Πανεπιστήμιο Πελοποννήσου

Σχολή Οικονομίας, Διοίκησης και Πληροφορικής
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων σε επίπεδο πηγαίου κώδικα για την βελτιστοποίηση της τοπικότητας αναφοράς σε αλγορίθμους όπου κυριαρχούν βρόχοι επανάληψης

Διδακτορική Διατριβή

Χρηστάκης Λέζος

Οκτώβριος 2018

Περίληψη

Στην παρούσα διδακτορική διατριβή παρουσιάζεται ένα εργαλείο για την εξαγωγή προτάσεων βελτιστοποίησης του πηγαίου κώδικα αλγορίθμων υλοποιημένων σε γλώσσες υψηλού επιπέδου και συγκεκριμένα στις γλώσσες C και MATLAB. Βασικό μέρος της διαδικασίας εξαγωγής αυτών των προτάσεων αποτελεί η ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων. Η γλώσσα C έχει χρησιμοποιηθεί ευρέως σαν γλώσσα εισόδου στην πλειονότητα των συναφών εργασιών. Η γλώσσα MATLAB είναι μια υψηλού επιπέδου γλώσσα, η οποία απλοποιεί κατά πολύ τις πράξεις μεταξύ πινάκων (array language) και χρησιμοποιείται κατά κόρον για την πρωτοτυποποίηση αλγορίθμων από επιστήμονες και μηχανικούς. Η ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων αφορά τη μέτρηση του αριθμού των μοναδικών στοιχείων δεδομένων του κώδικα που μεσολαβούν μεταξύ μιας επαναχρησιμοποίησης δεδομένων. Οι προτάσεις βελτιστοποίησης που παράγει το προτεινόμενο εργαλείο αφορούν κατά κύριο λόγο μετασχηματισμούς βρόχων επανάληψης και έχουν ως στόχο την μείωση των αποστάσεων επαναχρησιμοποίησης δεδομένων και την επακόλουθη βελτίωση της χρονικής τοπικότητας αναφοράς, τη μείωση των αστοχιών της λανθάνουσας μνήμης δεδομένων και τελικά τη βελτίωση της ταχύτητας εκτέλεσης ενός αλγορίθμου.

Στο πλαίσιο της διδακτορικής διατριβής αναπτύχθηκε επίσης ένα σύνολο διασυνδεδεμένων εργαλείων για την αυτόματη ανάπτυξη εργαλείων ανάλυσης και χειρισμού πηγαίου κώδικα, το οποίο περιλαμβάνει μια αυτόματη γεννήτρια εμπρόσθιων τμημάτων μεταγλωττιστών και μια γλώσσα προγραμματισμού ειδικού σκοπού για την εφαρμογή ερωτημάτων (query language) σε πηγαίο κώδικα. Τα εργαλεία αυτά αναπτύχθηκαν με σκοπό να χρησιμοποιηθούν για την κατασκευή του προτεινόμενου εργαλείου βελτιστοποίησης. Η γεννήτρια εμπρόσθιων τμημάτων παράγει έναν λεκτικό και συντακτικό αναλυτή για οποιαδήποτε δοθείσα γραμματική η οποία περιγράφεται με τη μορφή συμβολισμού BNF. Η γλώσσα ειδικού σκοπού μπορεί να χρησιμοποιηθεί για την εφαρμογή ερωτημάτων στο αφηρημένο συντακτικό δέντρο που παράγεται από το εμπρόσθιο τμήμα που δημιουργείται αυτόματα.

Το προτεινόμενο εργαλείο πραγματοποιεί ανάλυση σε C κώδικα και όχι σε MATLAB. Για το λόγο αυτό αναπτύχθηκε ένας MATLAB-σε-C μεταγλωττιστής για την υποστήριξη πηγαίου κώδικα σε γλώσσα MATLAB. Τα αποτελέσματα της ανά-

λυσης αντιστοιχίζονται έπειτα στον αρχικό MATLAB κώδικα. Εκτός από τη χρήση του στο πλαίσιο λειτουργίας του προτεινόμενου εργαλείου, ο συγκεκριμένος μεταγλωττιστής έχει τη δυνατότητα να παράγει C κώδικα με δυναμικά δεσμευμένους πίνακες, ο οποίος έχει κάποιες ιδιαιτερότητες και πλεονεκτήματα σε σχέση με αντίστοιχες υλοποιήσεις της βιβλιογραφίας.

Τέλος, αξιολογήθηκε η επίδοση (ως προς το χρόνο εκτέλεσης και την απόδοση των λανθανουσών μνημών) του βελτιστοποιημένου κώδικα χρησιμοποιώντας μια σειρά από ρεαλιστικούς αλγορίθμους σε τέσσερεις διαφορετικούς επεξεργαστές αρχιτεκτονικών ARM, MIPS και x86 με διαφορετικές ιεραρχίες μνήμης, καθώς και στον προσομοιωτή λανθανουσών μνημών Cachegrind για τρεις διαφορετικές ιεραρχίες μνήμης. Τα αποτελέσματα έδειξαν ότι ο βελτιστοποιημένος κώδικας επιτυγχάνει σημαντικές βελτιώσεις σε σχέση με την απόδοση του αρχικού κώδικα. Πιο συγκεκριμένα, επιτυγχάνεται μείωση των προσπελάσεων της κύριας μνήμης έως και 42%, με αντίστοιχη βελτίωση του χρόνου εκτέλεσης (speedup) έως και 3x για βελτιστοποιήσεις που αφορούν την τοπικότητα αναφοράς δεδομένων και έως 68x για βελτιστοποιήσεις που αφορούν την μείωση των περιττών επανυπολογισμών. Για την περίπτωση του MATLAB κώδικα, ο κώδικας C που παράγεται από τον βελτιστοποιημένο κώδικα MATLAB μέσω του MATLAB-σε-C μεταγλωττιστή MathWorks MATLAB Coder είναι επίσης αποδοτικότερος από αυτόν που παράγεται από την αρχική έκδοση του κώδικα MATLAB.

Abstract

This Ph.D. dissertation discusses a software profiling and optimization tool that infers source code optimizations for algorithms implemented in high level languages like C and MATLAB. Reuse distance analysis forms the main part of the optimization process. C has already been extensively used as the input specification on most similar systems. MATLAB is a high level array programming language used broadly for prototyping algorithms in scientific and engineering settings. Reuse distance analysis is the process of calculating the number of distinct data elements accessed between two consecutive uses of the same memory element during program execution. The notion of reuse distance is equivalent to temporal data reuse but in a machine independent manner. Reuse distance analysis provides quantitative measures of program locality that can be used to drive locality optimization. The optimizations proposed by the tool are mainly loop transformations which target the reduction of reuse distances and thus the improvement of temporal locality, the reduction of data cache misses and the improvement of the execution speed of a specific program.

A framework for the efficient development of code analysis software has also been developed. This framework includes a tool for automatically generating the front end of analysis tools for a given language grammar expressed in the BNF notation. It also provides a domain specific language to concisely express queries on the internal representation generated by the front end. This language tackles the problem of writing complex code in a general purpose programming language in order to retrieve information from the internal representation. Both tools of this framework have been developed and used primarily for the construction of the proposed optimization tool.

The required reuse distance analysis is always performed at C code level. To support MATLAB input sources, a MATLAB-to-C compiler has been developed. This compiler can relate the input MATLAB variables with output C code. Then reuse distance analysis is performed on the C code and the relevant optimization proposals are mapped back to the input MATLAB code.

A thorough performance evaluation (in terms of execution time and cache performance) has been performed on six different algorithms that were optimized using the proposed tool. Measurements have been carried out on four different processors with ARM, MIPS and x86 ISAs and different memory hierar-

chies but also using a memory simulator. The results show that a reduction of memory accesses up to 42% has been achieved, leading to a speedup up to 3x for locality related optimizations and up to 68x for optimizations regarding the reduction of recomputations. Furthermore, the C code generated from the optimized MATLAB code, using MathWorks MATLAB Coder, is also optimized compared to the code that is generated from the original MATLAB code.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

Πολλές βελτιστοποιήσεις μπορούν να προβλεφθούν και να εφαρμοστούν αυτόματα από τους σύγχρονους μεταγλωττιστές. Υπάρχουν, ωστόσο, ορισμένοι περιορισμοί όσον αφορά αυτή τους τη δυνατότητα. Ένας μεταγλωττιστής εφαρμογής βελτιστοποιήσεων, για παράδειγμα, δεν μπορεί να επιβάλει έναν μετασχηματισμό βρόχων επανάληψης αν υπάρχουν εξαρτήσεις δεδομένων, οι οποίες επηρεάζονται από αυτόν. Ανεξάρτητα από το πόσο σύνθετοι και εξελιγμένοι μπορεί να είναι οι αλγόριθμοι εντοπισμού πιθανών βελτιστοποιήσεων, πάντα θα υπάρχει η δυνατότητα επιπλέον βελτίωσης μέσω αλλαγών που μπορούν να πραγματοποιηθούν χειροκίνητα σε υψηλότερα επίπεδα από αυτό της μεταγλώττισης (π.χ. σε επίπεδο πηγαίου κώδικα ή σε αλγοριθμικό επίπεδο). Επομένως, υπάρχει μια ανάγκη για εφαρμογές που εξετάζουν τον πηγαίο κώδικα και παρέχουν προτάσεις στον χρήστη σχετικά με πιθανές βελτιστοποιήσεις που μπορούν να εισαχθούν χειροκίνητα. Ο προγραμματιστής έχει πιο σφαιρική γνώση του αλγορίθμου που υλοποίησε από ό,τι ο μεταγλωττιστής. Έτσι, είναι και πιο ευέλικτος στο να εισάγει αλλαγές στον κώδικα, οι οποίες δε θα επηρεάσουν τη λειτουργία και την έξοδο του προγράμματος.

Η πλειονότητα των σύγχρονων προσεγγίσεων στην βελτίωση της απόδοσης εφαρμογών που κυριεύονται από βρόχους επανάληψης εστιάζουν στην παραλληλοποίηση, αξιοποιώντας σχετικές αρχιτεκτονικές. Ωστόσο, για την επίτευξη μιας πιο συνολικής βελτίωσης ενός

αλγορίθμου πρέπει να ληφθούν υπόψη με ισορροπημένο τρόπο: (1) η βελτιστοποίηση της παραλληλίας, (2) η βελτιστοποίηση της τοπικότητας αναφοράς δεδομένων και (3) η μείωση των πλεοναζόντων επανυπολογισμών (recomputations) [126]. Σε ένα τέτοιο πλαίσιο, η βελτιστοποίηση της τοπικότητας αναφοράς δεδομένων και η αξιολόγηση της συμπεριφοράς της μνήμης μπορούν να θεωρηθούν κρίσιμα ζητήματα. Δεδομένου ότι οι μεταγλωττιστές υλοποιούν ήδη αρκετές σχετικές βελτιστοποιήσεις, σε συνδυασμό με την ανάγκη για επιπρόσθετες βελτιστοποιήσεις σε υψηλότερα επίπεδα, θα μπορούσαν να εφαρμοστούν μετασχηματισμοί βρόχων επανάληψης που βελτιώνουν την τοπικότητα αναφοράς δεδομένων σε επίπεδο πηγαίου κώδικα.

Η σειριακή εκτέλεση ενός προγράμματος μπορεί να ιδωθεί ως μια *ροή προσπελάσεων* στοιχείων δεδομένων στη μνήμη, όπου η έννοια του χρόνου καθορίζεται από τον αριθμό των προσπελάσεων και όχι από κύκλους ρολογιού. Μια προσπέλαση, δηλαδή, ισοδυναμεί με μια μονάδα του χρόνου. Ο αριθμός των ξεχωριστών *στοιχείων δεδομένων* που προσπελαύνονται μεταξύ δυο διαδοχικών εμφανίσεων του ίδιου στοιχείου ονομάζεται *απόσταση επαναχρησιμοποίησης δεδομένων (reuse distance, RD).* Αυτή η έννοια είναι ισοδύναμη με την χρονική επαναχρησιμοποίηση δεδομένων κατά έναν τρόπο ανεξάρτητο από το μηχάνημα στο οποίο πραγματοποιείται η εκτέλεση. Η *ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων* [28, 63] είναι μια πολύτιμη διαδικασία μέσω της οποίας παρέχονται ποσοτικές μετρήσεις της τοπικότητας αναφοράς δεδομένων ενός προγράμματος. Οι μετρήσεις αυτές μπορούν να χρησιμοποιηθούν για την καθοδήγηση διαδικασιών βελτιστοποίησης της τοπικότητας αναφοράς δεδομένων.

Σε αυτή την διατριβή παρουσιάζεται μια σειρά εργαλείων και μεθόδων για την εξαγωγή προτάσεων βελτιστοποίησης του πηγαίου κώδικα προγραμμάτων γραμμένων σε γλώσσες υψηλού επιπέδου. Πιο συγκεκριμένα, στοχεύονται οι γλώσσες C και MATLAB, ενώ βασικό μέρος της διαδικασίας εξαγωγής αυτών των προτάσεων αποτελεί η ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων. Η C έχει χρησιμοποιηθεί ευρέως σαν γλώσσα εισόδου στην πλειονότητα των παρεμφερών εργασιών. Η MATLAB είναι μια υψηλού επιπέδου γλώσσα, η οποία απλοποιεί κατά πολύ τις πράξεις μεταξύ πινάκων (array language) και χρησιμοποιείται κατά κόρον για την πρωτοτυποποίηση αλγορίθμων από επιστήμονες και μηχανικούς. Σε αυτό το επίπεδο, οι προγραμματιστές δεν λαμβάνουν υπόψη ζητήματα που αφορούν την υλοποίηση και εστιάζουν στην συνοπτική περιγραφή αλγορίθμων σε υψηλό επίπεδο αφαιρετικότητας. Η παροχή προτά-

σεων βελτιστοποίησης του κώδικα και η εφαρμογή τους σε αυτό το επίπεδο μπορεί να έχει μεγάλη επίδραση στην ποιότητα των υλοποιήσεων χαμηλότερων επιπέδων. Αυτό είναι ιδιαίτερα σημαντικό σε περιπτώσεις όπου παράγεται αυτόματα κώδικας C ή VHDL από κώδικα MATLAB με σκοπό την υλοποίηση του σε ενσωματωμένα συστήματα ή σε συσκευές όπως τα κινητά τηλέφωνα.

