Προηγμένα Θέματα Αρχιτεκτονικής Υπολογιστών

2η Σειρά Ασκήσεων
Μάνος Αραπίδης
ΑΜ : 03116071

- Μελέτη εντολών άλματος

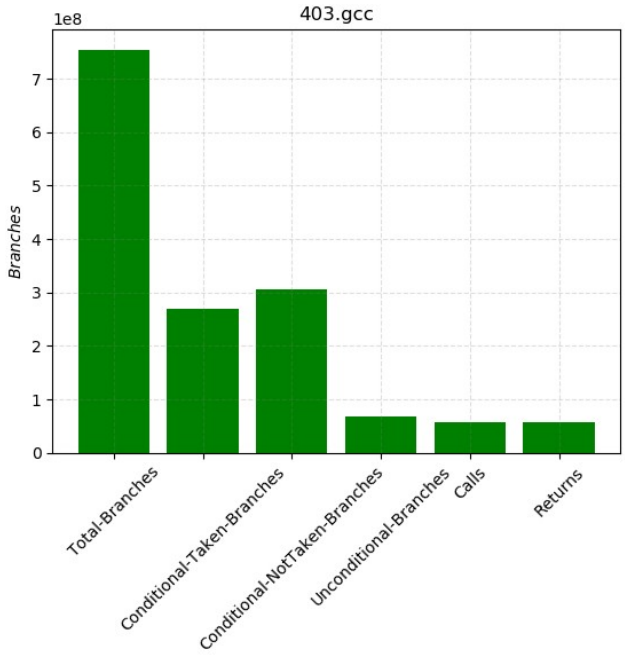
Σκοπός αυτού του ερωτήματος είναι συλλογή στατιστικών για τις εντολές άλματος . Συγκεκριμένα θα παρατηρήσουμε για κάθε benchmark το πλήθος branches εντολών που εκτελέστηκαν, καθώς και το ποσοστό αυτών που ανήκουν σε κάθε κατηγορία. Οι κατηγορίες που κατατάσσουμε τις εντολές άλματος είναι οι εξής :

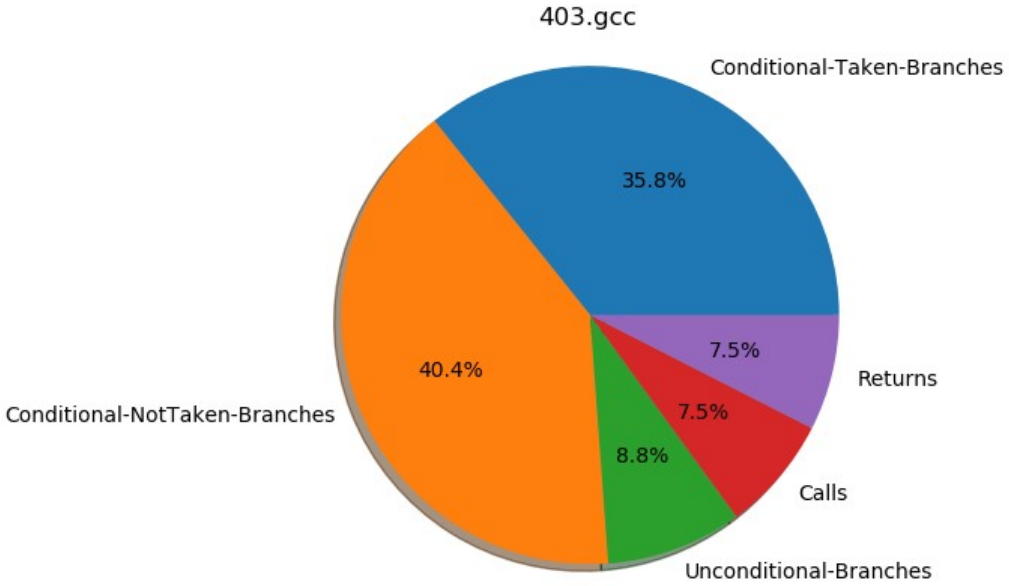
- Conditional Taken branches
- Conditional Not Taken branches
- UnConditional branches
- Calls
- Returns

Στην κατηγορία Unconditional αντιστοιχούμε τις εντολές direct jump. Ωστόσο οι εντολές Call και Return θεωρούνται και αυτές Unconditional , καθώς εκτελούνται χωρίς κάποια συνθήκη . Όποτε, στην ανάλυση μας (εκτός των διαγραμμάτων) όταν αναφερόμαστε στα Unconditional branches, θα συμπεριλαμβάνουμε τις περιπτώσεις direct jump , call και return .

Ακολουθούν διαγράμματα για κάθε benchmark

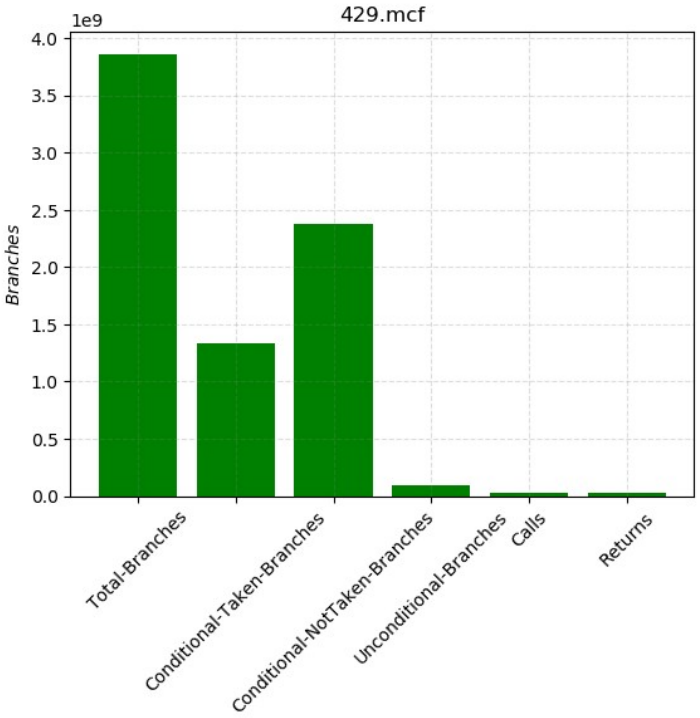
403.gcc

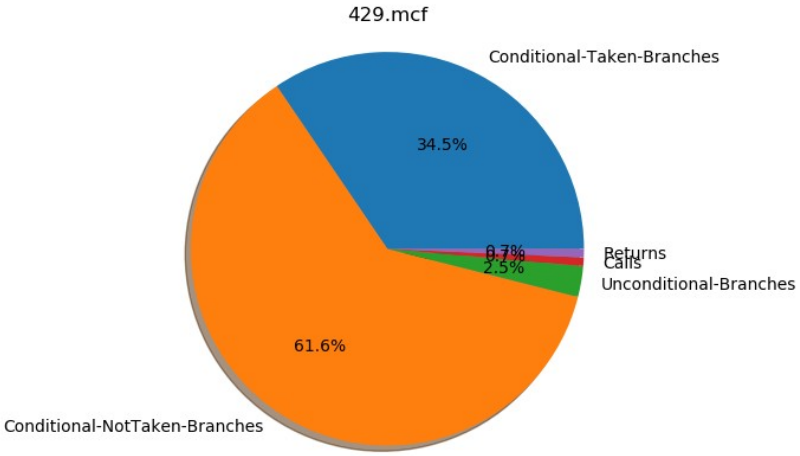




Παρατηρούμε ότι έχουμε σχεδόν ίσο πλήθος Not-Taken και Taken branches, με τα πρώτα να είναι ελαφρώς περισσότερα. Ακόμα, το μεγαλύτερο μέρος των branches είναι conditional .

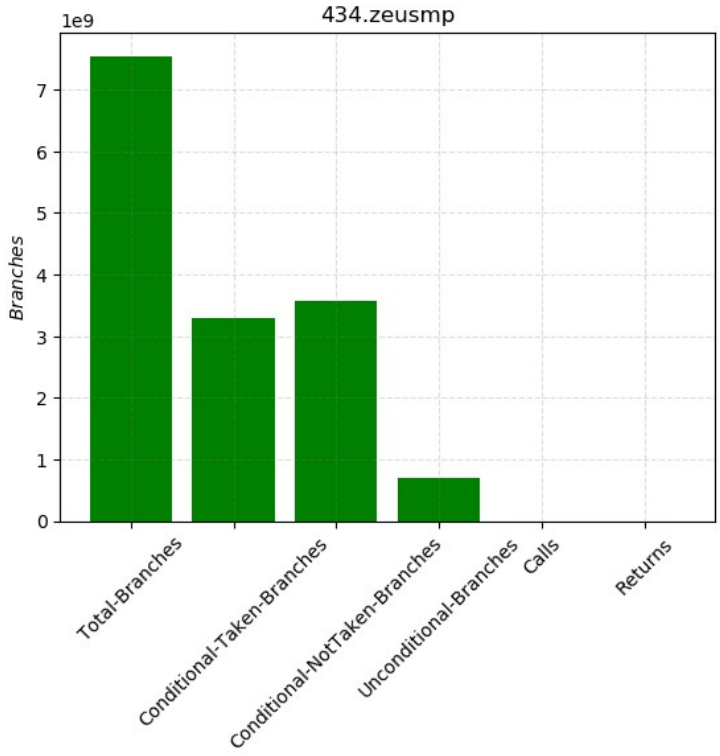
429.mcf

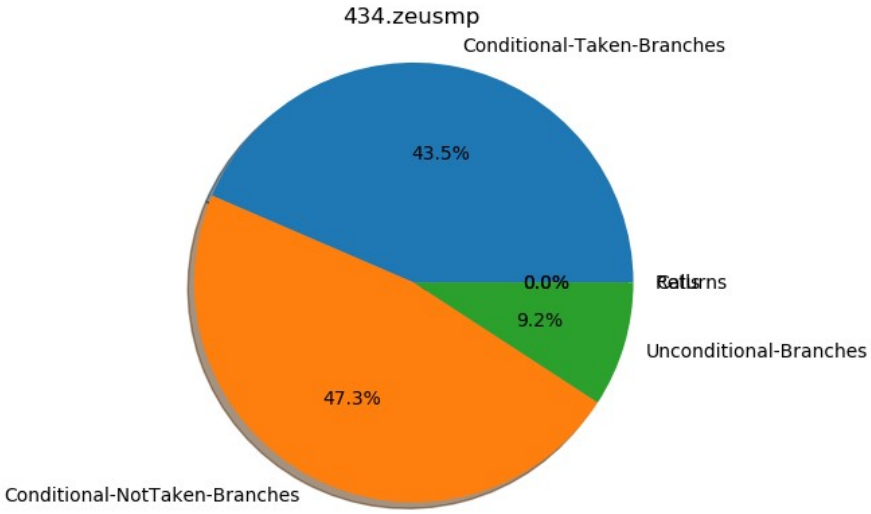




Παρατηρούμε ότι το πλήθος των Not-Taken branches είναι σχεδόν διπλάσιο από των Taken. Τα branches είναι κατά κύριο ρόλο conditional .

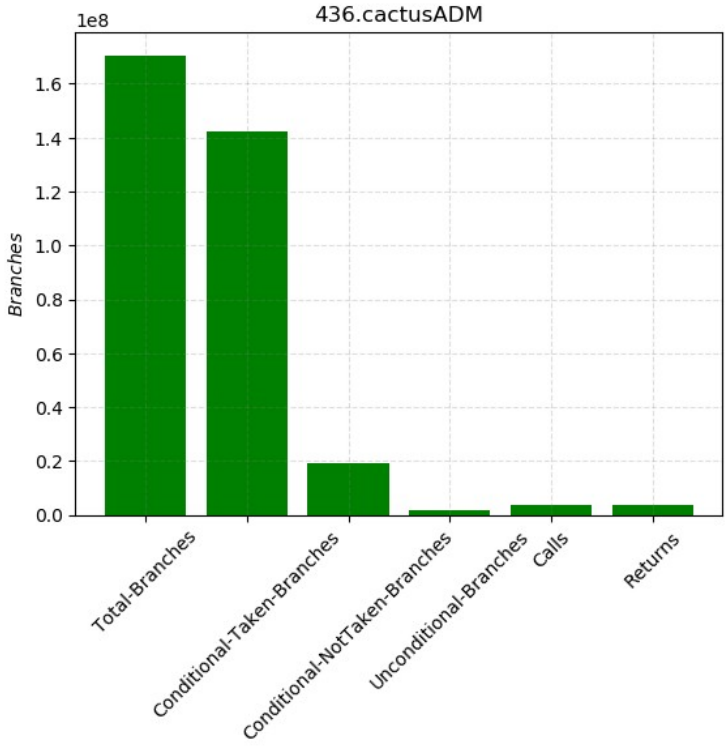
434.zeusmp

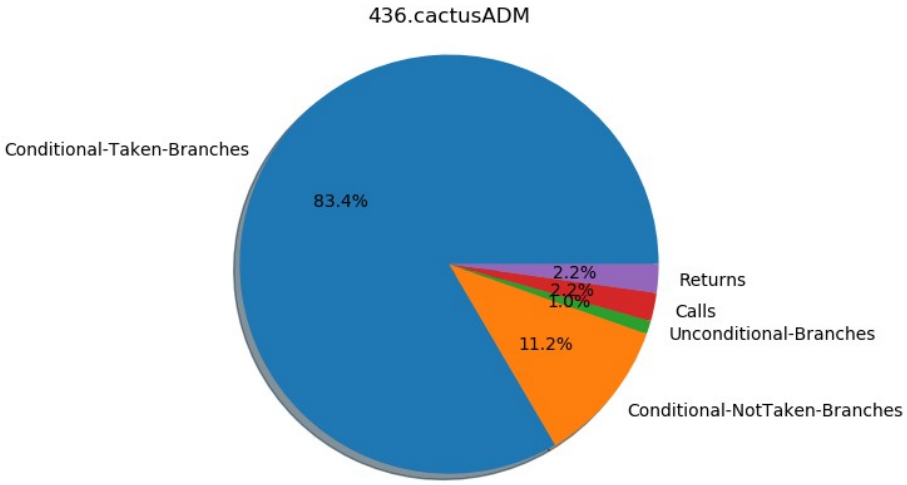




Παρατηρούμε ότι έχουμε σχεδόν ίσο αριθμό Not-Taken και Taken branches, με τα πρώτα να είναι ελαφρώς περισσότερα , ενώ φαίνεται να μην έχουμε Calls και αντίστοιχα Returns. Πάλι το μεγαλύτερο μέρος είναι conditional .

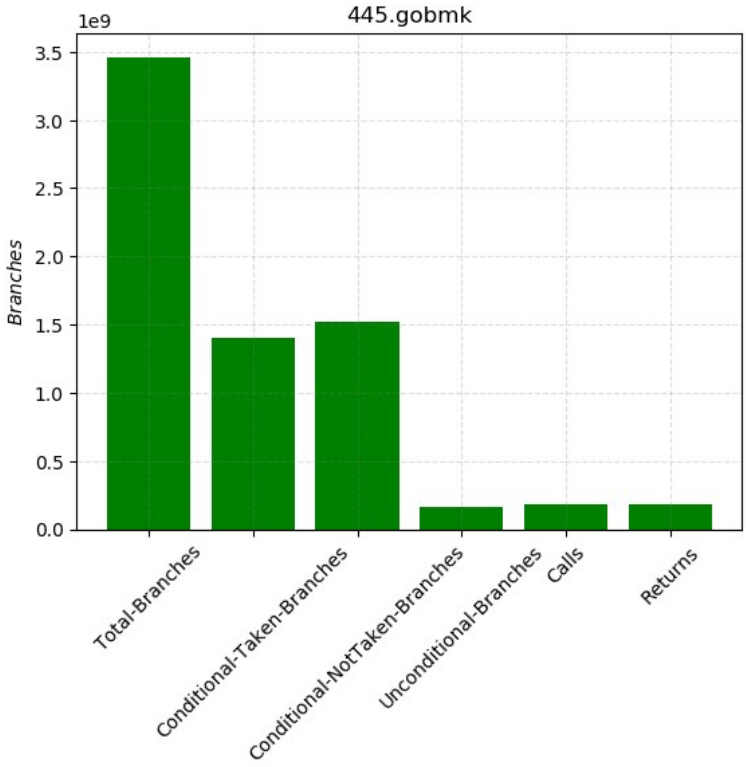
434.cactusAMD

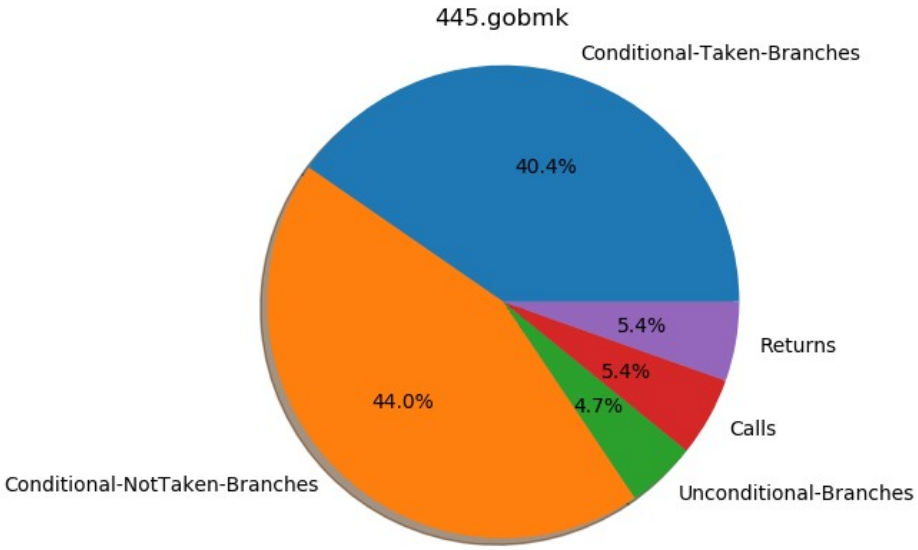




Παρατηρούμε ότι ο αριθμός των Taken branches είναι πολύ μεγαλύτερος των Not-Taken, πιθανώς λόγω πολλών loops στον κώδικα . Σχεδόν όλα είναι conditional branches .

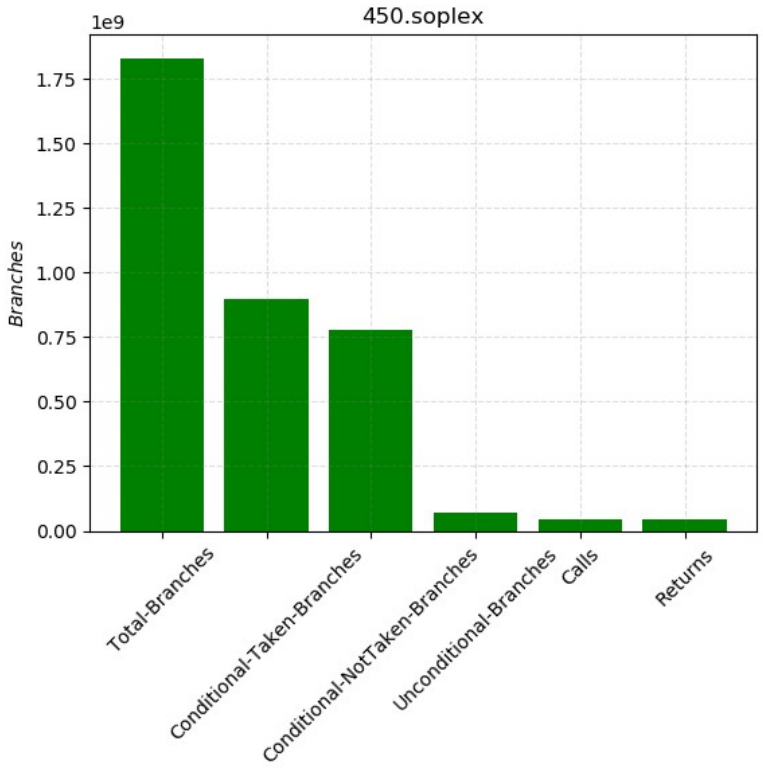
445.gobmk

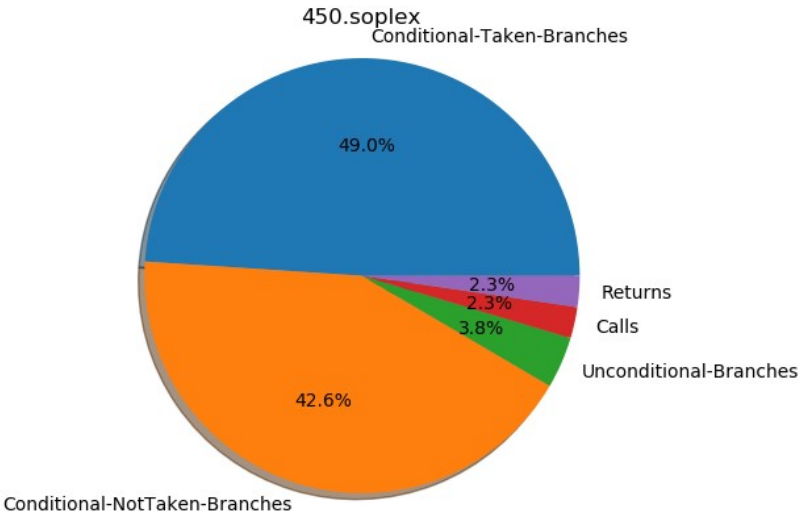




Παρατηρούμε ότι έχουμε σχεδόν ίσο αριθμό Not-Taken και Taken branches, με τα πρώτα να είναι ελαφρώς περισσότερα . Πάλι υπερτερούν σε πλήθος τα conditonal .

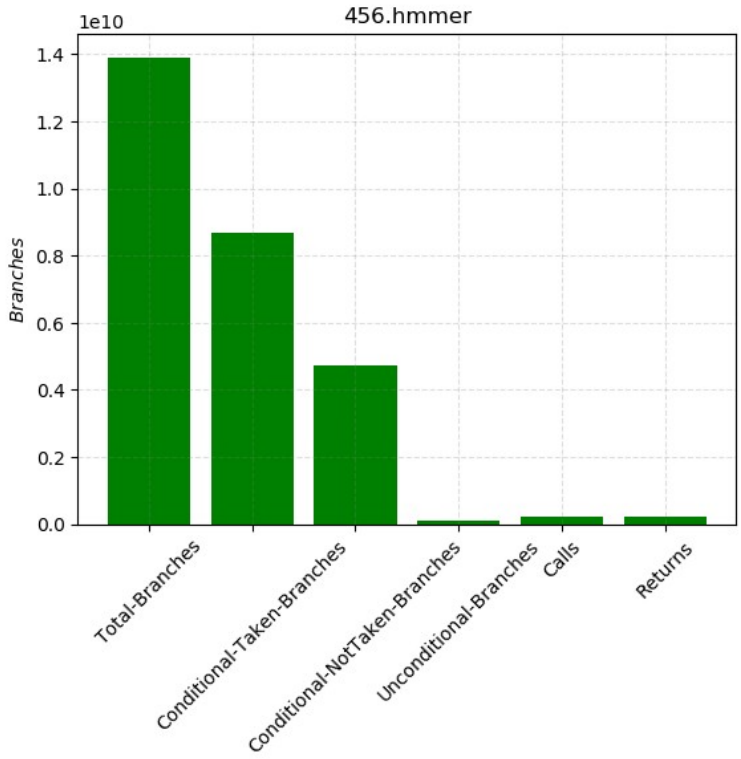
450.soplex

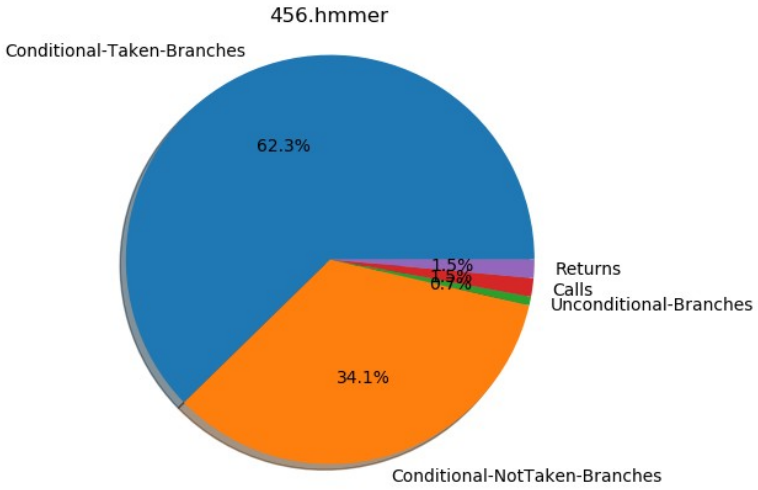




Παρατηρούμε ότι έχουμε σχεδόν ίσο αριθμό Taken και Not-Taken branches, με τα πρώτα να είναι ελάχιστα περισσότερα. Πάλι το μεγαλύτερο μέρος είναι conditiotal .

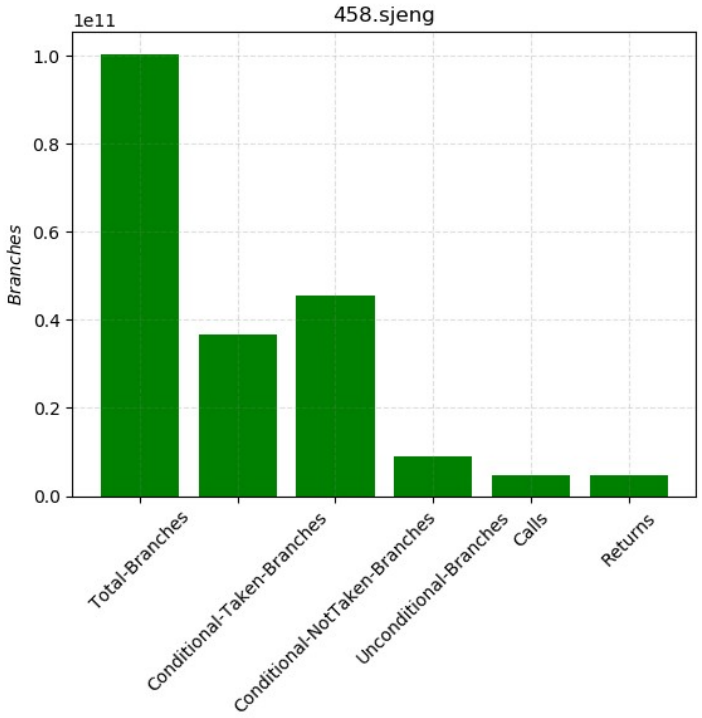
456.hmmer

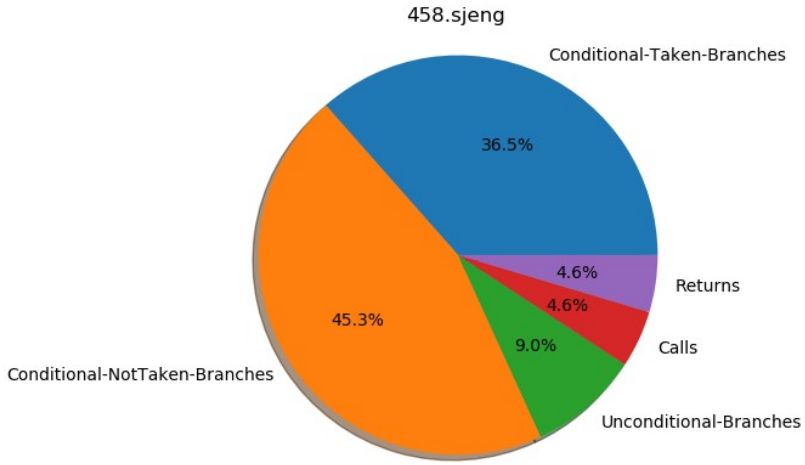




Παρατηρούμε ότι το πλήθος των Taken είναι σχεδόν διπλάσιο από των Not-Taken και πάλι το μεγαλύτερο μέρος είναι conditional branches .

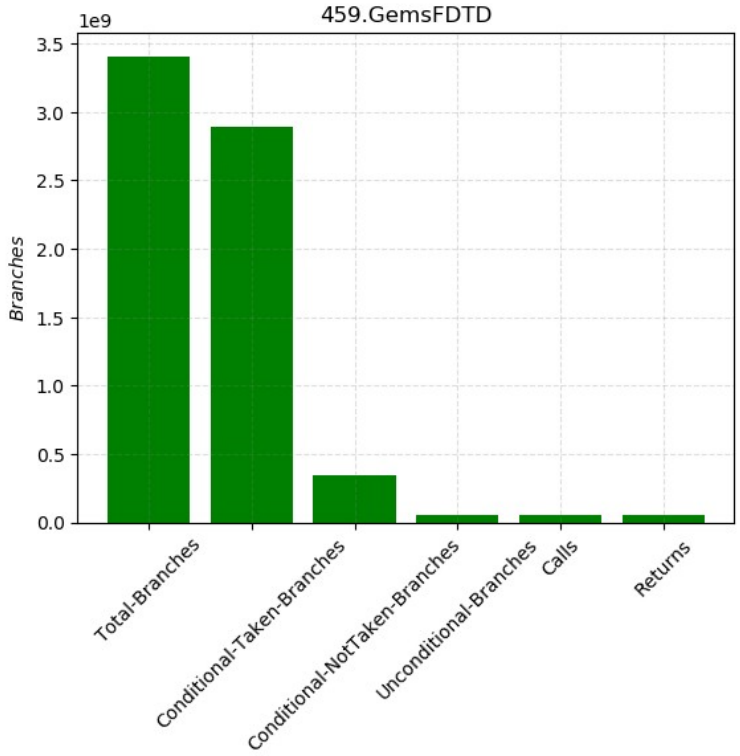
458.sjeng

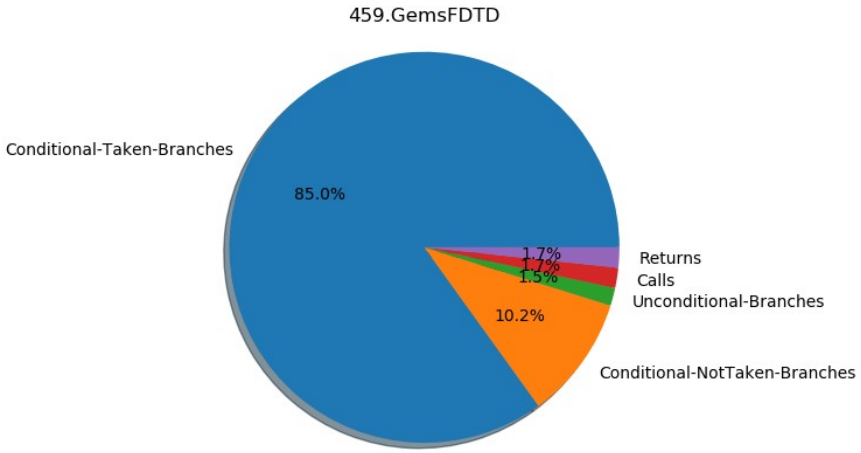




Παρατηρούμε ότι τα Not-Taken είναι περισσότερα από τα Taken, ενώ το πλήθος των Unconditional Branches(συμπεριλαμβανομένου των Call και Returns) εχεί αυξηθεί .

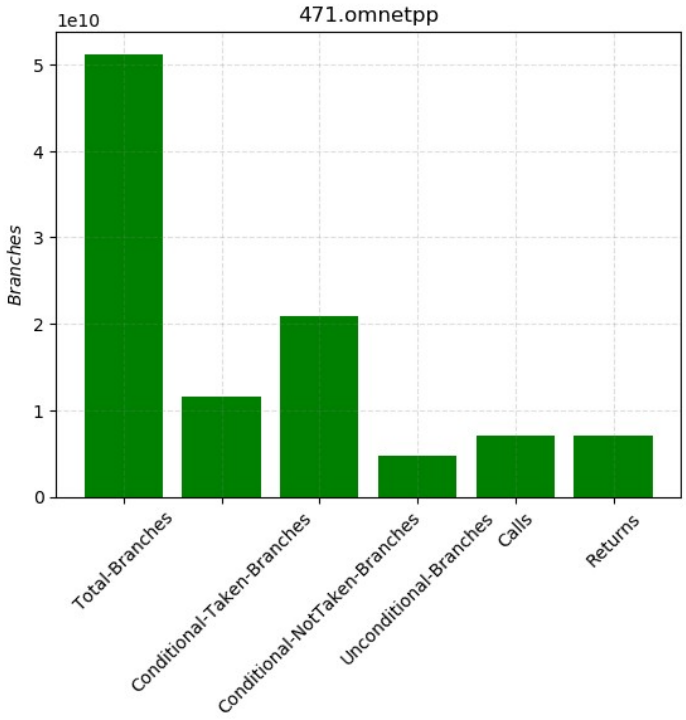
459.GemsFDTD

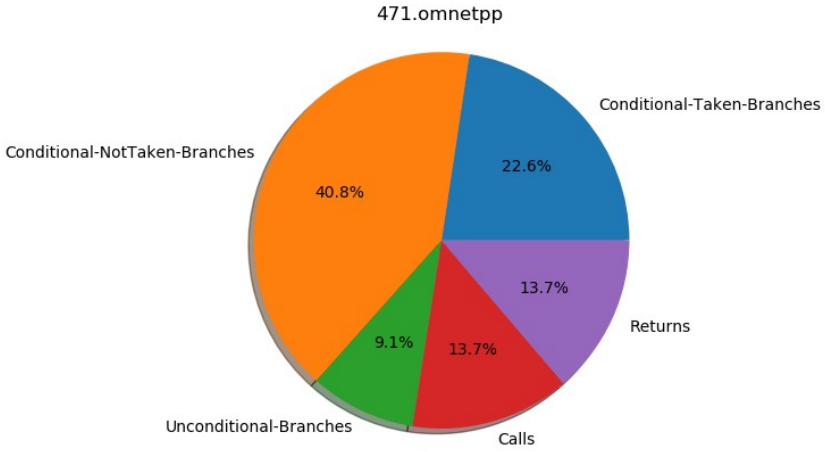




Παρατηρούμε ότι τα περισσότερα branches είναι τα Conditional-Taken, ενώ το πλήθος των Unconditional είναι ελάχιστο . Πιθανώς, υπάρχει μεγάλο κομμάτι κώδικα από loops .

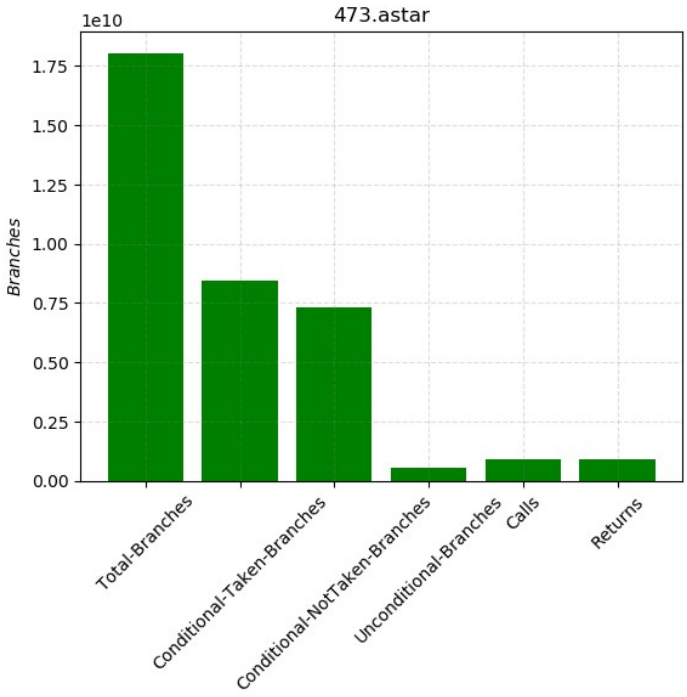
471.omnetpp

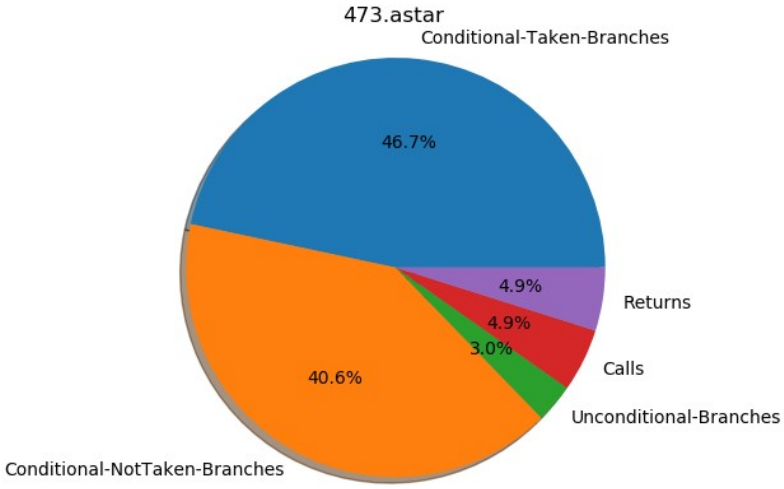




Παρατηρούμε ότι τα Conditional Not-Taken είναι διπλάσια από τα Conditional Taken , ενώ για πρώτη φορά έχουμε αυξημένο αριθμό Calls και αντίστοιχα Returns . Το πλήθος των Unconditional branches αυξήθηκε .

