 7,62% τους απαιτούμενους επεξεργαστικούς κύκλους και κατά 63,6% τα L1 Data TLB Misses. Τέλος, ένα S8_128FA μπορεί να αντικατασταθεί πάλι από ένα C8_32FA_32 προσφέροντας μείωση κατά 57,16% στο μέγεθος του L1 TLB, μειώνοντας παράλληλα κατά 10,63% τους απαιτούμενους επεξεργαστικούς κύκλους και κατά 63,45% τα L1 Data TLB Misses.

Σε σχέση με τα αποτελέσματα, πρέπει να επισημάνουμε δύο πράγματα. Αφ' ενός **μελετήσαμε την ποσοστιαία μεταβολή στο μέγεθος του L1 TLB και όχι στο σύνολο του ολοκληρωμένου, η οποία θα είναι σίγουρα μικρότερη**. Αφ' ετέρου τα FPGA τρέχουν σε διαφορετική συχνότητα ρολογιού (περίπου στα 100 MHz στο Alveo U250) σε σχέση με έναν πραγματικό επεξεργαστή (2-4 GHz), δηλαδή 20-40 φορές μικρότερη, ενώ η συχνότητα λειτουργίας της RAM παραμένει ίδια. Αυτό σημαίνει ότι αν ένα Page Table Walk διαρκεί 40 με 100 κύκλους ρολογιού ενός πραγματικού επεξεργαστή, αυτό μεταφράζεται σε 1 με 5 κύκλους ρολογιού στο FPGA. Πρακτικά αυτό σημαίνει πως **σε απόλυτο αριθμό οι διαφορές των επεξεργαστικών κύκλων μεταξύ των configurations που τρέξαμε είναι ακόμα μεγαλύτερη, κάτι το οποίο όμως δεν επηρεάζει την ποσοστιαία μεταβολή τους**.

Σημαντικό ρόλο στα αποτελέσματα κατέχει η **επιλογή του κατάλληλου cluster factor, του κατάλληλου μεγέθους του απλού TLB, καθώς και του cluster threshold**. Για αρχή, με βάση και τις ήδη υπάρχουσες μελέτες, η **ιδανική τιμή για το cluster factor είναι το 8**, αλλά για πολύ μικρά TLB 16 θέσεων θα μπορούσε να εξεταστεί η μείωσή του στο 4 προκειμένου να καλύπτονται περισσότερες περιοχές εικονικής/φυσικής μνήμης. Ταυτόχρονα **το μέγεθος του απλού TLB δεν θα πρέπει να είναι μικρότερο από 32 θέσεις,** καθώς σε τέτοιες περιπτώσεις δεν επαρκεί για την χαρτογράφηση σελίδων που δεν μπορούν να συγχωνευθούν με άλλες και αναγκαστικά αποθηκεύονται στο απλό TLB. Τέτοιου είδους καταστάσεις εμφανίζονται συχνότερα σε περιπτώσεις όπου ο κατακερματισμός της φυσικής μνήμης είναι μεγάλος. Από την άλλη δεν θα πρέπει να έχει πολύ μεγάλο μέγεθος καθώς κάτι τέτοιο θα επηρέαζε αρνητικά το critical path. Τέλος, **επιλέγουμε την τιμή 2 για το cluster threshold,** που υποδεικνύει τον ελάχιστο αριθμό έγκυρων μεταφράσεων που πρέπει να περιέχει ένα clustered entry, προκειμένου μια νέα μετάφραση με την οποία μοιράζονται το ίδιο VPN Mask αλλά διαφορετικό PPN Mask, να αποθηκευτεί στο απλό TLB και να μην την αντικαταστήσει. Πρακτικά, η τιμή που επιλέξαμε σημαίνει ότι ένα ήδη υπάρχων clustered entry θα αντικατασταθεί από ένα άλλο μονάχα αν περιέχει μόνο μια έγκυρη μετάφραση ή αν το υποδεικνύει η πολιτική αντικατάστασης LRU.

Κεφάλαιο 6

Μελλοντικές Επεκτάσεις

Στο κεφάλαιο αυτό παρουσιάζουμε πιθανές επεκτάσεις και βελτιστοποιήσεις της παρούσας διπλωματικής εργασίας καθώς και ευρύτερα του Rocket Chip Generator.