1.1 Ανάγκη για βελτιστοποίηση σε υψηλό επίπεδο

Η *βελτιστοποίηση προγραμμάτων* αναφέρεται στην τροποποίηση τους ώστε να λειτουργούν με αποδοτικότερο τρόπο ή να κάνουν χρήση λιγότερων πόρων του συστήματος. Μπορεί να πραγματοποιηθεί βελτιστοποίηση σε διάφορα επίπεδα αφαιρετικότητας. Σε όσο υψηλότερο επίπεδο εισάγονται βελτιστοποιήσεις σε ένα πρόγραμμα, τόσο μεγαλύτερη είναι και η επίδραση τους σε αυτό και είναι δυσκολότερο να αλλάξουν αργότερα. Επίσης, οι βελτιστοποιήσεις υψηλού επιπέδου δεν εξαρτώνται από το σύστημα στο οποίο θα εκτελεστεί η εφαρμογή. Όσο χαμηλότερο είναι το επίπεδο αφαιρετικότητας στο οποίο εισάγονται βελτιστοποιήσεις, τόσο αυξάνεται και η εξάρτηση τους από την πλατφόρμα υλικού στην οποία θα πραγματοποιηθεί η εκτέλεση. Τα επίπεδα στα οποία μπορούν να γίνουν βελτιστοποιήσεις στο πρόγραμμα συνοψίζονται παρακάτω:

- ***Επίπεδο σχεδιασμού***. Σε αυτό το επίπεδο, που είναι και το υψηλότερο, λαμβάνονται αποφάσεις σχετικά με την γενικότερη μορφή του τελικού προγράμματος. Αυτές οι αποφάσεις μπορεί να αφορούν την γλώσσα προγραμματισμού στην οποία θα υλοποιηθεί η εφαρμογή, τον μεταγλωττιστή ή τον διερμηνέα που θα χρησιμοποιηθεί, την πλατφόρμα υλικού στην οποία θα εκτελεστεί, τον γενικότερο τύπο και δομή της εφαρμογής κ.α. Είναι προφανές ότι αυτού του είδους οι αποφάσεις έχουν μεγάλη επίδραση στην τελική εφαρμογή και δεν είναι εύκολη η μετέπειτα αλλαγή τους.

- ***Αλγοριθμικό επίπεδο***. Το επίπεδο αυτό αφορά την επιλογή των αλγόριθμων και των δομών δεδομένων που υλοποιεί και χρησιμοποιεί η εφαρμογή. Η χρονική και χωρική πολυπλοκότητα των αλγορίθμων παίζει σημαντικό ρόλο στην απόδοση της εφαρμογής. Επιπλέον, οι δομές δεδομένων που διαμοιράζο-

νται μεταξύ διαφόρων τμημάτων του προγράμματος θα πρέπει να είναι αρκετά αφαιρετικές.

- ***Επίπεδο πηγαίου κώδικα***. Αυτό το επίπεδο αναφέρεται σε βελτιστοποιήσεις που μπορούν να γίνουν στον ίδιο τον κώδικα του προγράμματος. Εδώ υπάρχει εξάρτηση από τη γλώσσα προγραμματισμού που χρησιμοποιείται, τον μεταγλωττιστή (ή τον μεταφραστή) και τη γλώσσα μηχανής που παράγεται. Αρκετές βελτιστοποιήσεις αυτού του επιπέδου μπορούν να προβλεφθούν και να εφαρμοστούν από έναν *μεταγλωττιστή βελτιστοποιήσεων*. Σε επίπεδο μεταγλώττισης πραγματοποιείται συνήθως ο εντοπισμός και η εφαρμογή *μετασχηματισμών βρόχων επανάληψης*.

- ***Επίπεδο συμβολικής γλώσσας***. Αυτό το επίπεδο είναι άμεσα συνδεδεμένο με την πλατφόρμα υλικού στην οποία θα γίνει εκτέλεση της εφαρμογής και οι βελτιστοποιήσεις που εισάγονται είναι άμεσα εξαρτώμενες από αυτή.

- ***Επίπεδο εκτέλεσης***. Το επίπεδο αυτό έχει να κάνει κυρίως με την χρήση μεταγλωττιστών JIT (just-in-time). Κατά την εκτέλεση σε ένα JIT περιβάλλον παράγεται βελτιστοποιημένος κώδικας μηχανής δυναμικά ενώ τρέχει η εφαρμογή.

Οι βελτιστοποιήσεις προγραμμάτων αφορούν κυρίως τη βελτίωση της απόδοσης όσον αφορά τον χρόνο εκτέλεσης, τη χρήση της μνήμης, τη δυνατότητα παραλληλοποίησης, την κατανάλωση ενέργειας κ.α. Συνήθως, η ταυτόχρονη βελτιστοποίηση για όλες αυτές τις πλευρές είναι ανέφικτη, γι' αυτό πραγματοποιείται στόχευση σε ορισμένες από αυτές, ανάλογα με την σπουδαιότητά τους για το εκάστοτε πρόγραμμα. Επίσης, η εφαρμογή βελτιστοποιήσεων για κάθε μια από τις πλευρές αυτές πολλές φορές πρέπει να γίνεται με ισορροπημένο τρόπο.

Η αυτόματη εφαρμογή βελτιστοποιήσεων είναι πιο διαδεδομένη στα χαμηλότερα επίπεδα, ενώ στα πολύ υψηλά επίπεδα είναι μέχρι και ανέφικτη. Εκεί πρέπει ο ίδιος ο προγραμματιστής να έχει πλήρη γνώση του προγράμματος και να εισάγει κατάλληλες βελτιστοποιήσεις σε αυτό. Σε αυτή την περίπτωση πρέπει να αποφευχθεί η πλήρης εστίαση στην βελτιστοποίηση, καθώς κάτι τέτοιο μπορεί να επηρεάσει αρνητικά τη σχεδίαση του προγράμματος και να υποβαθμιστούν άλλες πλευρές της ανάπτυξης που πρέπει να λάβει υπόψη του

ο προγραμματιστής. Ο Donald Knuth αναφέρει στην εργασία [86] ότι «η *πρόωρη βελτιστοποίηση* είναι η πηγή όλων των κακών». Μια καλή προσέγγιση που μπορεί να ακολουθηθεί στη βελτιστοποίηση σε υψηλά επίπεδα είναι να γίνεται αρχικά ανάπτυξη της εφαρμογής χωρίς να λαμβάνεται υπόψη σε μεγάλο βαθμό η εισαγωγή πιθανών βελτιστοποιήσεων. Σε επόμενη φάση, μπορεί να γίνει χρήση εργαλείων αξιολόγησης του προγράμματος και με βάση τα αποτελέσματα που παρέχουν να εισαχθούν κατάλληλες βελτιστοποιήσεις από τον προγραμματιστή. Αυτή την προσέγγιση ακολουθεί και το εργαλείο βελτιστοποίησης που προτείνεται στην παρούσα διατριβή (*MemAssist*).

1.2 Πεδίο εφαρμογής

Οι γλώσσες C και MATLAB χρησιμοποιούνται ευρέως για την υλοποίηση εφαρμογών και αλγορίθμων επεξεργασίας σήματος και εικόνας. Τέτοιου είδους αλγόριθμοι έχουν πρακτική εφαρμογή στα πεδία των τηλεπικοινωνιών, της επεξεργασίας εικόνας, της συμπίεσης δεδομένων, της επεξεργασίας βίντεο, ήχου κ.α., ενώ συνήθως φέρουν ένα σύνολο κοινών χαρακτηριστικών: (1) Αποτελούνται από πολλούς *βρόχους επανάληψης*, η πλειοψηφία των οποίων τις περισσότερες φορές είναι *εμφωλευμένοι (nested loops)*. Θα μπορούσε να ειπωθεί πως οι εν λόγω εφαρμογές *κυριεύονται* από βρόχους επανάληψης (loop dominated applications). (2) Μέσα σε αυτούς τους βρόχους πραγματοποιούνται πολλαπλές προσπελάσεις σε δομές δεδομένων τύπου πίνακα, οι οποίοι συνήθως είναι πολυδιάστατοι. (3) Τα δεδομένα των εφαρμογών είναι συνήθως *στατικά*, με την έννοια ότι μπορεί να δεσμευθεί χώρος γι' αυτά κατά την μεταγλώττιση ή γενικότερα πριν από την εκτέλεση. (4) Πολλές από τις δομές δεδομένων των εφαρμογών αυτών περιέχουν *προσωρινά δεδομένα*, τα οποία χρησιμοποιούνται από ορισμένα μόνο στάδια κάθε αλγορίθμου. Για την επίδειξη της αποτελεσματικότητας των μεθόδων, των τεχνολογιών και των εργαλείων που προτείνονται στην παρούσα διατριβή χρησιμοποιήθηκαν εφαρμογές από το πεδίο της επεξεργασίας εικόνας, οι οποίες φέρουν τα παραπάνω χαρακτηριστικά. Πιο συγκεκριμένα, επιλέχθηκαν οι εφαρμογές:

- Cavity Detector (***cavity***) [41]. Ιατρική διαγνωστική εφαρμογή για τον εντοπισμό εγκεφαλικών όγκων.

- Image Segmentation (***segm***) [59]. Μια υλοποίηση του αλγόριθμου ομαδοποίησης K-μέσων (k-means) για την τμηματοποίηση (segmentation) εικόνας. Η υλοποίηση αυτή κάνει χρήση της μεθόδου αφαιρετικής ομαδοποίησης (subtractive clustering) για τον εντοπισμό των βέλτιστων αρχικών κέντρων για κάθε ομάδα.

- Edge Detect (***edge***). Αλγόριθμος ανίχνευσης ακμών (edge detection) από την σουίτα UTDSP [128], ο οποίος κάνει χρήση της διαδικασίας συνέλιξης (convolution) και τελεστών Sobel.

- DCT/JPEG Preview (***dct***). Μια υλοποίηση της μεθόδου απωλεστικής συμπίεσης JPEG [151], η οποία εφαρμόζει τον *διακριτό μετασχηματισμό συνημιτόνου (discrete cosine transform, DCT)* και τα βήματα της κβαντοποίησης, για έναν δοθέντα πίνακα κβαντοποίησης και έπειτα αντιστρέφει τη διαδικασία. Σκοπός της συγκεκριμένης εφαρμογής είναι η παροχή μιας προεπισκόπησης της συμπιεσμένης εικόνας για την αξιολόγηση της ποιότητας της για έναν πίνακα κβαντοποίησης.

- 2D DWT/IDWT (***dwt***,***dwtlegall***). Ο *διακριτός μετασχηματισμός κυματιδίων (2D discrete wavelet transform, DWT)* όπως υλοποιείται στο πρότυπο συμπίεσης εικόνας JPEG 2000 [137]. Η εικόνα εισόδου διασπάται με τον DWT και έπειτα ανασυντίθεται με τη διαδικασία του αντίστροφου DWT (inverse DWT, IDWT). Ο αλγόριθμος έχει δοκιμαστεί με τα DWT φίλτρα CDF 9/7 και LeGall 5/3.

Εφαρμογές όπως αυτές, μπορούν να επωφεληθούν αρκετά από βελτιστοποιήσεις σε υψηλό επίπεδο, οι οποίες αφορούν την αναδιάταξη του πηγαίου κώδικα. Πιο συγκεκριμένα, από μετασχηματισμούς βρόχων επανάληψης, με σκοπό τη βελτίωση της τοπικότητας αναφοράς δεδομένων.

Μια βασική διαφορά της C με την γλώσσα του MATLAB, σε ό,τι αφορά τα χαρακτηριστικά που αναφέρθηκαν παραπάνω, είναι ότι στην MATLAB δεν είναι κοινή πρακτική η χρήση βρόχων επανάληψης. Η γλώσσα παρέχει τελεστές που δρουν κατευθείαν σε πολυδιάστατους πίνακες χωρίς να είναι αναγκαία η χρήση βρόχων. Εσωτερικά, βέβαια, στον μεταφραστή του MATLAB οι τελεστές αυτοί υλοποιούνται με βρόχους επανάληψης και διάφορες τεχνικές παραλληλοποίησης. Επειδή οι μεθοδολογίες που αναφέρονται στην παρούσα

διατριβή βασίζονται κυρίως σε εφαρμογές που κυριεύονται από βρόχους επανάληψης, όλες οι υλοποιήσεις αλγορίθμων που έγιναν σε MATLAB έχουν παρόμοια μορφή με τις αντίστοιχες C εκδόσεις τους και περιλαμβάνουν βρόχους επανάληψης.

1.3 Πλεονεκτήματα βελτιστοποίησης σε υψηλό επίπεδο

Οι μετασχηματισμοί βρόχων επανάληψης ενδέχεται να αλλάξουν τη σειρά με την οποία εκτελούνται βασικές λειτουργίες / υπολογισμοί σε ένα πρόγραμμα. Πρόκειται για βελτιστοποιήσεις πολύ μεγάλης σημασίας για την επίτευξη παραλληλίας και την καλύτερη επαναχρησιμοποίηση της μνήμης. Το μεγαλύτερο πλεονέκτημα από την εφαρμογή τέτοιου είδους βελτιστοποιήσεων σε υψηλό επίπεδο είναι ότι τις περισσότερες φορές είναι ανεξάρτητες από το μηχάνημα στο οποίο θα πραγματοποιηθεί η εκτέλεση. Με αυτό τον τρόπο, μόλις γίνεται μετάφραση σε κάποιο χαμηλότερο επίπεδο τα οφέλη των βελτιστοποιήσεων συνήθως εξακολουθούν να υπάρχουν.

Η μεθοδολογία που ακολουθείται στην παρούσα διατριβή είναι να παρέχονται προτάσεις βελτιστοποιήσεων αυτόματα στον χρήστη και αυτός να είναι υπεύθυνος για την (χειροκίνητη) εφαρμογή τους στον πηγαίο κώδικα. Ο χρήστης μπορεί να υλοποιήσει βελτιστοποιήσεις που δεν μπορεί να υλοποιήσει αυτόματα ένας μεταγλωττιστής. Για παράδειγμα, ίσως να μην μπορεί να γίνει εφαρμογή κάποιον μετασχηματισμών βρόχων επανάληψης αν υπάρχουν πιθανές εξαρτήσεις δεδομένων στο πρόγραμμα. Ο προγραμματιστής, αντιθέτως, μπορεί να κάνει γενικότερες αλλαγές σε αλγοριθμικό επίπεδο, οι οποίες να καταστήσουν εφικτούς τους μετασχηματισμούς αυτούς. Τέτοιες αλλαγές είναι συνήθως ανέφικτες σε χαμηλό επίπεδο. Από την άλλη, όσο ψηλότερο είναι το επίπεδο αφαιρετικότητας, τόσο δυσκολότερη είναι η αυτοματοποίηση της εφαρμογής βελτιστοποιήσεων.

1.4 Σύντομη επισκόπηση σχετικής βιβλιογραφίας

Η πλειοψηφία όσων αναφέρονται στην παρούσα διατριβή έχουν υλοποιηθεί στο εργαλείο βελτιστοποίησης *MemAssist*, το οποίο παρουσιάζεται αναλυτικά στο κεφάλαιο 3. Το MemAssist προτείνει στο χρήστη μετασχηματισμούς βρόχων επανάληψης για πηγαίο κώδικα C ή MATLAB, οι οποίοι έχουν ως στόχο την βελτιστοποίηση της χρο-

νικής τοπικότητας αναφοράς δεδομένων και τη συνακόλουθη καλύτερη αξιοποίηση των λανθανουσών μνημών του συστήματος. Ένα πολύ σημαντικό κομμάτι της διαδικασίας εύρεσης αυτών των προτάσεων βελτιστοποίησης είναι η ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων. Ένα παρεμφερές με το MemAssist εργαλείο είναι το SLO [29, 30, 31, 32]. Κάνει και αυτό ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων και προτείνει παρόμοιους μετασχηματισμούς. Στη ενότητα 3.8 γίνεται μια λεπτομερής αναφορά στα δύο εργαλεία, αναλύοντας τις ομοιότητες και τις διαφορές τους. Υπάρχουν, ωστόσο, αρκετά ακόμα εργαλεία που έχουν ως στόχο την αξιολόγηση του πηγαίου κώδικα και την παροχή προτάσεων βελτιστοποίησης, ή ακόμα και την εφαρμογή βελτιστοποιήσεων, για την καλύτερη λειτουργία των λανθανουσών μνημών [8, 25, 26, 29, 30, 31, 32, 58, 66, 121, 133, 139, 159]. Μια κλασσική κατηγορία εργαλείων αξιολόγησης της απόδοσης ενός αλγορίθμου σε συγκεκριμένες ιεραρχίες μνήμης, είναι οι προσομοιωτές μνήμης [62, 65, 111]. Σε άλλες εργασίες επιχειρείται η πρόβλεψη του ρυθμού αστοχίας των λανθανουσών μνημών χωρίς να γίνεται προσομοίωση [8, 25, 26, 66, 133, 159]. Τα εργαλεία που προτείνονται συνήθως σε αυτές τις εργασίες, λειτουργούν κάνοντας ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων.