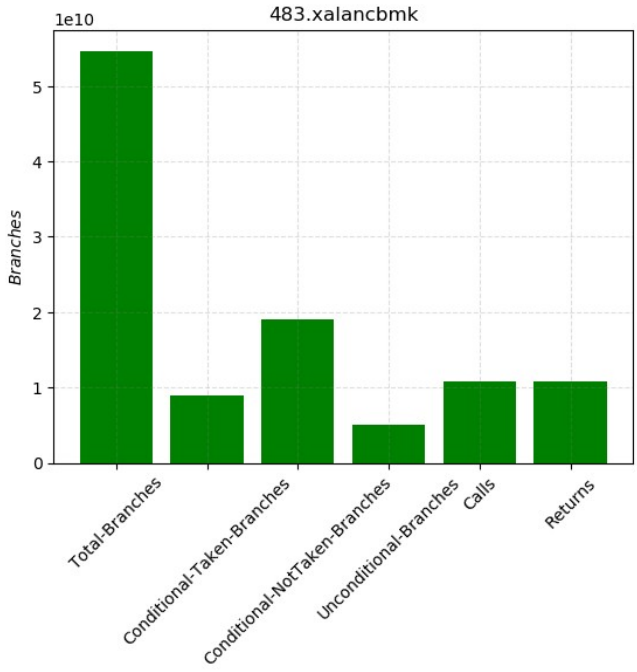
473.astar

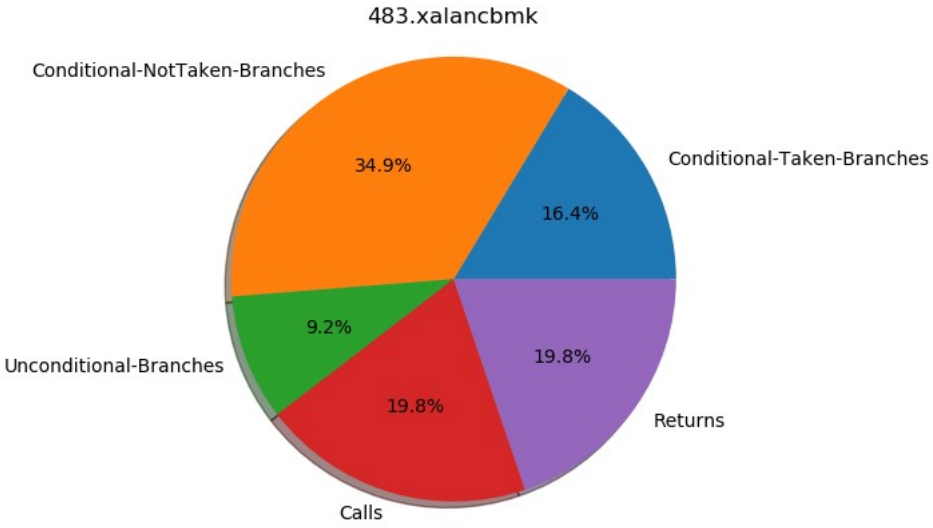




Παρατηρούμε ότι τα Conditional Taken είναι ελαφρώς περισσότερα από τα Conditional Not-Taken . Τα Conditional είναι το μεγαλύτερο μέρος των branches.

483.xalancbmk





Παρατηρούμε ότι τα branches έχουμε χωριστεί ομοιόμορφα σε Conditional και Unconditional .
Τα Conditional Not-Taken είναι διπλάσια των Conditional Taken και τα Calls – Returns αποτελούν το μεγαλύτερο κομμάτι των Unonditional branches .

Συμπεράσματα

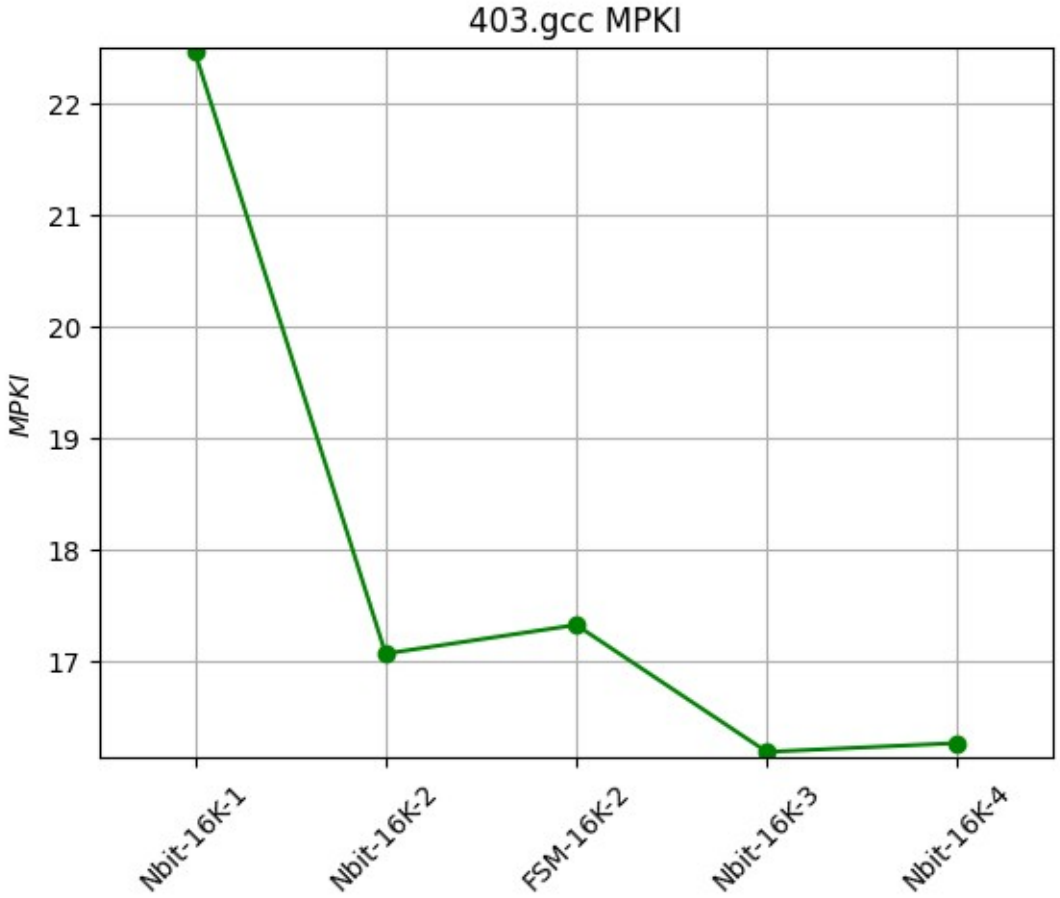
Από τα διαγράμματα παρατηρήσαμε ότι για τα περισσότερα benchmarks το μεγαλύτερο κομμάτι των branches είναι Conditional . Οπότε η απόδοση των εφαρμογών θα αυξηθεί αν συμπεριλάβουμε την χρήση branch predictors, οι οποίοι θα βοηθήσουν να κάνουν πρόβλεψη για τα conditional branches .

➢ Μελέτη N-bit Predictors

Στο συγκεκριμένο ερώτημα θα προσομοιώσουμε την λειτουργία ενός N-bit Predictor και θα εξετάσουμε πως μπορεί να βελτιώσει την απόδοση των εφαρμογών μας . Συγκεκριμένα, στο πρώτο κομμάτι θα χρησιμοποιήσουμε τους προβλεπτές N-bit = 1,2,3,4 και FSM, διατηρώντας σταθερό πλήθος entries σε 16K, για κάθε predictor . Ενώ στο δεύτερο θα χρησιμοποιήσουμε N-bit = 1,2,4 και FSM, έχοντας συνολικό hardware για κάθε predictor 32Kbits .

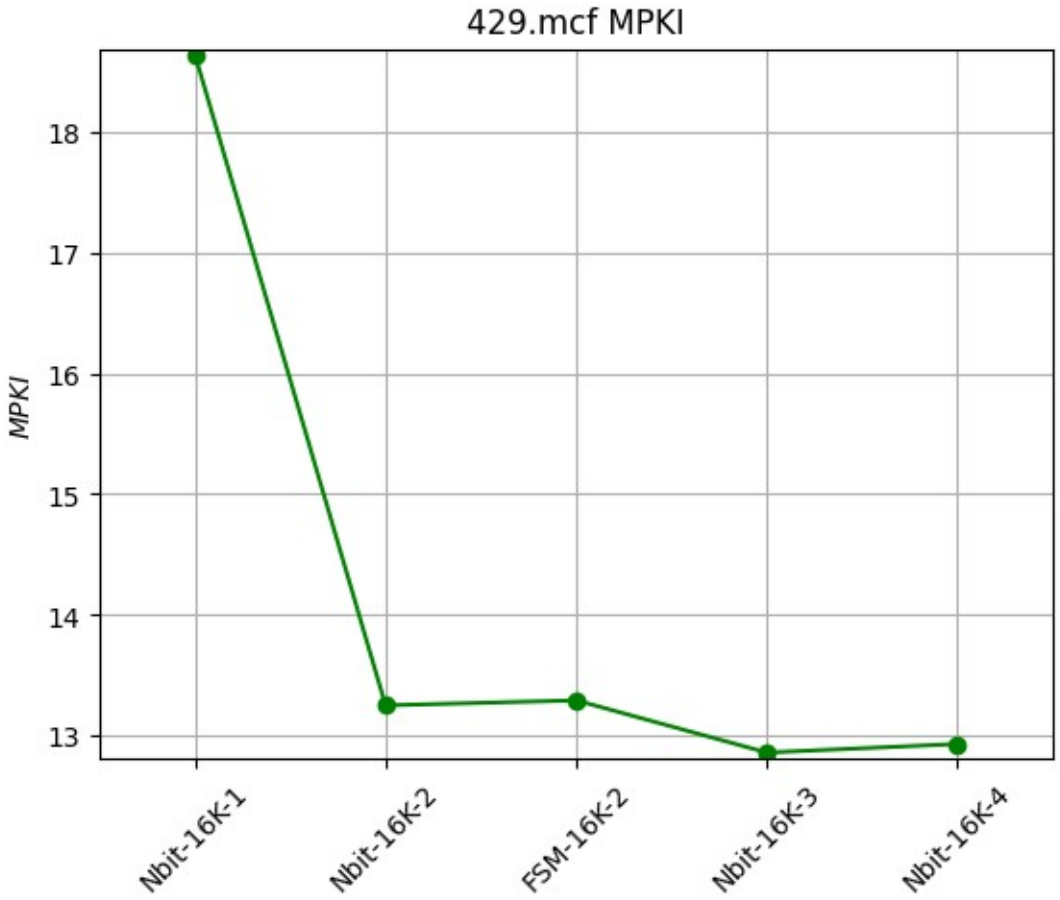
i. 16K entries ανά perdictor

403.gcc



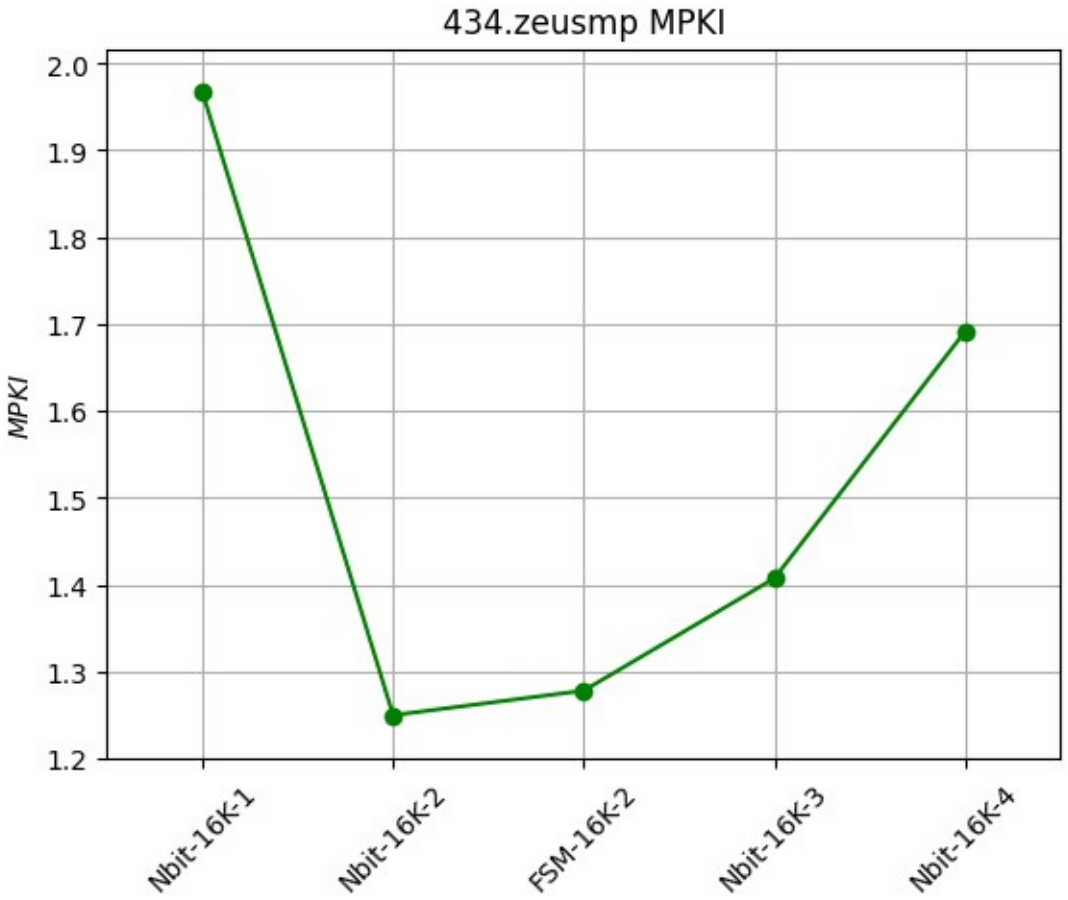
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM αν και χρησιμοποιεί το ίδιο πλήθος bits = 2 με τον 2-bit, έχει ελαφρώς χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχαμε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση επηρεάζει λιγότερο το MPKI (2-bit -> 3-bit) και στην περίπτωση 3-bit -> 4-bit διατηρείται σταθερό . Από πλευρά απόδοσης η βέλτιστη επιλογή είναι 3-bit .Όμως, τα αποτελέσματα για τους 3-bit και 2-bit διαφέρουν ελάχιστα, μπορούμε για εξοικονόμηση hardware, να επιλέξουμε τον 2-bit ,χωρίς να επηρεαστεί σημαντικά η απόδοση.

429.mcf



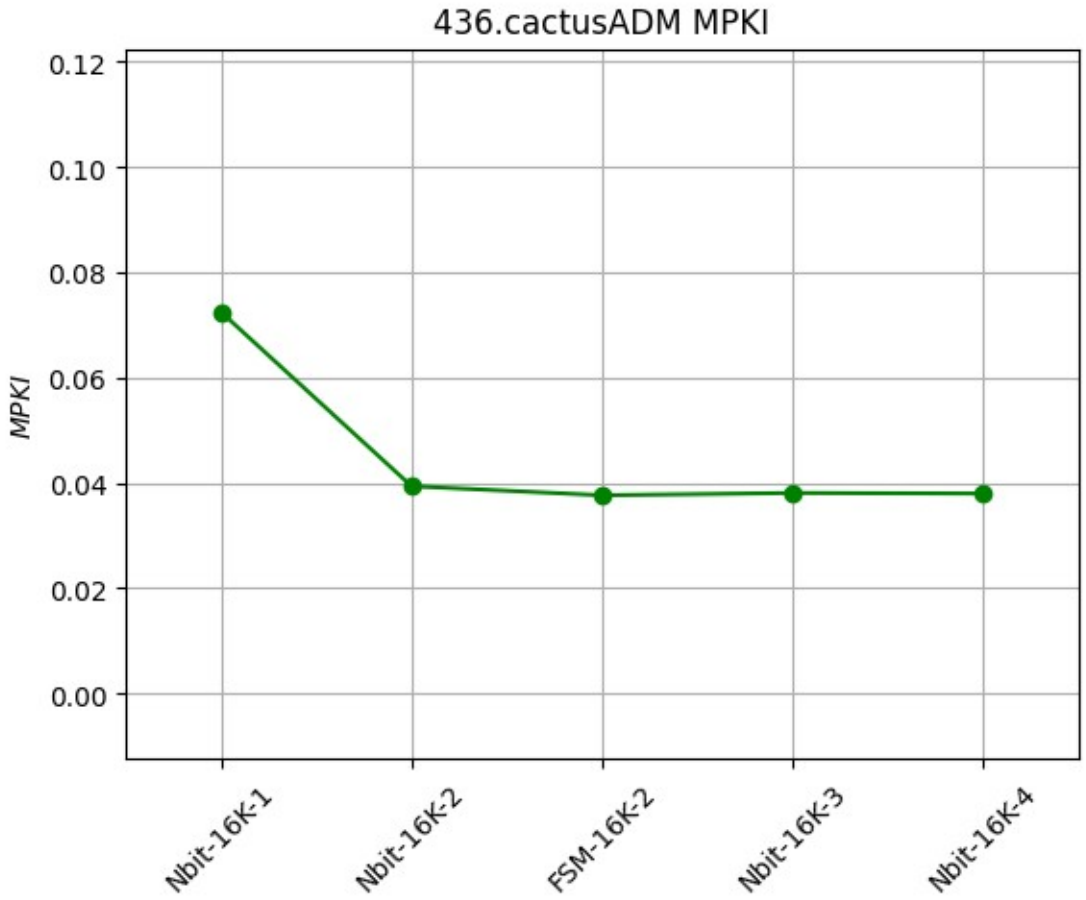
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM έχει την ίδια απόδοση με τον 2-bit. Η αρχική αύξηση από 1-bit σε 2-bit είχαμε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση επηρεάζει ελάχιστα το MPKI (2-bit -> 3-bit) και στην περίπτωση 3-bit -> 4-bit διατηρείται σταθερό . Από πλευρά απόδοσης η βέλτιστη επιλογή είναι 3-bit . Όμως οι 2-bit και 3-bit έχουν πολύ κοντινά αποτελέσματα,οπότε για περισσότερη εξοικονόμηση hardware μπορούμε να χρησιμοποιήσουμε τον 2-bit χωρίς ιδιαίτερη επίδραση στην απόδοση .

434.zeusmp



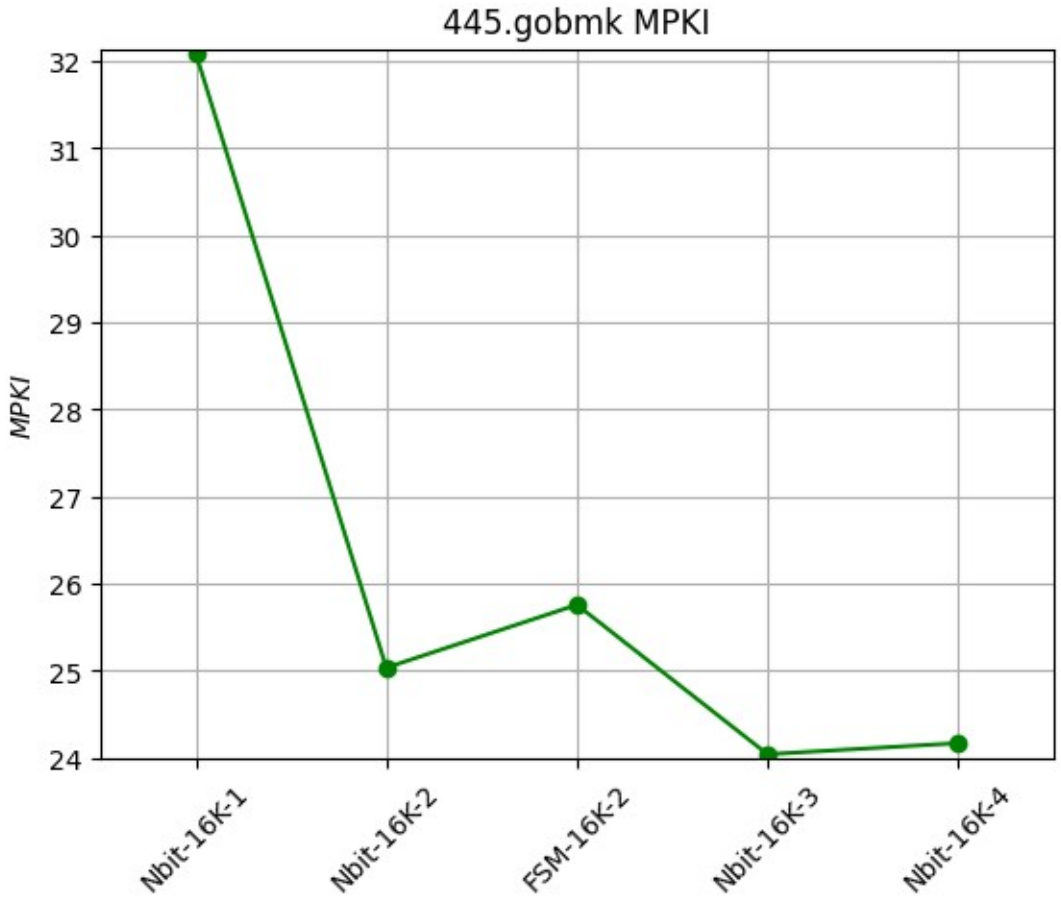
Παρατηρούμε ότι η αρχική αύξηση των bits μειώνει το MPKI . Ο 2-bit έχει την σχεδόν ίδια απόδοση με τον FSM . Η αύξηση πέραν του 2-bit οδηγεί σε αντίστοιχη αύξηση του MPKI, όμως λόγω του μικρού μεγέθους του MPKI μπορούμε να θεωρήσουμε την διαφορά αμελητέα . Η βέλτιστη επιλογή από πλευράς απόδοσης και hardware είναι ο 2-bit .

436.cactusAMD



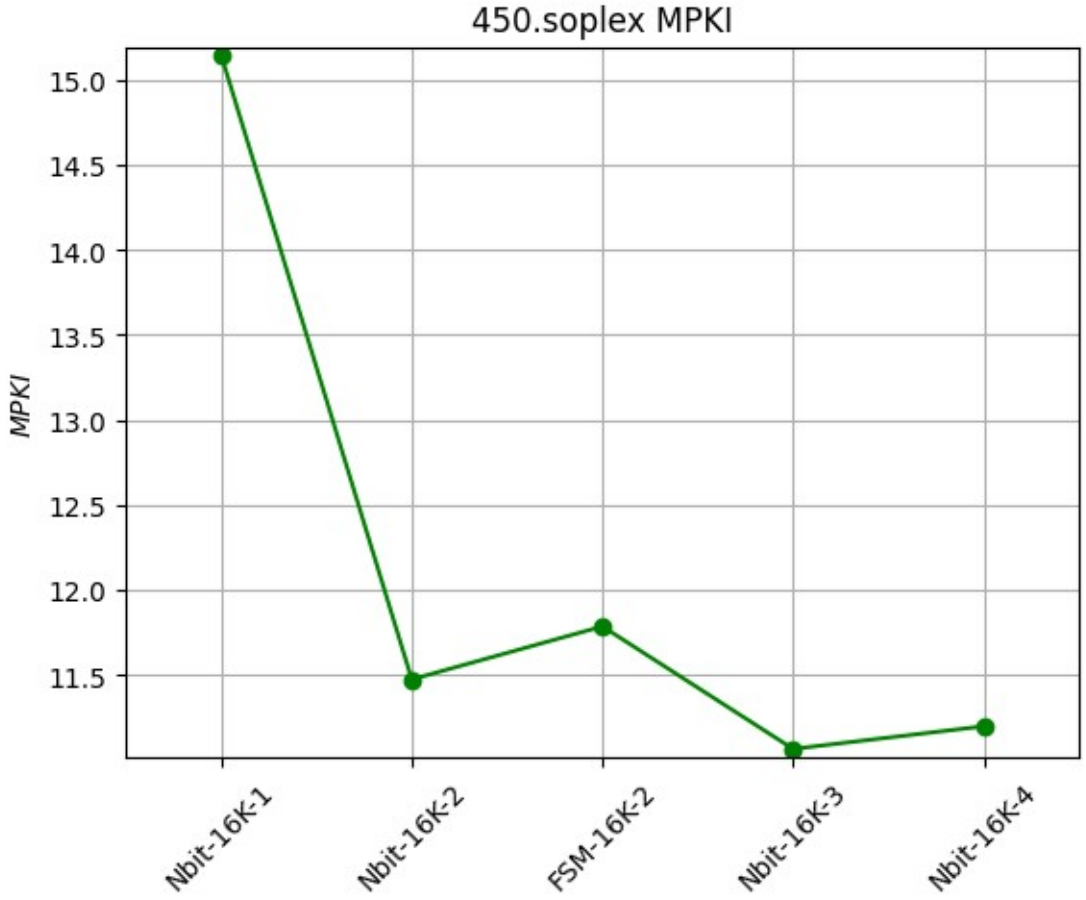
Παρατηρούμε ότι η αρχική αύξηση των bits μειώνει το MPKI . Ο 2-bit έχει την ίδια απόδοση με τον FSM . Η αύξηση πέραν του 2-bit δεν επηρεάζει το MPKI . Η βέλτιστη επιλογή από πλευράς απόδοσης και hardware είναι ο 2-bit . Όμως, το MPKI είναι εξαιρετικό μικρό οπότε για εξοικονόμηση hardware μπορούμε να διαλέξουμε το 1-bit χωρίς ουσιαστική διαφοροποίηση στην απόδοση .

445.gobmk



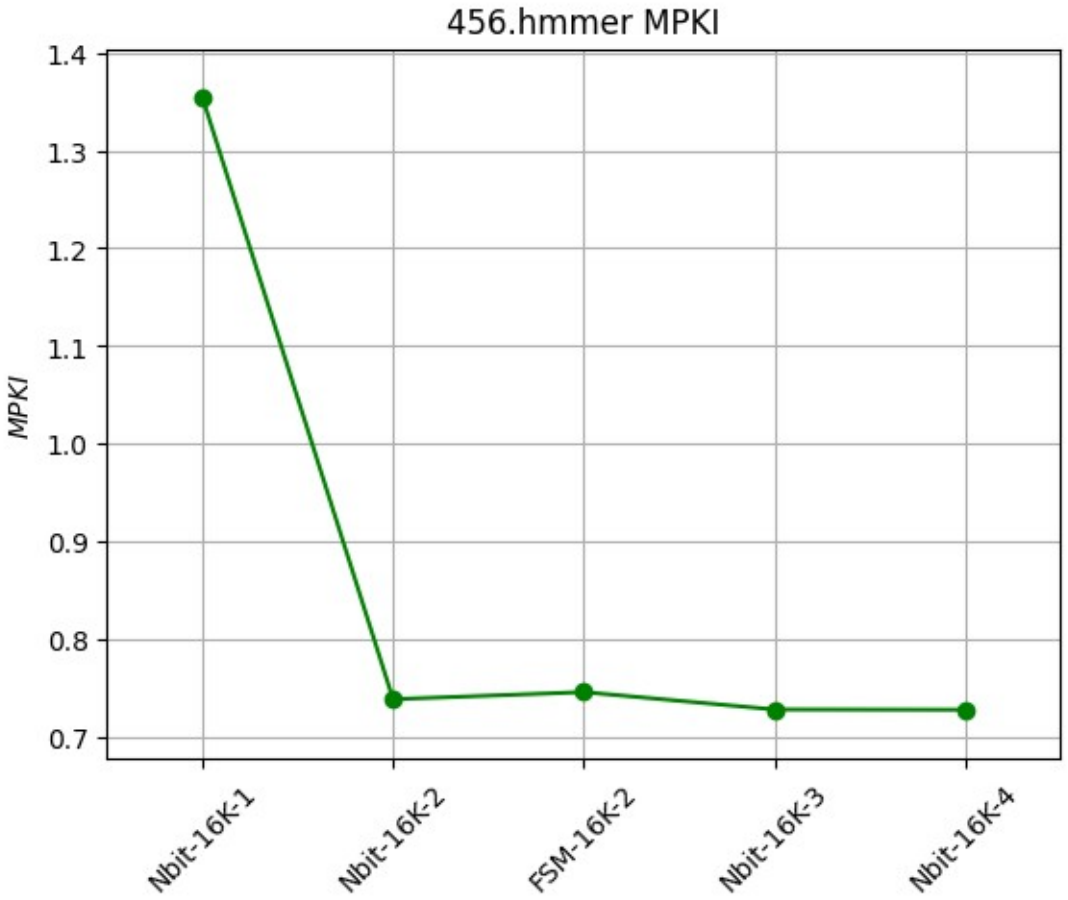
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM έχει ελάχιστα χειρότερη απόδοση από τον 2-bit. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση επηρεάζει λιγότερο το MPKI (2-bit -> 3-bit) και στην περίπτωση 3-bit -> 4-bit διατηρείται σταθερό . Από πλευρά απόδοσης η βέλτιστη επιλογή είναι 3-bit . Ομως, επειδή τα αποτελέσματα για τους 3-bit και 2-bit διαφέρουν ελάχιστα, μπορούμε για εξοικονόμηση hardware, να επιλέξουμε τον 2-bit ,χωρίς να επηρεάσουμε ιδιαίτερα την απόδοση.

450.soplex



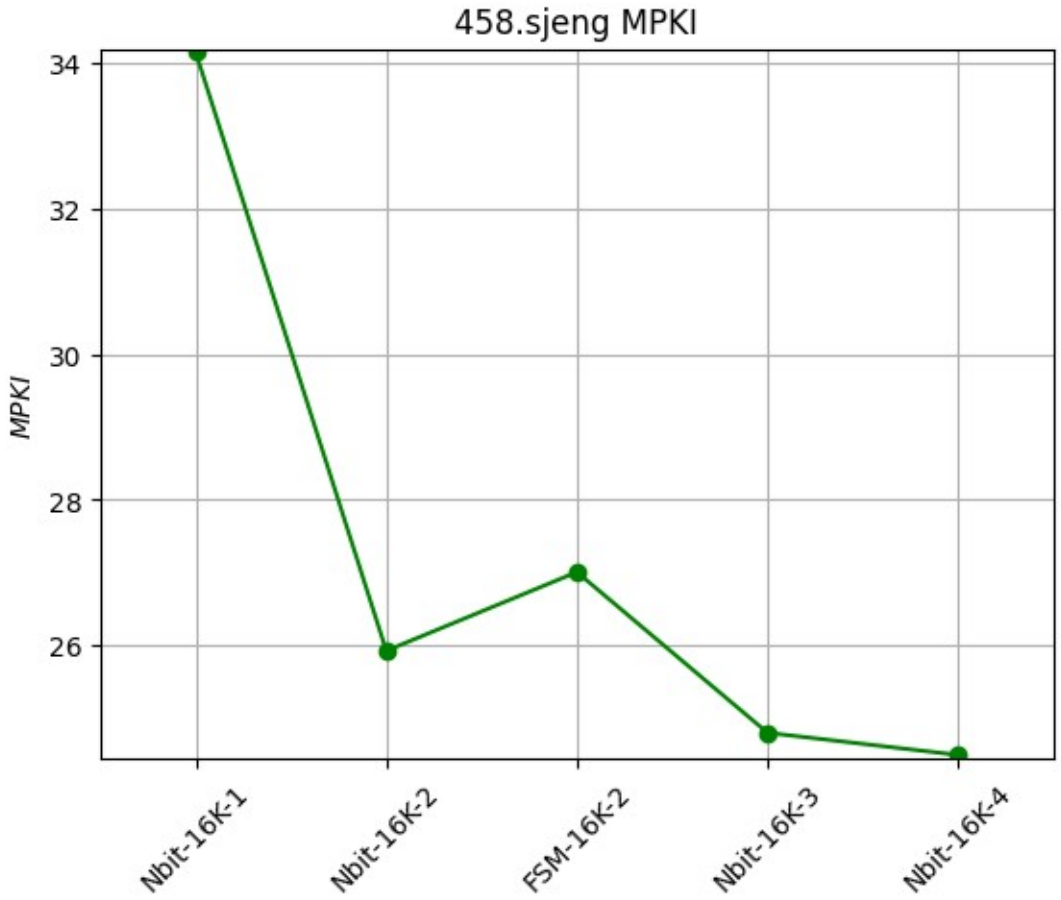
Ομοίως με το προηγούμενο 445.gobmk .

456.hmmer



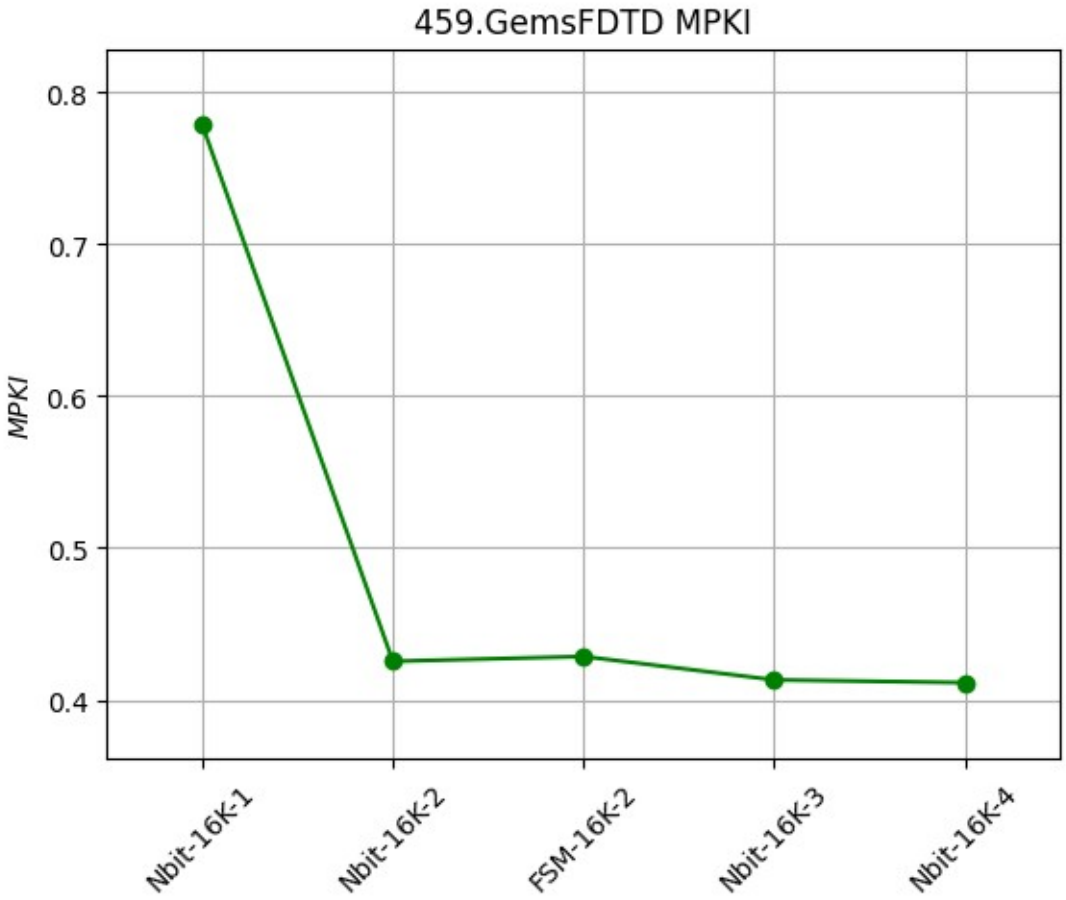
Παρατηρούμε ότι η αρχική αύξηση των bits μειώνει το MPKI . Ο 2-bit έχει την ίδια απόδοση με τον FSM . Η αύξηση πέραν του 2-bit δεν επηρεάζει το MPKI . Η βέλτιστη επιλογή από πλευράς απόδοσης και hardware είναι ο 2-bit .