6.1 Παραμετροποίηση του clustering logic - αναδιοργάνωση του L2 TLB και της PTW Cache

Στην παρούσα διπλωματική εργασία μελετήσαμε αποκλειστικά την επίδραση των αλλαγών μας στο L1 TLB. Παρ' όλα αυτά και το L2 TLB κατέχει εξαιρετικά σημαντικό ρόλο στην επιτάχυνση της διαδικασίας μετάφρασης των διευθύνσεων, ενώ ταυτόχρονα η τωρινή του οργάνωση στον Rocket (απλό παραμετροποιήσιμο set-associative TLB) επιδέχεται μεγάλα περιθώρια βελτίωσης. Συνεπώς, θα μπορούσε να εξεταστεί η υλοποίηση αντίστοιχου μηχανισμού clustering και για το L2 TLB. Επίσης μια σειρά μελετών [3] έχουν επισημάνει την σημασία της Page Table Walk Cache η οποία θα μπορούσε να αναδιοργανωθεί στον Rocket. Τέλος, ο μηχανισμός clustering στο Multigranular TLB θα μπορούσε να ελέγχει την δυνατότητα συγχώνευσης και των μεταφράσεων που είναι αποθηκευμένες στο απλό TLB. Σε περίπτωση που κάτι τέτοιο είναι εφικτό, οι μεταφράσεις αυτές θα μεταφέρονται στο Clustered TLB.

6.2 Υλοποίηση δυναμικού cluster threshold

Η επιλογή του κατάλληλου cluster threshold παίζει σημαντικό ρόλο στην σχεδίαση, καθώς καθορίζει το όριο του πλήθους των έγκυρων μεταφράσεων μιας καταχώρησης του Clustered TLB, πάνω από το οποίο μια νέα μετάφραση -με την οποία υπάρχει μόνο VPN Mask Match (χωρίς PPN Mask Match)- θα την αντικαταστήσει. Έτσι, όταν το όριο αυτό είναι υψηλό, αποθηκεύονται περισσότερες νέες μεταφράσεις στο απλό TLB, ενώ όταν είναι χαμηλό αποθηκεύονται περισσότερες στο Clustered TLB, πάντα με το κριτήριο του να υπάρχει VPN Mask Match χωρίς PPN Mask Match. Αυτή η διαδικασία μπορεί να περιγραφεί και ως εξής: όταν υπάρχει υψηλό memory contiguity, επομένως αξιοποιείται περισσότερο το Clustered TLB στο οποίο θα συμβαίνουν περισσότερα TLB Hit, το cluster threshold θα πρέπει να είναι υψηλό προκειμένου να είναι πιο σπάνιες οι αντικαταστάσεις καταχωρήσεων στο Clustered TLB. Αντίστοιχο σκεπτικό μπορεί να αναπτυχθεί και για το απλό TLB.

Στην παρούσα υλοποίηση, η τιμή του cluster threshold ορίζεται στατικά και προτείνεται η τιμή 2. Ένας μηχανισμός δυναμικής εκχώρησης της τιμής του cluster threshold, θα λαμβάνει υπ' όψιν το πόσα TLB Hits συμβαίνουν σε κάθε TLB και ανάλογα θα αυξομειώνει την τιμή του cluster threshold.

6.3 Προανάκληση διευθύνσεων (Address Prefetching)

Ένα Page Table Walk στον Rocket επιστρέφει στο TLB μονάχα την μετάφραση για την εικονική διεύθυνση που έχει ζητηθεί, σε αντίθεση με άλλες αρχιτεκτονικές (π.χ. x86-64) που ταυτόχρονα επιστρέφουν και γειτονικές μεταφράσεις ή μεταφράσεις που προκύπτουν με βάση μοτίβα πρόσβασης στην μνήμη. Με αυτό τον τρόπο δίνεται η δυνατότητα αποθήκευσης μιας μετάφρασης πρωτού αυτή χρειαστεί, γλιτώνοντας ένα τεράστιο κόστος σε κύκλους ρολογιού που προκύπτει από την διάσχιση του Πίνακα Σελίδων και την πρόσβαση στην φυσική μνήμη. Τεχνικές που έχουν προταθεί είναι το Sequential Prefetching, το Arbitrary Stride Prefetching, το Markov Prefetching, το Recency Based Prefetching και το Distance Prefetching. Ταυτόχρονα κάθε οργάνωση του TLB επωφελείται σε διαφορετικό βαθμό από τα διαφορετικά σχήματα προανάκλησης διευθύνσεων, επομένως αποτελεί ερώτημα ποιος είναι ο βέλτιστος συνδυασμός οργάνωσης του TLB και μηχανισμού prefetching για κάθε workload.