Ο υπολογισμός αποστάσεων επαναχρησιμοποίησης δεδομένων δεν είναι απλή διαδικασία και απαιτεί εξειδικευμένους αλγόριθμους για την πραγματοποίησή του [10, 24, 64, 103, 115, 142, 162]. Οι αλγόριθμοι αυτοί συνήθως κάνουν χρήση διάφορων δομών δεδομένων, όπως για παράδειγμα στοιβών και δέντρων διαφόρων τύπων. Έχουν αναπτυχθεί επίσης και κάποιοι παράλληλοι αλγόριθμοι για την επίλυση αυτού του προβλήματος [51, 114], καθώς και προσεγγίσεις που πραγματοποιούν κατά προσέγγιση υπολογισμό των αποστάσεων [64, 120, 135, 136, 162] και άλλες που βασίζονται στην εφαρμογή δειγματοληψίας [29, 160].

Για να καταστεί εφικτή η υποστήριξη της MATLAB σαν γλώσσα εισόδου στο MemAssist, κατασκευάστηκε ένας MATLAB-σε-C μεταγλωττιστής. Ο μεταγλωττιστής αυτός ονομάζεται *MAFE* και παρουσιάζεται αναλυτικά στο κεφάλαιο 5. Οι περισσότερες εργασίες γύρω από τη μεταγλώττιση κώδικα MATLAB έχουν να κάνουν με την παραγωγή βέλτιστου κώδικα χαμηλού επιπέδου για υλοποιήσεις υλικού [16, 17, 18, 19, 73, 140, 141]. Πολλές είναι όμως και οι εργασίες που ασχολούνται με την παραγωγή κώδικα C [9, 23, 33, 34, 35, 43, 82, 88, 89, 90, 117, 119, 123, 124, 134] και FORTRAN [55, 56, 57, 84, 97].

Η ανάπτυξη των MemAssist και MAFE έγινε με τη βοήθεια του γεννήτορα εμπρόσθιων τμημάτων μεταγλωττιστών *FEgen* και της γλώσσας ειδικού σκοπού CastQL για την εφαρμογή ερωτημάτων σε πηγαίο κώδικα. Τα δύο αυτά εργαλεία περιγράφονται στο κεφάλαιο 4. Υπάρχουν αρκετές παρόμοιες τεχνολογίες οι οποίες κλίνουν είτε προς την ανάλυση πηγαίου κώδικα [12, 13, 27, 46, 50, 52, 54, 76, 102, 150], είτε προς το μετασχηματισμό πηγαίου κώδικα [22, 36, 48, 53, 83, 85, 146].

1.5 Συνοπτική περιγραφή της διατριβής και καινοτομίες

Για την παρουσίαση της ανάλυσης απόστασης επαναχρησιμοποίησης δεδομένων με μεγαλύτερη συνοχή, καθώς και για την ευκολότερη κατανόησή της, περιγράφονται εν συντομία κάποιες βασικές έννοιες που χρησιμοποιούνται στο υπόλοιπο της διατριβής:

- Μια *προσπέλαση μνήμης* είναι μια προσπέλαση σε ένα συγκεκριμένο στοιχείο της μνήμης, η οποία συμβαίνει κατά την εκτέλεση.
- Μια *αναφορά μνήμης* ή *αναφορά δεδομένων* είναι το σημείο στον πηγαίο κώδικα, το οποίο προκαλεί προσπελάσεις μνήμης. Στην γλώσσα C για παράδειγμα, θα μπορούσε να είναι μια αναφορά σε μια βαθμωτή μεταβλητή ή σε ένα από τα στοιχεία ενός πίνακα.
- Η *ροή προσπελάσεων μνήμης* περιλαμβάνει όλες τις προσπελάσεις μνήμης που προκλήθηκαν κατά την εκτέλεση ενός προγράμματος. Οι προσπελάσεις αυτές είναι χρονικά ταξινομημένες μέσα στην ροή.
- Μια *επαναχρησιμοποίηση δεδομένων* συμβαίνει μεταξύ δυο διαδοχικών προσπελάσεων προς το ίδιο στοιχείο της μνήμης.
- Ένα *ζεύγος επαναχρησιμοποίησης* ή *ζεύγος προσπελάσεων* περιλαμβάνει τις δύο προσπελάσεις μιας επαναχρησιμοποίησης δεδομένων. Η πρώτη χρονικά ονομάζεται *προσπέλαση-πηγή* και η δεύτερη ονομάζεται *προσπέλαση-στόχος*. Οι αντίστοιχοι όροι για την αναφορά μνήμης η οποία σχετίζεται με την επαναχρησιμοποίηση είναι το *ζεύγος αναφορών*, η *αναφορά-πηγή* και η *αναφορά-στόχος*.

- *Χρονική απόσταση επαναχρησιμοποίησης δεδομένων* [136] ή *απόλυτη απόσταση επαναχρησιμοποίησης δεδομένων* [133] ονομάζεται ο αριθμός των προσπελάσεων που μεσολαβούν μεταξύ των δύο προσπελάσεων ενός ζεύγους επαναχρησιμοποίησης.

- Η *απόσταση επαναχρησιμοποίησης δεδομένων* [28, 63] ή *μοναδική απόσταση επαναχρησιμοποίησης δεδομένων* [133] χαρακτηρίζει τον αριθμό των ξεχωριστών στοιχείων που προσπελαύνονται ανάμεσα από την προσπέλαση-πηγή και την προσπέλαση-στόχο.

- Η κατανομή των αποστάσεων επαναχρησιμοποίησης για μια εκτέλεση μπορεί να ιδωθεί σαν ένα ιστόγραμμα, στο οποίο ο άξονας X δείχνει τη συνολική απόσταση επαναχρησιμοποίησης ενός στοιχείου της μνήμης ή ενός ζεύγους αναφορών και ο άξονας Y δείχνει τον συνολικό αριθμό επαναχρησιμοποιήσεων. Ένα τέτοιο ιστόγραμμα ονομάζεται *ιστόγραμμα αποστάσεων επαναχρησιμοποίησης δεδομένων (reuse distance histogram, RDH).*

Οι παραπάνω έννοιες περιγράφονται και στην ενότητα 3.1 με πιο συνεκτικό τρόπο. Για την επίτευξη καλής χρονικής τοπικότητας αναφοράς δεδομένων θα πρέπει οι προσπελάσεις προς το ίδιο στοιχείο να συμβαίνουν σε πολύ σύντομα χρονικά διαστήματα μεταξύ τους χωρίς να παρεμβαίνει μεγάλος αριθμός προσπελάσεων προς άλλα στοιχεία [79]. Ανάλογα με το μέγεθος μιας λανθάνουσας μνήμης, μπορεί να ειπωθεί ότι η καλή χρονική τοπικότητα αναφοράς δεδομένων μπορεί να προσφέρει μικρότερο αριθμό αστοχιών της λανθάνουσας μνήμης. Τόσο η χρονική, όσο και η κανονική / μοναδική απόσταση επαναχρησιμοποίησης δεδομένων έχουν προταθεί ως τρόποι ποσοτικής μέτρησης της τοπικότητας αναφοράς δεδομένων [28, 120, 136, 162]. Η κανονική απόσταση επαναχρησιμοποίησης όμως έχει μια ξεκάθαρη σύνδεση με τη συμπεριφορά της λανθάνουσας μνήμης. Μπορεί να προσδιορίσει τον ρυθμό αστοχίας για μια λανθάνουσα μνήμη με πλήρως συσχετιστική οργάνωση και πολιτική αντικατάστασης με βάση τον χρόνο τελευταίας προσπέλασης (LRU), συγκρίνοντας τις αποστάσεις επαναχρησιμοποίησης των στοιχείων δεδομένων με το μέγεθος της λανθάνουσας μνήμης. Οι Beyls και D'Hollander [28] έδειξαν ότι η απόσταση επαναχρησιμοποίησης μπορεί να προβλέψει επιτυχώς τον ρυθμό αστοχίας και για λανθάνουσες μνήμες με διαφορετικές πολιτικές αντικατάστασης και χαρτογράφηση. Το εργαλείο MemAssist πραγματοποιεί ανάλυση της απόστασης επαναχρησιμοποίησης

δεδομένων και εκμεταλλευόμενο όσα αναφέρθηκαν, προτείνει στον χρήστη μετασχηματισμούς βρόχων επανάληψης για τη μείωση των αποστάσεων επαναχρησιμοποίησης. Πιο συγκεκριμένα, διατίθενται δύο τύποι προτάσεων για την εφαρμογή μετασχηματισμών, οι οποίοι βασίζονται στη λογική ότι κάθε αναφορά-πηγή πρέπει να έρθει πιο κοντά με την αναφορά-στόχο που της αντιστοιχεί:

1. Όταν η αναφορά-πηγή περικλείεται από διαφορετικό βρόχο επανάληψης από την αναφορά-στόχο και οι βρόχοι αυτοί βρίσκονται σε διαφορετικές φωλιές (nests), προτείνεται η συγχώνευση (fusion) των βρόχων.

2. Αν η δύο αναφορές μνήμης ενός ζεύγους αναφορών βρίσκονται στην ίδια φωλιά βρόχων, τότε προτείνεται η εφαρμογή κάποιου μετασχηματισμού υπερκόμβων (tiling) στη συγκεκριμένη φωλιά.

Το MemAssist παρέχει τις προτάσεις αυτές σαν μια λίστα, ταξινομημένη με βάση τις αποστάσεις επαναχρησιμοποίησης και τον αριθμό των επαναχρησιμοποιήσεων. Προτάσεις που σχετίζονται με ζεύγη αναφορών με μικρές αποστάσεις ή λίγες επαναχρησιμοποιήσεις δεν έχει νόημα να ληφθούν υπόψη.

1.5.1 Καινοτομίες και συνεισφορές

Στην παρούσα διατριβή γίνονται οι ακόλουθες συνεισφορές:

- Παρουσιάζονται συγκεντρωτικά και με συνεκτικό τρόπο οι αλγόριθμοι υπολογισμού αποστάσεων επαναχρησιμοποίησης δεδομένων που έχουν προταθεί κατά καιρούς. Ο υπολογισμός αυτών των αποστάσεων δεν είναι απλή διαδικασία και είναι αναγκαία η χρήση εξειδικευμένων αλγορίθμων για την αποτελεσματικότερη πραγματοποίησή του. Κατά την παρουσίαση των αλγορίθμων αυτών γίνεται και αναλυτική σύγκρισή τους.

- Αναπτύχθηκε το εργαλείο MemAssist και ενσωματώθηκε σε ένα δημοφιλές περιβάλλον ανάπτυξης εφαρμογών. Διατίθεται σαν επέκταση (extension) για το Microsoft Visual Studio[1], παρέχοντας ένα ενοποιημένο περιβάλλον για την ανάπτυξη και την

[1]https://visualstudio.microsoft.com/

βελτιστοποίηση εφαρμογών. Το προτεινόμενο αυτό εργαλείο είναι διαθέσιμο και σαν διαδικτυακή εφαρμογή ενώ αναλύει εφαρμογές γραμμένες σε C ή MATLAB.

- Το βασικότερο βήμα της ροής λειτουργίας του MemAssist, που είναι η ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων, εκτελείται αποκλειστικά για κώδικα C. Για τον λόγο αυτό έχει αναπτυχθεί και ένας MATLAB-σε-C μεταγλωττιστής που ονομάζεται *MAFE*. Ο μεταγλωττιστής αυτός έχει τη δυνατότητα να συσχετίζει τις μεταβλητές του MATLAB κώδικα που δέχεται με κομμάτια του κώδικα C που παράγει. Έπειτα πραγματοποιείται η διαδικασία της ανάλυσης των αποστάσεων επαναχρησιμοποίησης δεδομένων στον C κώδικα και οι προτάσεις που παράγονται αντιστοιχίζονται στον MATLAB κώδικα της εισόδου.

- Προτείνεται μια μέθοδος για την αυτόματη επιλογή κατάλληλων μετασχηματισμών βρόχων επανάληψης, η οποία περιγράφεται λεπτομερώς.

- Πέρα από την παρουσίαση της σχετικής βιβλιογραφίας και παρόμοιων εργαλείων, γίνεται και μια αναλυτική σύγκριση του MemAssist με έναν παρεμφερή βελτιστοποιητή. Εξετάζονται τα χαρακτηριστικά των δύο εργαλείων, συμπεριλαμβανομένων και των προσεγγίσεων του καθενός για την επιλογή βελτιστοποιήσεων.

- Παρουσιάζονται επίσης κάποιες επιπλέον μετρικές για την εύρεση σημείων ενδιαφέροντος μέσα στον πηγαίο κώδικα, στα οποία ενδέχεται να πραγματοποιείται έντονη επεξεργασία δεδομένων.

- Πραγματοποιήθηκε βελτιστοποίηση έξι εφαρμογών επεξεργασίας εικόνας, κάνοντας χρήση του MemAssist, αποδεικνύοντας πειραματικά την αποτελεσματικότητά του.

- Παρουσιάζεται η γλώσσα προγραμματισμού ειδικού σκοπού *CastQL*. Μια γλώσσα εφαρμογής ερωτημάτων (query language) σε πηγαίο κώδικα που λειτουργεί σε συνδυασμό με κάποια απαραίτητα βήματα που πρέπει να ακολουθηθούν κατά την ανάπτυξη ερωτημάτων εξαρτώμενων από τη γλώσσα εισόδου. Τα βήματα αυτά ξεκινούν από την διατύπωση του προβλήματος

και φτάνουν στην υλοποίηση. Η CastQL λειτουργεί πάνω σε μια AST αναπαράσταση που λέγεται *cAST (contextual abstract syntax tree)*. Πρόκειται για μια εσωτερική/ενσωματωμένη (internal/embedded) γλώσσα προγραμματισμού ειδικού σκοπού [69] που κάνει χρήση της C++ ως γλώσσα υποδοχής.

- Αναπτύχθηκε ένας αυτόματος γεννήτορας εμπρόσθιων τμημάτων μεταγλωττιστών, με το όνομα *FEgen*, ο οποίος παράγει έναν λεκτικό και συντακτικό αναλυτή για οποιαδήποτε δοθείσα γραμματική κάνει χρήση της μορφής συμβολισμού BNF. Ο κώδικας που περιγράφει το εμπρόσθιο τμήμα και τις προδιαγραφές της cAST αναπαράστασης παράγεται αυτόματα και δεν απαιτείται η συγγραφή C++ κώδικα από τον χρήστη. Το FEgen πρακτικά καθιστά εφικτή την χρήση της CastQL για οποιαδήποτε γλώσσα εισόδου.