458.sjeng



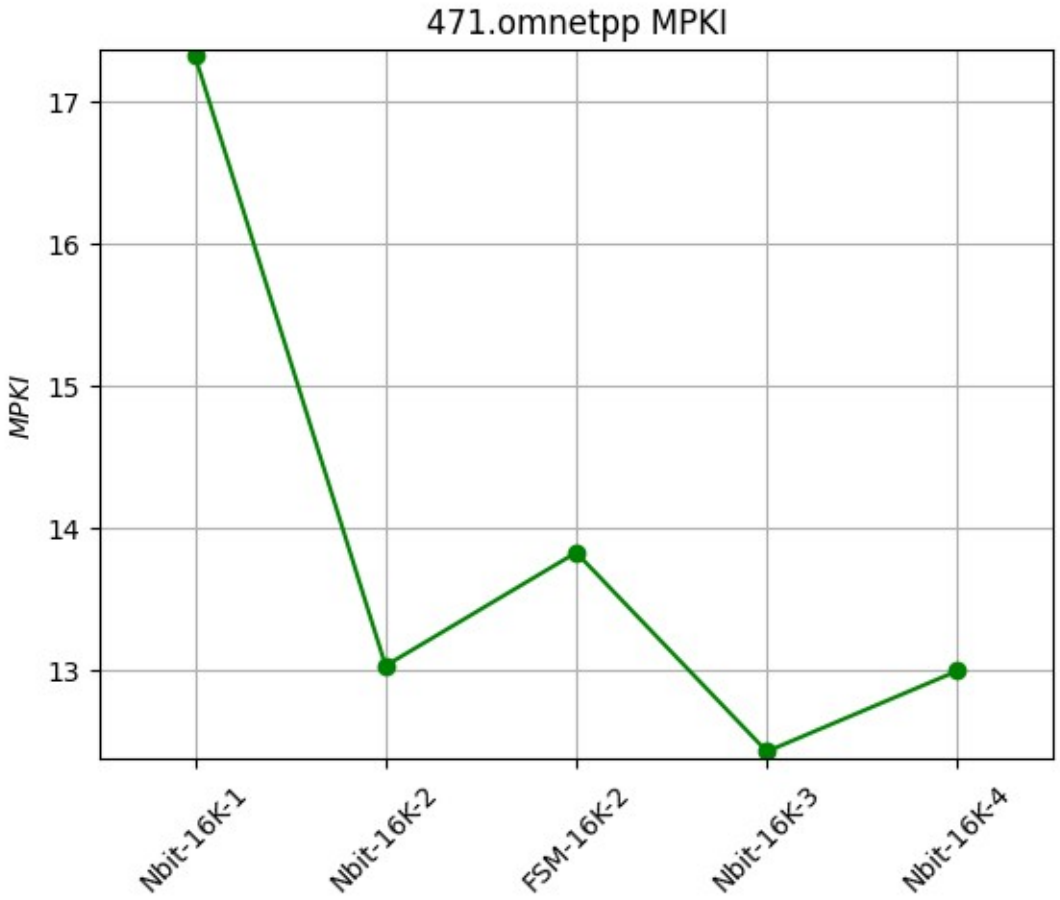
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σχέση με τον 2-bit, έχει ελαφρώς χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση μειώνει λιγότερο το MPKI . Από πλευρά απόδοσης η βέλτιστη επιλογή είναι 4-bit .’Ομως, επειδή τα αποτελέσματα για τους 3-bit και 4-bit διαφέρουν ελάχιστα, μπορούμε για εξοικονόμηση hardware, να επιλέξουμε τον 3-bit ,χωρίς να επηρεαστεί σημαντικά η απόδοση.

459.GemFDTD



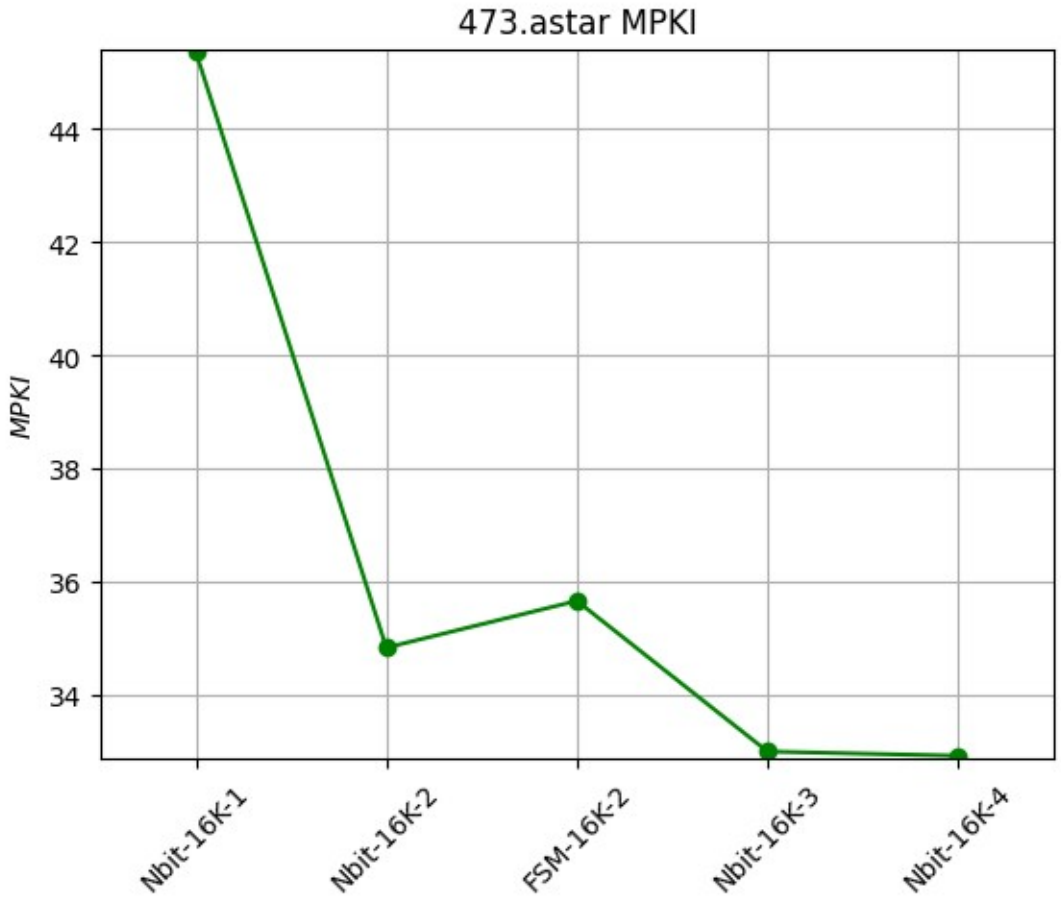
Παρατηρούμε ότι η αρχική αύξηση των bits μειώνει το MPKI . Ο 2-bit έχει την ίδια απόδοση με τον FSM . Η αύξηση πέραν του 2-bit δεν επηρεάζει το MPKI . Η βέλτιστη επιλογή από πλευράς απόδοσης και hardware είναι ο 2-bit .

471.omnetpp



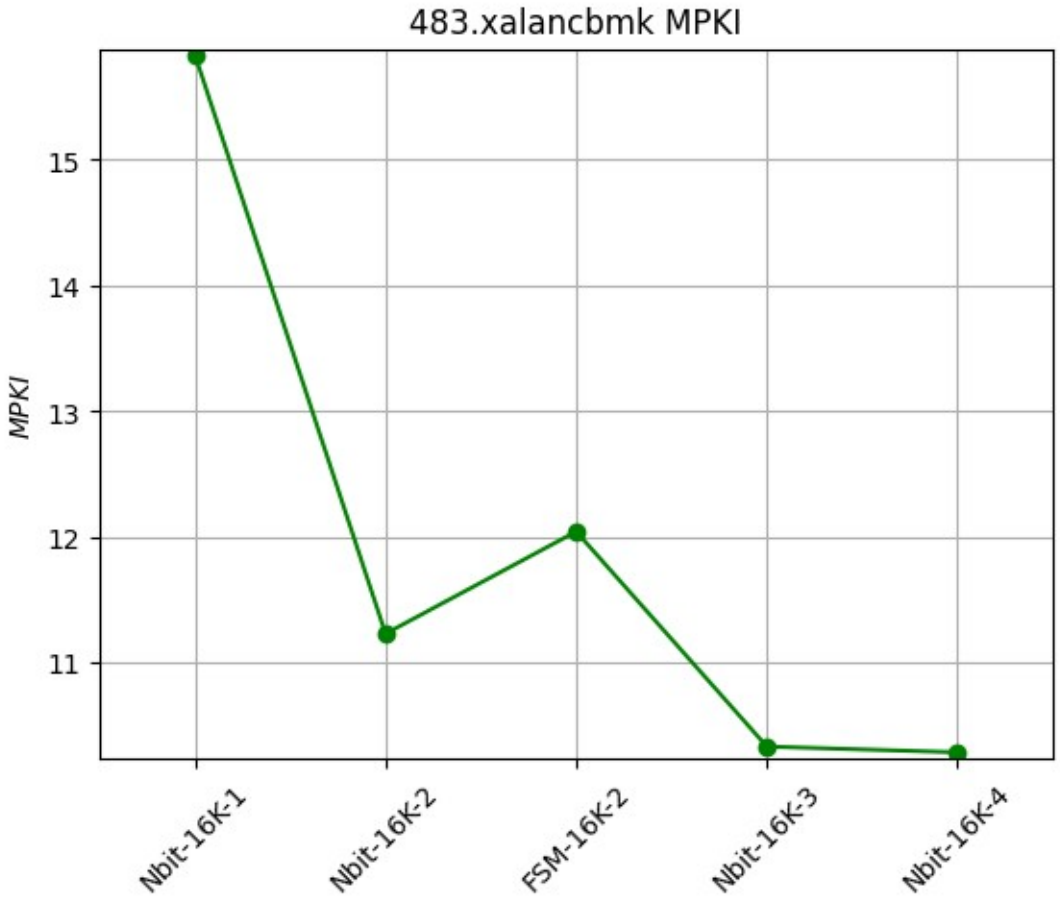
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σύγκριση με τον 2-bit, έχει χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση μειώνει λιγότερο το MPKI (2-bit -> 3-bit) και στην περίπτωση 3-bit -> 4-bit το αυξάνει . Από πλευρά απόδοσης, σε σχέση με το hardware, η βέλτιστη επιλογή είναι 2-bit .

473.astar



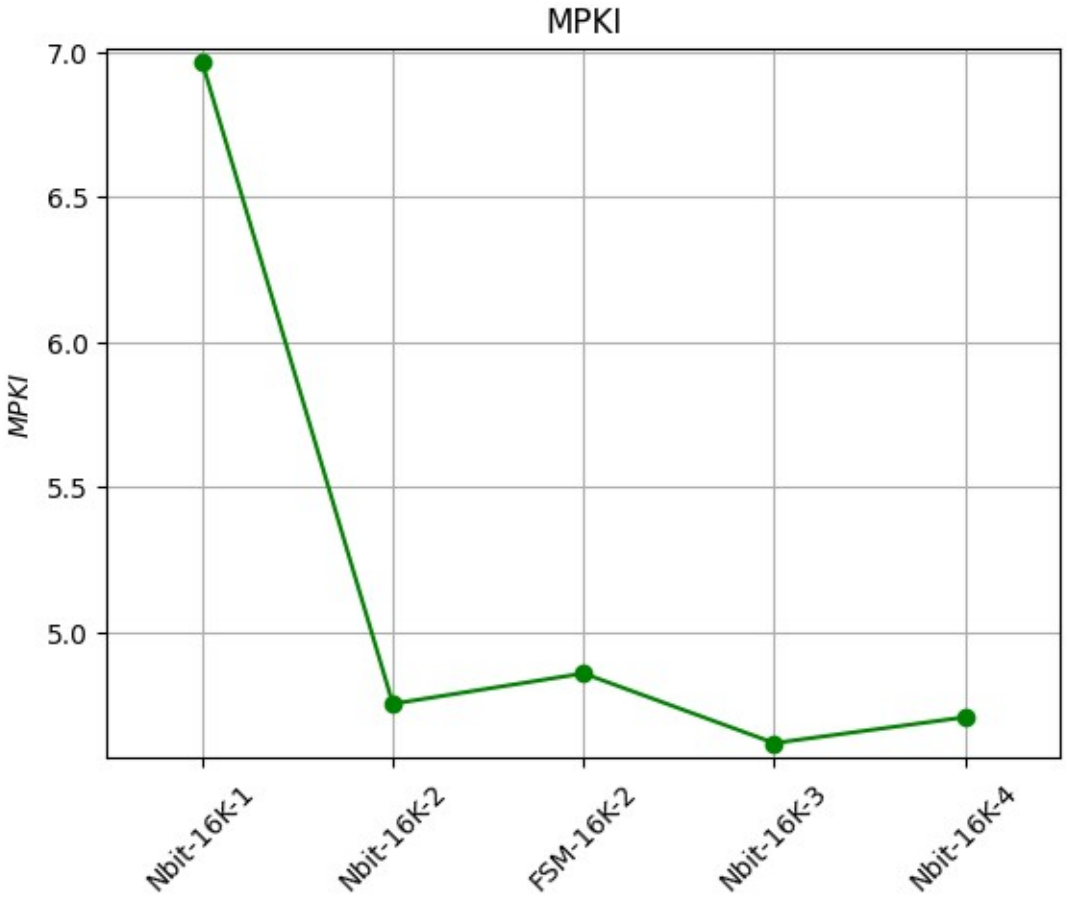
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σχέσημε τον 2-bit, έχει ελαφρώς χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση μειώνει λιγότερο το MPKI (2-bit -> 3-bit) και στην περίπτωση 3-bit -> 4-bit το διατηρεί σταθερό . Από πλευρά απόδοσης η βέλτιστη επιλογή είναι 3-bit . Για να εξοικονομήσουμε hardware μπορούμε να επιλέξουμε τον 2-bit, χωρίς ουσιαστική διαφορά στην απόδοση.

483.xalancbmk



Ομοίως με το προηγούμενο 473.astar .

Συμπεράσματα



Στο παραπάνω διάγραμμα έχουμε τον γεωμετρικό μέσο για αποτελέσματα των benchmarks . Το διάγραμμα συμβαδίζει με την ανάλυση που κάναμε για τις περισσότερες εφαρμογές. Πιο συγκεκριμένα, παρατηρούμε μείωση του MPKI για την αρχική αύξηση 1-bti -> 2-bit. Ο FSM έχει σχεδόν την ίδια απόδοση με τον 2-bit . Η αύξηση πέρα των 2 bits στον N-bit φαίνεται να μην επηρεάζει ιδιαίτερα το MPKI .

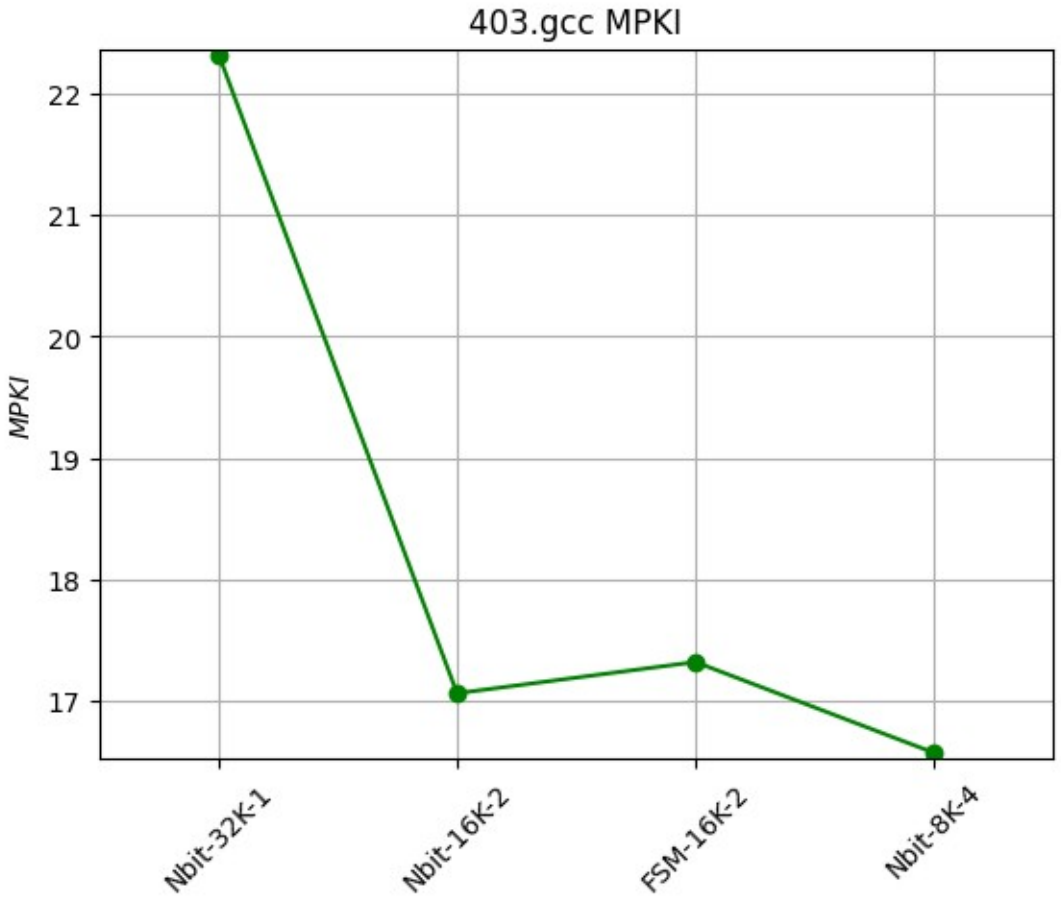
Στην υλοποίηση μας είχαμε ένα πίνακα από predictors σταθερών entries και κάθε ένας από αυτούς μπορεί να χρησιμοποιηθεί από διαφορετικά branches. Μια πιθανή αιτία για την αύξηση του MKPI στην περίπτωση 3-bit -> 4-bit είναι, ότι ο 4-bit μετά από ένα loop θα χρειαστεί περισσότερες μεταβάσεις για να προβλέψει Not-Taken . Όποτε ο 4-bit θα αποτυχαίνει σε περισσότερα branches που είναι Not-Taken μετά από ένα loop . Παρόλο αυτά, η διαφορά των αποτελεσμάτων για τους 3-bit και 4-bit είναι ελάχιστη και μπορεί να θεωρηθεί αμελητέα .

Την βέλτιστη απόδοση την έχουμε για τον 3-bit, ωστόσο η διαφορά του από τον 2-bit είναι πολύ μικρή και δεν δικαιολογεί το overhead σε hardware . Οπότε επιλέγουμε τον 2-bit ,που συμφωνεί με την ανάλυση που κάναμε για τις περισσότερες εφαρμογές .

ii. 32Kbits σταθερό hardware

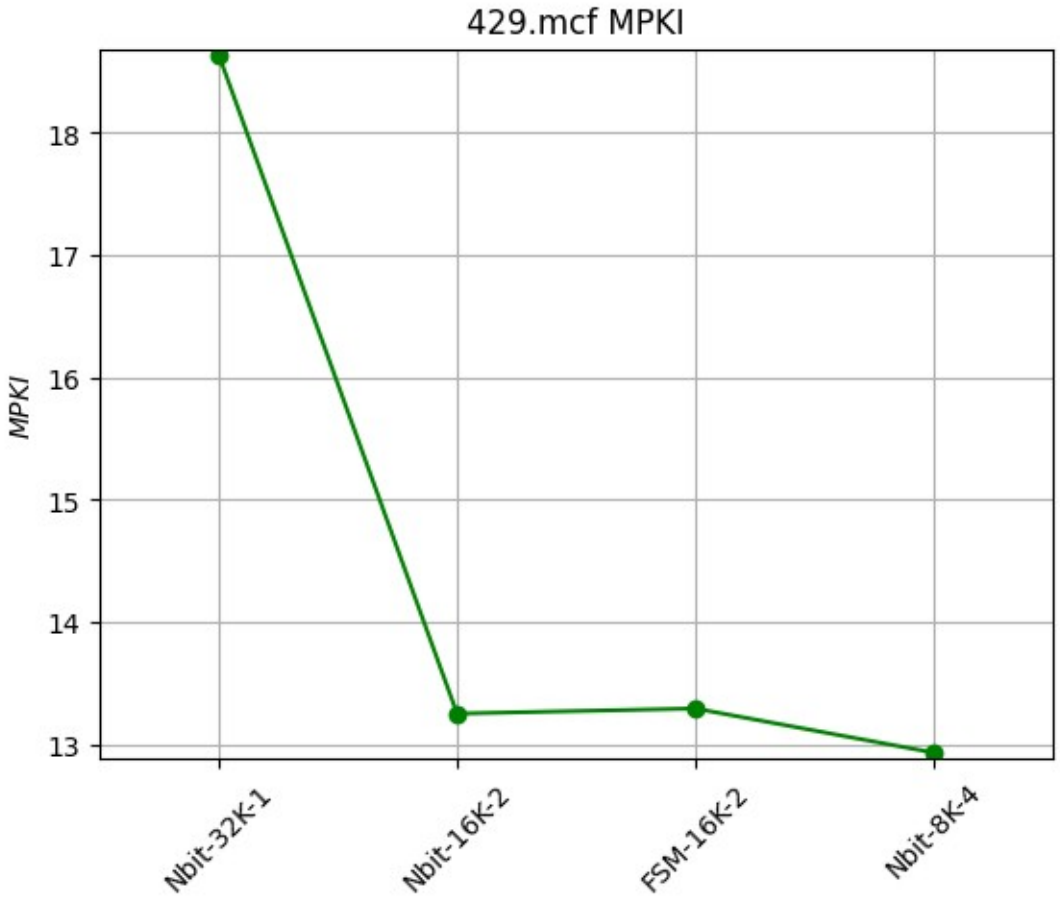
- 1-bit : $\frac{32\,Kbits}{1\,bit} \Rightarrow 32\,K\,entries$
- 2-bit : $\frac{32\,Kbits}{2\,bits} \Rightarrow 16\,K\,entries$
- FSM : $\frac{32\,Kbits}{2\,bits} \Rightarrow 16\,K\,entries$
- 4-bit : $\frac{32\,Kbits}{4\,bits} \Rightarrow 8\,K\,entries$

403.gcc



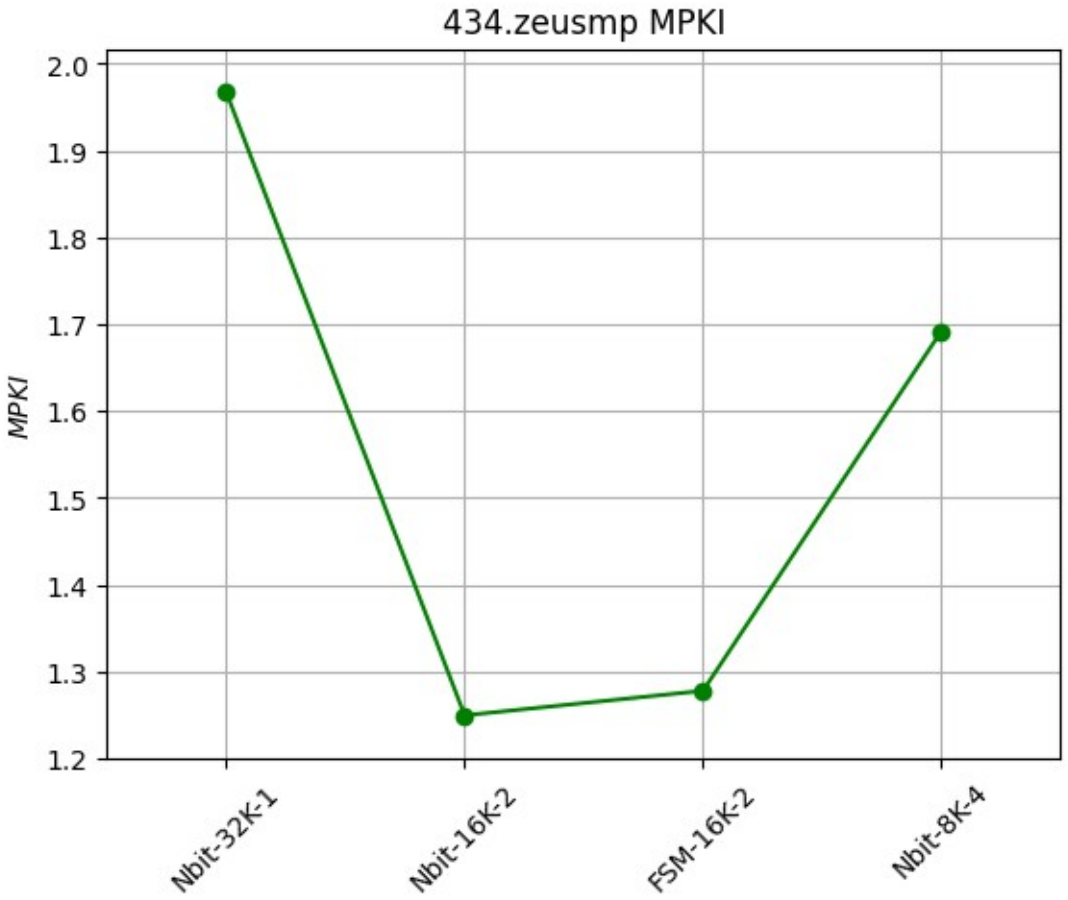
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σχέση με τον 2-bit, έχει ελάχιστα χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση μειώνει πιο λίγο το MPKI (2-bit -> 4-bit) . Η βέλτιστη επιλογή είναι 4-bit .

429.mcf



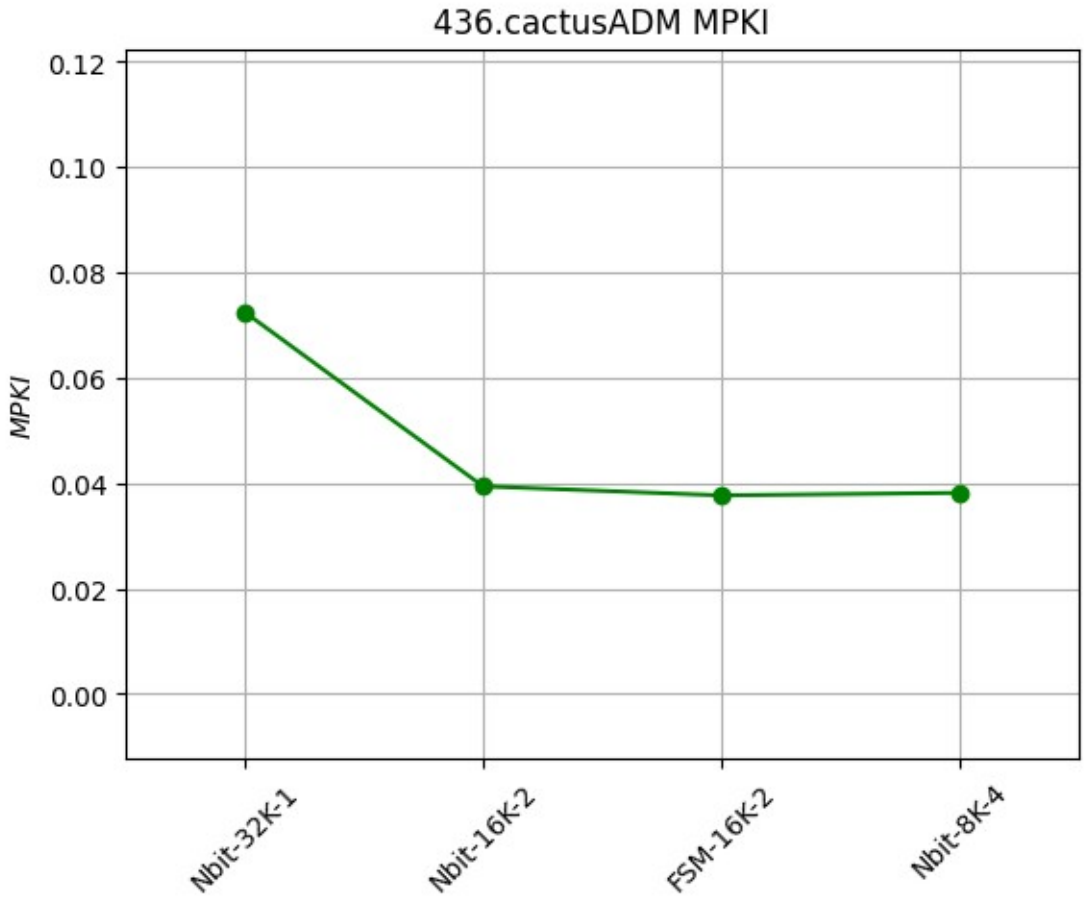
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM έχει την ίδια απόδοση με τον 2-bit . Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση των bit διατηρεί πρακτικά σταθερό το MPKI (2-bit -> 4-bit) . Η βέλτιστη επιλογή είναι 4-bit .

434.zeusmp



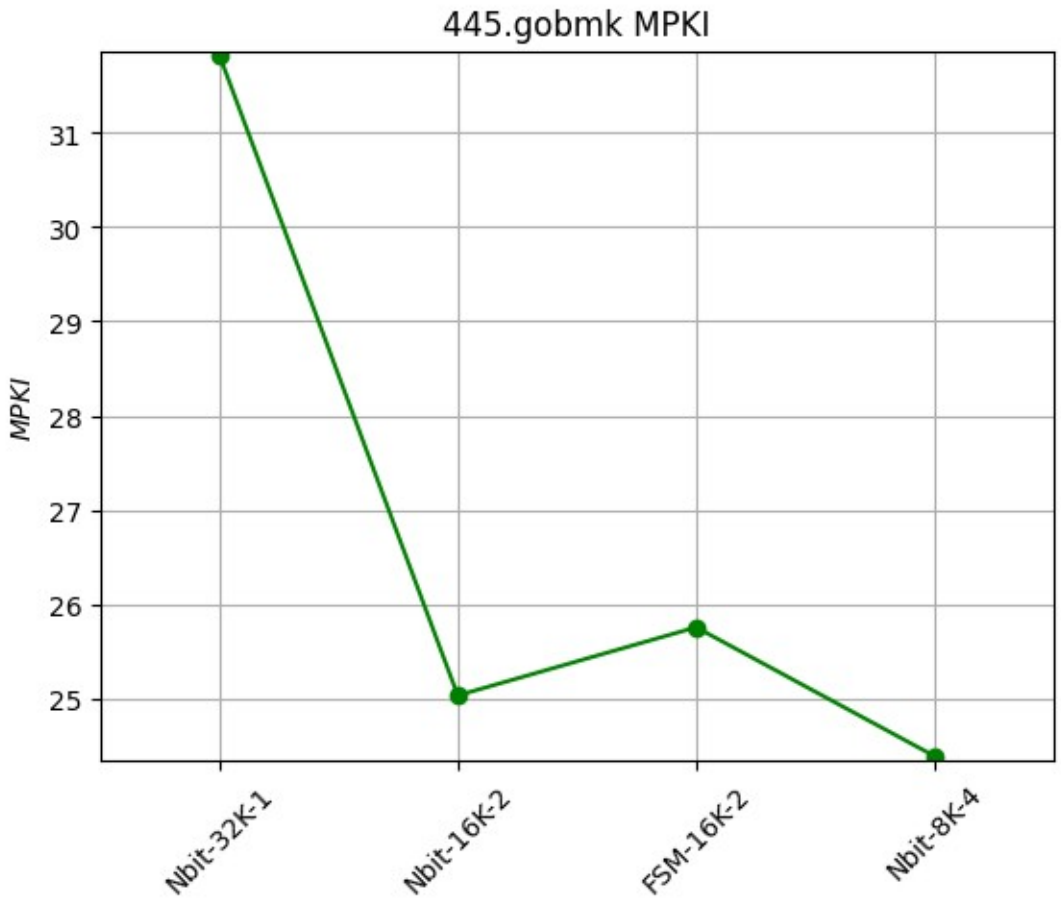
Παρατηρούμε ότι η αρχική αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σχέση με τον 2-bit, έχει ελάχιστα χειρότερη απόδοση. Ωστόσο, η αύξηση 2-bit -> 4-bit αυξάνει το MPKI . Η βέλτιστη επιλογή είναι 2-bit .

436.cactusAMD



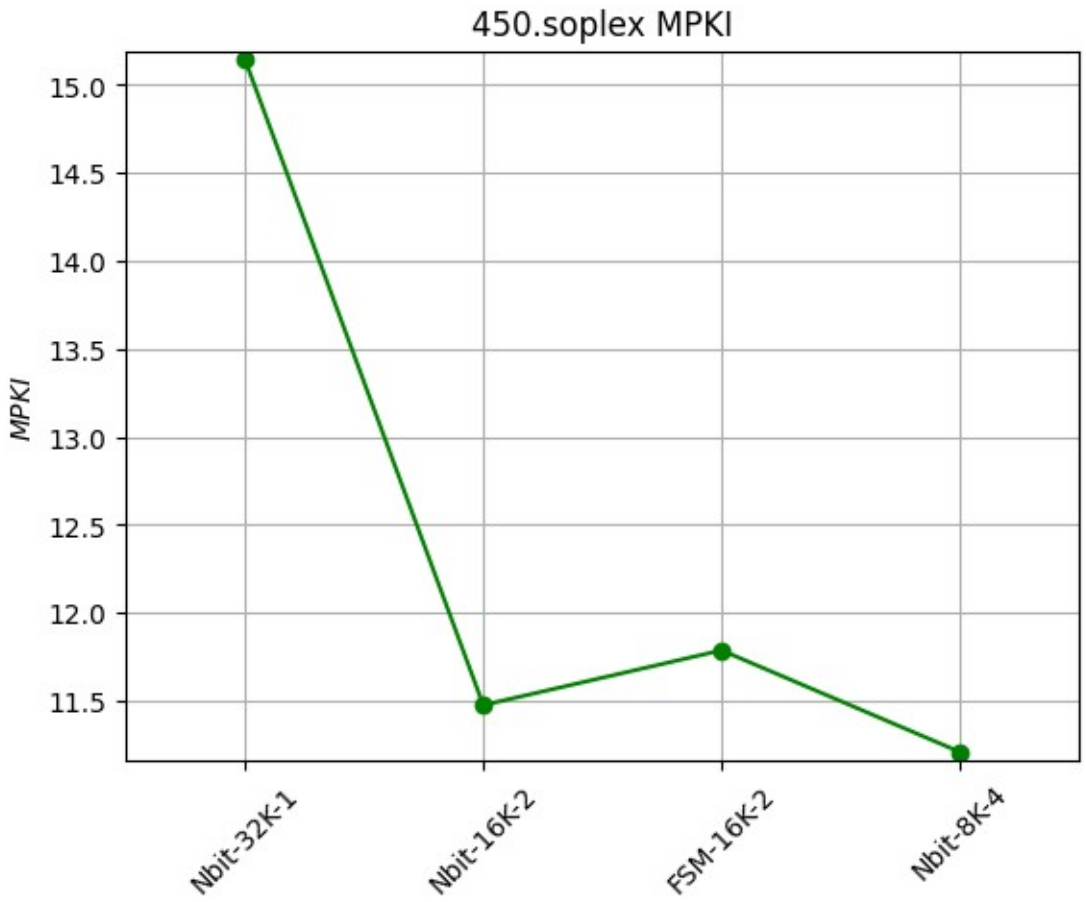
Παρατηρούμε ότι η αρχική αύξηση των bits μειώνει το MPKI . Ο 2-bit έχει την ίδια απόδοση με τον FSM . Η αύξηση πέραν του 2-bit δεν επηρεάζει το MPKI . Μπορούμε να επιλέξουμε οποιονδήποτε από τους 2-bit, FSM και 4-bit .