6.4 Μελέτη προηγμένων πολιτικών αντικατάστασης

Η παρούσα πολιτική αντικατάστασης (LRU) λαμβάνει υπ' όψιν της μονάχα το πόσο πρόσφατα έχει προσπελαστεί μια καταχώρηση και όταν χρειαστεί διώχνει εκείνη που έχει προσπελαστεί νωρίτερα από όλες τις άλλες. Πιο προηγμένες τεχνικές μπορούν να συνυπολογίζουν το πλήθος των μεταφράσεων που είναι αποθηκευμένες σε μία καταχώρηση με βάση τα valid bit ή/και να αξιοποιήσουν μηχανισμούς πρόβλεψης αντίστοιχους με εκείνους που χρησιμοποιούνται για το address prefetching. Ένας τέτοιος μηχανισμός πρέπει να δίνει διαφορετικό βάρος σε κάθε παράγοντα που προαναφέρθηκε. Σκοπός μιας τέτοιας αλλαγής είναι το να υπάρχουν ανά πάσα στιγμή όσο το δυνατόν περισσότερες χρήσιμες μεταφράσεις λαμβάνοντας υπ΄ όψιν περισσότερους από έναν παράγοντες, προκειμένου να περιοριστούν τα TLB Misses.

Βιβλιογραφία

[1] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. The Rocket Chip Generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[2] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: Constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1216–1225, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1199-1. doi: 10.1145/2228360.2228584. URL http://doi.acm.org/10.1145/2228360.2228584.

[3] Thomas Barr, Alan Cox, and Scott Rixner. Translation caching: Skip, don't walk (the page table). volume 38, pages 48–59, 06 2010. doi: 10.1145/1815961.1815970.

[4] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, page 41–42, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356299. doi: 10.1145/3185768.3185771. URL https://doi.org/10.1145/3185768.3185771.

[5] Christopher Celio, David A. Patterson, and Krste Asanović. The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor. Technical Report UCB/EECS-2015-167, EECS Department, University of California, Berkeley, Jun 2015. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html.

[6] Andrew Waterman and Krste Asanovic. The RISC-V Instruction Set Manual, Volume 1: User-Level ISA, Document Version 2.2. RISC-V Foundation, May 2017.

[7] Andrew Waterman and Krste Asanovic. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. RISC-V Foundation, May 2017.

[8] Neel Gala, Arjun Menon, Rahul Bodduna, G. S. Madhusudan, and V. Kamakoti. Shakti processors: An open-source hardware initiative. In *Proceedings of the 2016 29th International*

Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), VLSID '16, pages 7–8, Washington, DC, USA, 2016. IEEE Computer Society. ISBN 978-1-4673-8700-2. doi: 10.1109/VLSID.2016.130. URL http://dx.doi.org/10.1109/VLSID.2016.130.

[9] Nikolaos Charalampos Papadopoulos, Vasileios Karakostas, Konstantinos Nikas, Nectarios Koziris, and Dionisios N. Pnevmatikatos. A configurable tlb hierarchy for the risc-v architecture. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 85–90, 2020. doi: 10.1109/FPL50879.2020.00024.

[10] Binh Pham, Viswanathan Vaidyanathan, Aamer Jaleel, and Abhishek Bhattacharjee. Colt: Coalesced large-reach tlbs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 258–269, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4924-8. doi: 10.1109/MICRO.2012.32. URL https://doi.org/10.1109/MICRO.2012.32.

[11] Binh Pham, Abhishek Bhattacharjee, Yasuko Eckert, and Gabriel H. Loh. Increasing tlb reach by exploiting clustering in page translations. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 558–567, 2014. doi: 10.1109/HPCA.2014.6835964.

[12] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini. Pulp: A parallel ultra low power platform for next generation iot applications. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–39, Aug 2015. doi: 10.1109/HOTCHIPS.2015.7477325.

[13] Sifive. RISC-V Core IP Products, 2017. URL https://cdn2.hubspot.net/hubfs/3020607/SiFive-RISCVCoreIP.pdf.

[14] Madhusudhan Talluri and Mark D. Hill. Surpassing the tlb performance of superpages with less operating system support. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VI, page 171–182, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916603. doi: 10.1145/195473.195531. URL https://doi.org/10.1145/195473.195531.

[15] Wikipedia contributors. Instruction set architecture — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Instruction_set_architecture&oldid=919994629.

[16] Wikipedia contributors. Verilator — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Verilator.