1.6 Διάρθρωση της διατριβής

Τα υπόλοιπα κεφάλαια της διατριβής οργανώνονται ως ακολούθως:

- Το κεφάλαιο 2 παρέχει μια ανασκόπηση της σχετικής βιβλιογραφίας. Εξετάζονται διάφορα εργαλεία αξιολόγησης της ιεραρχίας μνήμης και εφαρμογής σχετικών βελτιστοποιήσεων. Γίνεται μια λεπτομερής περιγραφή των αλγόριθμων υπολογισμού αποστάσεων επαναχρησιμοποίησης δεδομένων. Ακόμα, εξετάζονται εργασίες που αφορούν την κατασκευή εργαλείων ανάλυσης και χειρισμού πηγαίου κώδικα. Τέλος, γίνεται μια αναδρομή στην ερευνητική δραστηριότητα που αφορά τη μεταγλώττιση κώδικα MATLAB και παρεμφερών γλωσσών.

- Στο κεφάλαιο 3 περιγράφεται αναλυτικά το εργαλείο βελτιστοποίησης της τοπικότητας αναφοράς δεδομένων, MemAssist. Εξηγούνται οι αρχές πάνω στις οποίες βασίζεται, καθώς και ο τρόπος που λειτουργεί εσωτερικά. Στο τέλος του κεφαλαίου συγκρίνεται με ένα παρεμφερές εργαλείο.

- Το κεφάλαιο 4 αναφέρεται στο εργαλείο αυτόματης παραγωγής εμπρόσθιων τμημάτων μεταγλωττιστών, FEgen και στην γλώσσα ειδικού σκοπού, για την εφαρμογή ερωτημάτων σε πηγαίο κώδικα, CastQL. Παρέχεται μια γενική περιγραφή και των

δύο εργαλείων, ενώ για την CastQL γίνεται αναφορά στα δομικά της στοιχεία και παρουσιάζεται ένα απλό παράδειγμα χρήσης της.

- Στο κεφάλαιο 5 περιγράφεται ο μεταγλωττιστής MAFE και τα στάδια λειτουργίας του, ενώ γίνεται αναφορά και στον κώδικα που παράγει.
- Στο κεφάλαιο 6 αξιολογούνται τα εργαλεία MemAssist, FEgen και CastQL. Γίνεται χρήση του MemAssist για την βελτιστοποίηση έξι εφαρμογών και σχολιάζονται τα αποτελέσματα. Τα FEgen και CastQL δοκιμάζονται σε ορισμένα πρακτικά σενάρια χρήσης.
- Το κεφάλαιο 7 περιλαμβάνει τον επίλογο και προτάσεις για μελλοντική επέκταση της παρούσας δουλειάς.

Περιλαμβάνονται επίσης και τα εξής παραρτήματα:

- Στο παράρτημα Α΄ παρουσιάζονται οι παράμετροι που δέχονται οι διεπαφές γραμμής εντολών των εργαλείων που αναπτύχθηκαν στα πλαίσια της παρούσας διατριβής.
- Στο παράρτημα Β΄ παρουσιάζονται διάφορα παραδείγματα εκτέλεσης των αλγόριθμων υπολογισμού αποστάσεων επαναχρησιμοποίησης δεδομένων, που αναφέρονται στο κεφάλαιο 2.
- Στο παράρτημα Γ΄ παρατίθενται αναλυτικά οι μετρήσεις που χρησιμοποιήθηκαν για την πειραματική αξιολόγηση στην ενότητα 6.1.
- Το παράρτημα Δ΄ απαριθμεί τις επιστημονικές δημοσιεύσεις που σχετίζονται με την διατριβή.

Κεφάλαιο 2

Ανασκόπηση σχετικής βιβλιογραφίας

Η ερευνητική δραστηριότητα στο χώρο της βελτιστοποίησης αλγορίθμων ώστε να κάνουν καλύτερη χρήση της ιεραρχίας μνήμης επικεντρώνεται κυρίως στην εφαρμογή μετασχηματισμών για την βελτίωση της τοπικότητας αναφοράς δεδομένων [40, 87, 92, 105, 155]. Πληθώρα εργαλείων και μεθόδων που στοχεύουν στην αξιολόγηση αλγορίθμων με βάση τις επιδόσεις τους ως προς την τοπικότητα αναφοράς δεδομένων έχουν παρουσιαστεί κατά καιρούς. Σε αυτά περιλαμβάνονται: εργαλεία πρόβλεψης του ρυθμού αστοχίας της λανθάνουσας μνήμης [8, 25, 26, 66, 133, 159], εργαλεία οπτικοποίησης της λανθάνουσας μνήμης [58, 121], εργαλεία εξομοίωσης της ιεραρχίας μνήμης [62, 65, 111], καθώς και εργαλεία που παρέχουν προτάσεις στο χρήστη σχετικά με την βελτιστοποίηση των αλγορίθμων [29, 30, 31, 32, 139] ή και κάποια που πραγματοποιούν αυτόματη εφαρμογή βελτιώσεων. Όλοι οι κύριοι σύγχρονοι μεταγλωττιστές της γλώσσας C, για παράδειγμα, εφαρμόζουν βελτιώσεις που αφορούν την τοπικότητα αναφοράς δεδομένων.

Σε κάποιες από αυτές τις εργασίες και τα εργαλεία πραγματοποιείται εξομοίωση της εκτέλεσης μιας εφαρμογής σε μια προκαθορισμένη ιεραρχία μνήμης για τη μέτρηση της τοπικότητας αναφοράς δεδομένων της εφαρμογής αυτής. Η μέθοδος αυτή περιορίζει την αξιολόγηση της εφαρμογής όσον αφορά την τοπικότητα αναφοράς σε συγκεκριμένο μηχάνημα και ιεραρχία μνήμης. Η επαναχρησιμοποίηση

δεδομένων είναι όμως ένα έμφυτο χαρακτηριστικό ενός προγράμματος και δεν εξαρτάται από τις ιδιότητες του συστήματος στο οποίο εκτελείται. Μια άλλη μέθοδος για την μέτρηση της τοπικότητας αναφοράς, η οποία δεν εξαρτάται από το μηχάνημα εκτέλεσης, είναι η *ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων*. Αυτού του είδους η ανάλυση αφορά τη μέτρηση του αριθμού των μοναδικών στοιχείων δεδομένων του προγράμματος που μεσολαβούν μεταξύ μιας επαναχρησιμοποίησης. Πολλά από τα παραπάνω εργαλεία πραγματοποιούν τέτοια ανάλυση κατά τη λειτουργία τους, μεταξύ τους και το MemAssist. Όντας μια υπολογιστικά δαπανηρή διαδικασία, διάφορες μέθοδοι και αλγόριθμοι έχουν προταθεί για την αποτελεσματικότερη μέτρηση της απόστασης επαναχρησιμοποίησης δεδομένων [24, 103]. Οι μέθοδοι αυτές κάνουν χρήση διάφορων δομών δεδομένων όπως στοιβών, δέντρων κλπ. [10, 24, 64, 103, 115, 142, 162] είτε εκμεταλλεύονται παράλληλες αρχιτεκτονικές [51, 114].

Ένα από τα βασικά χαρακτηριστικά του προτεινόμενου εργαλείου βελτιστοποίησης είναι η δυνατότητα να δέχεται σαν είσοδο κώδικα σε γλώσσα MATLAB, πέραν της C. Το MemAssist πραγματοποιεί ανάλυση σε C κώδικα και όχι σε MATLAB. Για το λόγο αυτό αναπτύχθηκε ένας MATLAB-σε-C μεταγλωττιστής όπου προηγείται της ανάλυσης επαναχρησιμοποίησης δεδομένων στη ροή του MemAssist. Τα αποτελέσματα της ανάλυσης αντιστοιχίζονται έπειτα στον αρχικό MATLAB κώδικα. Εκτός από τη χρήση του στο πλαίσιο λειτουργίας του MemAssist, ο συγκεκριμένος μεταγλωττιστής έχει τη δυνατότητα να παράγει C κώδικα με δυναμικά δεσμευμένους πίνακες, ο οποίος έχει κάποιες ιδιαιτερότητες και πλεονεκτήματα σε σχέση με αντίστοιχες υλοποιήσεις. Η έρευνα στον τομέα της μεταγλώττισης MATLAB κώδικα σε κώδικα γλωσσών χαμηλότερων επιπέδων εστιάζει κυρίως στην παραγωγή βέλτιστου κώδικα για υλοποιήσεις υλικού [16, 17, 18, 19, 73, 140, 141]. Πολλές είναι όμως και οι εργασίες που αφορούν την παραγωγή κώδικα σε επίπεδο γλώσσας C [9, 23, 33, 34, 35, 43, 82, 88, 89, 90, 117, 119, 123, 124, 134]. Στην παρούσα διατριβή είναι η πρώτη φορά που ένας μεταγλωττιστής τέτοιου είδους χρησιμοποιείται για την ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων σε MATLAB εφαρμογές. Η μόνη σχετική εργασία είναι αυτή των Chauhan και Shei [42], στην οποία προβλέπουν αποστάσεις επαναχρησιμοποίησης δεδομένων σε MATLAB εφαρμογές αλλά μέσω στατικής ανάλυσης κώδικα.

Για την ανάπτυξη του MemAssist έγινε χρήση μιας συλλογής εργαλείων και τεχνολογιών που δημιουργήθηκαν στα πλαίσια της πα-

ρούσας διατριβής. Τα εργαλεία αυτά περιλαμβάνουν μια γλώσσα προγραμματισμού ειδικού σκοπού για την εφαρμογή ερωτημάτων σε πηγαίο κώδικα και έναν αυτόματο γεννήτορα εμπρόσθιων τμημάτων μεταγλωττιστών και εργαλείων ανάλυσης κώδικα. Υπάρχουν αρκετές παρόμοιες τεχνολογίες οι οποίες κλίνουν είτε προς την ανάλυση πηγαίου κώδικα [12, 13, 27, 46, 50, 52, 54, 76, 102, 150], είτε προς το μετασχηματισμό πηγαίου κώδικα [22, 36, 48, 53, 83, 85, 146]. Η προτεινόμενη συλλογή εργαλείων στοχεύει στη γρήγορη ανάπτυξη εργαλείων και των δύο περιοχών.

Στο κεφάλαιο αυτό γίνεται μια ανασκόπηση και συγκριτική μελέτη των παραπάνω εργασιών και γενικότερα της σχετικής βιβλιογραφίας. Αρχικά, στην ενότητα 2.1 παρουσιάζονται διάφορες εργασίες που αφορούν τη μεταγλώττιση MATLAB κώδικα σε χαμηλότερα επίπεδα αφαιρετικότητας και κυρίως σε κώδικα επιπέδου C. Έπειτα, στην ενότητα 2.2 γίνεται εισαγωγή στο πεδίο της ανάλυσης της απόστασης επαναχρησιμοποίησης δεδομένων και παρουσιάζονται αναλυτικά όλοι οι γνωστοί σχετικοί αλγόριθμοι και μέθοδοι που έχουν προταθεί για τον αποδοτικότερο υπολογισμό αποστάσεων επαναχρησιμοποίησης δεδομένων. Παράλληλα γίνεται και σύγκριση των μεθόδων αυτών. Στη συνέχεια, στην ενότητα 2.3 γίνεται αναφορά και συγκριτική μελέτη ενός μεγάλου εύρους εργαλείων βελτιστοποίησης της τοπικότητας αναφοράς δεδομένων λανθάνουσας μνήμης. Η σύγκριση που πραγματοποιείται μεταξύ όλων των εργαλείων είναι πιο γενική, ενώ μια εκτενέστερη σύγκριση είναι εφικτή μεταξύ του προτεινόμενου εργαλείου και ενός παρόμοιου εργαλείου, που λέγεται SLO [29, 30, 31, 32], λόγω των αρκετά όμοιων δυνατοτήτων που παρέχουν. Η σύγκριση αυτή παρουσιάζεται στο κεφάλαιο 3, όπου γίνεται και εκτενής περιγραφή του εργαλείου MemAssist και των λειτουργιών του. Ακόμα, στην ενότητα 2.4 παρουσιάζονται διάφορες τεχνολογίες και εργαλεία ανάλυσης πηγαίου κώδικα και συγκρίνονται με την αντίστοιχη συλλογή εργαλείων που προτείνεται στην παρούσα διατριβή.

2.1 Μεταγλώττιση MATLAB-σε-C

Σε αυτή την ενότητα γίνεται μια επισκόπηση των υπαρχουσών εργασιών σχετικά με τη μεταγλώττιση της γλώσσας MATLAB και εναλλακτικών της όπως η Scilab. Οι εργασίες αυτές αφορούν κυρίως τη μεταγλώττιση σε χαμηλότερα επίπεδα και ειδικότερα στο επίπεδο

γλωσσών όπως οι C/C++ και η Fortran.

Η MathWorks παρέχει ένα σύνολο εμπορικών εργαλείων μεταγλώττισης MATLAB κώδικα και διαγραμμάτων Simulink σε κώδικα γλωσσών χαμηλότερου επιπέδου. Πιο συγκεκριμένα, το MATLAB Coder [5] είναι το MATLAB-σε-C εργαλείο της MathWorks. Ο παραγόμενος C και C++ κώδικας μπορεί να χρησιμοποιηθεί αυτούσιος ή να μεταγλωττιστεί κατευθείαν σε στατική ή δυναμική C/C++ βιβλιοθήκη ή να μετατραπεί σε αρχείο MEX. Όσον αφορά τη διαχείριση μνήμης, το MATLAB Coder μπορεί να παράξει κώδικα που κάνει στατική δέσμευση μνήμης καθώς και κώδικα που κάνει δυναμική δέσμευση μνήμης. Το Simulink Coder [7] είναι το αντίστοιχο εργαλείο για παραγωγή C/C++ κώδικα από διαγράμματα Simulink. Το Embedded Coder [1] παράγει C/C++ κώδικα για χρήση σε ενσωματωμένους επεξεργαστές ενώ μπορεί να χρησιμοποιηθεί σε συνδυασμό με το MATLAB Coder αλλά και με το Simulink Coder. Τέλος, το HDL Coder [2] παράγει κώδικα VHDL και Verilog για χρήση στον προγραμματισμό FPGAs και τη σχεδίαση ASIC κυκλωμάτων. Όλα τα εργαλεία μεταγλώττισης της MathWorks είναι διαθέσιμα ως πρόσθετα στο περιβάλλον του MATLAB.

Στις εργασίες [88] και [90] παρουσιάζεται ένας MATLAB-σε-C μεταγλωττιστής ο οποίος παράγει κώδικα που περιέχει διανυσματικές εντολές SIMD (single instruction - multiple data, μια εντολή - πολλά δεδομένα). Οι εντολές αυτές αναπαριστώνται από εσωτερικές (intrinsic) συναρτήσεις. Επιπλέον, ο εν λόγω μεταγλωττιστής κάνει χρήση ενός παραμετρικού μοντέλου για την περιγραφή των SIMD εντολών ενός επεξεργαστή-στόχου. Έχει έτσι τη δυνατότητα να παράξει κώδικα για οποιαδήποτε αρχιτεκτονική μπορεί να περιγραφεί με το συγκεκριμένο μοντέλο. Ενδεικτικά, στην διδακτορική διατριβή του Ιωάννη Λατίφη [89] παρουσιάζονται μοντέλα για τις αρχιτεκτονικές Bot, tinyBot, ARM (NEON) και x86 (SSE) ενώ παράγεται και κώδικας για κάθε μια από αυτές.