445.gobmk



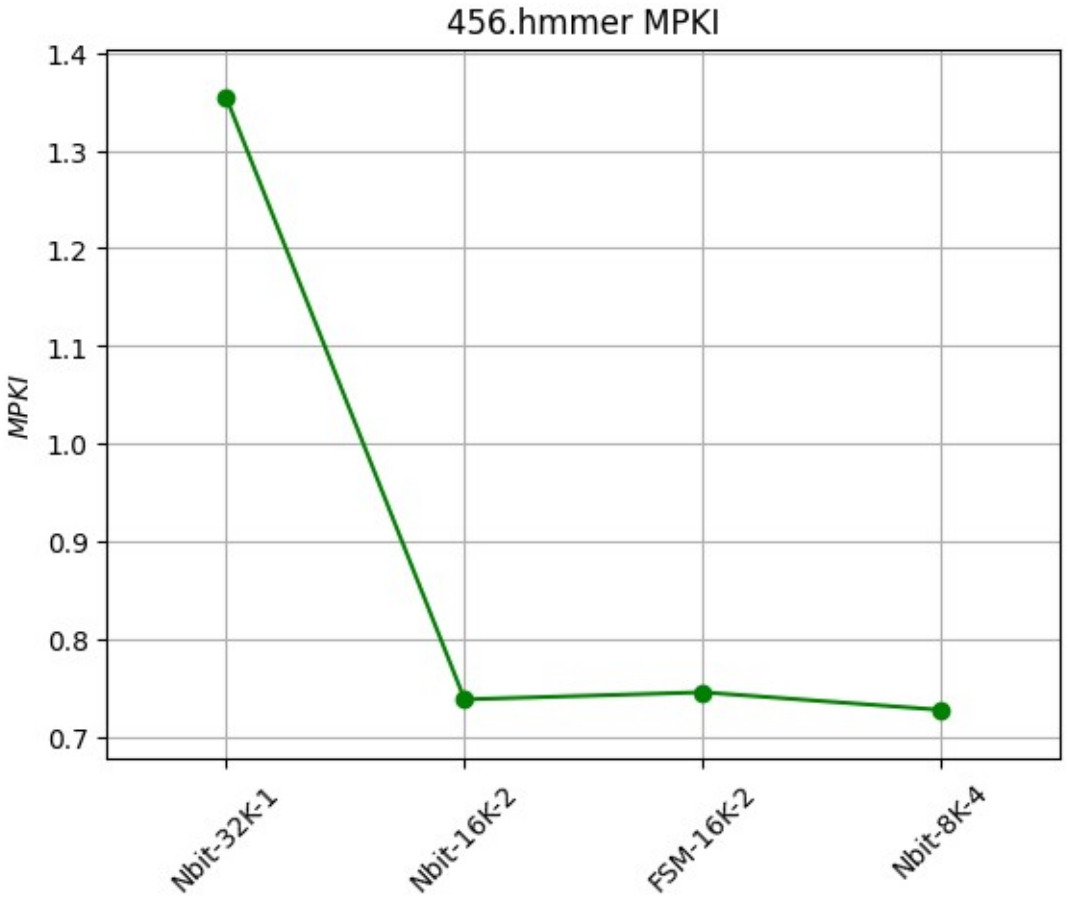
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σχέση με τον 2-bit, έχει ελάχιστα χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η περαιτέρω αύξηση μειώνει πιο λίγο το MPKI (2-bit -> 4-bit) . Η βέλτιστη επιλογή είναι 4-bit .

450.soplex



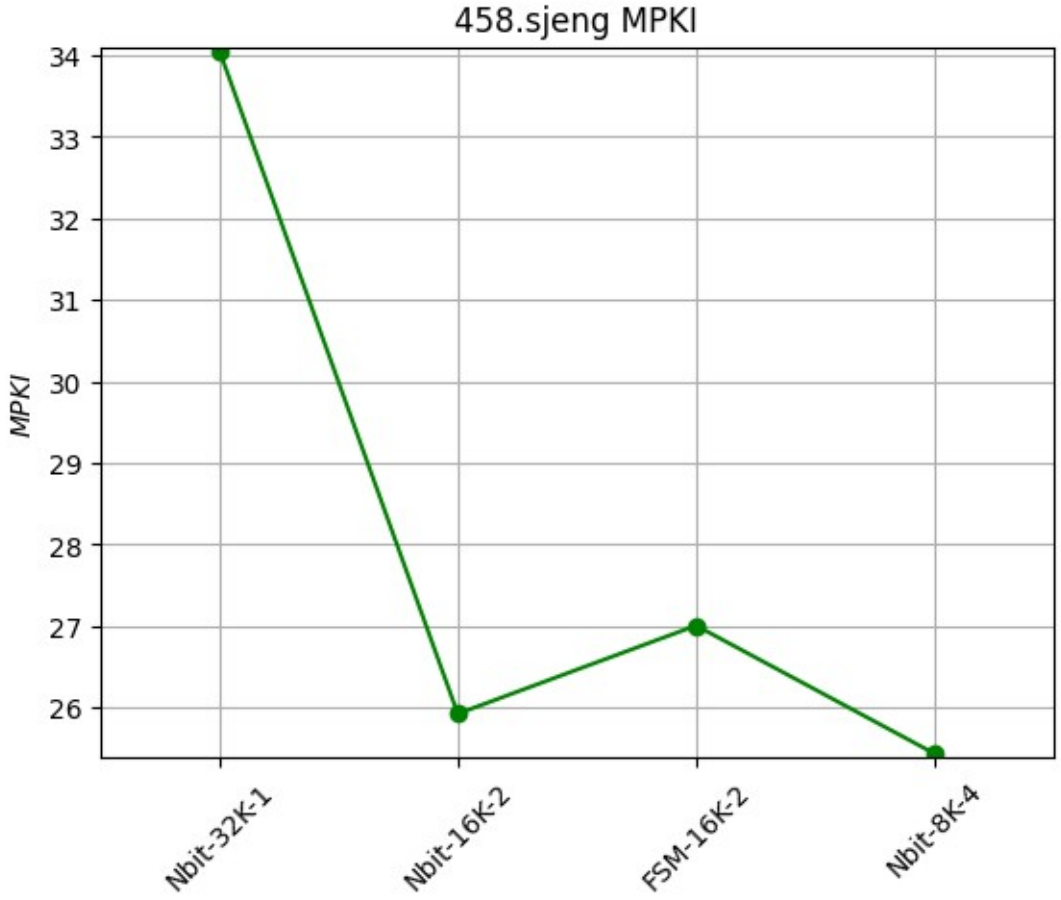
Ομοίως με το προηγούμενο 445.gobmk

456.hmmer



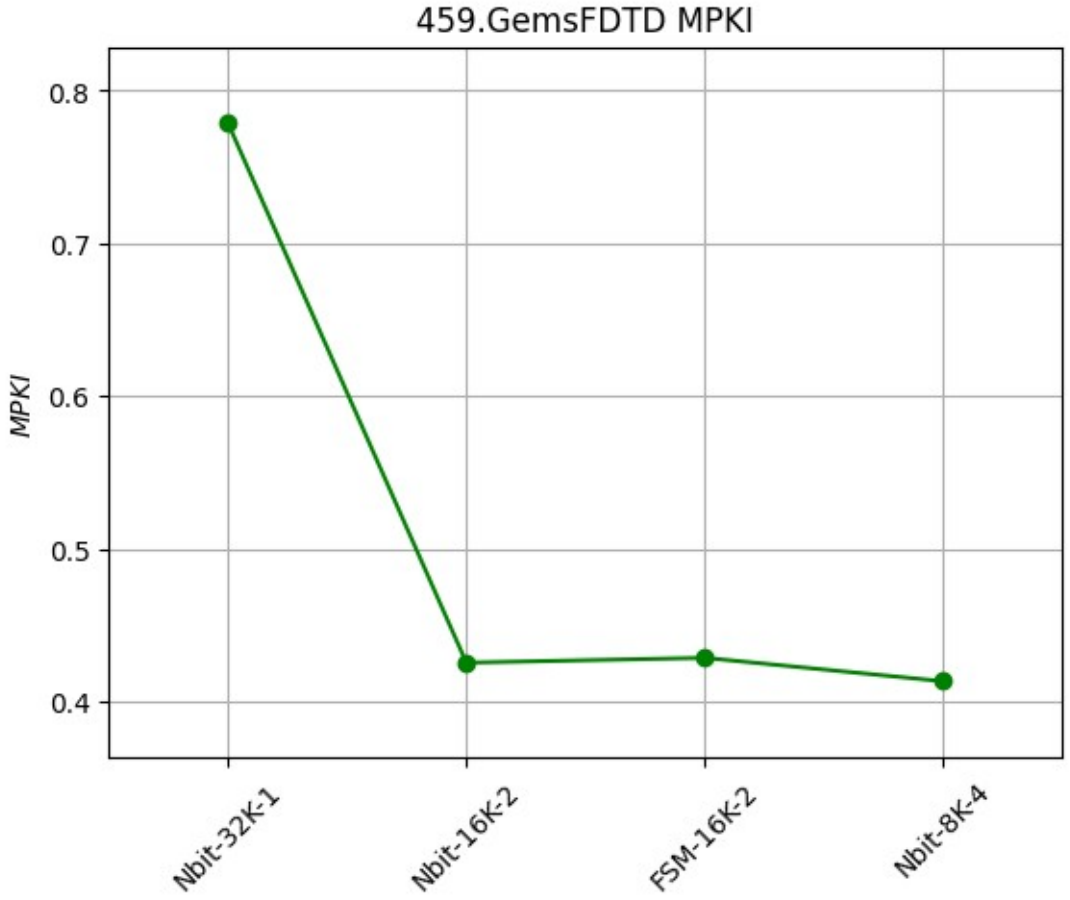
Παρατηρούμε ότι η αρχική αύξηση των bits μειώνει το MPKI . Ο 2-bit έχει την ίδια απόδοση με τον FSM . Η αύξηση πέραν του 2-bit δεν επηρεάζει το MPKI . Μπορούμε να επιλέξουμε οποιονδήποτε από τους 2-bit, FSM και 4-bit .

458.sjeng



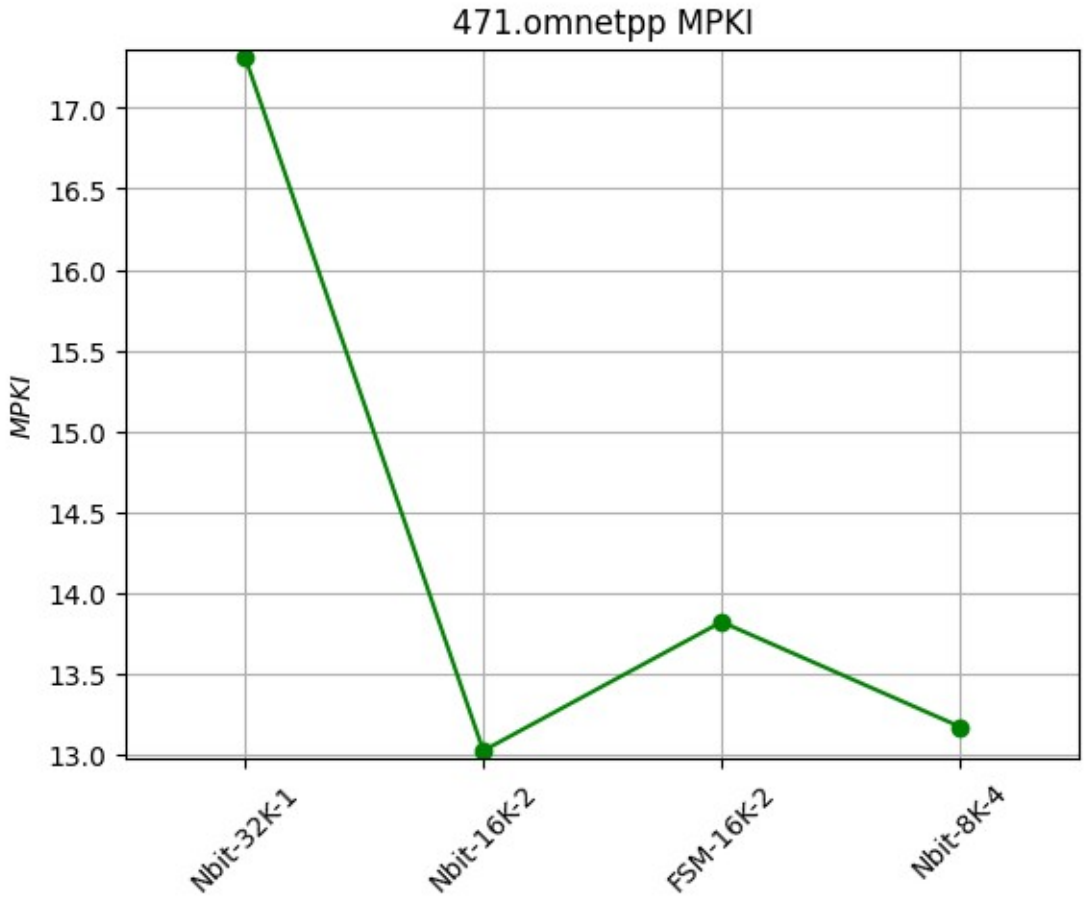
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σχέση με τον 2-bit, έχει ελαφρώς χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Η περαιτέρω αύξηση μειώνει λιγότερο το MPKI (2-bit -> 4-bit) . Η βέλτιστη επιλογή είναι 4-bit .

459.gemsFDTD



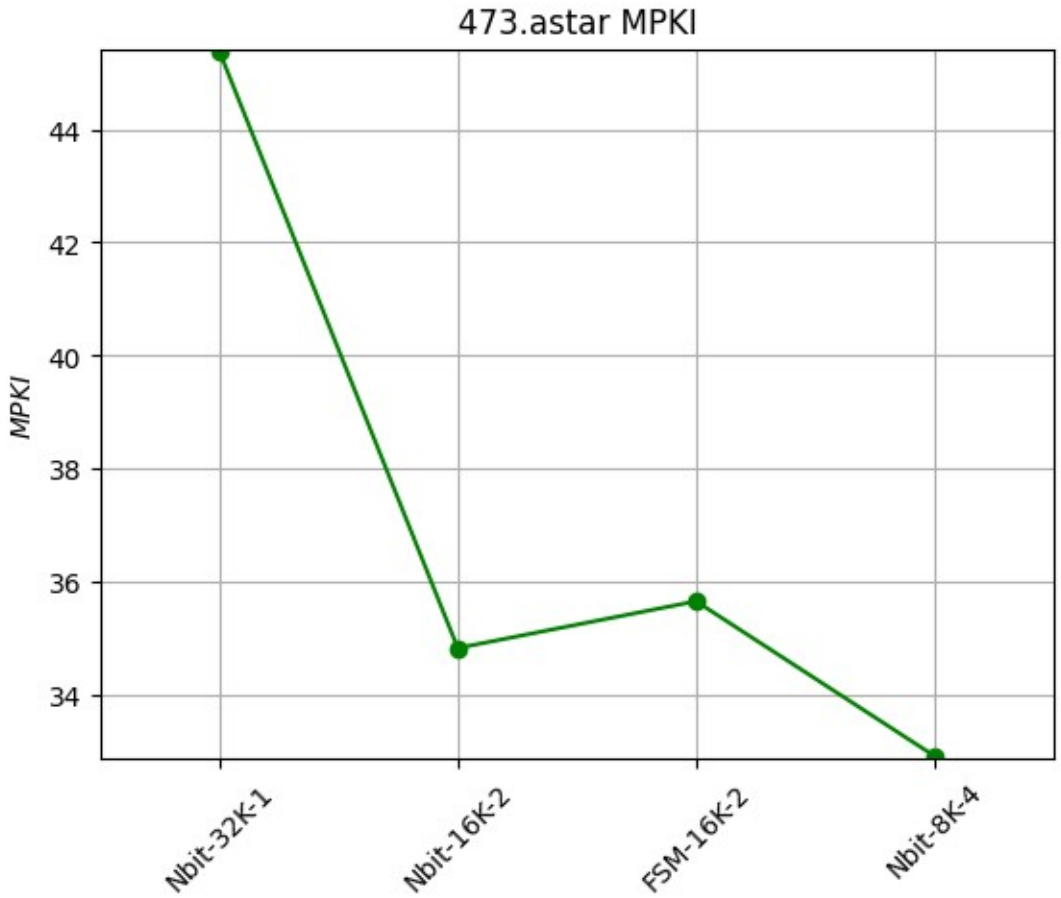
Παρατηρούμε ότι η αρχική αύξηση των bits μειώνει το MPKI . Ο 2-bit έχει την ίδια απόδοση με τον FSM . Η αύξηση πέραν του 2-bit δεν βελτιώνει πρακτικά το MPKI . Μπορούμε να επιλέξουμε οποιονδήποτε από τους 2-bit, FSM και 4-bit .

471.omnetpp



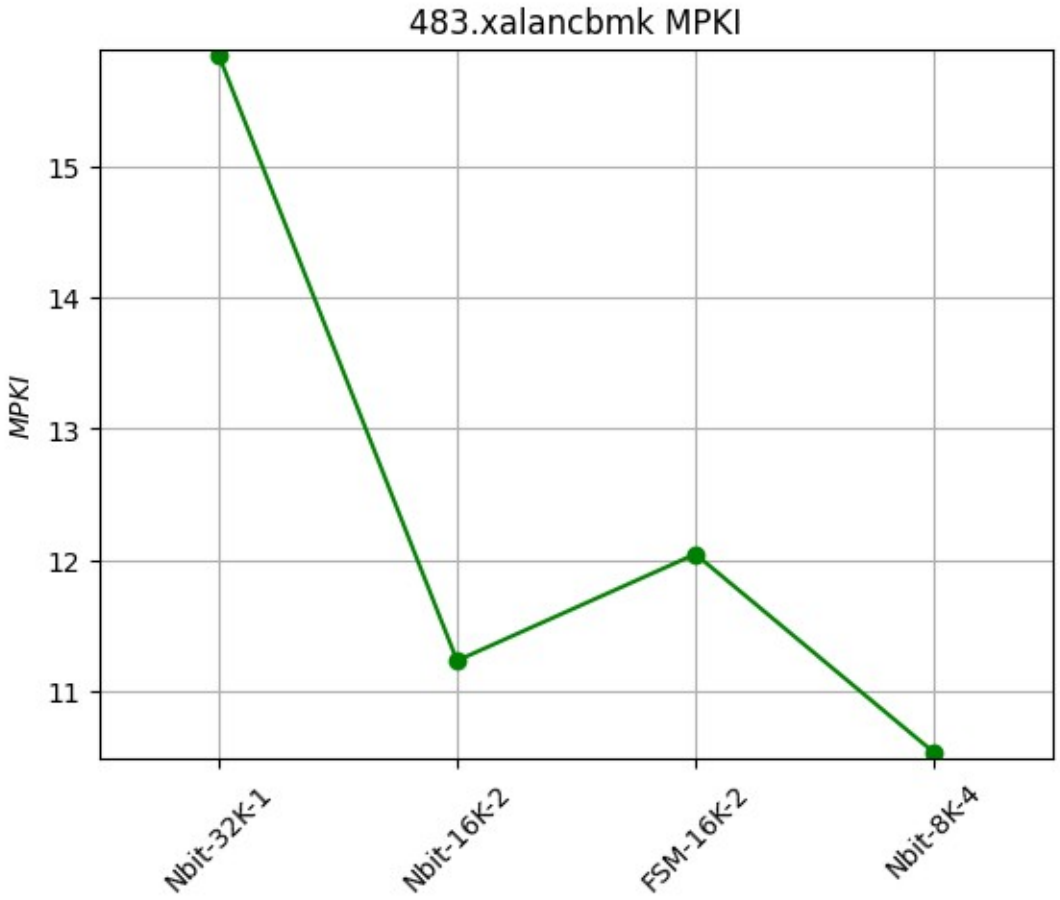
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σχέση με τον 2-bit, έχει ελαφρώς χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Ωστόσο, η αύξηση 2-bit -> 4-bit φαίνεται να αυξάνει ελάχιστα το MPKI . Η βέλτιστη επιλογή είναι 2-bit .

473.astar



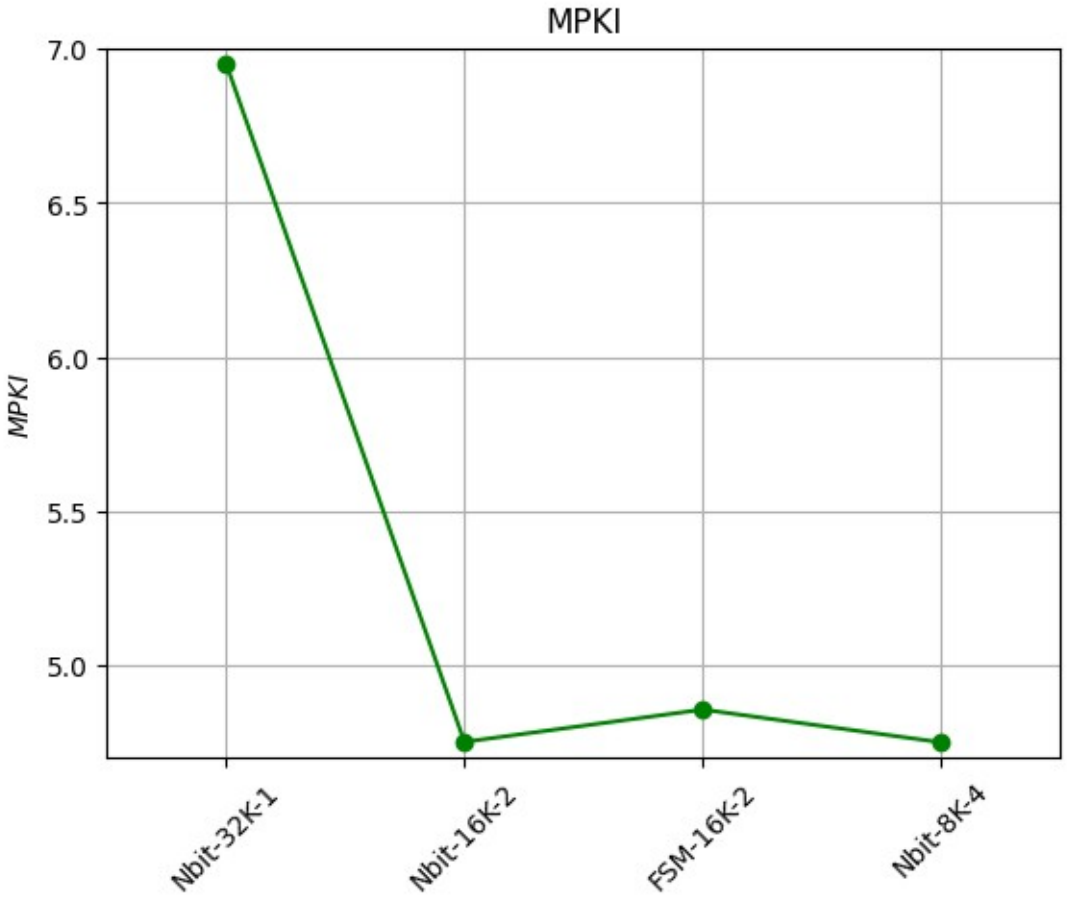
Παρατηρούμε ότι η αύξηση των bits βελτιώνει το MPKI . Ο FSM σε σχέση με τον 2-bit, έχει ελαφρώς χειρότερη απόδοση. Η αρχική αύξηση από 1-bit σε 2-bit είχε ως αποτέλεσμα την μείωση του MPKI . Η περαιτέρω αύξηση βοηθαέι λιγότερο το MPKI (2-bit -> 4-bit) . Η βέλτιστη επιλογή είναι 4-bit .

483.xalancbmk



Ομοίως με το προηγούμενο 473.astar .

Συμπεράσματα



Στο παραπάνω διάγραμμα έχουμε τους γεωμετρικούς μέσους του MPKI για τα αποτελέσματα των benchmarks . Το διάγραμμα συμφωνεί με την ανάλυση που κάναμε στις περισσότερες εφαρμογές. Ειδικότερα, η αύξηση 1-bit -> 2-bit μειώνει το MPKI . Ο FSM έχει ελάχιστα μεγαλύτερο MPKI από τον 2-bit, όμως η διαφορά είναι πρακτικά αμελητέα . Οι 2-bit και 4-bit έχουν την ίδια απόδοση, χρησιμοποιώντας σταθερά 32Kbits ο καθενας, αλλά με 16K και 8K entries αντίστοιχα. Όπως ,είδαμε και στο ερώτημα i) ο 4-bit για 16K entries και 64Kbits hardware πετυχαίνει τα ίδια αποτελέσματα με τον 2-bit .

Άρα, καταλήγουμε στο συμπέρασμα ότι αξιοποιώντας λιγότερο hardware μπορούμε να έχουμε την βέλτιστη απόδοση με τον 2-bit .

- Μελέτη BTB

Στο συγκεκριμένο ερώτημα υλοποιήσαμε ένα BTB και μελετήσαμε την ακρίβεια πρόβλεψης για τις εξής περιπτώσεις :

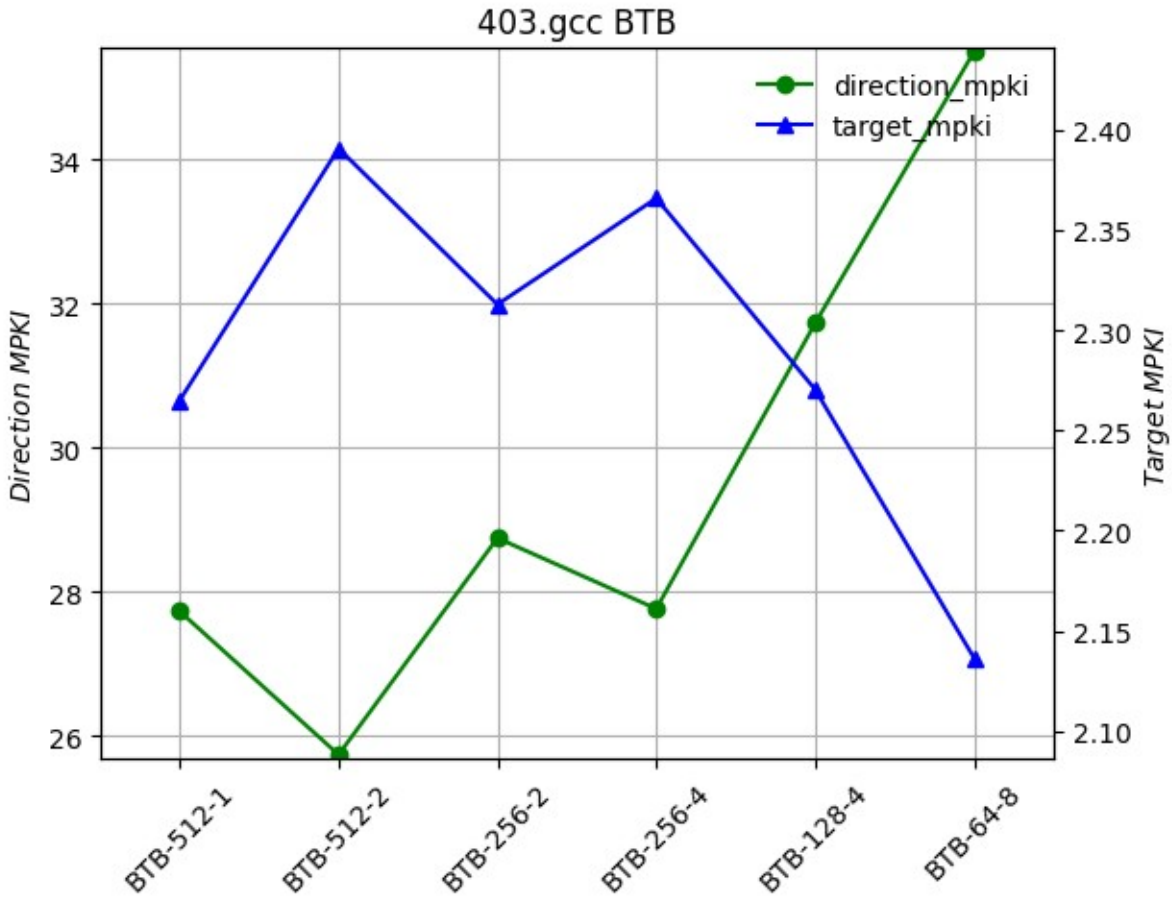
BTB entries	**BTB associativity**
512	1,2
256	2,4
128	4
64	8

Στον πίνακα BTB εισάγαμε μόνο τα branches που είναι Taken . Χρησιμοποιήσαμε τον BTB ως προβλεπτή και άμα βρούμε entry το οποίο περιέχει την διεύθυνση του branch, κάναμε πρόβλεψη Taken, ενω αντίθετα Not-Taken . Στην περίπτωση που χρησιμοποιούμε τον BTB ως προβλεπτή διακλάδωσης και διεύθυνση προορισμού, έχουμε δυο κατηγορίες misses : τα direction misprediction και target misprediction στην περίπτωση direction hit . Σαν πολιτική αντικατάστασης στις set-associative υλοποιήσεις ορίσαμε FIFO .

Τα direction misprediction είναι όταν έχουμε κάνει λάθος πρόβλεψη για το branch. Τα target misprediction συμβαίνουν όταν έχουμε προβλέψει Taken και ηταν όντως Taken ,αλλά έχουμε λάθος διεύθυνση προορισμού .

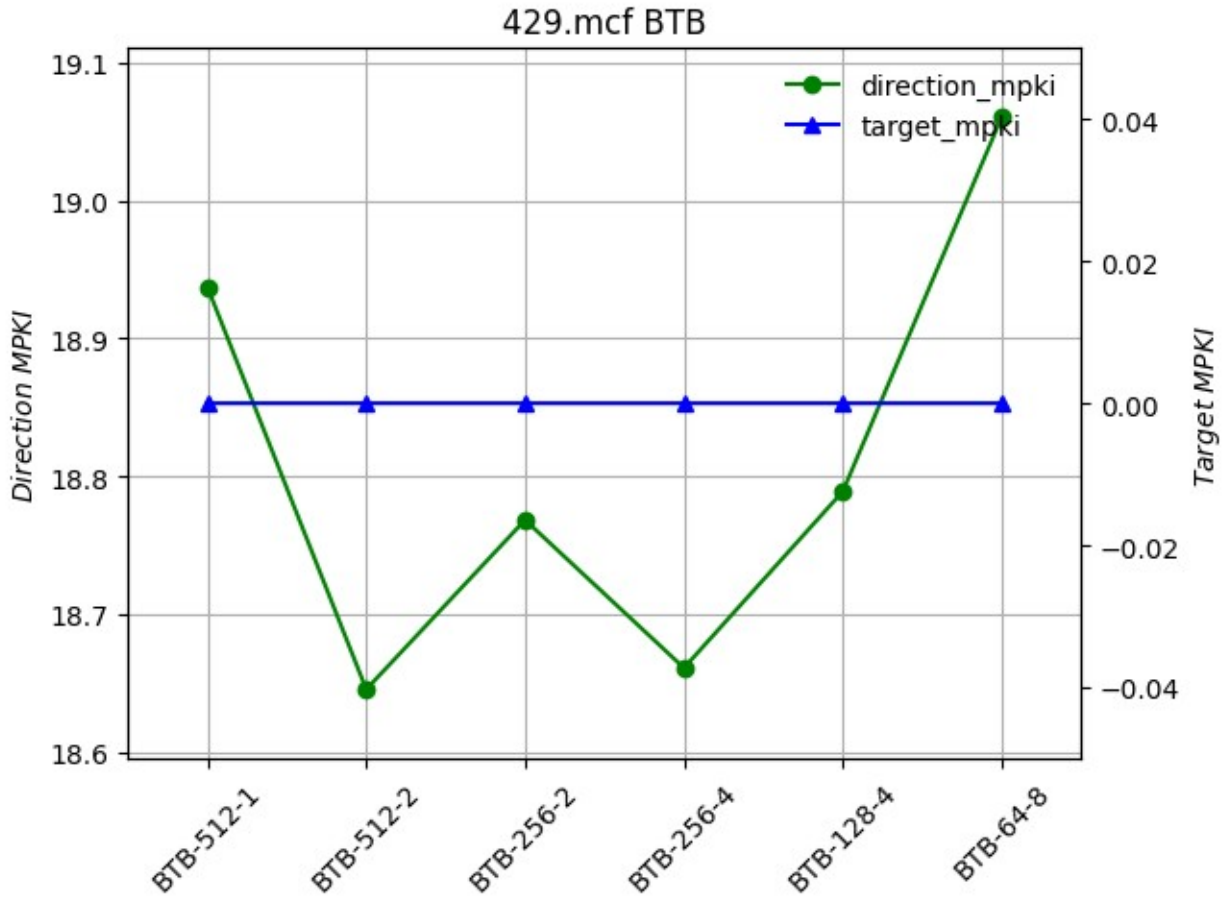
Για την ανάλυση των αποτελεσμάτων μας έχουμε χρησιμοποιήσει τις μετρικές direction_mpki και target_mpki . Ακολουθούν διαγράμματα για κάθε benchmark .

403.gcc



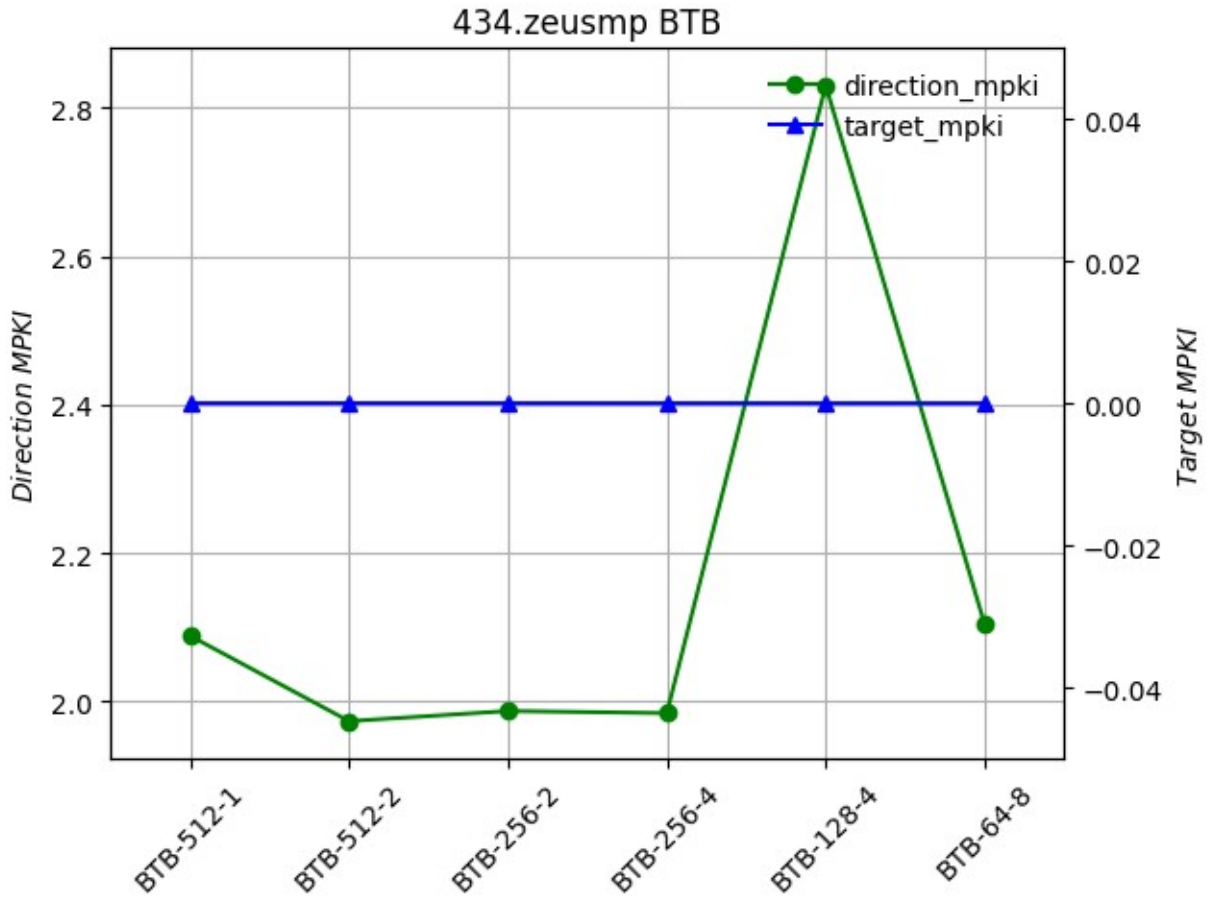
Παρατηρούμε ότι η αύξηση του associtivity μειώνει το direction-MPKI, καθώς έχουμε λιγότερα conflict misses : 512-1 -> 512-2 και 256-2 -> 256-4 . Η μείωση του μεγέθους προκαλεί αύξηση του direction-MPKI, αφού έχουμε έχουμε περισσότερα capacity misses :512-2 -> 256-2 και 256-4 ->128-4. Ακόμα φαίνεται το direction και target MPKI να έχουν αντιστρόφως ανάλογη συμπεριφορά, καθώς στην περίπτωση του direction misprediction δεν λαμβάνουμε υπόψιν το target misprediction, οπότε λογικό είναι να αυξάνεται . Η βέλτιστη υλοποίηση από πλευράς direction prediction είναι η 512-2, ενώ από tagret είναι 64-8 .

429.mcf



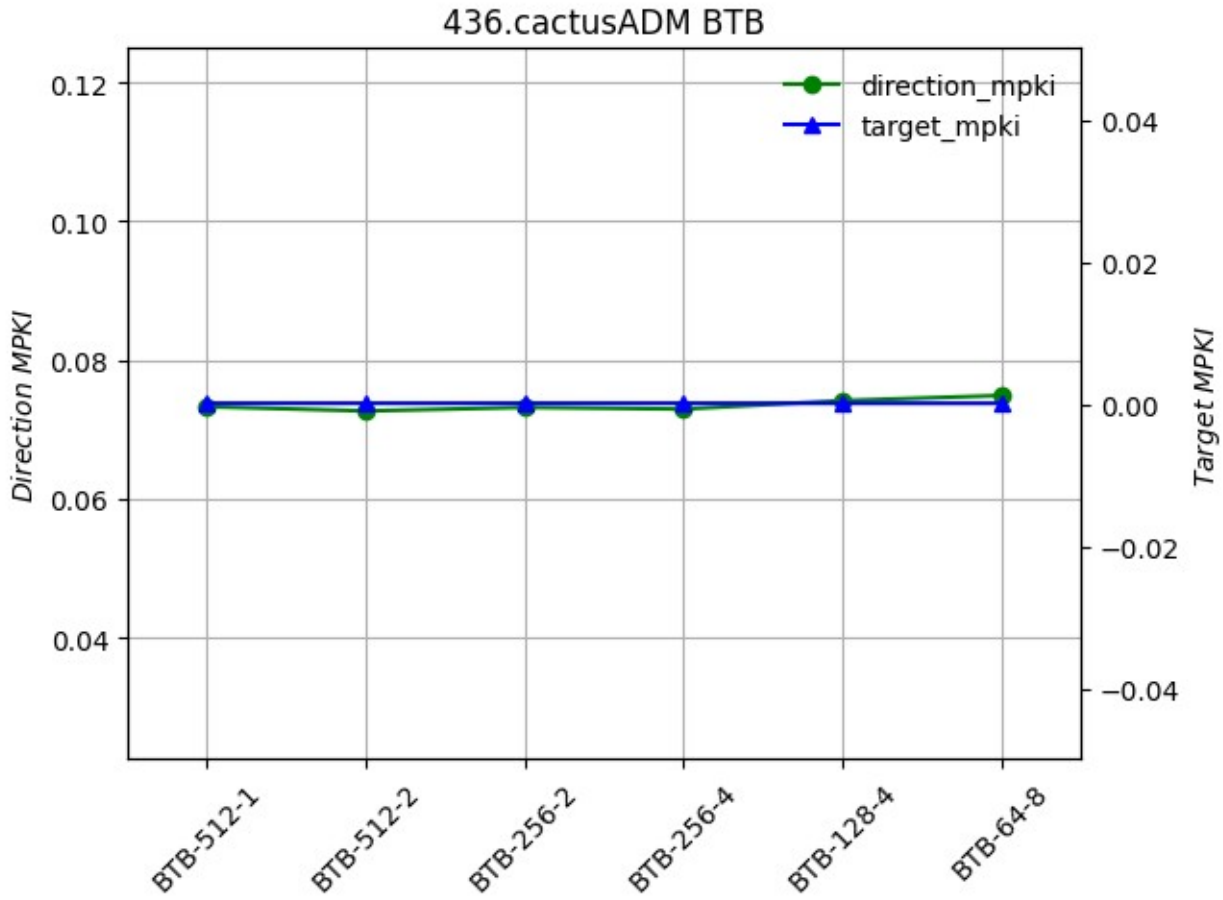
Παρατηρούμε ότι το driection misprediction παρουσιάζει την ίδια συμπεριφορά με το προηγούμενο benchmark 403.gcc . Το taget misprediction παραμένει σταθερά μηδέν . Οι τιμές 512-2 και 256-4 έχουν την ίδια απόδοση, οπότε για εξοικονόμηση hardware επιλέγουμε την υλοποίηση 256-4 . Ομοίως ,από πλευράς traget prediction, θα επιλέξουμε 64-8 .

434.zeusmp



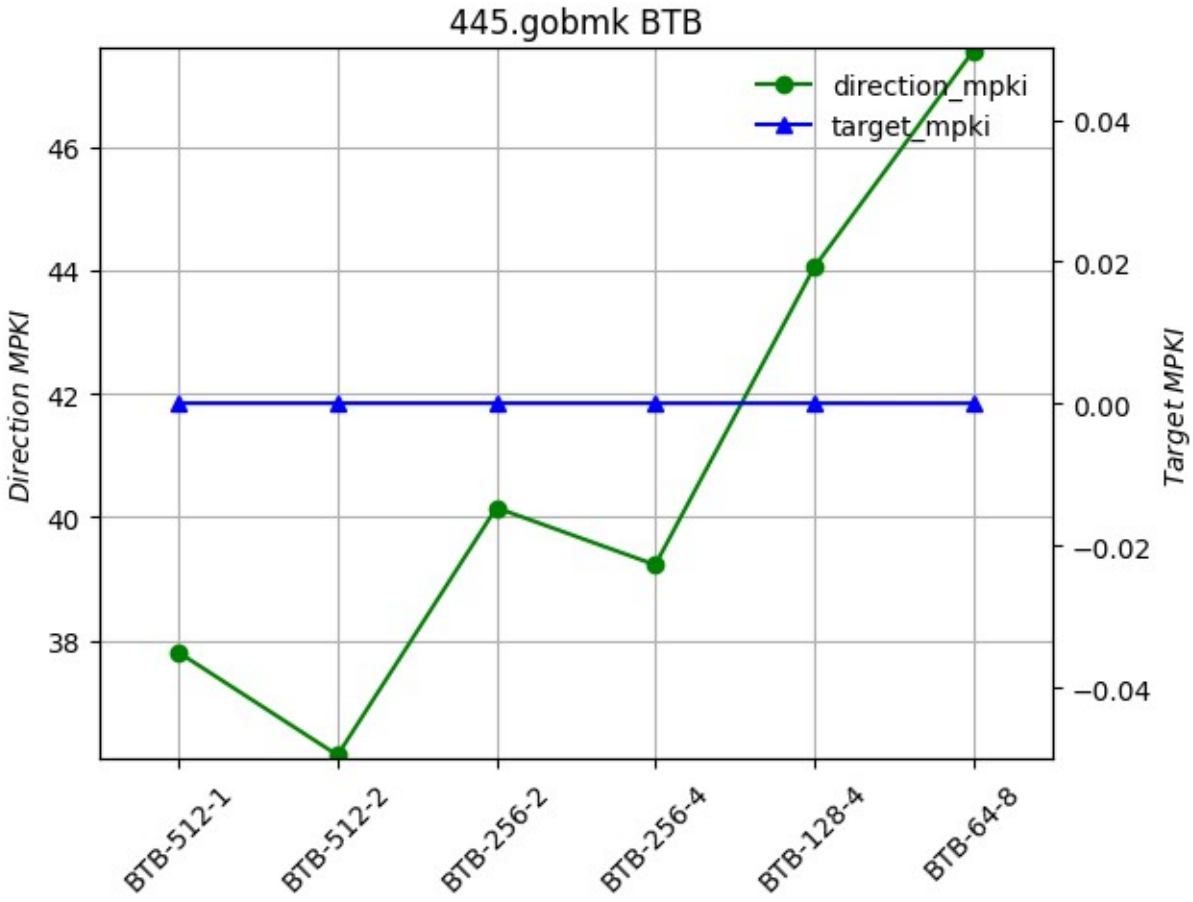
Παρατηρούμε ότι η αρχική αύξηση του associativity (512-1 -> 512-2) βελτιώνει ελάχιστα το driection MKPI , ενώ περαιτέρω αύξηση (256-2 -> 256-4) το διατηρεί σταθερό . Αντιθέτως, η αρχική μείωση του μεγέθους (512-2 -> 256-2) δεν επηρεάζει το MKPI, ενώ μετά (256-4 -> 128-4) έχουμε αύξηση . Το tagret MPKI παραμένει σταθερά μηδέν. Η βέλτιστη επιλογή για το direction prediction είναι μια από τις 512-2, 256-2 και 256-4 , όμως επειδή τα αποτελέσματα για 64-8 διαφέρουν ελάχιστα και το MKPI έχει μικρή τιμή , μπορούμε για εξοικονόμηση υλικού να επιλέξουμε τον 64-8 . Οπότε για πρώτη φορά η επιλογή για direction και target prediction συμβαδίζει .

436.cactusADM



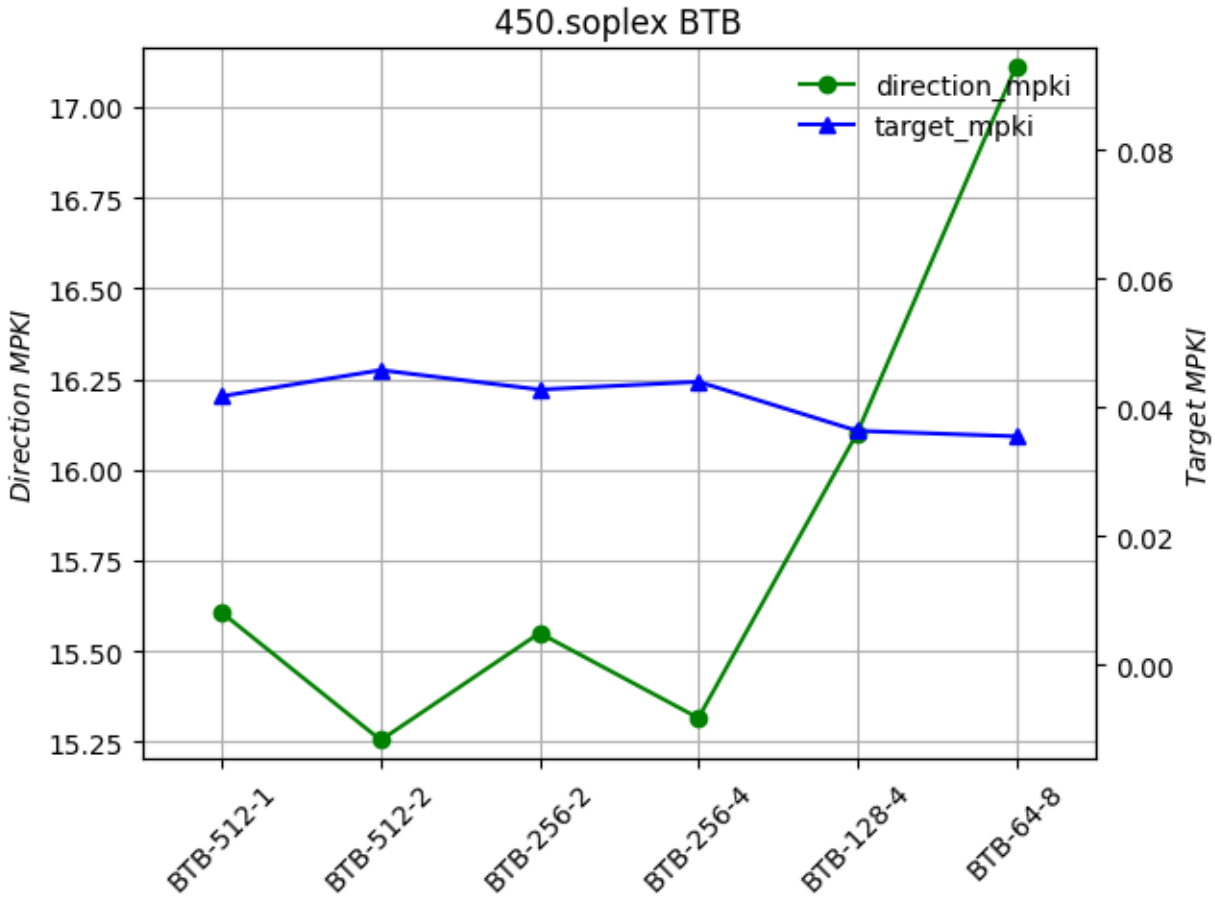
Παρατηρούμε ότι οι μετρικές direction και target MPKI παρεμένουν σταθερές . Οπότε για εξοικονόμηση υλικού επιλέγουμε τον 64-8 .

445gobmk



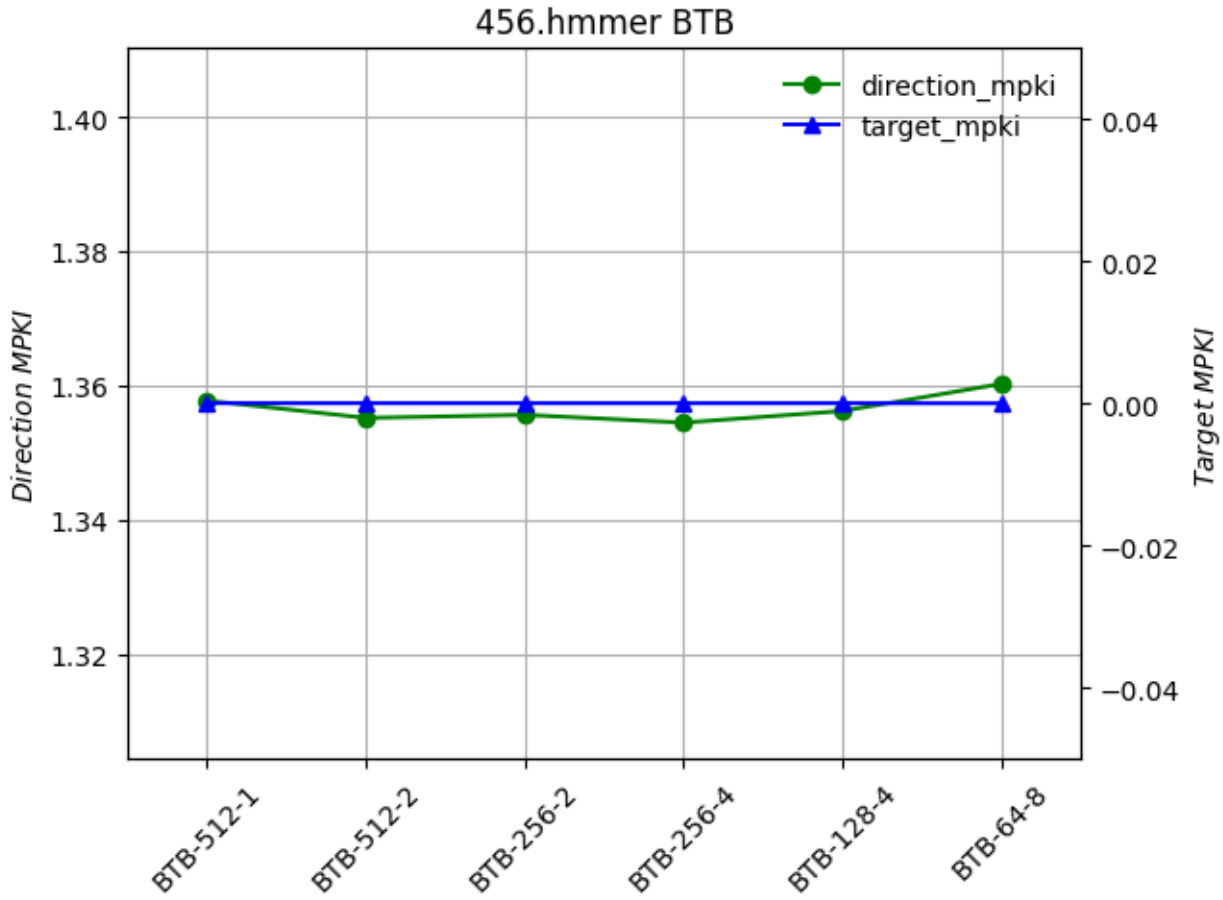
Παρατηρούμε ότι το driection MPKI εχει την ίδια συμπεριφορά με το benchmark 403.gcc . Το target MPKI παραμένει σταθερά μηδέν . Η βέλτιστη υλοποίηση από πλευράς direction prediction είναι η 512-2, ενώ από tagret predicton είναι 64-8 .

450.soplex



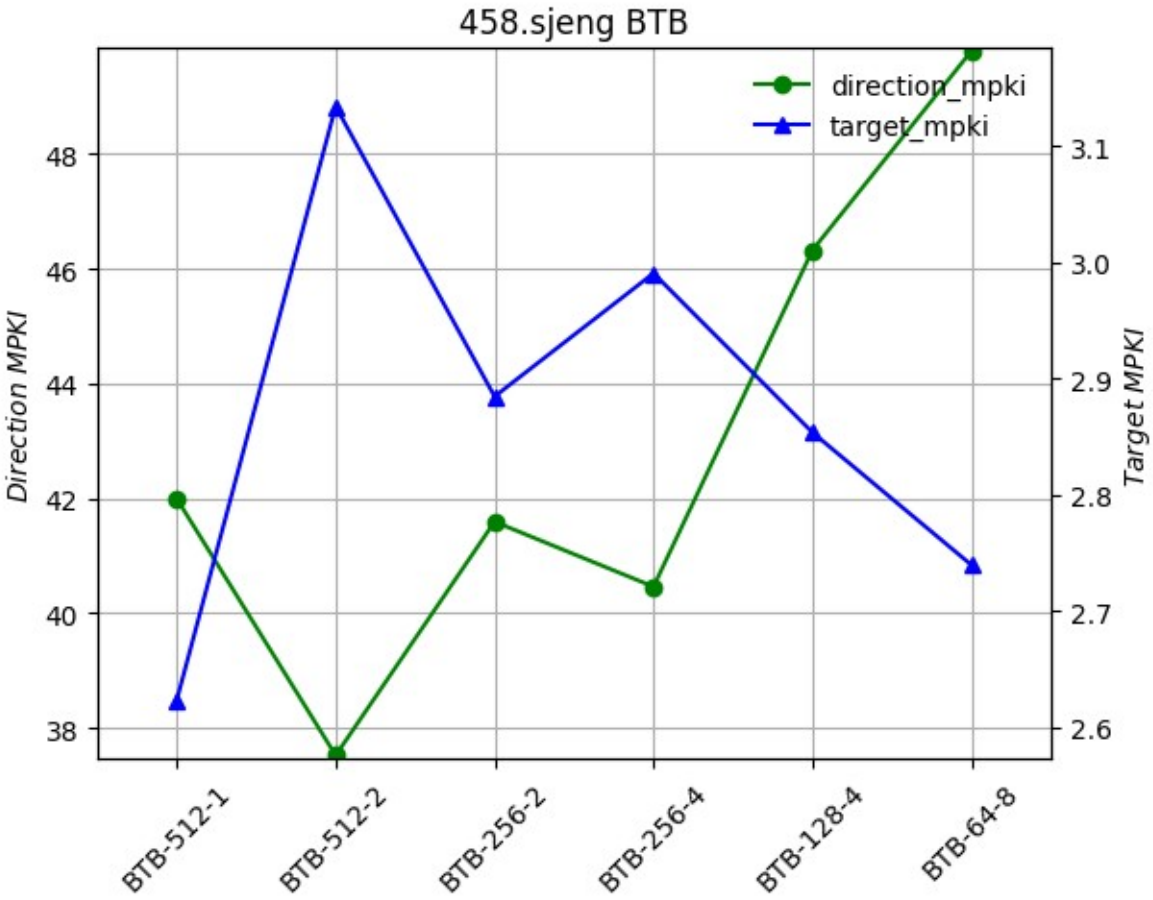
Παρατηρούμε ότι το driection MPKI εχει την ίδια συμπεριφορά με το benchmark 403.gcc . Το target misprediction φαίνεται να παρουσιάζει μια μικρή ταλάντωση, όμως λόγω της πολύ μικρής τιμής του θεωρείται αμελητέα . Η βέλτιστη υλοποίηση από πλευράς direction prediction είναι η 512-2, ενώ από tagret predicton είναι 64-8 .

456.hmmer



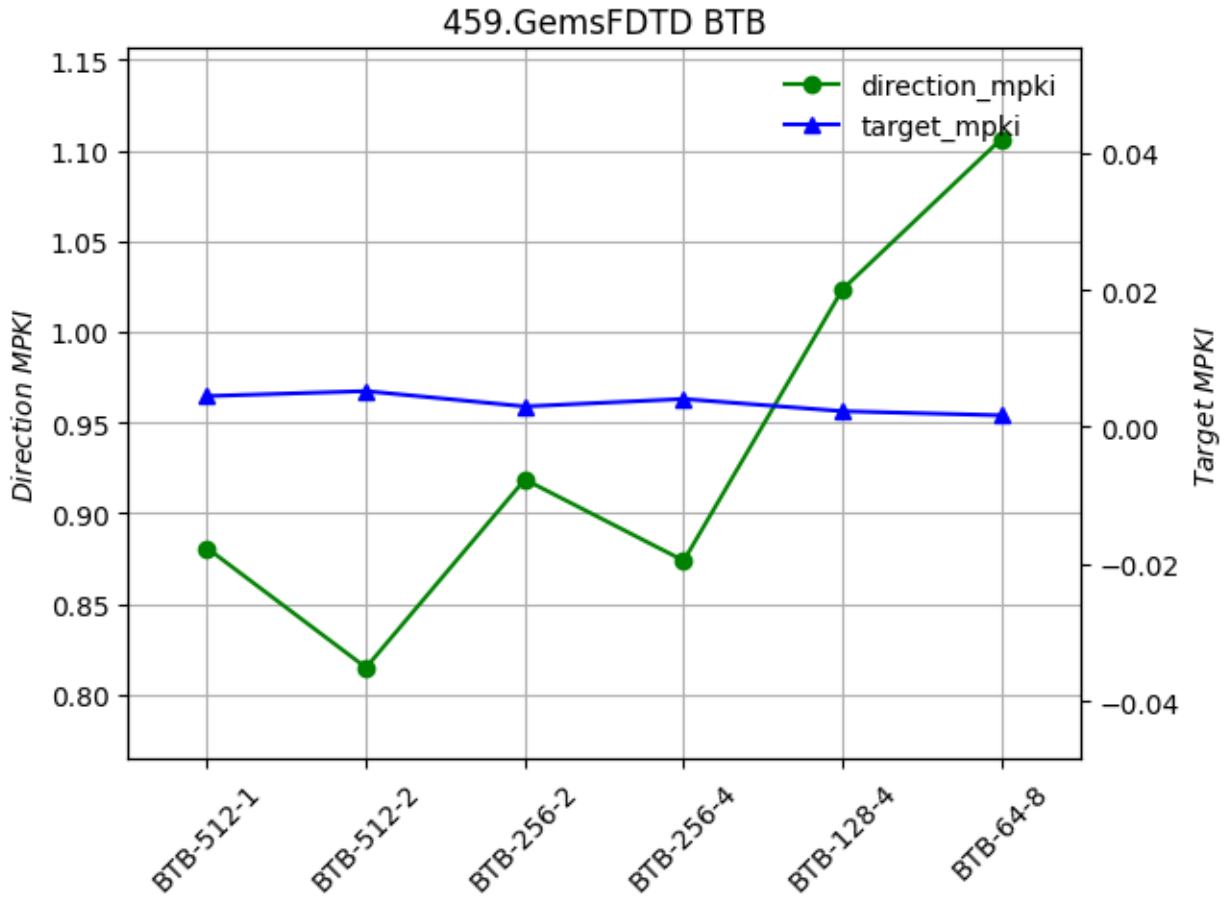
Παρατηρούμε ότι οι μετρικές direction και target MPKI παρεμένουν σταθερές . Οπότε για εξοικονόμηση υλικού επιλέγουμε τον 64-8 .

458.sjeng



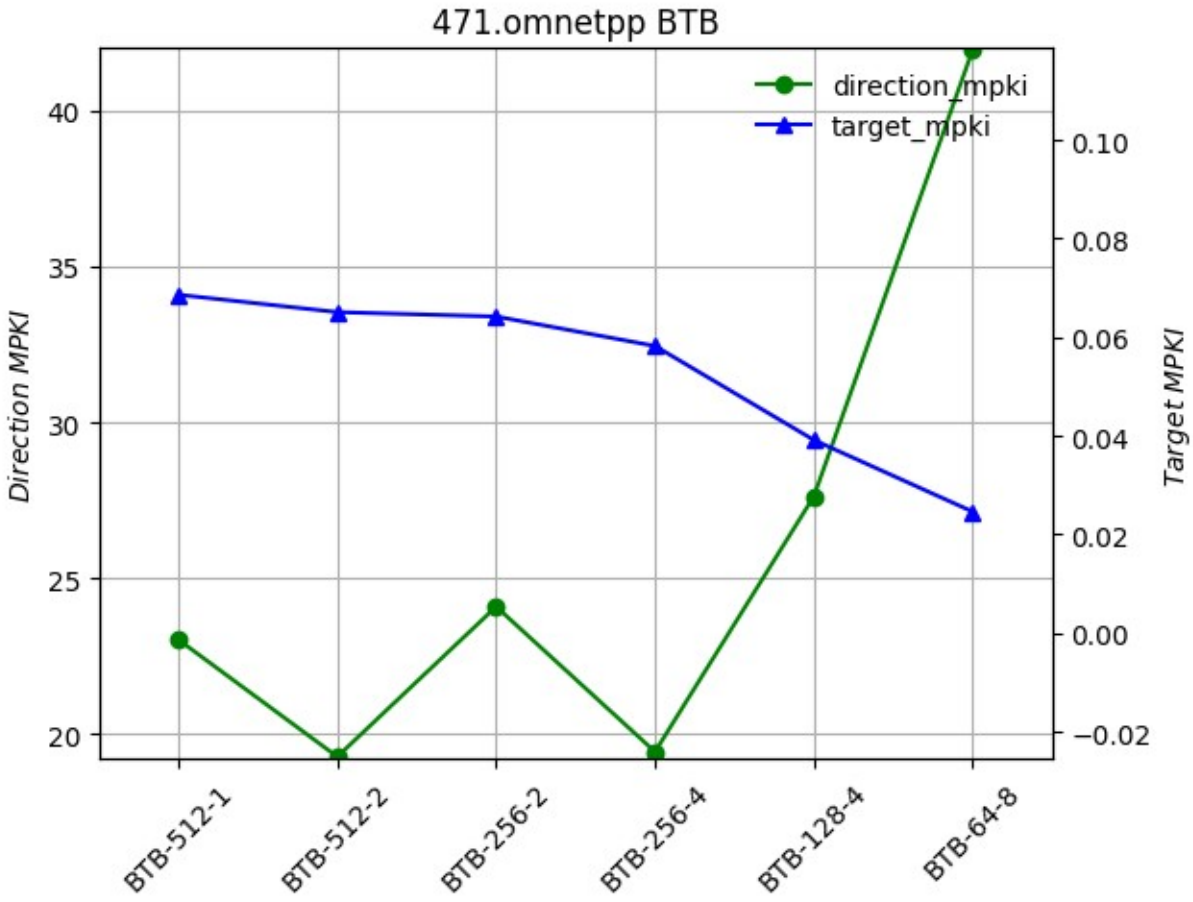
Ίδια ανάλυση με το benchmark 403.gcc .

459.GemsFDTD



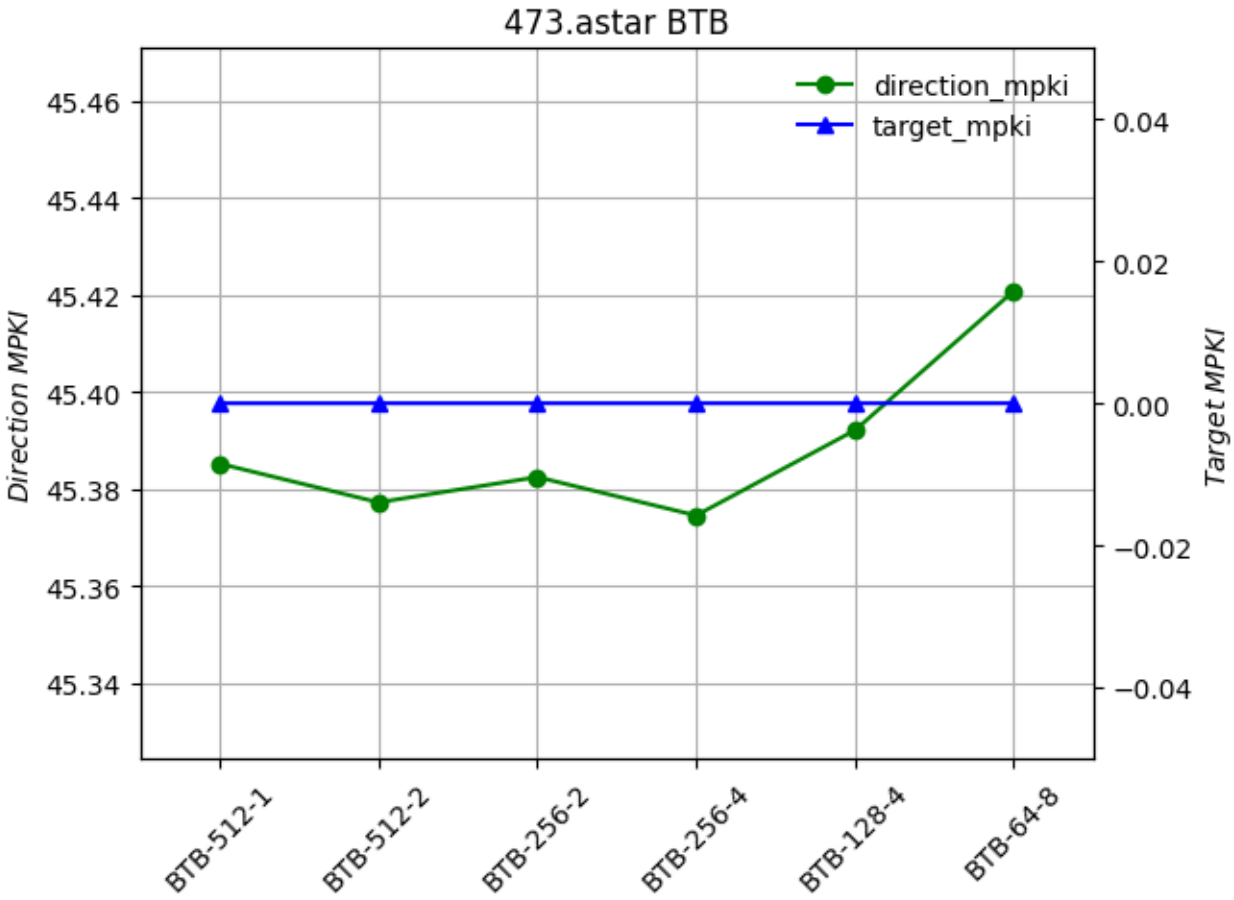
Παρατηρούμε ότι το driection MPKI εχει την ίδια συμπεριφορά με το benchmark 403.gcc . Το target MPKI παραμένει σταθερά μηδέν . Η βέλτιστη υλοποίηση από πλευράς direction prediction είναι η 512-2, ενώ από tagret predicton είναι 64-8 .

471.omnetpp



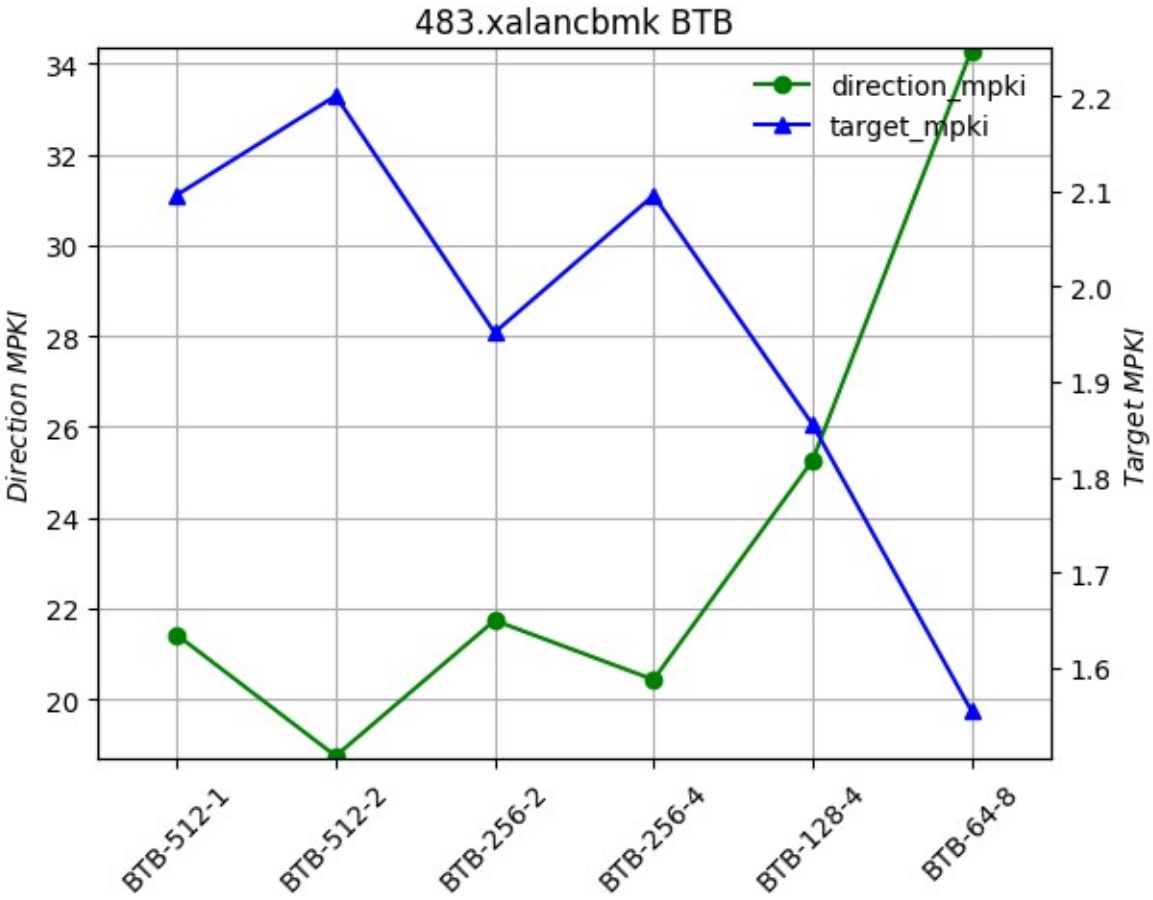
Παρατηρούμε ότι το driection MPKI εχει την ίδια συμπεριφορά με το benchmark 403.gcc . Το target MPKI μειώνεται σταθερά . Την βέλτιστη απόδοση από πλευράς direction prediction την έχουμε για 512-2 και 256-4, όμως για εξοικονόμηση υλικού θα επιλέξουμε την 256-4 . Για το tatget prediction θα διαλέξουμε την 64-8 .