Ο Matlab2cpp [117] είναι ένας μεταφραστής για το MATLAB που παράγει C++ κώδικα. Ο παραγόμενος C++ κώδικας κάνει χρήση της βιβλιοθήκης γραμμικής άλγεβρας Armadillo [130]. Η συγκεκριμένη βιβλιοθήκη έχει παρόμοια σύνταξη με το MATLAB ενώ για να επιτύχει υψηλή απόδοση κάνει χρήση των LAPACK [4], Intel MKL [3], AMD ACML και OpenBLAS [6]. Διάφοροι περιορισμοί που υφίστανται στον Matlab2cpp οφείλονται στο γεγονός ότι η Armadillo δεν καλύπτει όλες τις λειτουργίες που παρέχει το MATLAB.

Στο ερευνητικό έργο ALMA [73, 140, 141] προτείνεται μια σειρά

εργαλείων για την εκτέλεση Scilab εφαρμογών σε ενσωματωμένα πολυπύρηνα συστήματα. Το πρώτο εργαλείο της ροής είναι ένας Scilab-σε-C μεταγλωττιστής που παράγει ακολουθιακό κώδικα C. Έπειτα ο κώδικας αυτός τροφοδοτείται στον μεταγλωττιστή Gecos [67] ο οποίος λειτουργεί σαν εργαλείο παραλληλοποίησης.

Το εργαλείο MATISSE [33, 34, 35], για τη μεταγλώττιση κώδικα MATLAB σε C, ελέγχεται μέσω μιας θεματοστρεφούς (aspect-oriented) γλώσσας προγραμματισμού που ονομάζεται LARA [39]. Το MATISSE χρησιμοποιεί τη LARA για τον καθορισμό μετασχηματισμών στον MALTAB κώδικα που δέχεται ως είσοδο αλλά και για την έκφραση πληροφοριών που αφορούν τους τύπους και τα σχήματα των μεταβλητών του MATLAB κώδικα. Ακόμα, το MATISSE έχει τη δυνατότητα να παράγει διαφορετικό κώδικα C ανάλογα με την πλατφόρμα που στοχεύει κάθε φορά. Εκτός από την παραγωγή C κώδικα, οι συγγραφείς εργάζονται και για την παραγωγή κώδικα OpenCL.

Ο MAT2C [82] είναι ένας MATLAB-σε-C μεταφραστής που πραγματοποιεί στατική ανάλυση για την *εξαγωγή τύπων* από MATLAB κώδικα. Για τους τύπους που δεν μπορεί να βρει στατικά κατά τη μεταγλώττιση παράγεται κατάλληλος κώδικας για το χειρισμό τους κατά την εκτέλεση. Η εξαγωγή των τύπων γίνεται μέσω του εργαλείου MAGICA [81], που αποτελεί μέρος του MAT2C. Οι τύποι αυτοί χρησιμοποιούνται ως βάση για την πραγματοποίηση μιας σειράς βελτιστοποιήσεων.

Άλλα εργαλεία σχετικά με τη μεταγλώττιση MATLAB κώδικα σε C είναι τα Otter [123, 124], RTExpress [23], Menhir [43], MEGHA [119], MATCH [17, 18] και AccelDSP [16, 19] καθώς και αυτά που περιγράφονται στις εργασίες των Shei et al. [134] και Allen [9]. Ο Otter [123, 124] είναι ένας μεταγλωττιστής που παράγει C κώδικα ο οποίος κάνει χρήση του πρωτοκόλλου MPI για την παράλληλη επεξεργασία δεδομένων. Το RTExpress [23] είναι ένα εργαλείο που δουλεύει σε συνδυασμό με τον μεταγλωττιστή MCC για την εκτέλεση MATLAB κώδικα σε υπολογιστές υψηλής απόδοσης (high performance computers). Ο μεταγλωττιστής Menhir [43] παράγει κώδικα C ή Fortran, ενώ επιτρέπει την παραγωγή ακολουθιακού και παράλληλου κώδικα που βασίζεται σε βιβλιοθήκες όπως η ScaLAPACK. Ο Allen στην εργασία [9] παρουσιάζει μια προσέγγιση για τη μεταγλώττιση MATLAB-σε-C που στοχεύει σε πλατφόρμες ψηφιακής επεξεργασίας σήματος (digital signal processors, DSPs). Στην εργασία [134] οι Shei et al. μετατρέπουν MATLAB πίνακες σε βαθμωτά ώστε να είναι ευκολότερη η παραλληλοποίηση του παραγόμε-

νου C++ κώδικα. Ο MEGHA [119] είναι ενας μεταγλωττιστής που παράγει ένα συνδυασμό από κώδικα C++ και CUDA ενώ επιτρέπει την εκτέλεση του προγράμματος εισόδου σε πολλαπλούς ετερογενείς επεξεργαστές. Ο μεταγλωττιστής MATCH [17, 18] και η εμπορική του έκδοση, με το όνομα AccelDSP [16, 19], επιτρέπουν σε MATLAB κώδικα να εκτελείται σε κατανεμημένα, ετερογενή υπολογιστικά συστήματα. Το σύνολο αυτών των συστημάτων μπορεί να απαρτίζεται από συστοιχίες επιτόπια προγραμματιζόμενων πυλών (field-programmable gate arrays, FPGAs), ενσωματωμένους επεξεργαστές και πλατφόρμες ψηφιακής επεξεργασίας σήματος.

Οι μεταγλωττιστές FALCON [55, 56, 57], CMC [84] και Mc2FOR [97] παράγουν κώδικα Fortran από MATLAB είσοδο. Η συνεισφορά του FALCON έγκειται στο σύστημα στατικής εξαγωγής τύπων που παρέχει και στον τρόπο που χειρίζεται τα σχήματα των πινάκων για τους οποίους δεν μπορεί να καθοριστεί ο τύπος στατικά. Ο CMC υποστηρίζει εφαρμογές που πραγματοποιούν υπολογισμούς που περιλαμβάνουν αραιούς πίνακες (sparse matrixes). Το Mc2FOR είναι μέρος του έργου McLAB [77], ενός συνόλου εργαλείων που περιλαμβάνει επίσης έναν μεταγλωττιστή JIT (just-in-time) για το MATLAB (McVM [45]) και μια θεματοστρεφής (aspect-oriented) γλώσσα προγραμματισμού (AspectMatlab [15]) για την εφαρμογή μετασχηματισμών στον πηγαίο κώδικα και τη δυναμική ανάλυση του προγράμματος.

2.2 Ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων

Η ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων έχει χρησιμοποιηθεί ευρέως ως μετρική για τη μοντελοποίηση της τοπικότητας αναφοράς. Αυτή η ενότητα εστιάζει στην παρουσίαση των υπαρχόντων αλγορίθμων, που έχουν προταθεί κατά καιρούς, για τον βέλτιστο υπολογισμό αποστάσεων επαναχρησιμοποίησης δεδομένων. Εργασίες που αφορούν εργαλεία που κάνουν χρήση της ανάλυσης απόστασης επαναχρησιμοποίησης δεδομένων για σκοπούς όπως η αξιολόγηση της απόδοσης διάφορων εφαρμογών παρουσιάζονται στην επόμενη ενότητα.

Πρέπει να αναφερθεί ότι υπάρχουν αρκετές εργασίες σχετικά με τον υπολογισμό των στοιχείων που εμφανίζονται μια ή παραπάνω φορές σε ένα ρεύμα δεδομένων, οι οποίες δεν σχετίζονται άμεσα με

την μέτρηση της απόστασης επαναχρησιμοποίησης δεδομένων, την αξιολόγηση της τοπικότητας αναφοράς δεδομένων, ή γενικότερα τις λανθάνουσες μνήμες [20, 44, 72]. Τέτοιες εργασίες υπάγονται κυρίως στα πεδία των βάσεων δεδομένων [152] και των δικτύων ηλεκτρονικών υπολογιστών.

2.2.1 Αλγόριθμοι υπολογισμού απόστασης επαναχρησιμοποίησης δεδομένων

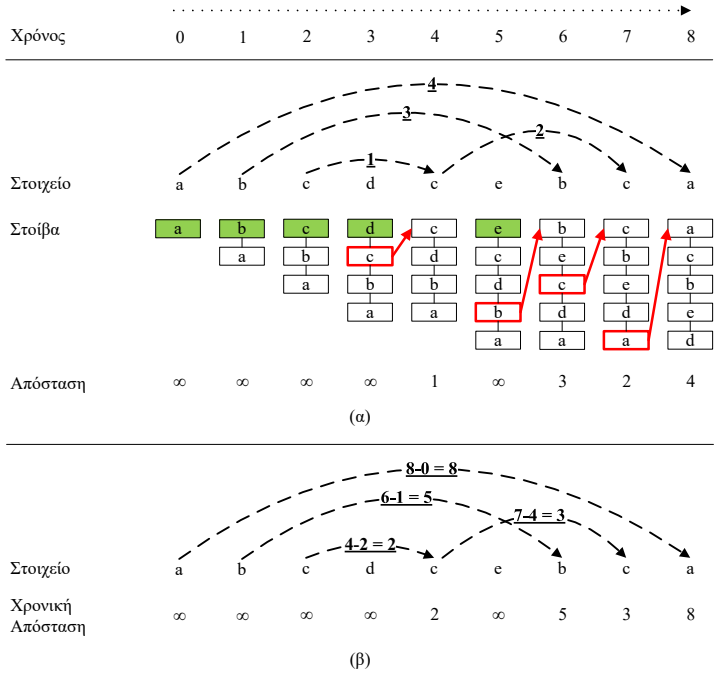
Ο υπολογισμός των *μοναδικών* στοιχείων δεδομένων που προσπελαύνονται μεταξύ δυο διαδοχικών προσπελάσεων στο ίδιο στοιχείο, δηλαδή η *απόσταση επαναχρησιμοποίησης δεδομένων*, αποτελεί έναν πολύ ακριβή τρόπο εκτίμησης της χρονικής τοπικότητας μιας εφαρμογής. Η διαδικασία αυτή όμως είναι υπολογιστικά δαπανηρή και είναι αναγκαία η υλοποίηση περίπλοκων λύσεων για να καταστεί εφαρμόσιμη. Ειδικά στην ανάλυση μεγάλων εφαρμογών που κατά την εκτέλεση τους πραγματοποιούν εκατομμύρια ή και δισεκατομμύρια προσπελάσεις μνήμης, η ανάλυση των αποστάσεων επαναχρησιμοποίησης πρέπει να γίνεται με έξυπνο και αποδοτικό τρόπο. Εκτός από τον απλό αλλά εξαιρετικά δαπανηρό αλγόριθμο, που μετριούνται με αφελή τρόπο τα μοναδικά στοιχεία μέσω πολλαπλών περασμάτων από τις καταγεγραμμένες προσπελάσεις μνήμης, διάφοροι πιο αποδοτικοί αλγόριθμοι έχουν προταθεί κατά καιρούς [10, 24, 64, 103, 115, 142, 162]. Μια σφαιρική εικόνα αυτών των αλγορίθμων δίνεται στον πίνακα 2.1. Η παλιότερη και πιο κοινή μέθοδος μέτρησης αποστάσεων επαναχρησιμοποίησης βασίζεται στην *απόσταση στοίβας LRU (least recently used stack distance)* που παρουσιάζεται στην εργασία [103] από τους Mattson et al. Η απόσταση στοίβας LRU είναι πρακτικά το ίδιο με την απόσταση επαναχρησιμοποίησης. Μεταγενέστεροι αλγόριθμοι για τον υπολογισμό αποστάσεων επαναχρησιμοποίησης δεδομένων βελτίωσαν αισθητά και επιτάχυναν τη διαδικασία [10, 24, 64, 115, 142, 162]. Η πλειοψηφία αυτών των αλγορίθμων λειτουργεί πάνω σε δενδροειδείς δομές δεδομένων. Έχουν επίσης προταθεί μέθοδοι κατά προσέγγιση υπολογισμού της απόστασης επαναχρησιμοποίησης, οι οποίες θυσιάζουν την απόλυτη ακρίβεια για χάρη της αποδοτικότητας ως προς το χώρο, ώστε να γίνεται εφικτός ο υπολογισμός μεγαλύτερων αποστάσεων επαναχρησιμοποίησης. Αυτές οι μέθοδοι βασίζονται επίσης σε δενδροειδείς δομές δεδομένων. Μια άλλη κατά προσέγγιση μέθοδος είναι να γίνεται υπολογισμός της *χρονικής απόστασης επαναχρησιμοποίησης δεδομένων*

[120, 135, 136] αντί της κανονικής. Η χρονική απόσταση είναι ο συνολικός αριθμός των προσπελάσεων μνήμης που παρεμβαίνουν μεταξύ δύο συνεχόμενων προσπελάσεων στο ίδιο στοιχείο. Αυτή είναι και η πιο αποδοτική μέθοδος καθώς δεν απαιτείται η δαπανηρή διαδικασία υπολογισμού των *μοναδικών* στοιχείων δεδομένων όπως στην κανονική απόσταση επαναχρησιμοποίησης. Η χρονική απόσταση μπορεί να υπολογιστεί απλά αφαιρώντας το χρόνο που πραγματοποιήθηκε η ακριβώς προηγούμενη προσπέλαση στο στοιχείο που εξετάζεται από τον χρόνο στον οποίο έγινε η επαναχρησιμοποίηση του στοιχείου αυτού. Σε γενικές γραμμές, οι αλγόριθμοι υπολογισμού απόστασης επαναχρησιμοποίησης δεδομένων μπορούν να κατηγοριοποιηθούν σύμφωνα με τα χαρακτηριστικά που φαίνονται στον πίνακα 2.1. Στην τέταρτη στήλη το *Α* υποδηλώνει ότι ο αλγόριθμος υπολογίζει τις αποστάσεις με απόλυτη ακρίβεια ενώ το *ΚΠ* ότι τις υπολογίζει κατά προσέγγιση. Οι δύο τελευταίες στήλες παρουσιάζουν τη χρονική και χωρική πολυπλοκότητα κάθε αλγόριθμου. Σε αυτές τις στήλες *N* είναι το συνολικό πλήθος των προσπελάσεων μνήμης και *M* είναι ο αριθμός των στοιχείων δεδομένων του προγράμματος. Ακολουθεί μια συνοπτική περιγραφή αυτών των αλγορίθμων ενώ στο παράρτημα Β΄ δίνονται εκτενή παραδείγματα.