473.astar



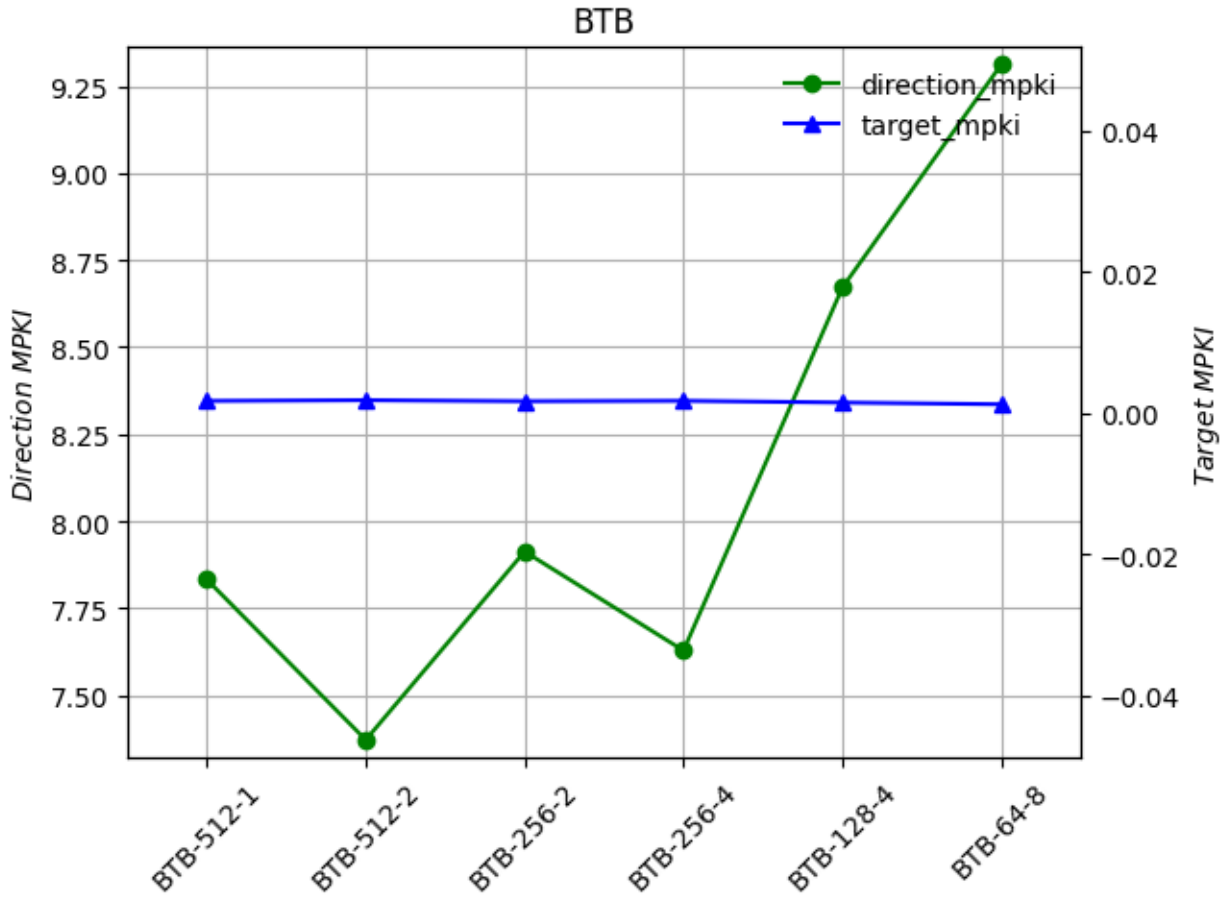
Παρατηρούμε ότι το direction MPKI παρουσιάζει μικρές αυξομοιώσεις, παρόλο αυτά επειδή οι διαφορές είναι πρακτικά αμελητέες , μπορούμε να θεωρήσουμε το MPKI πρακτικά σταθερο . Το target MPKI παραμένει σταθερά μηδέν . Μπορούμε για εξοικονόμηση υλικού να επιλέξουμε την υλοποίηση 64-8, χωρίς επίπτωση στην απόδοση .

483.xalacbmk



Ίδια ανάλυση με το benchmark 403.gcc .

Συμπεράσματα



Στο παραπάνω διάγραμμα έχουμε τους γεωμετρικούς μέσους των direction και target MPKI για τα αποτελέσματα των benchmarks . Για το direction MPKI παρατηρούμε την ίδια συμπεριφορά που συναντήσαμε στις περισσότερες εφαρμογές. Πιο συγκεκριμένα η αύξηση του associativity μειώνει το MPKI , ενώ η μείωση των entries προκαλεί την αύξηση του . Τα target mispredictions παραμένουν σταθερά στο μηδέν . Από την περίπτωση 128-4 -> 64-8 ,όπου έχουμε ταυτόχρονα υποδιπλασιασμό των entries, διπλασιασμό του associativtiy και το direction MPKI αυξάνεται, μπορούμε συμπεράνουμε, ότι το πλήθος των entries παίζει σημαντικότερο ρόλο στην απόδοση του BTB ως προβλεπτή διακλάδωσης .

Ο BTB προβλέπει με εξαιρετική ακρίβεια την διεύθυνση προορισμού , όμως σαν προβλεπτής διακλάδωσης έχει χαμηλότερη απόδοση από αυτούς που έχουμε συναντήσει μέχρι τώρα (πχ N-bit ,FSM) . Οπότε, θα ήταν ιδανικό να τον συνδυάσουμε με κάποιον άλλο προβλεπτή. Όμως αν θέλουμε να τον υλοποιήσουμε μόνο του, πιθανώς για εξοικονόμηση υλικού, η βέλτιστη επιλογή είναι ο 512-2 .

- **Μελέτη του RAS**

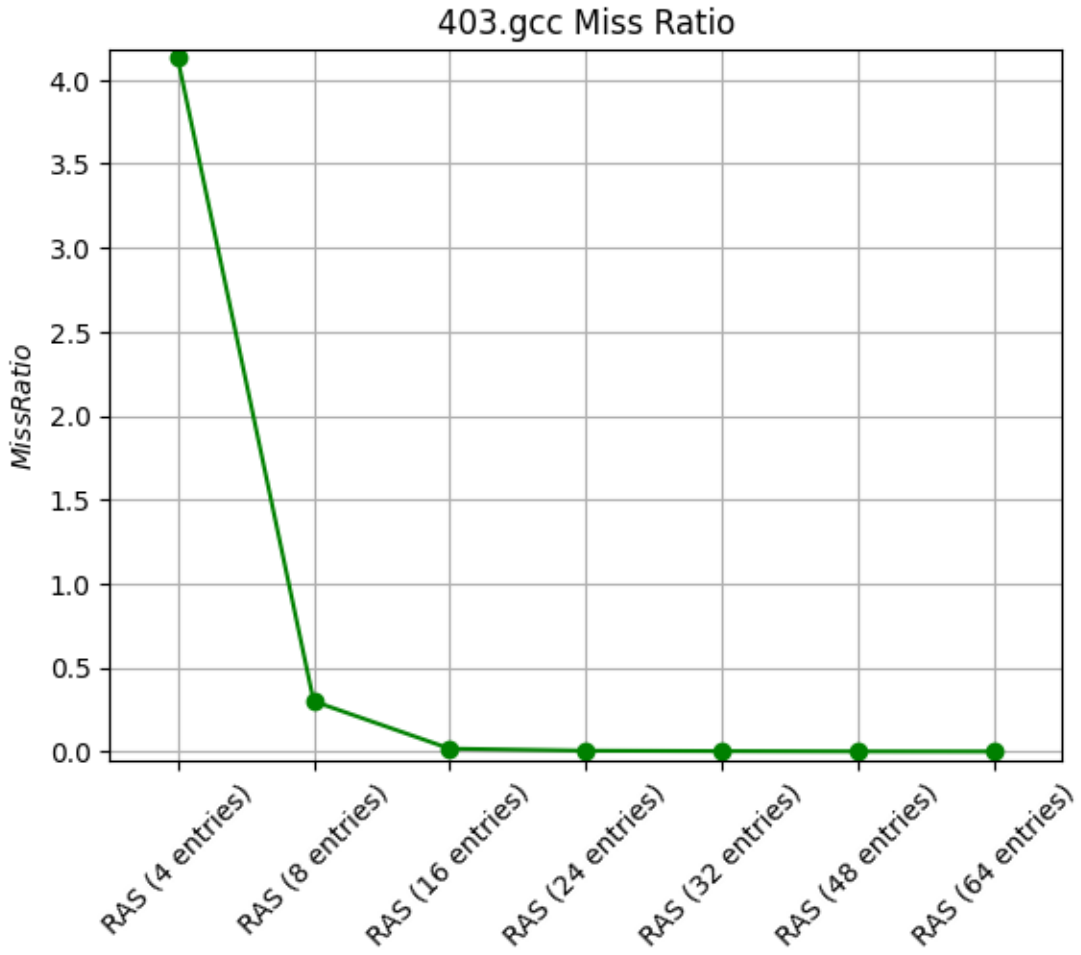
Στο συγκεκριμένο ερώτημα υλοποιήσαμε έναν RAS και καταγράψαμε τα ποσοστά αστοχίας για τις εξής περιπτώσεις :

Αριθμός εγγραφών
4
8
16
24
32
48
64

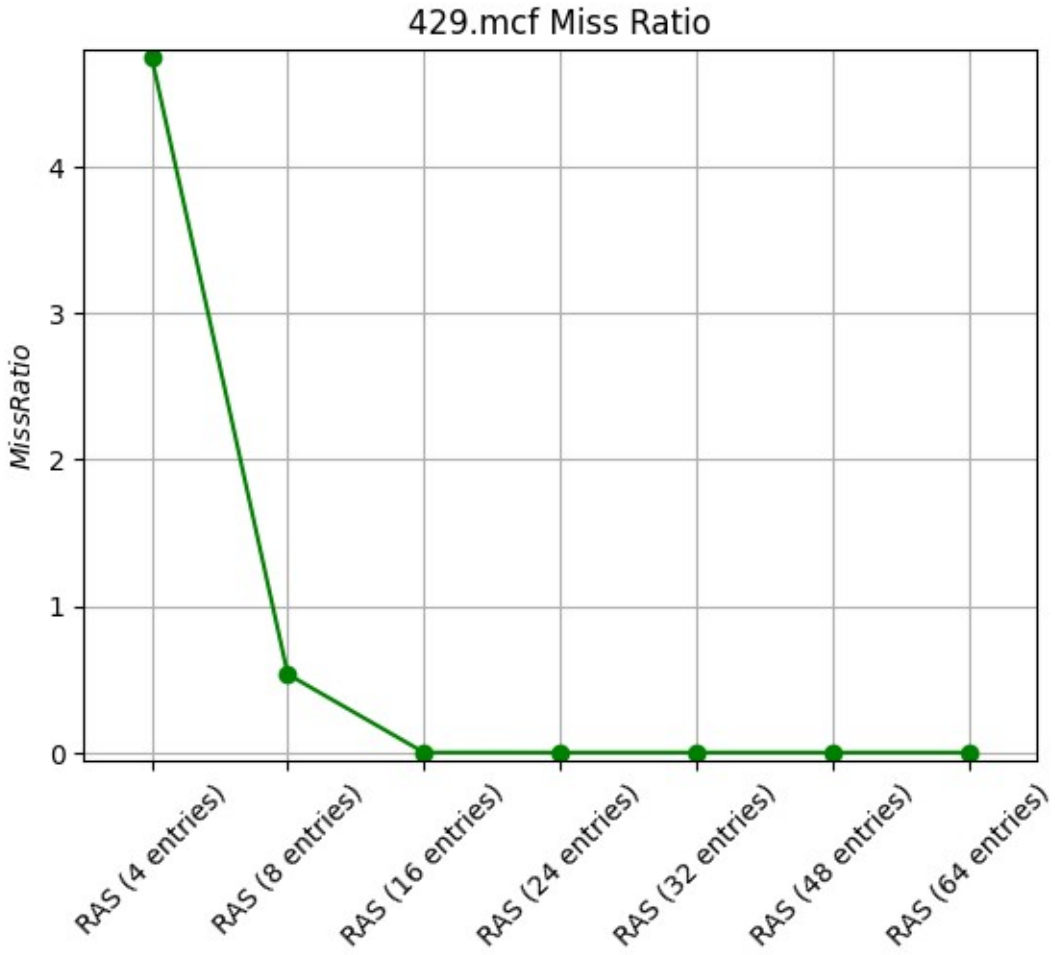
Ο προορισμός επιστροφής ενός καλέσματος συνάρτησης είναι πάντα η επόμενη διεύθυνση της επόμενης της εντολής call. Θα χρησιμοποιήσουμε τον RAS, οποίος έχει δομή LIFO, για να κάνουμε push την διεύθυνση επιστροφής της κλήσης call . Οπότε, στην κορυφή της στοίβας βρίσκεται πάντα η return address της τελευταίας κλίσης call .

Θα χρησιμοποιήσουμε ως μετρική το Miss Ratio (%) όπου έχουμε λάθος διεύθυνση επιστροφής. Ακολουθούν διαγράμματα για κάθε benchmark .

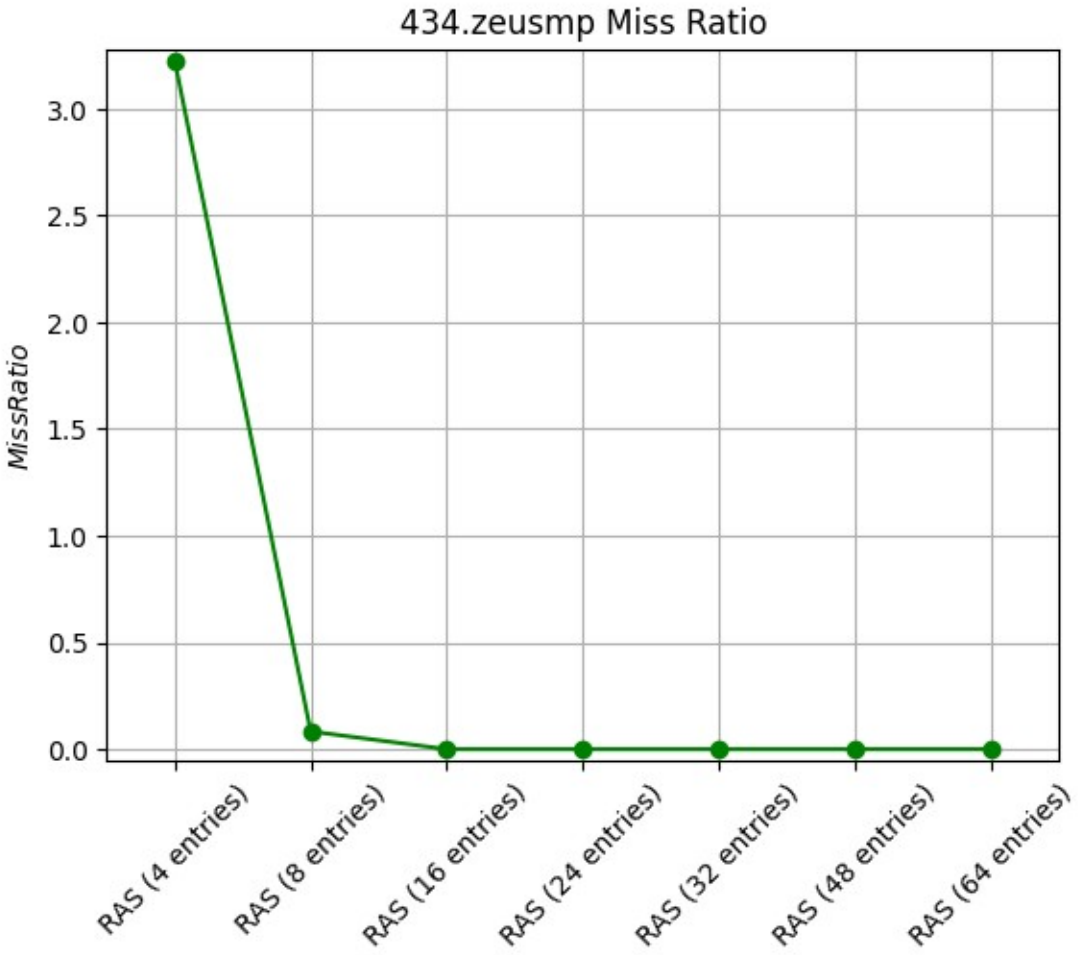
403.gcc


403.gcc Miss Ratio
MissRatio
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

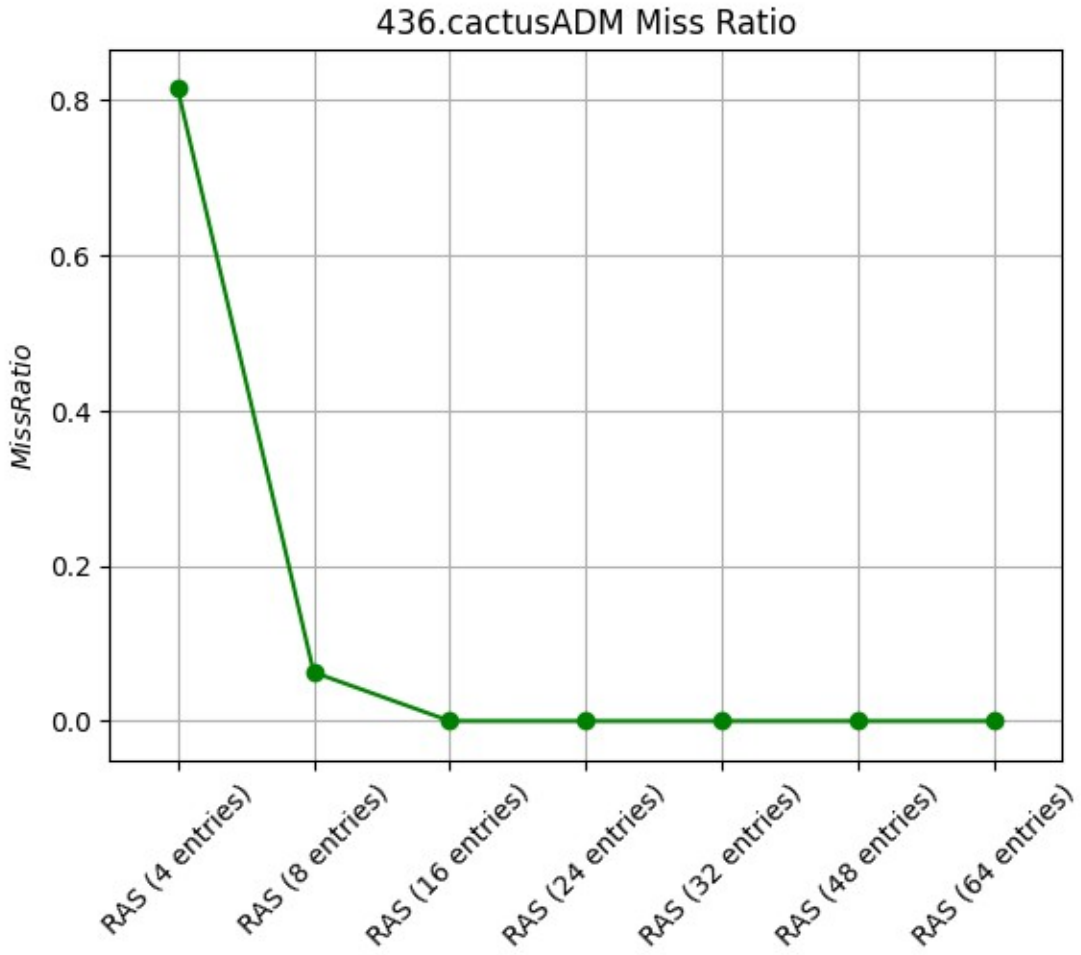
429.mcf


429.mcf Miss Ratio
MissRatio
0
1
2
3
4
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

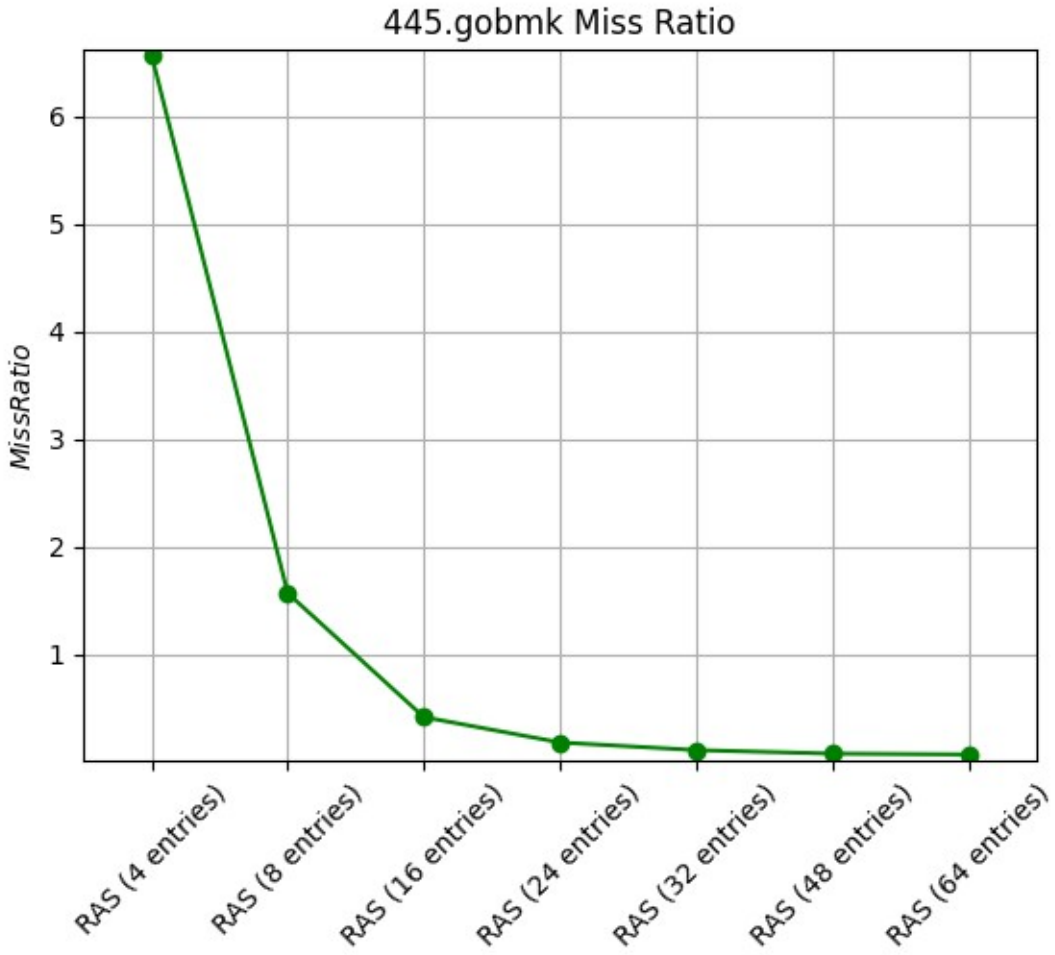
434.zeusmp


434.zeusmp Miss Ratio
MissRatio
3.0
2.5
2.0
1.5
1.0
0.5
0.0
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

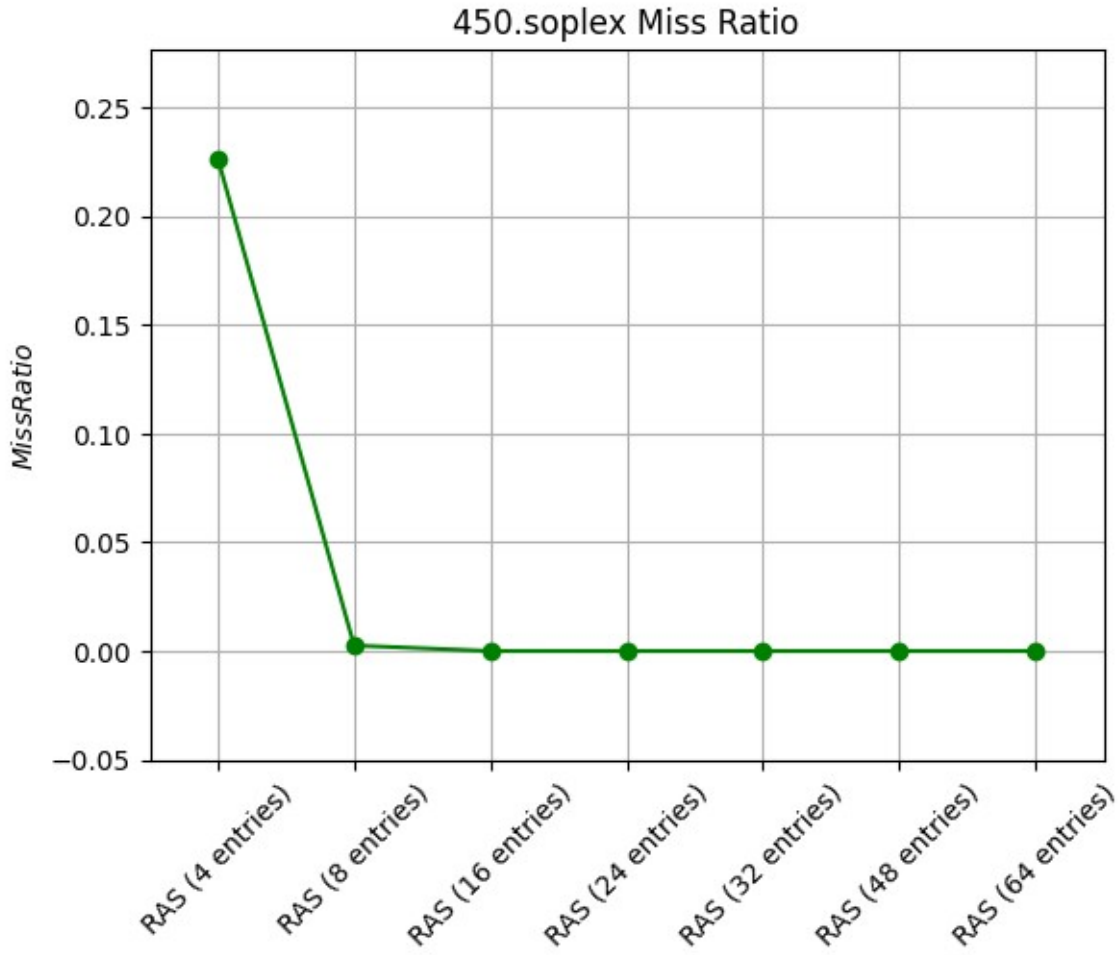
436.cactusADM



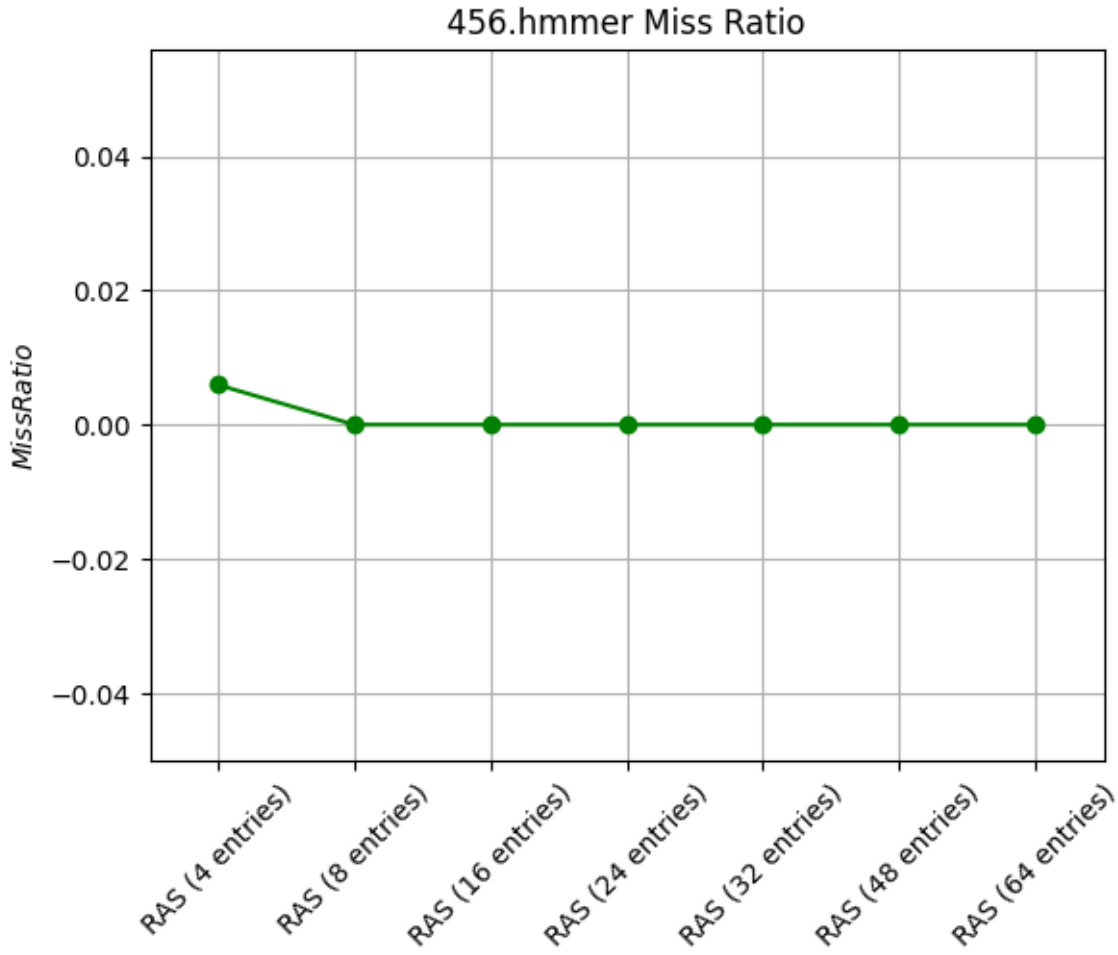
445.gobmk


445.gobmk Miss Ratio
MissRatio
6
5
4
3
2
1
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

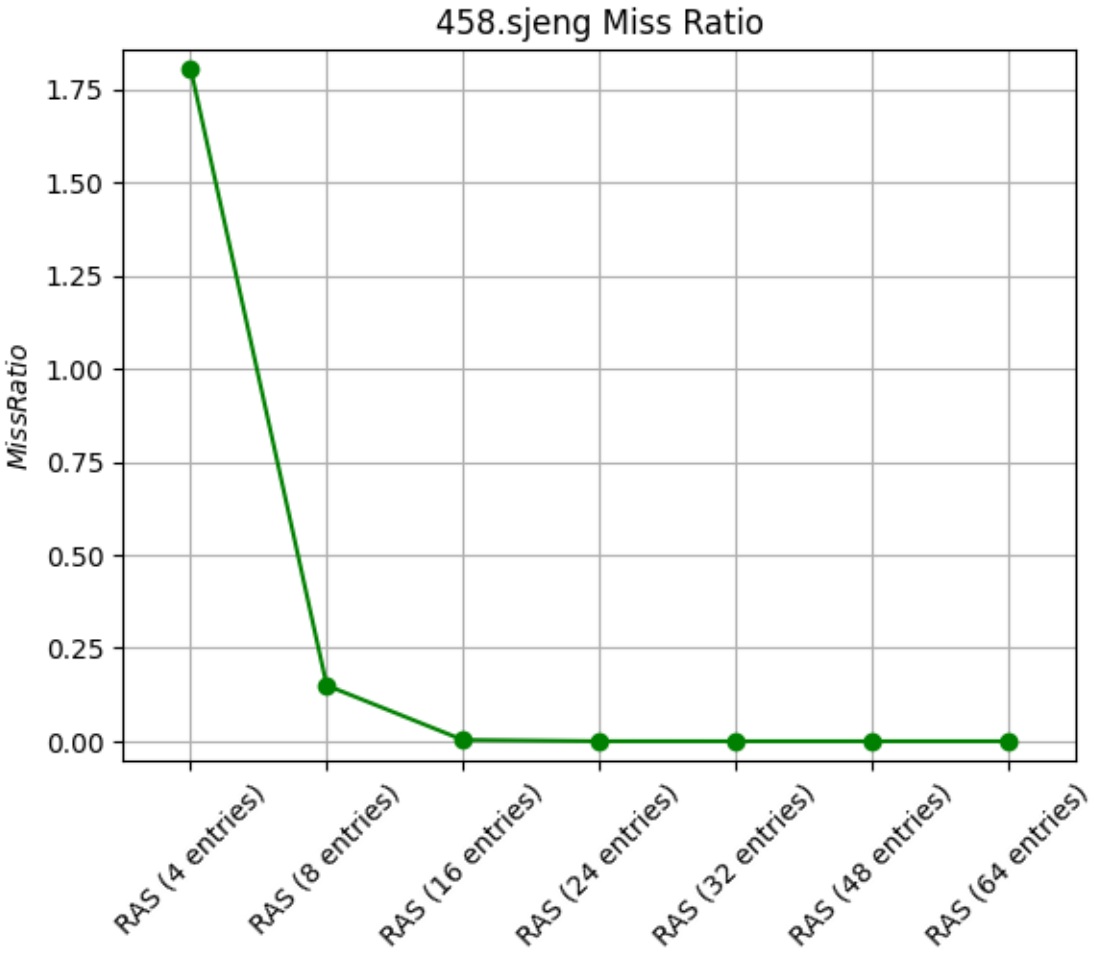
450.soplex


450.soplex Miss Ratio
MissRatio
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

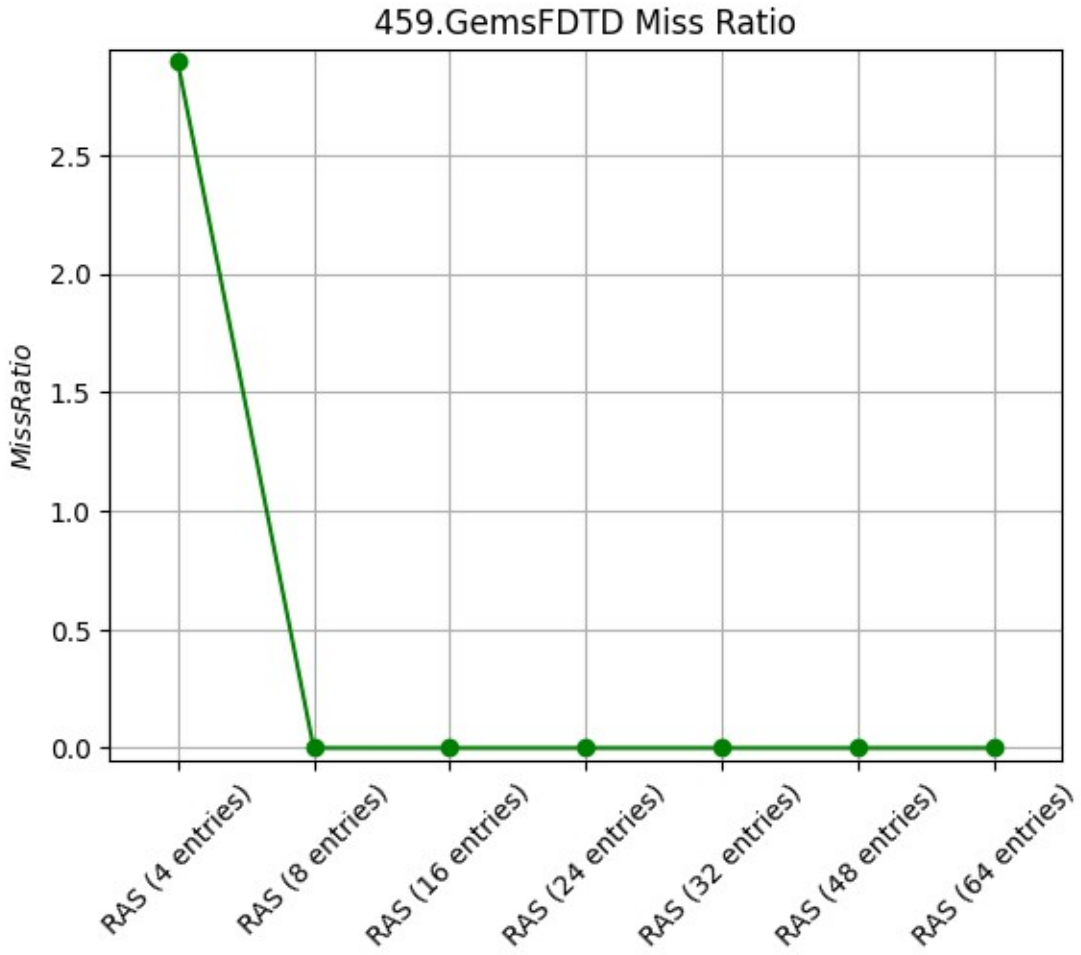
456.hmmer


456.hmmer Miss Ratio
MissRatio
0.04
0.02
0.00
−0.02
−0.04
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

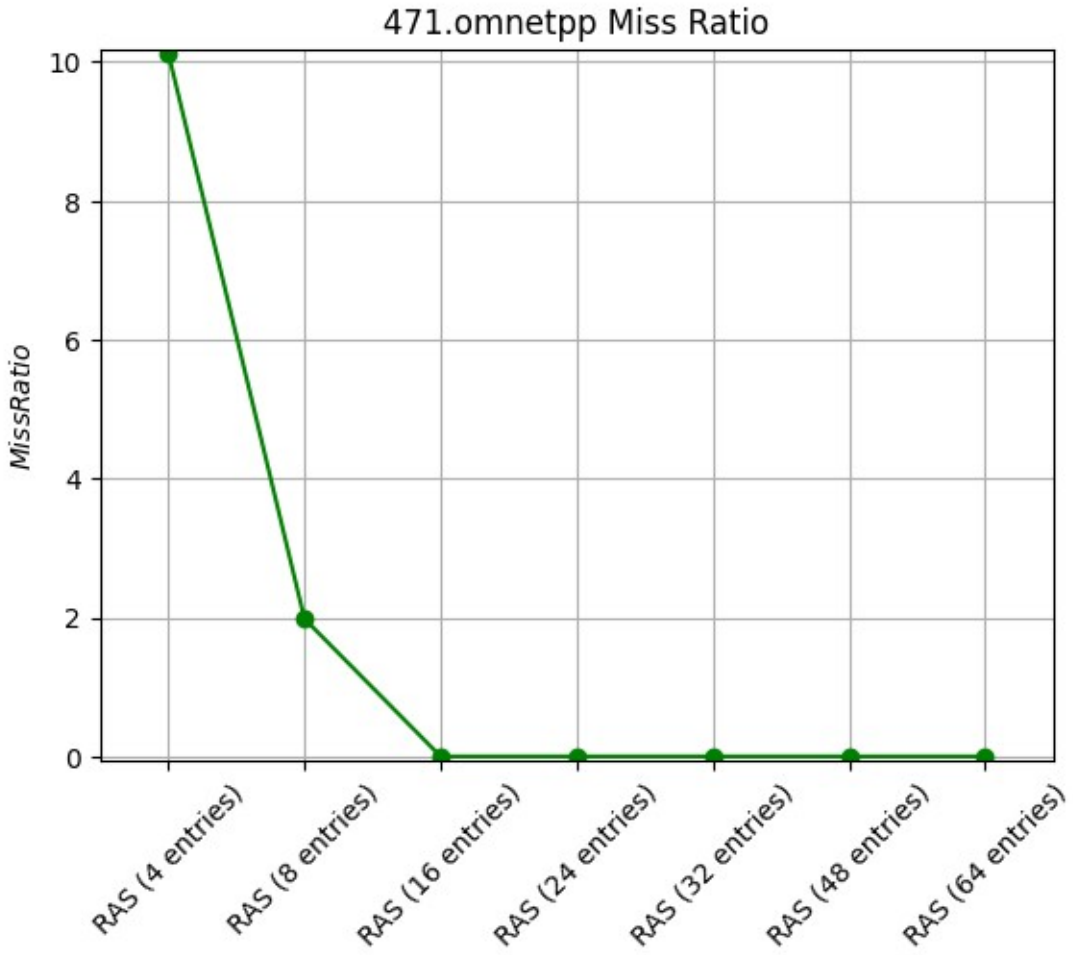
458.sjeng


458.sjeng Miss Ratio
MissRatio
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

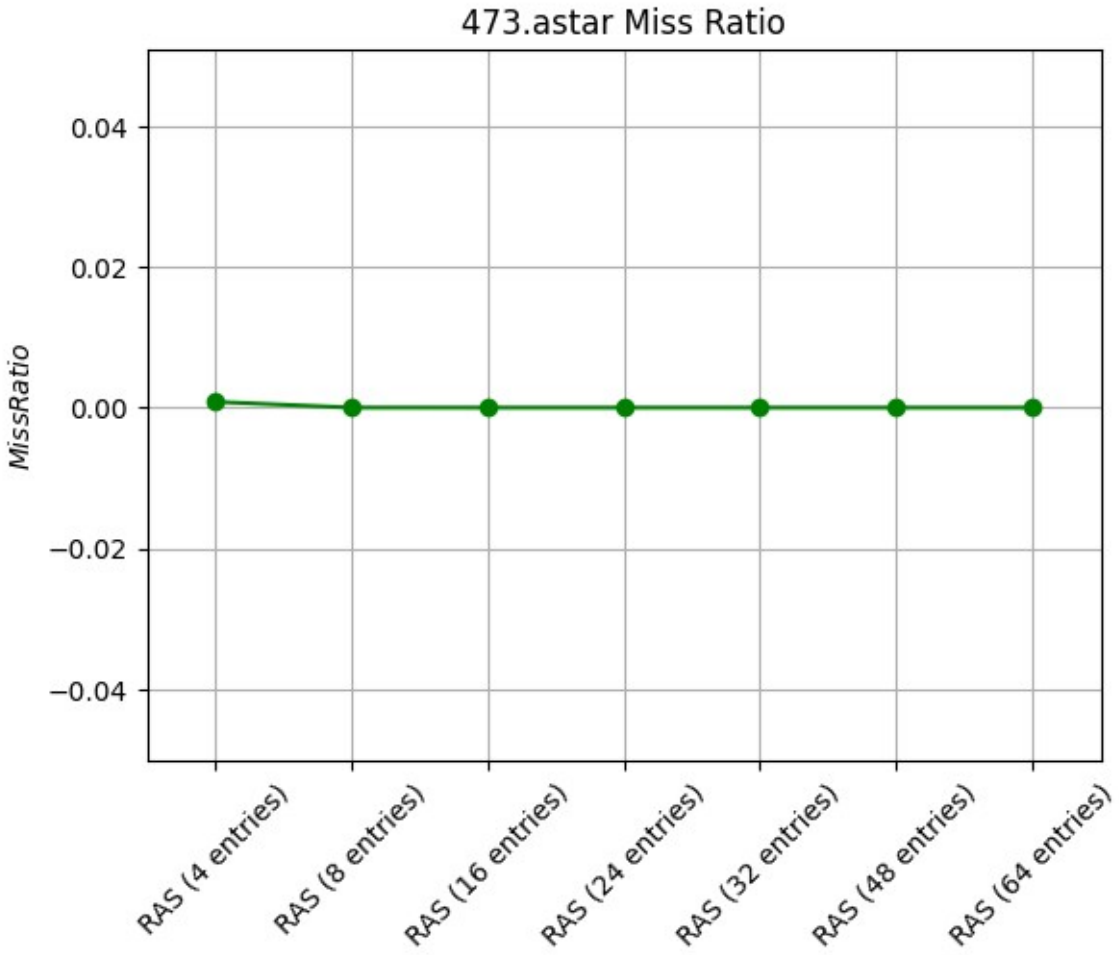
459.GemsFDTD



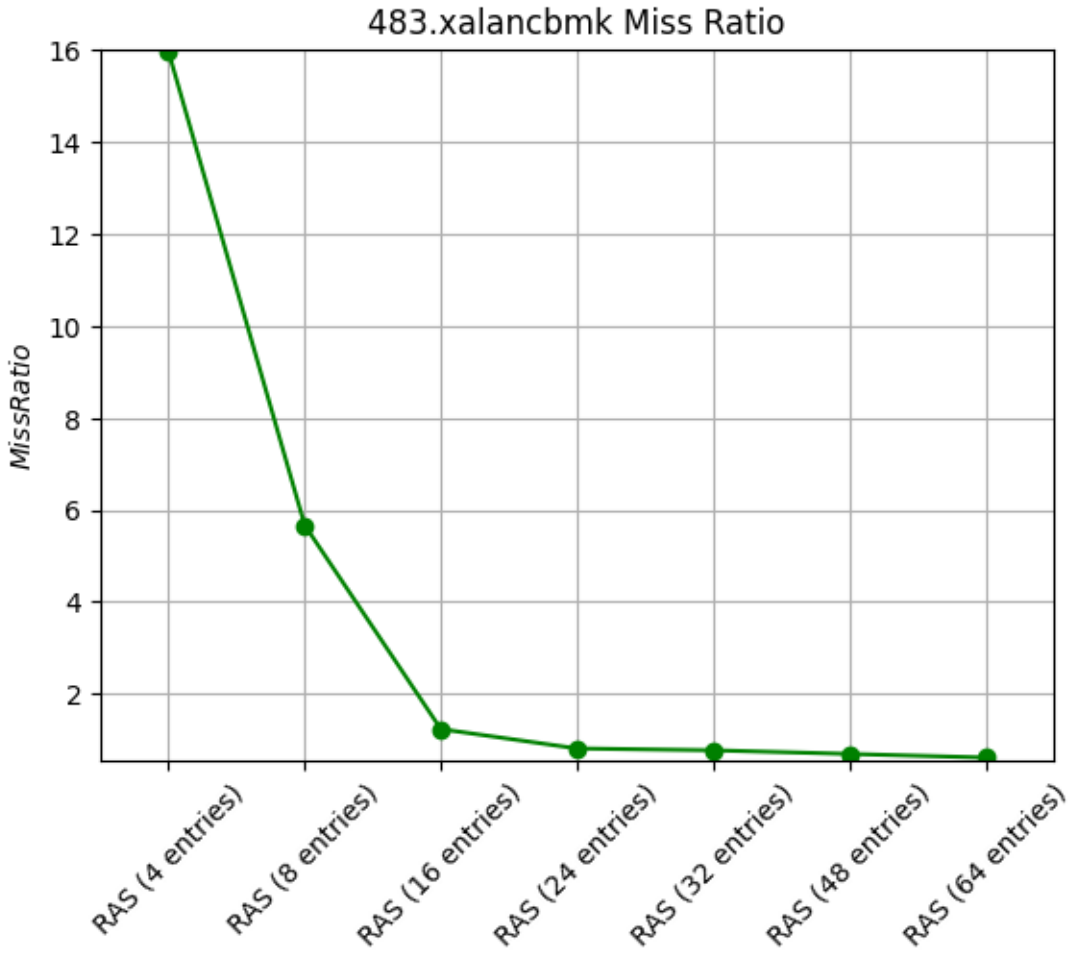
471.omneptt


471.omnetpp Miss Ratio
MissRatio
10
8
6
4
2
0
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

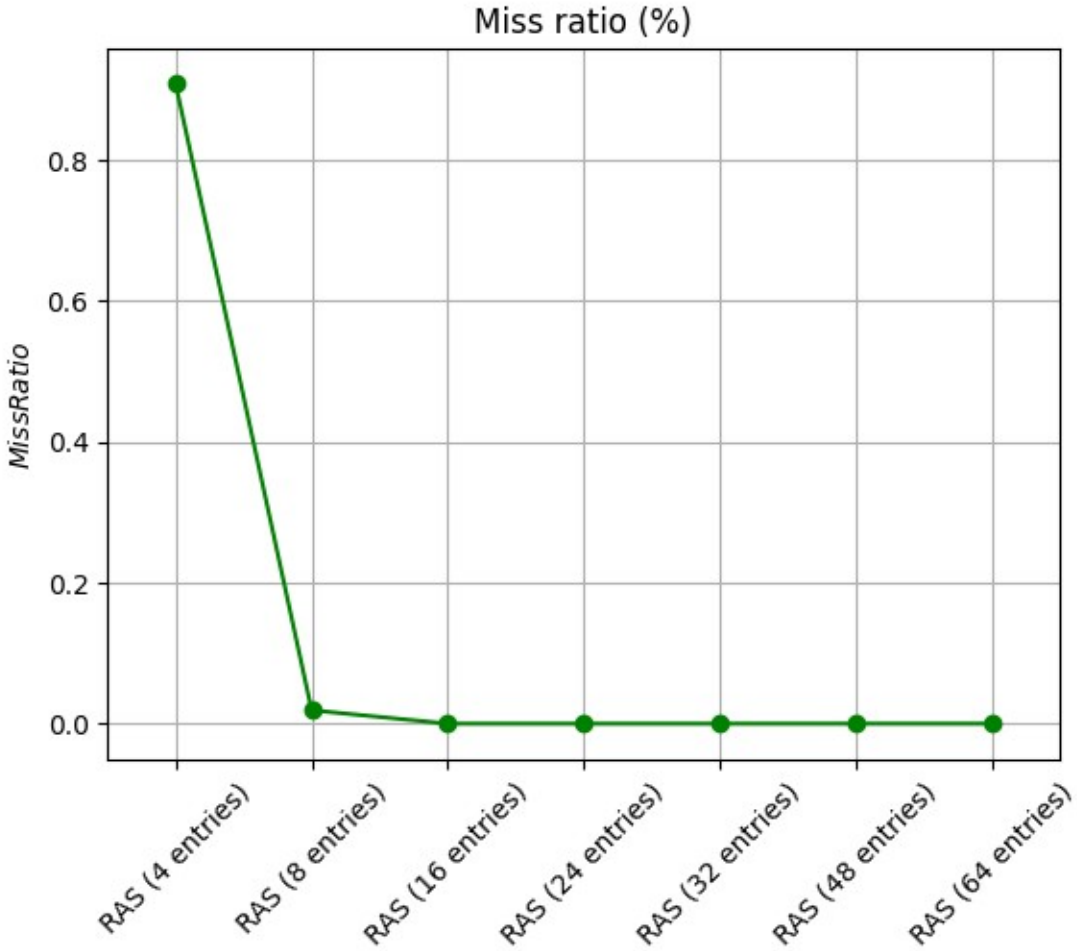
473.astar


473.astar Miss Ratio
MissRatio
0.04
0.02
0.00
−0.02
−0.04
RAS (4 entries)
RAS (8 entries)
RAS (16 entries)
RAS (24 entries)
RAS (32 entries)
RAS (48 entries)
RAS (64 entries)

483.xalancbmk



Συμπεράσματα



Στο παραπάνω διάγραμμα παρουσιάζουμε τον γεωμετρικό μέσο για τα αποτελέσματα όλων των benchmarks . Σε όλες τις εφαρμογές παρατηρήσαμε την ίδια συμπερίφορα, να σταθεροποιείται το Miss-Ratio στο μήδεν . Στις περισσότερες περιπτώσεις η σύγκλιση στο μήδεν ξεκίνησε από τα 16 entries, αν και σε κάποιες συνέβη από τις 4 και 8 entries .

Η μηδενισμός του Miss Ratio που παρατηρήσαμε σε όλες τις γραφικές είναι αναμενόμενος, καθώς αυξάνοντας το πλήθος εγγραφών στη στοίβα εξαλείφουμε τον ενδεχόμενο stack-overflow, που μπορεί να προκύψει από φωλιασμένες κλήσεις συναρτήσεων. Έτσι κάθε συνάρτηση εντοπίζει την σωστή διεύθυνση επιστροφής , μέσω την κλήση της εντολής reutrn, χωρίς τον κίνδυνο να έχει επανεγγραφεί από κάποια άλλη εντολή call.

Η χαμηλότερη τιμή entries, όπου όλα τα benchmarks είχαν μηδενικό Miss Ratio είναι 16. Όποτε η βέλτιστη επιλογή για την υλοποίηση της δομής RAS είναι 16 entries .

➢ Σύγκριση διαφορετικών predictors

Στο τελευταίο κομμάτι της άσκησης καλούμαστε να προσομοιώσουμε και να συγκρίνουμε την απόδοση διαφορετικών predictors . Οι προβλεπτές χωρίζονται στις κατηγορίες στατικοί και δυναμικοί .

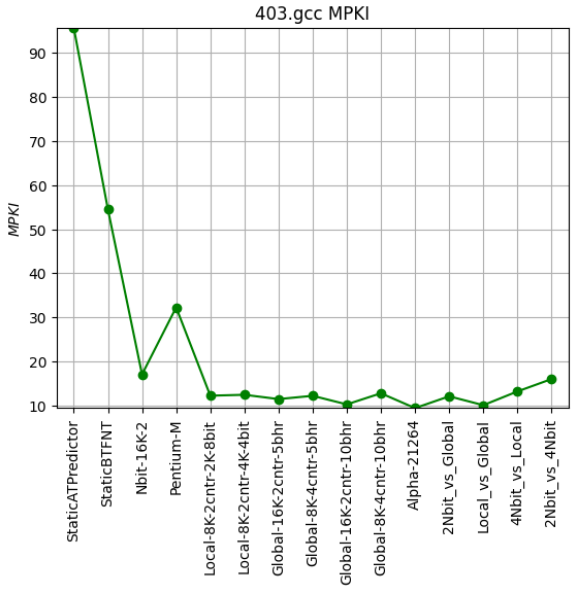
- Στατικοί
  - ➔ Static Always Taken : κάνει πάντα Taken prediction
  - ➔ Static BackwardTaken-ForwardNotTaken : αν η διακλάδωση έχει προορισμό με μικρότερη διεύθυνση προορισμού από αυτή του branch, κάνει πρόβλεψη Taken, σε αντίθετη περίπτωση Not-Taken

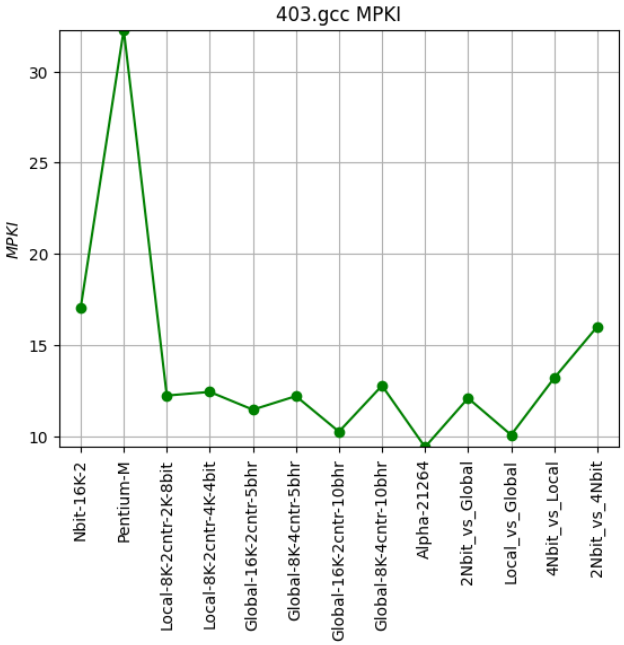
- Δυναμικοί
  - ➔ 2-bit Predicror με 16K entries
  - ➔ Pentium-M
  - ➔ Local-History Two-level :
    - PHT = 8K entriess , PHT n-bit counter length = 2, BHT = 2K entries, BHT entry length = 8bits
    - PHT = 8K entriess , PHT n-bit counter length = 2, BHT = 4K entries, BHT entry length = 4bits
  - ➔ Global-History Two-level :
    - PHT = 16K entries, PHT n-bit counter length = 2, BHR length = 5bits
    - PHT = 8K entries, PHT n-bit counter length = 4, BHR length = 5bits
    - PHT = 16K entries, PHT n-bit counter length = 2, BHR length = 10bits
    - PHT = 8K entries, PHT n-bit counter length = 4, BHR length = 10bits
  - ➔ Alpha 21264 Predictor
    - Local-History Two-level :
      - PHT = 1K entriess , PHT n-bit counter length = 3, BHT = 1K , BHT entry length = 10bits
    - Global-History Two-level :
      - PHT = 4K entries, PHT n-bit counter length = 2, BHR length = 12bits
    - Meta predictor :
      - Ο Meta predictor έχει 4K enrties των 2-bit predictors, και η προσπέλαση γίνεται μέσω του BHR .
  - ➔ Tournament Hybrid predictors:
    - Ο Meta predictor έχει 1K entries στις δυο πρώτες υλοποιήσεις και 2K entries στις δυο τελευταίες . Χρησιμοποιεί 2-bit Predictor και η προσπέλαση γίνεται μέσω της διεύθυνσης του branch.
    - 2-bit vs Global :
      - 2-bit Predicror με 8K entries
      - Global-History Two-level : PHT = 8K entries, PHT n-bit counter length = 2, BHR length = 4bits
    - Local vs Global
      - Local-History Two-level : PHT = 4K entriess , PHT n-bit counter length = 2, BHT = 2K , BHT entry length = 4bits
      - Global-History Two-level : PHT = 4K entries, PHT n-bit counter length = 4, BHR length = 6bits
    - 4-bit vs Local:
      - 4-bit Predicror με 4K entries

    - Local-History Two-level : PHT = 6K entriess , PHT n-bit counter length = 2, BHT = 2K , BHT entry length = 2bits
- 2-bit vs 4-bit
    - 2-bit Predicror με 8K entries
    - 4-bit Predicror με 4K entries

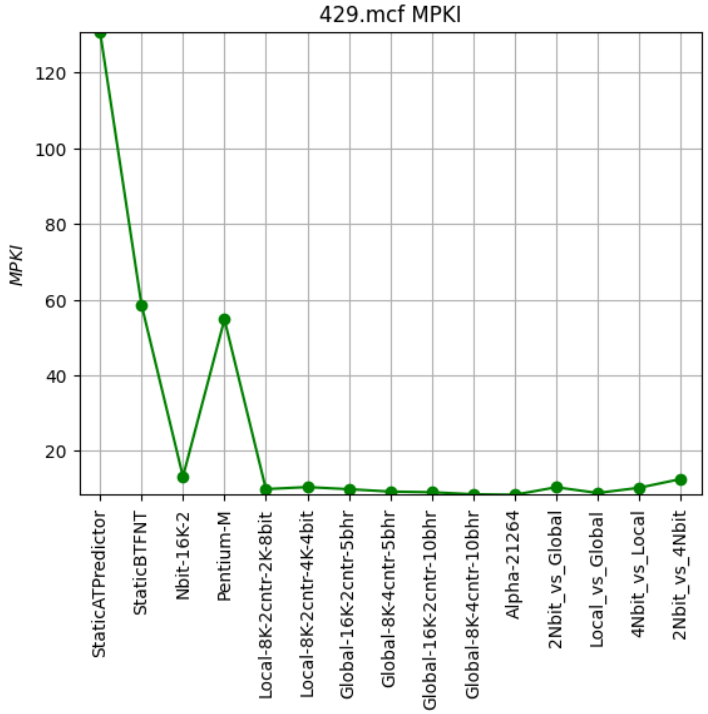
Η μετρική που χρησιμοποιούμε για παρουσίαση των αποτελεσμάτων είναι η MPKI . Για κάθε benchmark θα αναπαριστούμε τα αποτελέσματα σε ένα κοινό διάγραμμα όλους τους predictors και σε ένα ξεχωριστό μόνο τους δυναμικούς .

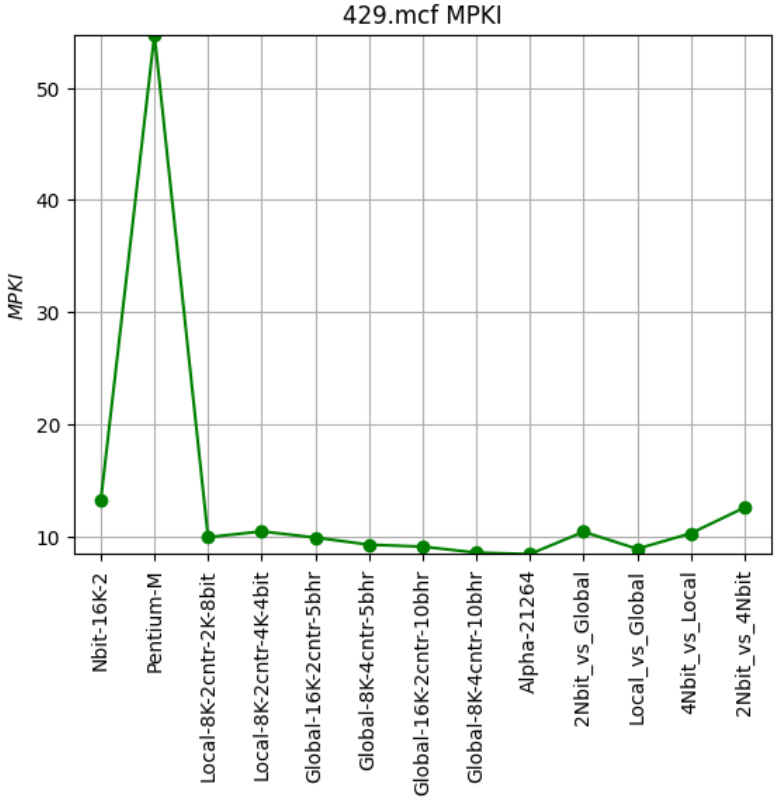
403.gcc



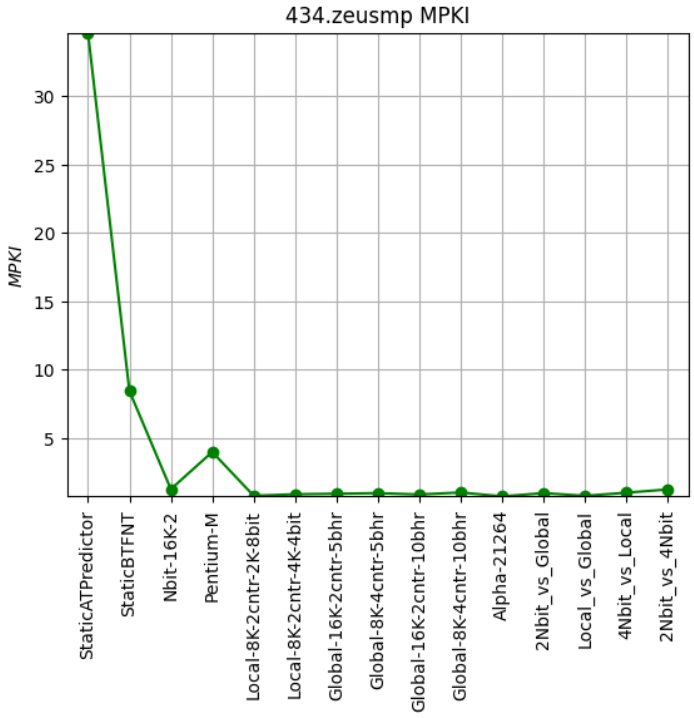


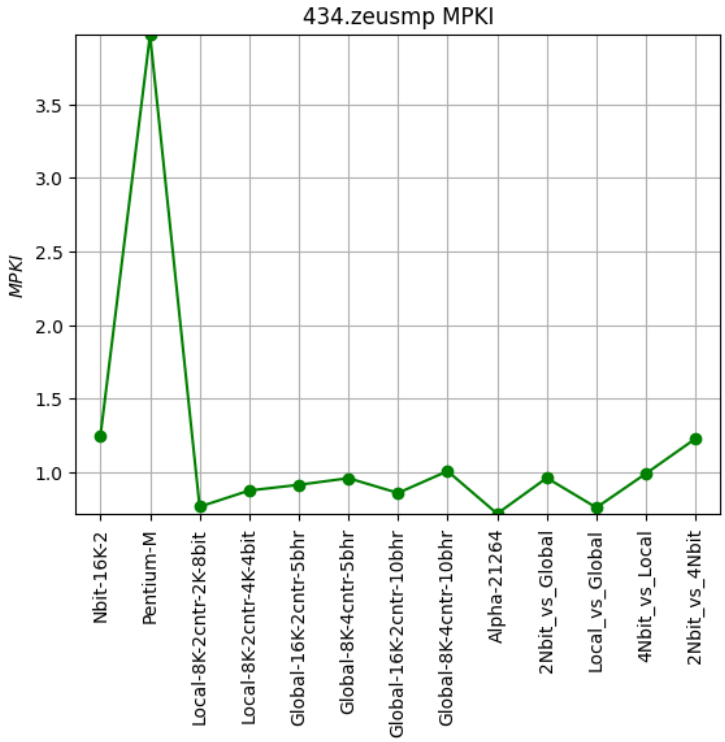
429.mcf


429.mcf MPKI
MPKI
120
100
80
60
40
20
StaticATPredictor
StaticBTFNT
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

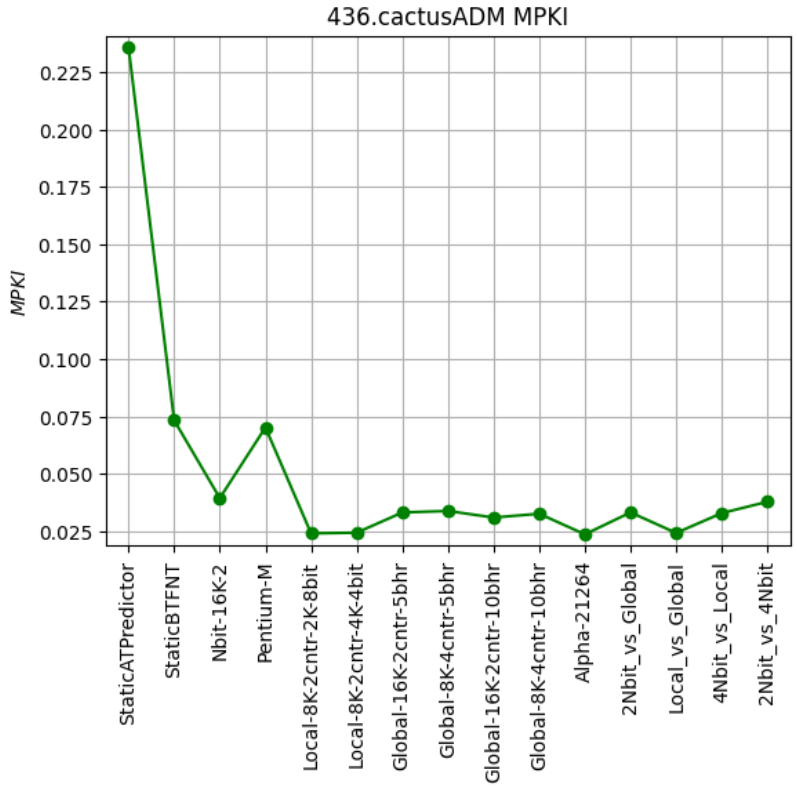

429.mcf MPKI
MPKI
50
40
30
20
10
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

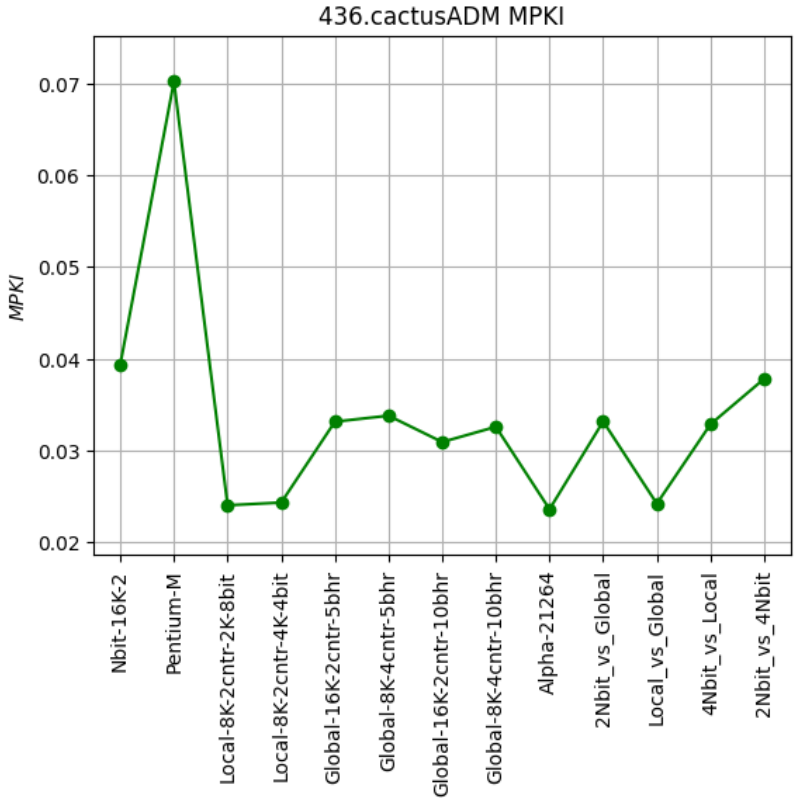
434.zeusmp


434.zeusmp MPKI
MPKI
30
25
20
15
10
5
StaticATPredictor
StaticBTFNT
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

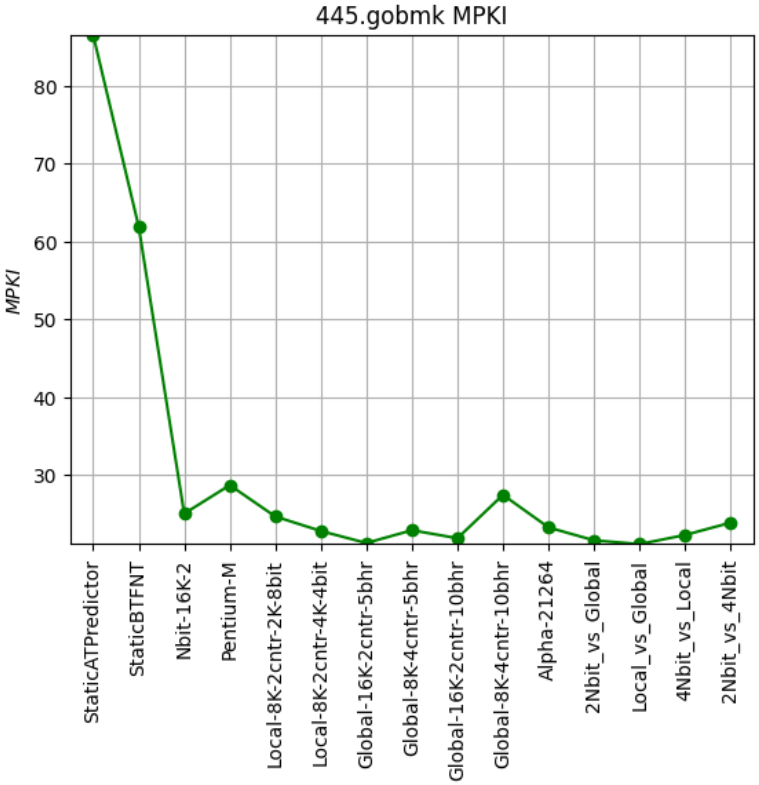

434.zeusmp MPKI
MPKI
3.5
3.0
2.5
2.0
1.5
1.0
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