Απόσταση στοίβας [103]. Η μέτρηση αποστάσεων επαναχρησιμοποίησης, κάνοντας χρήση δομής δεδομένων τύπου στοίβας, είναι μια αρκετά απλή διαδικασία. Για μια υποθετική ροή προσπελάσεων μνήμης, όπως αυτή στο σχήμα 2.1, εξετάζονται χρονικά οι προσπελάσεις μια προς μία. Σε κάθε βήμα γίνεται αναζήτηση στη στοίβα, ξεκινώντας από την κεφαλή, για το στοιχείο δεδομένων που προσπελαύνεται εκείνη τη στιγμή. Αν το στοιχείο αυτό υπάρχει ήδη στη στοίβα, τότε διαγράφεται από την τρέχουσα θέση του και προστίθεται στην κεφαλή. Αυτό σημαίνει ότι το εν λόγω στοιχείο έχει προσπελαστεί ξανά στο παρελθόν και υπάρχει επαναχρησιμοποίηση δεδομένων. Κατά την αναζήτηση στη στοίβα καταγράφεται και το βάθος στο οποίο βρέθηκε το κάθε στοιχείο. Το βάθος είναι ουσιαστικά η απόσταση επαναχρησιμοποίησης για την κάθε επαναχρησιμοποίηση. Στην περίπτωση που η αναζήτηση φτάσει στο τέλος της στοίβας και δεν βρεθεί το στοιχείο, γίνεται απλή προσθήκη του στην κεφαλή. Αυτό δείχνει ότι η παρούσα προσπέλαση είναι η πρώτη που γίνεται στο συγκεκριμένο στοιχείο και δεν υπάρχει επαναχρησιμοποίηση. Η μη ύπαρξη επαναχρησιμοποίησης αναπαρίσταται στο σχήμα 2.1 με το σύμβολο του άπειρου (∞). Ορίζοντας τον συνολικό αριθμό προσπελάσεων που υπάρχουν σε μια ροή ως *N* και τον συνολικό αριθμό των

Πίνακας 2.1: Αλγόριθμοι υπολογισμού απόστασης επαναχρησιμοποίησης δεδομένων.

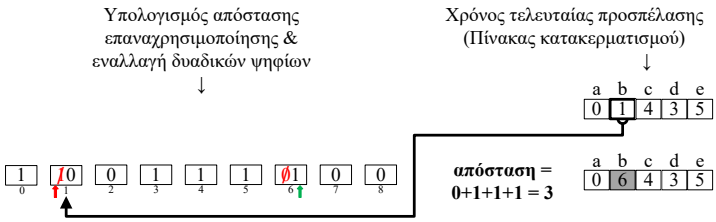
Δομή δεδομένων	Αλγόριθμος	Σχετική εργασία	Ακρίβεια	Χρόνος	Χώρος
	Χρονική απόσταση	[25, 120, 136]	ΚΠ	$O(N)$	$O(1)$
Στοίβα	Στοίβα LRU (με λίστα)	*Mattson et al., 1970* [103]	Α	$O(NM)$	$O(M)$
	Στοίβα με σημάδια (δύο λίστες)	*Almasi et al., 2002* [10]	Α	$O(N\sqrt{M})$	$O(M)$
Πίνακας δυαδικών ψηφίων	Πίνακας δυαδικών ψηφίων / m-αδικό δέντρο	*Bennett & Kruskal, 1975* [24]	Α	$O(NlogN)$	$O(N)$
	Στατικά δεσμευμένο δέντρο τρυπών	*Almasi et al., 2002* [10]	Α	$O(NlogN)$	$O(N)$
Ισοζυγισμένο δυαδικό δέντρο αναζήτησης	Δυαδικό δέντρο αναζήτησης AVL	*Olken, 1981* [115]	Α	$O(NlogM)$	$O(M)$
	Δυαδικό δέντρο αναζήτησης splay	*Sugumar, 1993* [143]	Α	$O(NlogM)$	$O(M)$
	Δέντρο αναπαράστασης τρυπών (AVL ή κόκκινο-μαύρο δέντρο)	*Almasi et al., 2002* [10]	Α	$O(NlogM)$	$O(M)$
	Δυναμική συμπίεση δέντρου	*Zhong et al., 2009* [162]	ΚΠ	$O(Nlog^2M)$	$O(logM)$
Δειγματοληψία (διάφορες δομές)	Βασισμένη στο reservoir sampling [149]	*Beyls & D'Hollander 2006* [29]	ΚΠ		
	Βασισμένη στο bursty tracing [78]	*Zhong & Chang, 2008* [160]	ΚΠ		
Παράλληλη επεξεργασία (διάφορες δομές)	Δυαδικό δέντρο αναζήτησης Αλγόριθμος PARDA	*Niu et al, 2012* [114]	Α		
	Τοπικοί πίνακες Αλγόριθμος HP-RDA	*Cui et al., 2012* [51]	ΚΠ		



Σχήμα 2.1: Παράδειγμα ροής προσπελάσεων μνήμης. α) Υπολογισμός των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση στοίβας. β) Υπολογισμός των χρονικών αποστάσεων.

στοιχείων δεδομένων που προσπελαύνονται ως M, η χρονική πολυπλοκότητα του συγκεκριμένου αλγορίθμου είναι $O(N \times M)$. Εκτελούνται συνολικά N βήματα για τον έλεγχο όλων των προσπελάσεων και σε κάθε ένα από αυτά πραγματοποιούνται το πολύ έως M βήματα για την αναζήτηση ενός στοιχείου στη στοίβα.

Αφελής αλγόριθμος με πίνακα κατακερματισμού. Ένας εντελώς αφελής αλγόριθμος χωρίς τη χρήση στοίβας είναι σε κάθε ένα από τα N βήματα να διασχίζεται, για κάθε στοιχείο (M φορές), το κομμάτι της ροής από την τρέχουσα προσπέλαση μέχρι την προηγούμενη προς το ίδιο στοιχείο. Έτσι εξετάζεται ποια και πόσα στοιχεία εμφανίζονται σε αυτό το κομμάτι. Το κομμάτι αυτό μπορεί βέβαια να φτάσει ακόμα και το μέγεθος N. Αυτός ο εξαιρετικά δαπανηρός αλ-



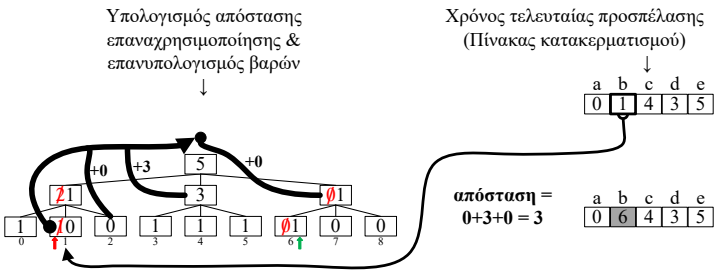
Σχήμα 2.2: Υπολογισμός των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση πίνακα δυαδικών ψηφίων.

γόριθμος έχει χρονική πολυπλοκότητα $O(N \times M \times N)$. Ένα διαφορετικό χαρακτηριστικό αυτού του αλγορίθμου σε σχέση με τη χρήση στοίβας είναι ότι σε κάθε βήμα πρέπει να είναι γνωστός ο χρόνος κατά τον οποίο έγινε η τελευταία προσπέλαση προς το στοιχείο που εξετάζεται εκείνη τη στιγμή. Αυτό μπορεί να γίνει χρησιμοποιώντας έναν πίνακα κατακερματισμού μεγέθους M όπου για κάθε στοιχείο θα φαίνεται ανά πάσα στιγμή ο χρόνος τελευταίας προσπέλασης προς αυτό. Η πολυπλοκότητα για την αναζήτηση στον πίνακα κατακερματισμού είναι $O(1)$ ενώ γίνεται χρήση του σε όλους τους αλγορίθμους που περιγράφονται στη συνέχεια.

Αλγόριθμος με πίνακα δυαδικών ψηφίων [24]. Ο αφελής αλγόριθμος μπορεί να βελτιωθεί κάνοντας χρήση ενός πίνακα δυαδικών ψηφίων μεγέθους N για την αναπαράσταση της ροής προσπελάσεων. Όλες οι θέσεις του συγκεκριμένου πίνακα είναι αρχικοποιημένες με την τιμή 0. Σε κάθε βήμα του αλγορίθμου η θέση στον πίνακα που αντιστοιχεί στον τρέχοντα χρόνο αλλάζει τιμή σε 1, ενώ η θέση που αντιστοιχεί στον χρόνο της αμέσως προηγούμενης προσπέλασης προς το ίδιο στοιχείο με το τρέχον μετατρέπεται από 1 σε 0. Με αυτό τον τρόπο τα 1 αναπαριστούν τις πιο πρόσφατες προσπελάσεις σε κάθε στοιχείο ενώ τα 0 πιο παλιές χρονικά. Ο χρόνος/θέση για τις προηγούμενες προσπελάσεις δίνεται από τον σχετικό πίνακα κατακερματισμού. Για τον υπολογισμό της απόστασης επαναχρησιμοποίησης αρκεί πλέον ένα πέρασμα, αντί για M, από το κομμάτι ανάμεσα στην τρέχουσα και την αμέσως προηγούμενη προσπέλαση στο ίδιο στοιχείο. Κατά το πέρασμα αυτό μετριέται ο αριθμός των 1 που βρέθηκαν και το σύνολο αυτό είναι ίδιο με την απόσταση επαναχρησι-

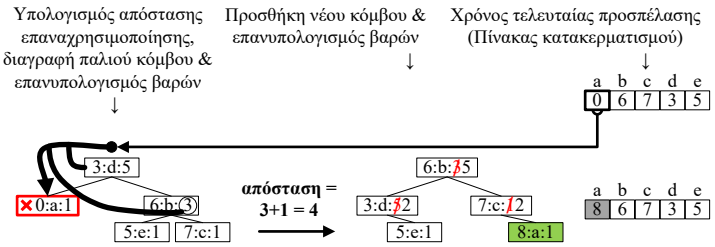
μοποίησης. Με τη χρήση στατικά δεσμευμένου πίνακα δυαδικών ψηφίων ο χρόνος εκτέλεσης γίνεται $O(N \times N)$. Στο σχήμα 2.2 δίνεται ένα παράδειγμα στο οποίο ο τρέχων χρόνος είναι 6 και η προηγούμενη προσπέλαση στο ίδιο στοιχείο έγινε στον χρόνο 1. Η απόσταση επαναχρησιμοποίησης σε αυτή την περίπτωση είναι *0+1+1+1 = 3*.

Αλγόριθμος με πίνακα δυαδικών ψηφίων / m-αδικό δέντρο [24]. Για την περαιτέρω βελτίωση του παραπάνω αλγόριθμου, οι Bennett και Kruskal [24] κατασκεύασαν ένα στατικά δεσμευμένο *m*-αδικό δέντρο στο οποίο οι τιμές του πίνακα δυαδικών ψηφίων αποτελούν τα φύλλα (το πρώτο επίπεδο). Οι εσωτερικοί κόμβοι δεν έχουν δυαδικές τιμές αλλά κρατούν το άθροισμα των τιμών των παιδιών τους. Κάθε κόμβος έχει ακριβώς *m* παιδιά ενώ το ύψος του δέντρου είναι $L = \lceil log_m N \rceil + 1$. Για τον υπολογισμό των αποστάσεων επαναχρησιμοποίησης ακολουθείται παρόμοια διαδικασία με πριν. Η διαφορά έγκειται στο ότι εκτός από τις αλλαγές στις δυαδικές τιμές των φύλλων του δέντρου, ανανεώνονται και οι τιμές των κόμβων που περιλαμβάνονται στα μονοπάτια πού ξεκινάνε από τα δυαδικά φύλλα και καταλήγουν στη ρίζα. Πιο συγκεκριμένα, από το φύλλο που αντιπροσωπεύει την αμέσως προηγούμενη προσπέλαση στο ίδιο στοιχείο με το τρέχον (που από 1 γίνεται 0) μέχρι τη ρίζα οι τιμές των κόμβων θα μειωθούν κατά 1. Από το φύλλο που αντιπροσωπεύει την τρέχουσα προσπέλαση (που από 0 γίνεται 1) μέχρι τη ρίζα οι τιμές θα αυξηθούν κατά 1. Κατά το πρώτο πέρασμα προς τη ρίζα θα γίνει και η μέτρηση της απόστασης επαναχρησιμοποίησης, αθροίζοντας για κάθε κόμβο του μονοπατιού τις τιμές των παιδιών του που βρίσκονται δεξιά από τον κόμβο που προηγήθηκε. Στο παράδειγμα του σχήματος 2.3 φαίνεται η διαδικασία υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων με χρήση *m*-αδικού δέντρου. t_{cur} είναι ο τρέχων χρόνος ενώ t_{prev} είναι ο χρόνος της προηγούμενης προσπέλασης στο ίδιο στοιχείο. Στο σχήμα αυτό η προσπέλαση στο χρόνο t_{cur} υποδεικνύεται με πράσινο βέλος ενώ με κόκκινο βέλος υποδεικνύεται η προσπέλαση στο χρόνο t_{prev}. Στους κόμβους που αλλάζουν τιμή, φαίνονται με κόκκινο χρώμα οι παλιές τιμές και με μαύρο χρώμα οι καινούριες τιμές. Η χοντρή μαύρη γραμμή δείχνει τους κόμβους που αθροίζονται κατά τον υπολογισμό της απόστασης επαναχρησιμοποίησης. Το άθροισμα αυτό, για το συγκεκριμένο παράδειγμα, είναι *0+3+0 = 3*. Με τη χρήση στατικά δεσμευμένου *m*-αδικού δέντρου η χρονική πολυπλοκότητα βελτιώνεται σε $O(N \times logN)$, ξεπερνώντας σε απόδοση τη μέθοδο υπολογισμού με χρήση στοίβας για μεγάλες αποστάσεις επαναχρησιμοποίησης.



Σχήμα 2.3: Υπολογισμός των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση πίνακα δυαδικών ψηφίων / στατικά δεσμευμένου m-αδικού δέντρου.

Αλγόριθμος με δυαδικό δέντρο αναζήτησης [115]. Ο πρώτος αλγόριθμος με δυναμικά δεσμευμένο δέντρο δόθηκε από τον Olken [115]. Πρόκειται για ισοζυγισμένο δυαδικό δέντρο αναζήτησης (ΔΔΑ) τύπου AVL, κάθε κόμβος του οποίου αναπαριστά την τελευταία χρονικά προσπέλαση προς ένα στοιχείο δεδομένων. Κάθε κόμβος έχει ένα *βάρος* καθώς και πληροφορία για τον χρόνο της προσπέλασης που αναπαριστά και το στοιχείο δεδομένων που αντιστοιχεί σε αυτή την προσπέλαση. Το δέντρο αυτό είναι ταξινομημένο ως προς το χρόνο προσπέλασης ενώ το βάρος κάθε κόμβου αντιστοιχεί στον αριθμό των απογόνων του συν 1. Σε κάθε βήμα του αλγορίθμου ανακτάται ο χρόνος τελευταίας προσπέλασης στο τρέχον στοιχείο από τον πίνακα κατακερματισμού. Έπειτα, ξεκινώντας από τη ρίζα, γίνεται αναζήτηση στο δέντρο για τον κόμβο που αντιστοιχεί στον συγκεκριμένο χρόνο. Ταυτόχρονα με την αναζήτηση γίνεται και ο υπολογισμός της απόστασης επαναχρησιμοποίησης. Για κάθε κόμβο που επισκέπτεται η αναζήτηση προστίθεται το βάρος του δεξιού του παιδιού συν 1 (για τον τρέχοντα κόμβο). Αυτό γίνεται μόνο στην περίπτωση που η αναζήτηση θα συνεχίσει στο αριστερό υποδέντρο του κόμβου. Μόλις βρεθεί ο υπό αναζήτηση κόμβος καταγράφεται το άθροισμα της απόστασης επαναχρησιμοποίησης. Ταυτόχρονα διαγράφεται ο κόμβος αυτός και ανανεώνεται κατάλληλα το AVL δέντρο και τα βάρη των κόμβων. Επόμενο βήμα είναι η προσθήκη ενός νέου κόμβου στο δέντρο που θα αναπαριστά την τρέχουσα προσπέλαση. Το σχήμα 2.4 παραθέτει ένα παράδειγμα εκτέλεσης του αλγόριθμου όπου $t_{prev} = 0$ και



Σχήμα 2.4: Υπολογισμός των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση ισοζυγισμένου δυαδικού δέντρου AVL.

t_{cur} = *8*. Κάθε κόμβος περιέχει τις ιδιότητες *<χρόνος:στοιχείο:βάρος>*. Η διάσχιση για την αναζήτηση του κόμβου με χρόνο t_{prev} και τον ταυτόχρονο υπολογισμό της απόστασης επαναχρησιμοποίησης αναπαρίσταται με χοντρές μαύρες γραμμές. Ο προς διαγραφή κόμβος με χρόνο t_{prev} έχει κόκκινο περίγραμμα ενώ ο νέος κόμβος με χρόνο t_{cur} έχει πράσινο χρώμα. Ακόμα, στους κόμβους των οποίων τα βάρη ανανεώνονται, τα παλιά βάρη απεικονίζονται με κόκκινο χρώμα και τα νέα με μαύρο. Ο αλγόριθμος με ΔΔΑ μειώνει το χρόνο που απαιτείται για μια διάσχιση από $O(M)$ στον αλγόριθμο με στοίβα σε $O(logM)$. Η χρονική πολυπλοκότητα του συνολικού αλγόριθμου για *N* βήματα είναι $O(N \times logM)$. Τέλος, οι Sugumar και Abraham [142, 143] έδειξαν πώς χρησιμοποιώντας δέντρο splay [138], αντί για δέντρο AVL, επιτυγχάνεται ακόμα καλύτερη απόδοση για τον συγκεκριμένο αλγόριθμο.

Αλγόριθμος με δυαδικό δέντρο τρυπών [10]. Οι Almasi et al. στην εργασία [10] έδειξαν πως καταγράφοντας τις *κενές περιοχές* σε μια ροή προσπελάσεων, αντί για τις τελευταίες προσπελάσεις σε κάθε στοιχείο, είναι εφικτή η βελτίωση της απόδοσης των μεθόδων που χρησιμοποιούν στατικά δεσμευμένα *m*-αδικά δέντρα αλλά και δυναμικά δεσμευμένα δυαδικά δέντρα. Σαν *τρύπες* ορίζουν όλες τις προσπελάσεις όπου βρίσκονται χρονικά πριν από την τελευταία σε κάθε στοιχείο. Σε έναν πίνακα δυαδικών ψηφίων όλες οι προσπελάσεις με τιμή 0, ανάμεσα από δυο προσπελάσεις με χρόνο t_{prev} και t_{cur}, αποτελούν τρύπες. Στο σχήμα 2.2, για παράδειγμα, η προσπέλαση που γίνεται στον χρόνο 2 θεωρείται τρύπα. Στην ίδια λογική, μια *κενή περιοχή* είναι ένα σύνολο συνεχόμενων τρυπών. Χρησιμο-

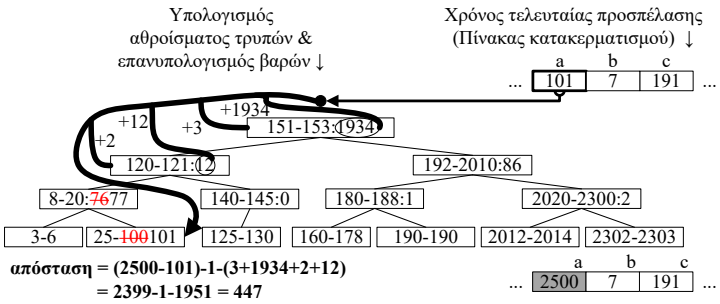
ποιώντας την έννοια των τρυπών, μπορεί να ειπωθεί ότι η απόσταση επαναχρησιμοποίησης από το χρόνο t_{prev} στον t_{cur} ισούται με την *χρονική απόσταση* μείον τον *αριθμό των τρυπών* ανάμεσα από αυτούς τους δύο χρόνους. Η χρονική απόσταση είναι εύκολα υπολογίσιμη. Για τον υπολογισμό του συνόλου των τρυπών μεταξύ t_{prev} και t_{cur} οι Almasi et al. έχτισαν ένα δυαδικό δέντρο αναζήτησης του οποίου οι κόμβοι αναπαριστούν κενές περιοχές. Όπως και στον αλγόριθμο του Olken [115], το δέντρο είναι ταξινομημένο ως προς το χρόνο προσπέλασης. Κάθε κόμβος έχει ένα *βάρος* και πληροφορία για το χρόνο έναρξης και το χρόνο λήξης της κενής περιοχής που αναπαριστά. Το βάρος κάθε κόμβου αντιστοιχεί στον αριθμό των τρυπών που αναπαριστώνται στο δεξιό του υποδέντρο. Σε κάθε βήμα του αλγόριθμου η προσπέλαση που γίνεται στο χρόνο t_{prev} μετατρέπεται σε τρύπα και προστίθεται στο δέντρο. Κατά την προσθήκη αυτή γίνεται και ο υπολογισμός του συνόλου των τρυπών μεταξύ t_{prev} και t_{cur}. Για την εύρεση του κόμβου στον οποίο θα προστεθεί η τρύπα πραγματοποιείται αναζήτηση ξεκινώντας από τη ρίζα του δέντρου. Υπάρχουν τρείς πιθανές περιπτώσεις που πρέπει να ληφθούν υπόψιν κατά την εισαγωγή μιας τρύπας στο δέντρο:

1. Η τρύπα είναι γειτονική σε μια μόνο κενή περιοχή. Σε αυτή την περίπτωση η τρύπα προστίθεται στον κόμβο που αναπαριστά αυτή την κενή περιοχή.

2. Η τρύπα είναι γειτονική σε δύο κενές περιοχές. Σε αυτή την περίπτωση ο ένας εκ των δύο κόμβων που αναπαριστούν τις περιοχές αυτές διαγράφεται. Οι δυο περιοχές ενώνονται σε μια, περιλαμβάνοντας και την νέα τρύπα. Η νέα αυτή περιοχή αναπαρίσταται πλέον από τον εναπομείναντα κόμβο. Υπάρχει επίσης η περίπτωση από τη διαγραφή του ενός κόμβου να χρειαστεί εξισορρόπηση το δέντρο.

3. Η τρύπα δεν γειτνιάζει με καμία κενή περιοχή οπότε ένας καινούριος κόμβος προστίθεται στο δέντρο. Ο κόμβος αυτός περιέχει την κενή περιοχή που περιλαμβάνει μόνο τη συγκεκριμένη τρύπα. Και πάλι το δέντρο, λόγω της προσθήκης του νέου κόμβου, ενδέχεται να χρειαστεί εξισορρόπηση.

Στο σχήμα 2.5 φαίνεται ένα στιγμιότυπο εκτέλεσης του αλγορίθμου τη στιγμή που ο χρόνος t_{prev}, που έχει ανακτηθεί από τον πίνακα κατακερματισμού, είναι 101 και ο τρέχων χρόνος t_{cur} ισούται με 2500.



Σχήμα 2.5: Υπολογισμός των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση ισοζυγισμένου δυαδικού δέντρου τρυπών.

Η διαδρομή που ακολουθείται κατά την αναζήτηση του κόμβου στον οποίο θα προστεθεί σαν τρύπα ο χρόνος t_{prev} αναπαρίσταται με το χοντρό μαύρο βέλος. Οι γραμμές που εφάπτουν στο βέλος δείχνουν τα βάρη που προστίθενται για τον υπολογισμό του συνόλου των ενδιάμεσων τρυπών. Στους κόμβους που αλλάζουν τιμές τα βάρη, φαίνονται με κόκκινο χρώμα οι παλιές τιμές και με μαύρο χρώμα οι καινούριες τιμές. Η συγκεκριμένη εισαγωγή τρύπας στο δέντρο εμπίπτει στην πρώτη περίπτωση του αλγόριθμου, καθώς ο χρόνος 101 γειτνιάζει μόνο με την κενή περιοχή 25-100. Συνεπώς πραγματοποιείται εισαγωγή της τρύπας στον κόμβο που αναπαριστά τη συγκεκριμένη περιοχή χωρίς να γίνει κάποια άλλη αλλαγή στο δέντρο. Η απόσταση επαναχρησιμοποίησης που θα μετρηθεί ισούται με:

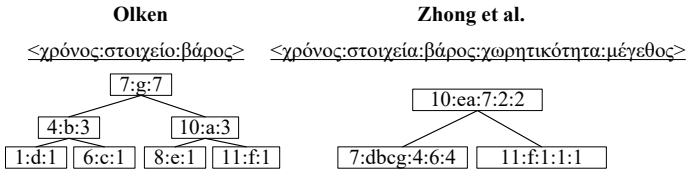
$$\begin{aligned} &(t_{cur} - t_{prev}) - 1 - Τρυπες(t_{prev}, t_{cur}) \\ =&(2500 - 101) - 1 - (3 + 1934 + 2 + 12) \\ =&2399 - 1 - 1951 \\ =&447 \end{aligned} \tag{2.1}$$

Ο αλγόριθμος με δυαδικό δέντρο τρυπών έχει χρονική πολυπλοκότητα $O(N \times logM)$. Στην εργασία [10] παρουσιάζεται επίσης και αλγόριθμος με στατικά δεσμευμένο *m*-αδικό δέντρο τρυπών και αποδεικνύεται πειραματικά ότι είναι γρηγορότερος από τον αρχικό αλ-

γόριθμο των Bennett και Kruskal [24] αλλά και από αλγόριθμους με δυναμικά δεσμευμένο δυαδικό δέντρο.

2.2.2 Κατά προσέγγιση υπολογισμός απόστασης επαναχρησιμοποίησης δεδομένων

Δυναμική συμπίεση δέντρου [64, 162]. Οι αλγόριθμοι που έχουν προταθεί κατά καιρούς αναμφίβολα συντομεύουν το χρόνο που απαιτείται για την ακριβή μέτρηση αποστάσεων επαναχρησιμοποίησης δεδομένων. Για την περίπτωση μεγάλων αποστάσεων όμως, η ακριβής μέτρηση εξακολουθεί να είναι μια δαπανηρή διαδικασία. Οι Ding, Zhong et al. [64, 162] παρατήρησαν ότι σπάνια έχει χρησιμότητα ο ακριβής υπολογισμός μιας απόστασης επαναχρησιμοποίησης. Αν, για παράδειγμα, μια απόσταση είναι κοντά στο ένα εκατομμύριο συνήθως δεν έχει σημασία αν η ακριβής τιμή είναι ένα εκατομμύριο ή ένα εκατομμύριο και ένα. Με βάση αυτή την παρατήρηση πρότειναν μια μέθοδο για τον κατά προσέγγιση υπολογισμό αποστάσεων επαναχρησιμοποίησης επεκτείνοντας τον αλγόριθμο με δυαδικό δέντρο αναζήτησης του Olken [115]. Στο τροποποιημένο αυτό δέντρο κάθε κόμβος έχει πέντε ιδιότητες έναντι των τριών που έχει στο δέντρο του Olken. Πιο συγκεκριμένα, στο δέντρο του Olken ένας κόμβος χαρακτηρίζεται από τις ιδιότητες *<χρόνος:στοιχείο:βάρος>* ενώ στο δέντρο των Zhong et al. από τις ιδιότητες *<χρόνος:στοιχεία:βάρος:χωρητικότητα:μέγεθος>*. Η βασική διαφορά είναι ότι ένας κόμβος αντιστοιχεί στις τελευταίες προσπελάσεις ενός *συνόλου στοιχείων* ενώ στο δέντρο του Olken κάθε κόμβος αντιπροσωπεύει μια μόνο προσπέλαση. Οι ιδιότητες *χωρητικότητα* και *μέγεθος* δείχνουν τον μέγιστο και τον τρέχοντα αριθμό των τελευταίων προσπελάσεων που αναπαριστά ο κόμβος. Η ιδιότητα *χρόνος* δείχνει τον χρόνο της πιο πρόσφατης χρονικά από αυτές τις προσπελάσεις. Τέλος, το *βάρος* είναι το άθροισμα των *μεγεθών* του υποδέντρου, με ακριβώς την ίδια λογική όπως στον αλγόριθμο του Olken. Ο συνολικός αριθμός των κόμβων ενός τέτοιου δέντρου ισούται με N δια το μέσο *μέγεθος* των κόμβων. Στο σχήμα 2.6 συγκρίνεται ένα δέντρο τύπου Olken με ένα τύπου Zhong et al. για τον υπολογισμό μιας απόστασης επαναχρησιμοποίησης. Το δέντρο των Zhong et al. είναι πολύ μικρότερο σε συνολικό αριθμό κόμβων κάνοντας κατά πολύ ταχύτερη τη διάσχιση του και την εισαγωγή και διαγραφή κόμβων. Εκεί που χάνεται η ακρίβεια είναι στο τέλος της μέτρησης της απόστασης επαναχρησιμοποίησης όταν ανακαλύπτεται ο κόμβος που περιλαμβάνει την τελευταία προσπέλαση. Επειδή



Σχήμα 2.6: Σύγκριση δέντρου Olken με συμπιεσμένο δέντρο Zhong et al.

κάθε κόμβος αντιπροσωπεύει πολλές προσπελάσεις, μπορεί να υπάρξει σφάλμα κατά τη μέτρηση, που μπορεί να ισούται μέχρι και με το *μέγεθος* του κόμβου. Αυτό θα συμβεί αν, για παράδειγμα, η προσπέλαση που αναζητείται έχει συμβεί χρονικά πιο παλιά από όλες τις άλλες προσπελάσεις που περιέχονται στον ίδιο κόμβο. Κάθε νέα προσπέλαση προστίθεται πάντα σε καινούριο κόμβο που αντιπροσωπεύει αρχικά μόνο αυτή. Μετά από κάθε νέα προσθήκη ελέγχεται το μέγεθος του δέντρου και εάν έχει επεκταθεί αρκετά, ο αλγόριθμος το συμπιέζει κατάλληλα. Η πολυπλοκότητα του κατά προσέγγιση αλγόριθμου με δέντρο αναζήτησης είναι $O(N \times log^2 M)$.