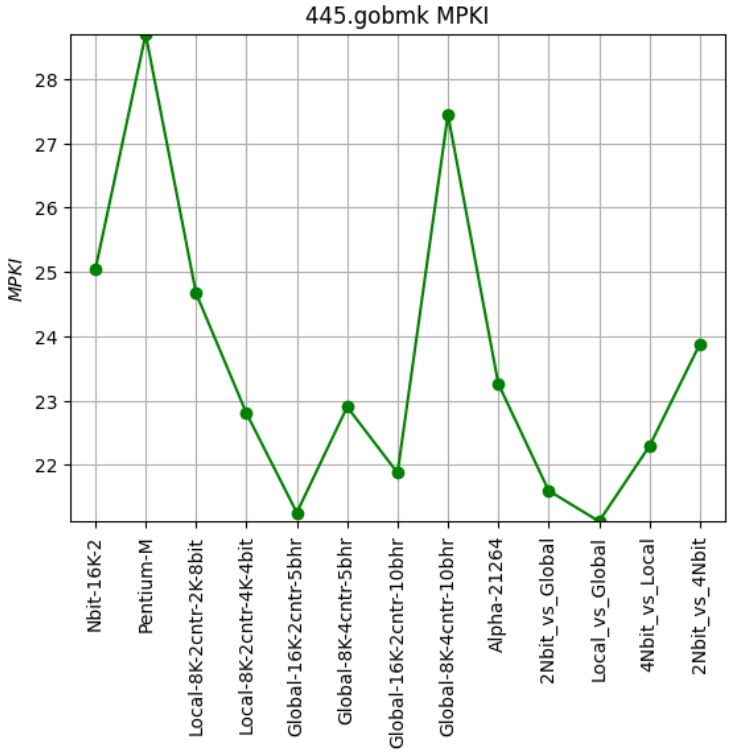
436.cactusADM



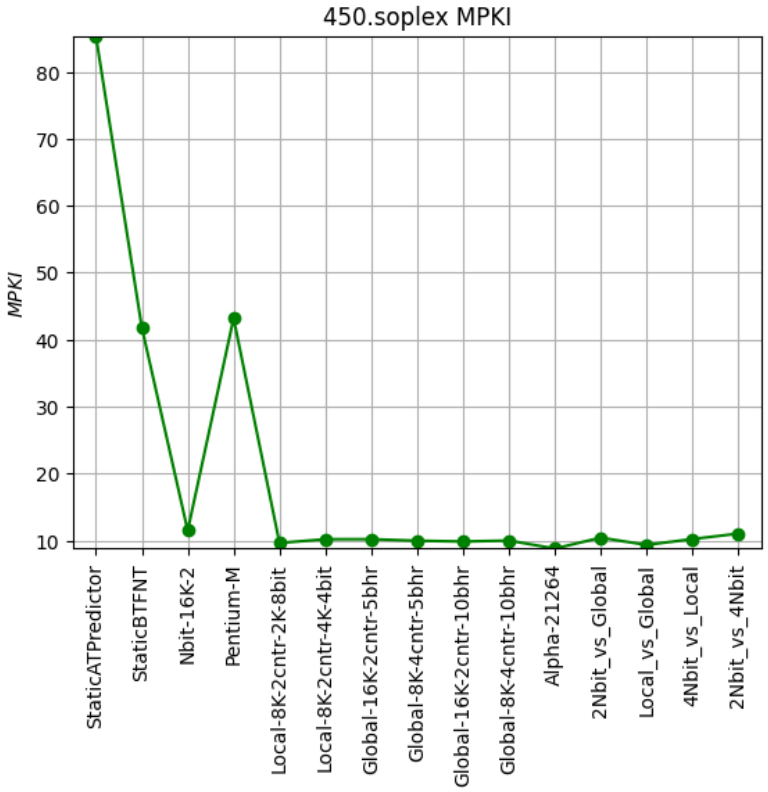


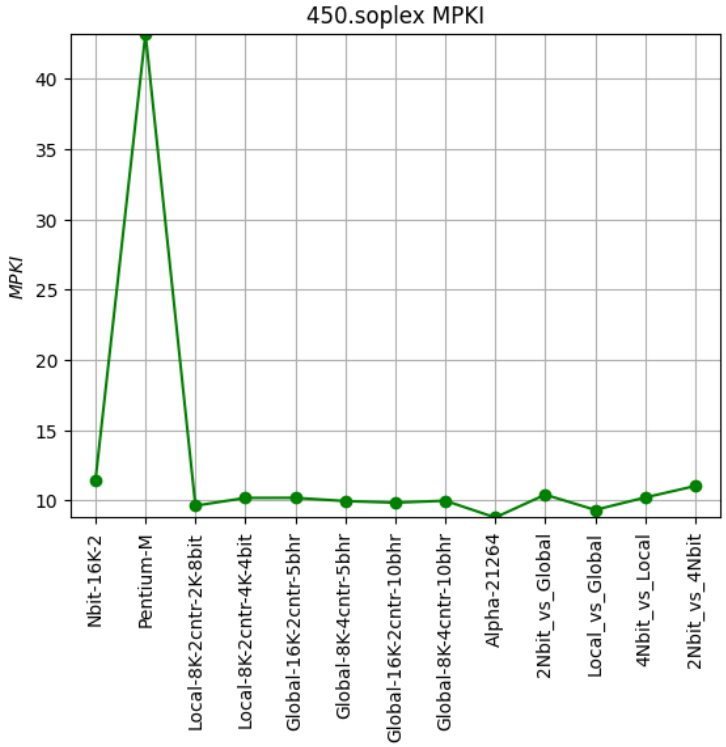
445.gobmk


445.gobmk MPKI
MPKI
80
70
60
50
40
30
StaticATPredictor
StaticBTFNT
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

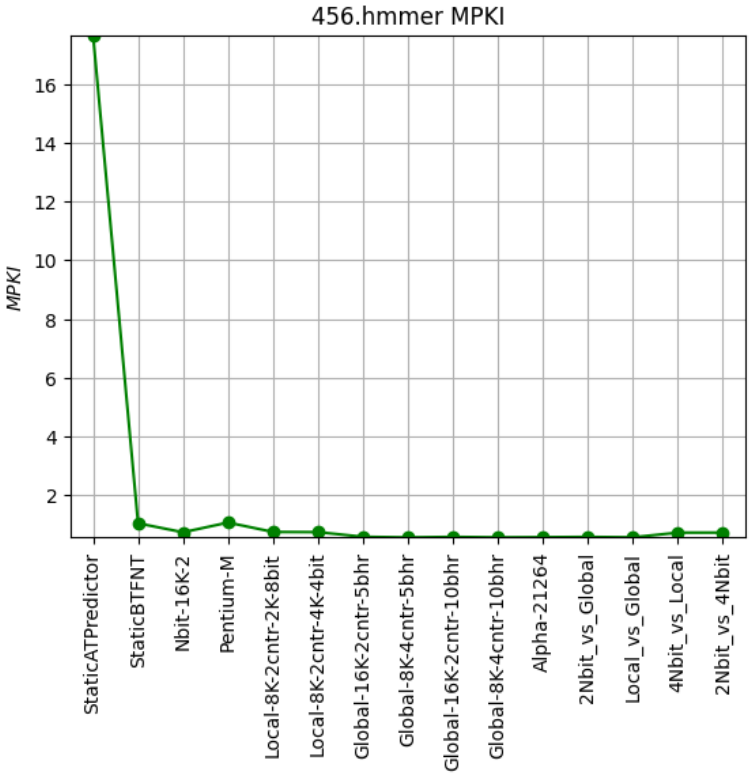

445.gobmk MPKI
MPKI
28
27
26
25
24
23
22
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

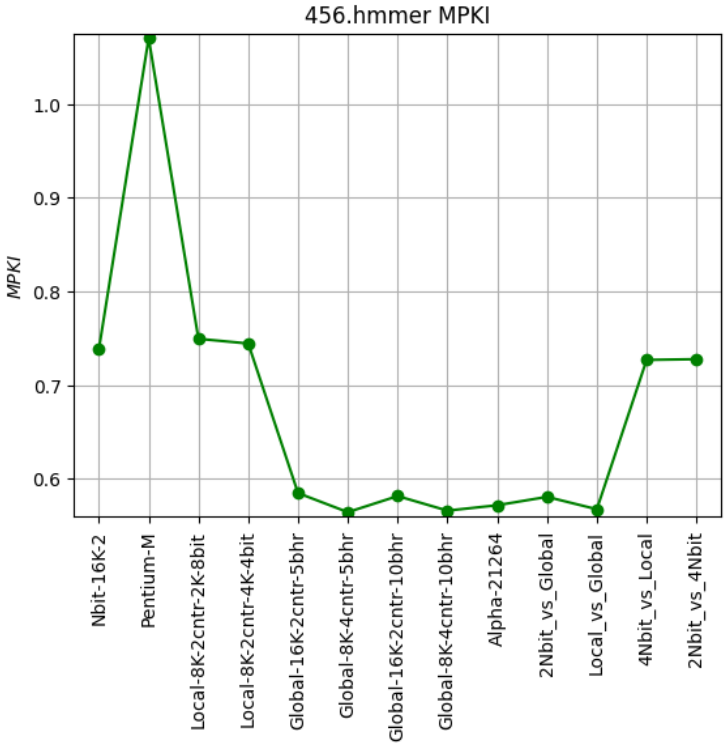
450.soplex


450.soplex MPKI
MPKI
80
70
60
50
40
30
20
10
StaticATPredictor
StaticBTFNT
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

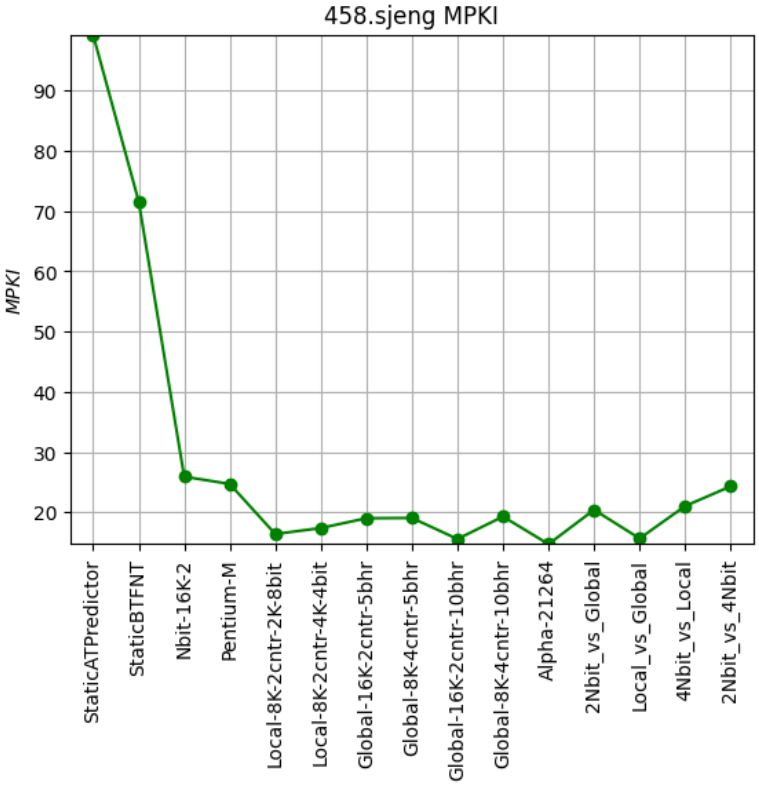

450.soplex MPKI
MPKI
40
35
30
25
20
15
10
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

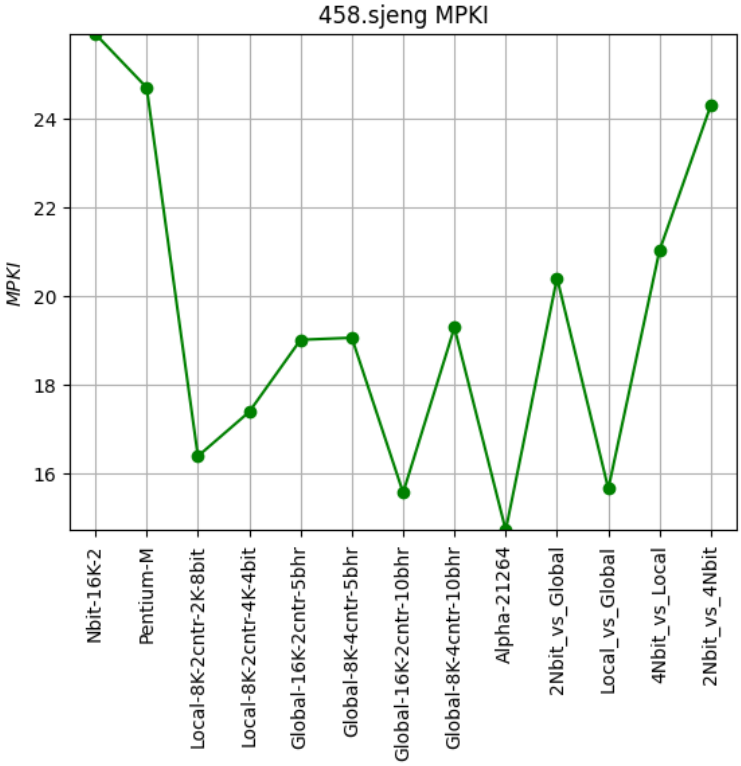
456.hmmer


456.hmmer MPKI
MPKI
16
14
12
10
8
6
4
2
StaticATPredictor
StaticBTFNT
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

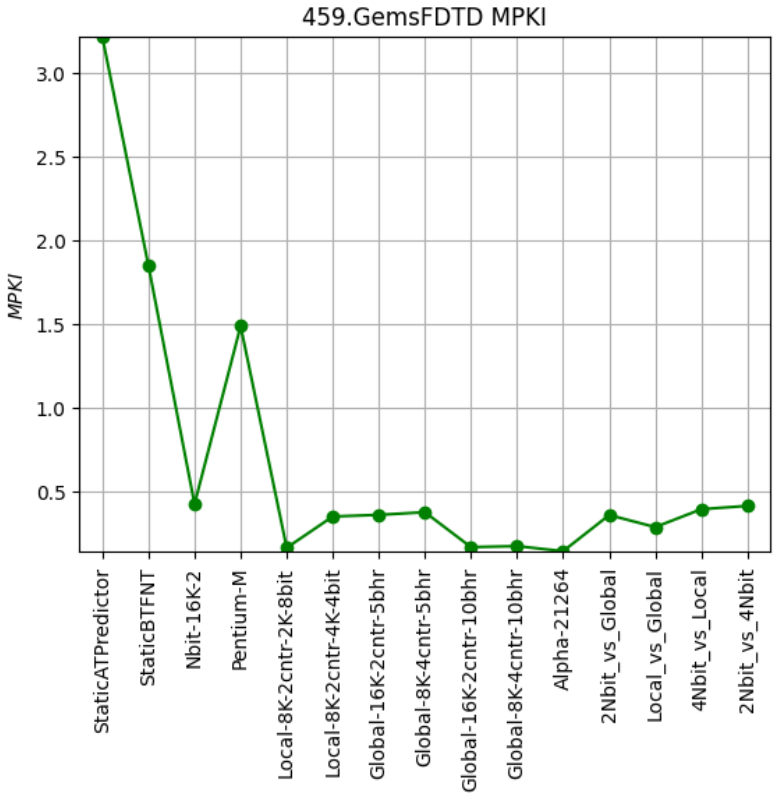

456.hmmer MPKI
MPKI
1.0
0.9
0.8
0.7
0.6
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

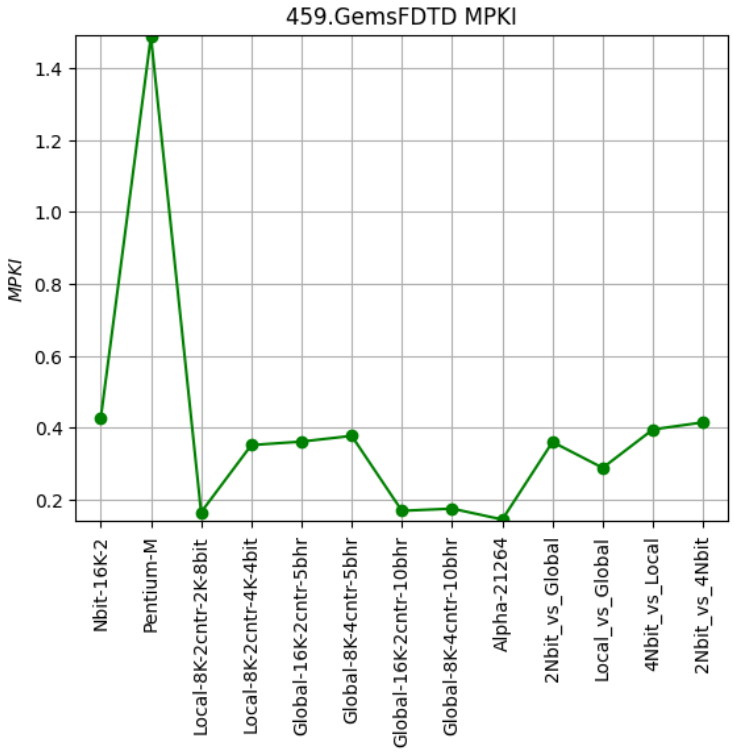
458.sjeng


458.sjeng MPKI
MPKI
90
80
70
60
50
40
30
20
StaticATPredictor
StaticBTFNT
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

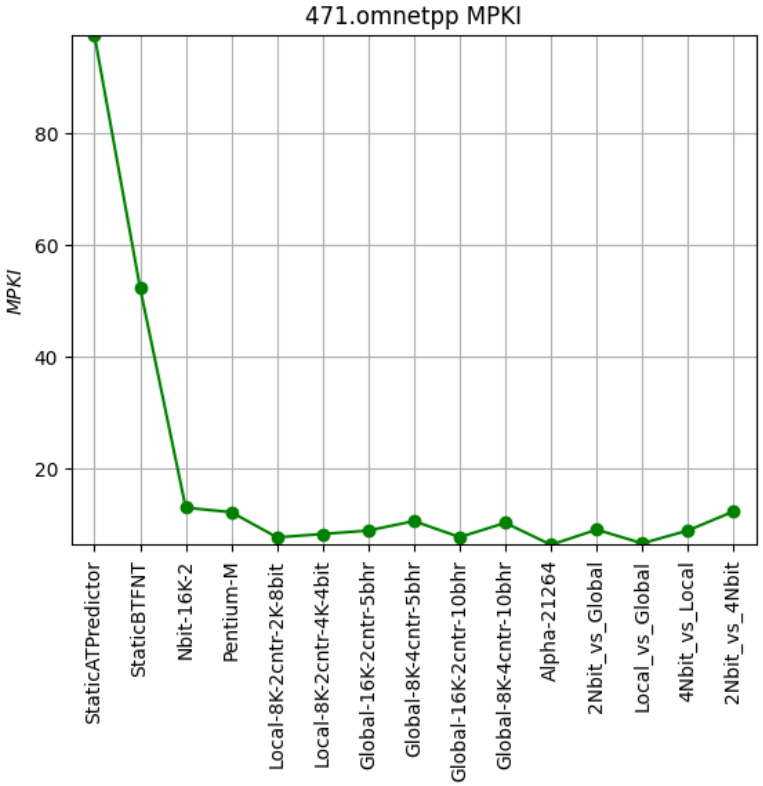

458.sjeng MPKI
MPKI
24
22
20
18
16
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

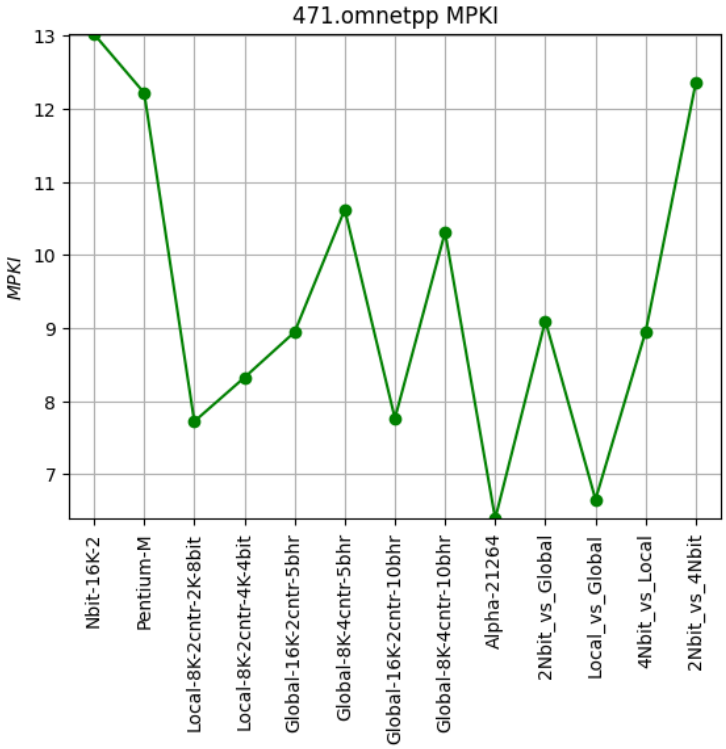
459.GemsFDTD



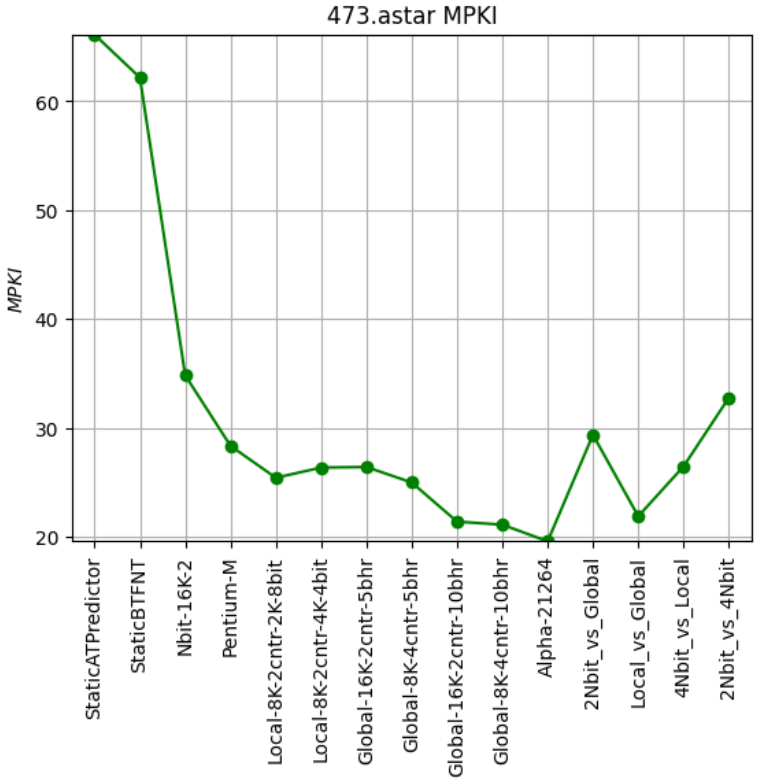


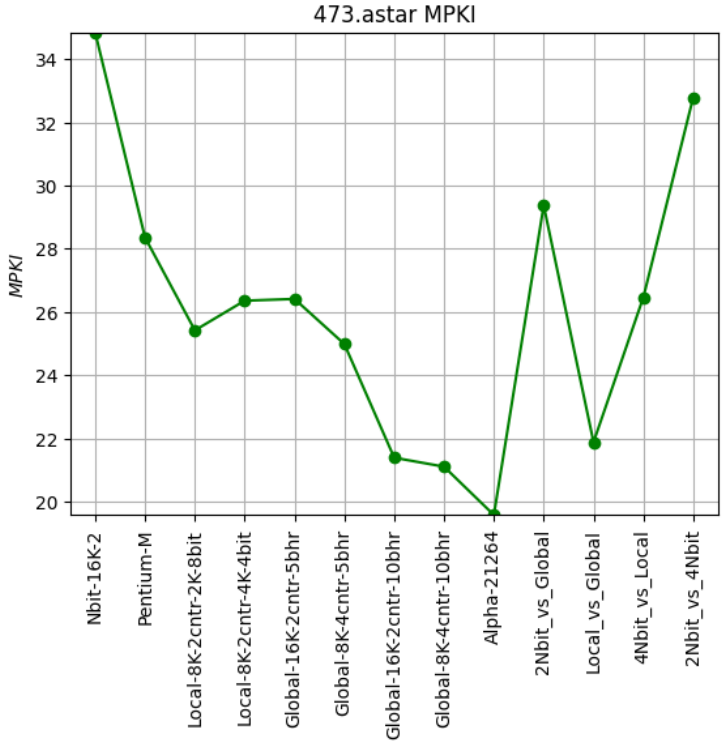
471.omnetpp


471.omnetpp MPKI
MPKI
80
60
40
20
StaticATPredictor
StaticBTFNT
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

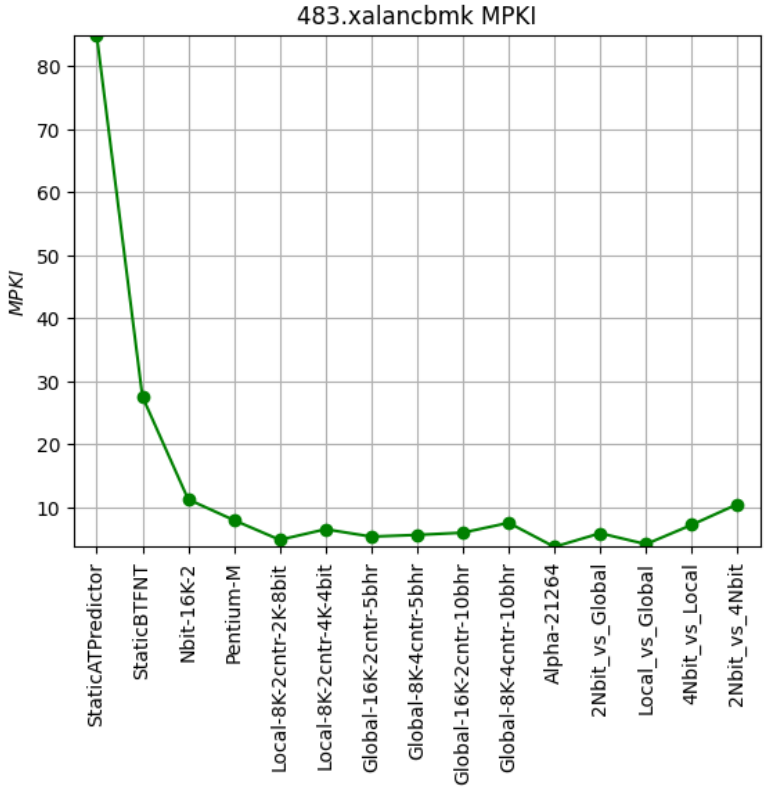

471.omnetpp MPKI
MPKI
13
12
11
10
9
8
7
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

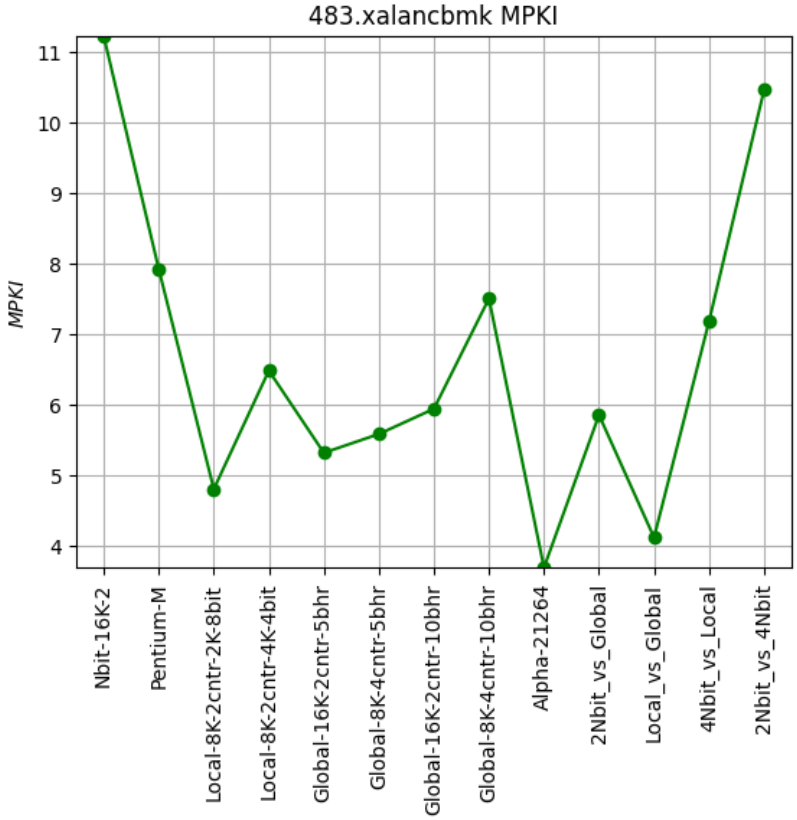
473.astar


473.astar MPKI
MPKI
60
50
40
30
20
StaticATPredictor
StaticBTFNT
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

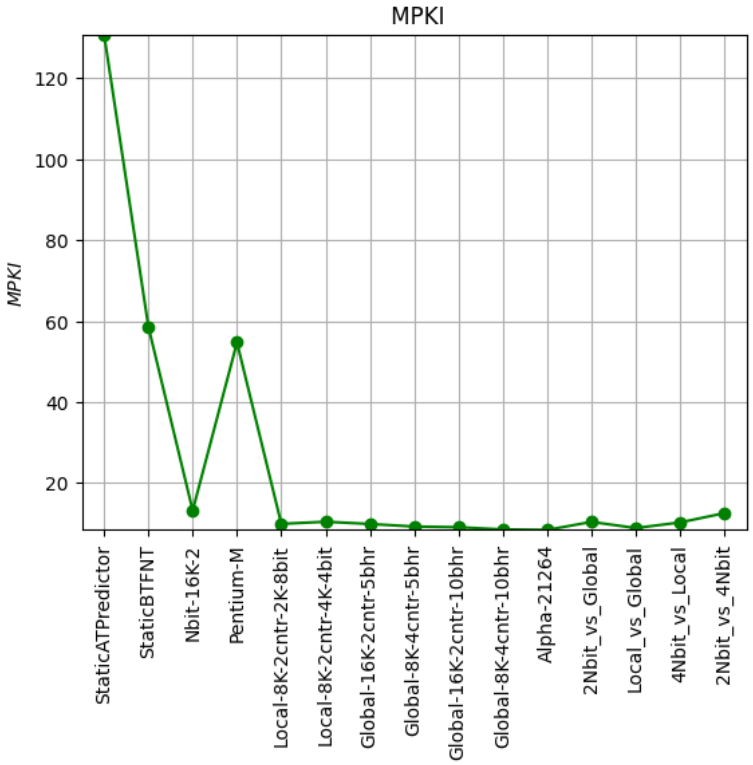

473.astar MPKI
MPKI
34
32
30
28
26
24
22
20
Nbit-16K-2
Pentium-M
Local-8K-2cntr-2K-8bit
Local-8K-2cntr-4K-4bit
Global-16K-2cntr-5bhr
Global-8K-4cntr-5bhr
Global-16K-2cntr-10bhr
Global-8K-4cntr-10bhr
Alpha-21264
2Nbit_vs_Global
Local_vs_Global
4Nbit_vs_Local
2Nbit_vs_4Nbit

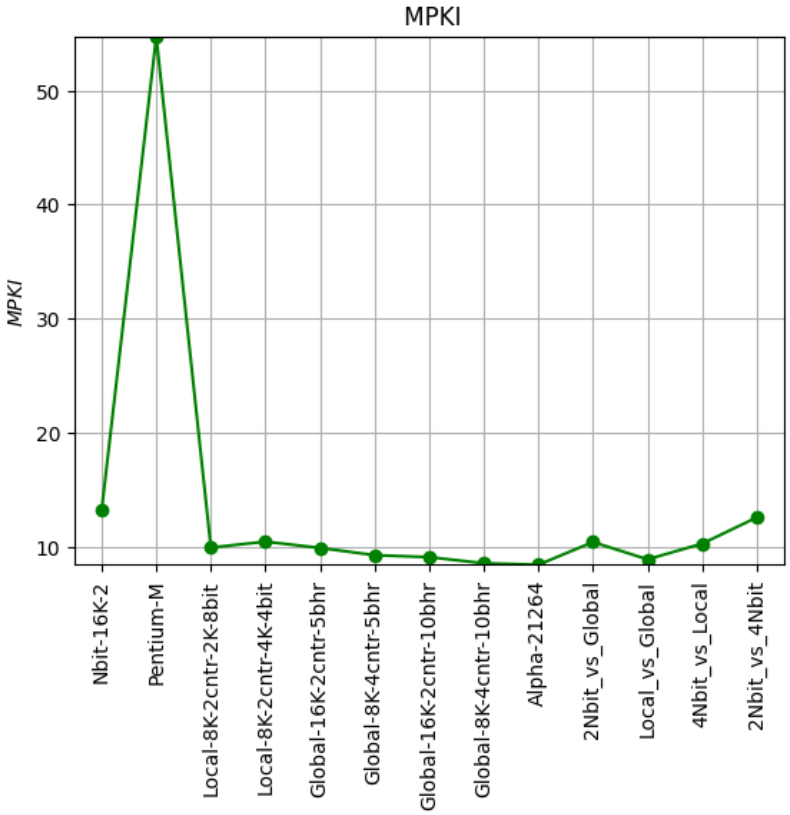
483.xalancbmk





Συμπεράσματα





Στα παραπάνω δύο διαγράμματα έχουμε τον γεωμετρικό μέσο MPKI για κάθε perdictor .

Όπως ήταν αναμενόμενο οι στατικοί έχουν τις χειρότερες επιδόσεις. Όμως Static BTFNT καταφέρνει να έχει το μισό ποσοστό αστοχίας από τον Static AT .

Ο Pentium-M έχει την χειρότερη απόδοση από τους δυναμικούς και έχει αποτελέσματα πολυ κοντινά με τον Static BTFNT . Οι υπόλοιποι αποδίδουν καλύτερα, χωρίς ιδιαίτερη διαφοροποιήση στον δείκτη MPKI .

Την βέλτιστη απόδοση την έχουμε για τους προβλεπτές : Alpha 21264, Global-8K-4cntr-10bhr και Local vs Global . Μιας και τα αποτελέσματα τους είναι πολύ κοντινά θα επιλέξουμε με βάση την κατανάλωση υλικού .

- Alpha 21264 : 29Kbits
- Global-8K-4cntr-10bhr : 32Kbits
- Local vs Global : 32Kbits

Οπότε επιλέγουμε τον Alpha 21264 .

ΥΓ.
Τα ορθογραφικά και συντακτικά λάθη που μπορεί να έχω κάνει οφείλονται στην δυσλεξία που έχω. Επίσης, στον ίδιο οφείλονται και κάποια εκφραστικά λάθη που μπορεί να έχω κάνει διότι μου είναι δύσκολο να εκφράσω την σκέψη μου μέσω του γραπτού λόγου. Σε περίπτωση που υπάρχει κάποια ασάφεια ή δυσκολία κατανόησης ,παρακαλώ να επικοινωνήσετε μαζί μου στο email: arapidismanolis@gmail.com