Χρονική απόσταση. Η μέτρηση της χρονικής απόστασης μεταξύ δύο προσπελάσεων σε μια ροή αποτελεί μια ακόμα μέθοδο για τον κατά προσέγγιση υπολογισμό της απόστασης επαναχρησιμοποίησης δεδομένων [120, 135, 136]. Στην χρονική απόσταση μετριέται ο συνολικός αριθμός των προσπελάσεων που πραγματοποιήθηκαν μεταξύ δυο διαδοχικών προσπελάσεων προς το ίδιο στοιχείο, σε αντίθεση με την απόσταση επαναχρησιμοποίησης που μετριέται ο αριθμός των στοιχείων που προσπελάστηκαν. Με την χρήση της απόστασης επαναχρησιμοποίησης είναι εφικτό να βγουν συμπεράσματα σχετικά με τα ποσοστά αστοχιών στις λανθάνουσες μνήμες, καθώς αν η απόσταση επαναχρησιμοποίησης ξεπεράσει το μέγεθος μιας λανθάνουσας μνήμης με πλήρως συσχετιστική οργάνωση και πολιτική αντικατάστασης με βάση το χρόνο τελευταίας προσπέλασης (LRU) υπάρχει σίγουρα αστοχία [28, 63]. Η χρονική απόσταση δεν είναι τόσο ακριβής. Είναι όμως κατά κάποιο τρόπο ανάλογη της απόστασης επαναχρησιμοποίησης. Όταν αυξάνει η χρονική απόσταση, αυξάνει και η πιθανότητα να είναι μεγαλύτερη και η απόσταση επαναχρησιμοποίησης. Σε υψηλό επίπεδο λοιπόν, αποτελεί μια χρήσιμη μέθοδο

αξιολόγησης της χρονικής τοπικότητας αναφοράς παρόμοια με την απόσταση επαναχρησιμοποίησης. Η χρονική απόσταση μπορεί να βρεθεί κάνοντας την απλή αφαίρεση: *χρόνος παρούσας προσπέλασης - χρόνος προηγούμενης προσπέλασης στο ίδιο στοιχείο.* Έτσι η χρονική πολυπλοκότητα του αλγορίθμου για τον υπολογισμό των χρονικών αποστάσεων σε μια ροή προσπελάσεων είναι $O(N)$. Εκτελούνται συνολικά *N* βήματα όπου σε κάθε ένα γίνεται μια αφαίρεση. Αυτό προϋποθέτει φυσικά την ύπαρξη πίνακα κατακερματισμού για την γρήγορη ανάκτηση των χρόνων προηγούμενης προσπέλασης σε κάθε στοιχείο. Στη ροή προσπελάσεων του σχήματος 2.1 φαίνεται και η χρονική απόσταση κάθε επαναχρησιμοποίησης. Ακόμα, στις εργασίες [135] και [136] προτείνεται ένα στατιστικό μοντέλο για τον κατά προσέγγιση υπολογισμό των αποστάσεων επαναχρησιμοποίησης κάνοντας χρήση των χρονικών αποστάσεων. Η μέθοδος που ακολουθείται σε αυτές τις εργασίες βασίζεται στην ακόλουθη παρατήρηση: Στη ροή προσπελάσεων *abbbba* η χρονική απόσταση από το πρώτο *a* στο δεύτερο είναι 5. Με αυτή μόνο την πληροφορία ως δεδομένη, η αντίστοιχη απόσταση επαναχρησιμοποίησης θα μπορούσε να είναι οτιδήποτε από 0 έως 4. Εάν όμως είχαν αρχικά υπολογιστεί οι χρονικές αποστάσεις όλων των επαναχρησιμοποιήσεων και ήταν γνωστό πως υπάρχει μια επαναχρησιμοποίηση με χρονική απόσταση 5 και άλλες τρεις με χρονική απόσταση 1, θα ήταν πιο εύκολος ο καθορισμός της απόστασης επαναχρησιμοποίησης. Αν και δεν είναι πάντα εφικτός ο ακριβής υπολογισμός των αποστάσεων επαναχρησιμοποίησης μέσω των χρονικών αποστάσεων, οι παραπάνω εργασίες δείχνουν ότι η γνώση των χρονικών αποστάσεων είναι αρκετή για τον κατά προσέγγιση υπολογισμό των αποστάσεων επαναχρησιμοποίησης.

2.2.3 Υπολογισμός απόστασης επαναχρησιμοποίησης δεδομένων με δειγματοληψία

Η εφαρμογή δειγματοληψίας στις προσπελάσεις μνήμης έχει χρησιμοποιηθεί σε ορισμένες εργασίες για τη βελτίωση της απόδοσης των αλγορίθμων υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων [29, 160]. Το εργαλείο SLO [29] κάνει χρήση μιας μεθόδου ομοιόμορφης τυχαίας δειγματοληψίας που λέγεται *reservoir sampling* ενώ το προτεινόμενο εργαλείο ενσωματώνει επίσης ικανότητες εφαρμογής δειγματοληψίας. Οι Zhong και Chang [160] προτείνουν έναν αλγόριθμο δειγματοληψίας που βασίζεται στον καταμερισμό της ροής προσπελάσεων σε περιόδους που διαδοχικά επιτρέπουν ή

όχι τη μέτρηση των αποστάσεων επαναχρησιμοποίησης δεδομένων. Παλιότερες μέθοδοι, που λαμβάνουν υπόψιν διάφορες παραμέτρους του υλικού και ενδέχεται να μην είναι άμεσα εφαρμόσιμες σε υψηλό επίπεδο αφαιρετικότητας, παρουσιάζονται στην εργασία [144].

Δειγματοληψία βασισμένη στο reservoir sampling. Στην εργασία [29] οι Beyls και D'Hollander παρουσιάζουν τη μέθοδο δειγματοληψίας που χρησιμοποιούν στο εργαλείο SLO [30, 31, 32] για τον ταχύ υπολογισμό αποστάσεων επαναχρησιμοποίησης δεδομένων. Το συγκεκριμένο εργαλείο πραγματοποιεί ομοιόμορφη τυχαία δειγματοληψία με τη μέθοδο *reservoir sampling* [149]. Στη μέθοδο αυτή επιλέγονται αρχικά οι *n* πρώτες προσπελάσεις και προστίθενται στο δείγμα. Έπειτα εξετάζονται μία προς μία οι επόμενες *N-n* προσπελάσεις. Κάθε προσπέλαση έχει την ευκαιρία να προστεθεί στο δείγμα με πιθανότητα η οποία μειώνεται σταδιακά. Αν μια προσπέλαση επιλεγεί για προσθήκη στο δείγμα, τότε μια άλλη προσπέλαση αφαιρείται από το δείγμα (με την ίδια πιθανότητα) και η νέα παίρνει τη θέση της. Ο ακριβής αλγόριθμος που χρησιμοποιείται στο SLO περιγράφεται στην εργασία [96] (αλγόριθμος L). Η γνώση του μεγέθους της ροής προσπελάσεων δεν είναι απαραίτητη. Παρακάμπτεται έτσι μια επιπλέον διάσχιση για την μέτρησή του. Επίσης, στο SLO όπως και στο προτεινόμενο εργαλείο η ανάλυση πραγματοποιείται σε πραγματικό χρόνο (online) και δεν υπάρχει ολόκληρη η ροή προσπελάσεων αποθηκευμένη εκ των προτέρων. Αυτό κάνει αναγκαία τη χρήση μεθόδων όπως το reservoir sampling. Οι Beyls και D'Hollander περιγράφουν επίσης στην ίδια εργασία και μια μέθοδο για τον υπολογισμό του αριθμού *n* των δειγμάτων που πρέπει να ληφθούν ώστε να επιτευχθεί μια δοθείσα ακρίβεια στις προβλέψεις που πραγματοποιεί το εργαλείο που προτείνουν.

Δειγματοληψία βασισμένη στο bursty tracing. Στην εργασία [160] οι Zhong και Chang προτείνουν έναν αλγόριθμο δειγματοληψίας ο οποίος επεκτείνει τη μέθοδο *bursty tracing* [78]. Στον συγκεκριμένο αλγόριθμο η ροή προσπελάσεων χωρίζεται σε *ενεργές* και *μη ενεργές περιόδους* οι οποίες εναλλάσσονται συνεχώς μεταξύ τους. Η βασική λογική του είναι ότι στις ενεργές περιόδους πραγματοποιείται δειγματοληψία ενώ οι μη ενεργές παρακάμπτονται. Αναπαριστώντας με I_E μια ενεργή περίοδο και με I_M μια μη ενεργή περίοδο ο ρυθμός δειγματοληψίας είναι $r = |I_E|/(|I_E| + |I_M|)$, όπου το $|I_X|$ υποδεικνύει το μήκος της περιόδου I_X. Ο αφελής τρόπος δειγματοληψίας με βάση τα παραπάνω είναι να μην λαμβάνονται καθόλου υπόψη οι προσπελάσεις που βρίσκονται σε μη ενεργές περιόδους και

να πραγματοποιείται μέτρηση των αποστάσεων επαναχρησιμοποίησης δεδομένων μόνο για τις προσπελάσεις που βρίσκονται σε ενεργές περιόδους. Αυτή η προσέγγιση, παρότι επιταχύνει τη μέτρηση, δεν εγγυάται την ακρίβεια του αποτελέσματος. Θα μπορούσε να οδηγήσει ακόμα και σε 100% ρυθμό σφάλματος, από τη στιγμή που δεν καταγράφονται οι προσπελάσεις κατά τις μη ενεργές περιόδους. Μόνο στην περίπτωση που η τρέχουσα προσπέλαση και η προηγούμενη προσπέλαση προς το ίδιο στοιχείο βρίσκονται μέσα στην ίδια περίοδο μπορεί να πραγματοποιηθεί ακριβής μέτρηση της απόστασης επαναχρησιμοποίησης. Στην περίπτωση που η προηγούμενη προσπέλαση προς το ίδιο στοιχείο βρίσκεται σε προηγούμενη ενεργή περίοδο, η απόσταση επαναχρησιμοποίησης που θα υπολογιστεί θα είναι εσφαλμένη. Οι μη ενεργές περίοδοι που μεσολαβούν περιέχουν μη καταγεγραμμένες προσπελάσεις που ενδεχομένως να είναι χρήσιμες για τον υπολογισμό της απόστασης επαναχρησιμοποίησης. Ακόμα και η ίδια η προηγούμενη προσπέλαση μπορεί να είναι λανθασμένη και η πραγματική να βρίσκεται σε κάποια από τις ενδιάμεσες μη ενεργές (και μη καταγεγραμμένες) περιόδους. Μια διαφορετική προσέγγιση είναι να καταγράφονται όλες οι προσπελάσεις και να πραγματοποιείται πάντα υπολογισμός της απόστασης επαναχρησιμοποίησης εκτός από την περίπτωση που η τρέχουσα προσπέλαση αλλά και η προηγούμενη προσπέλαση προς το ίδιο στοιχείο βρίσκονται εντός της ίδιας μη ενεργής περιόδου. Αυτή η παραλλαγή δίνει στις επαναχρησιμοποιήσεις με μεγάλες αποστάσεις επαναχρησιμοποίησης μεγαλύτερη πιθανότητα να ληφθούν υπόψη από ότι αυτές με μικρές αποστάσεις. Πιο συγκεκριμένα, οι επαναχρησιμοποιήσεις με απόσταση μεγαλύτερη ή ίση του I_M έχουν 100% πιθανότητα να επιλεγούν. Συνολικά η ακρίβεια υπολογισμού των αποστάσεων για όλη τη ροή με το συγκεκριμένο τρόπο είναι μόνο 24.8%. Με βάση τα παραπάνω, οι συγγραφείς παραθέτουν δυο παρατηρήσεις οι οποίες τους οδήγησαν στην τελική μορφή του αλγόριθμου: (1) Η καταγραφή των προσπελάσεων σε όλες τις περιόδους, ενεργές και μη ενεργές, κρίνεται απαραίτητη και (2) η πιθανότητα να γίνει επιλογή μιας επαναχρησιμοποίησης για μέτρηση πρέπει να είναι ομοιόμορφη για όλες τις αποστάσεις. Στον αλγόριθμο που παρουσιάζουν οι Zhong και Chang γίνεται καταγραφή όλων των προσπελάσεων. Υπολογισμός της απόστασης επαναχρησιμοποίησης πραγματοποιείται μόνο όταν η προσπέλαση-πηγή της επαναχρησιμοποίησης βρίσκεται σε ενεργή περιοχή. Επιτυγχάνεται έτσι ομοιόμορφη δειγματοληψία και εγγυάται ότι ο ρυθμός δειγματοληψίας θα είναι $r = |I_E|/(|I_E| + |I_M|)$ για όλες τις

πιθανές αποστάσεις. Με αυτό τον τρόπο η ακρίβεια της δειγματοληψίας ξεπερνά το 99%.

2.2.4 Παράλληλοι αλγόριθμοι υπολογισμού απόστασης επαναχρησιμοποίησης δεδομένων

Τα τελευταία χρόνια έχει παρατηρηθεί ένα αυξανόμενο ενδιαφέρον προς την παραλληλία όσον αφορά την ανάλυση απόστασης επαναχρησιμοποίησης. Σχετικές εργασίες είτε αφορούν την εφαρμογή παράλληλων αλγόριθμων για τον υπολογισμό αποστάσεων επαναχρησιμοποίησης δεδομένων [51, 114], είτε εξετάζουν την εφαρμοσιμότητα των εννοιών της απόστασης επαναχρησιμοποίησης στην ανάλυση της τοπικότητας παράλληλων εφαρμογών [80, 131, 132, 156, 157].

Ο πρώτος παράλληλος αλγόριθμος για τον υπολογισμό αποστάσεων επαναχρησιμοποίησης δεδομένων με ακρίβεια προτάθηκε από τους Niu et al. [114] και ονομάζεται PARDA. Ο συγκεκριμένος αλγόριθμος έχει υλοποιηθεί χρησιμοποιώντας την διεπαφή μεταβίβασης μηνυμάτων (message passing interface, MPI) ώστε να εκτελείται σε συστήματα κατανεμημένης μνήμης. Ο PARDA βασίζεται στην ιδιότητα μιας προσπέλασης που γίνεται σε χρόνο t_{cur} να είναι ανεξάρτητη από όλες τις προσπελάσεις που γίνονται πριν από τον χρόνο t_{prev} κατά τη μέτρηση της απόστασης επαναχρησιμοποίησης. Η ροή προσπελάσεων χωρίζεται σε p κομμάτια, κάθε ένα από τα οποία ανατίθεται σε έναν επεξεργαστή/διεργασία. Σε κάθε κομμάτι εφαρμόζεται ξεχωριστά (και παράλληλα) ο αλγόριθμος του Olken [115] και υπολογίζονται με ακρίβεια οι αποστάσεις επαναχρησιμοποίησης δεδομένων όλων των επαναχρησιμοποιήσεων όπου ο χρόνος-πηγή t_{prev} και ο χρόνος της τρέχουσας προσπέλασης t_{cur} εμπίπτουν στο ίδιο κομμάτι. Όσες αποστάσεις από κάθε κομμάτι δεν κατάφεραν να υπολογιστούν ονομάζονται *τοπικά άπειρα*. Για να υπολογιστούν όλες οι αποστάσεις επαναχρησιμοποίησης της ροής προσπελάσεων θα πρέπει τα τοπικά άπειρα από κάθε κομμάτι να στέλνονται σταδιακά στη διεργασία που είναι υπεύθυνη για το αμέσως προηγούμενο κομμάτι. Τα τοπικά άπειρα που φτάνουν με αυτό τον τρόπο στο κομμάτι 0 χωρίς ακόμα να είναι εφικτός ο υπολογισμός αποστάσεων επαναχρησιμοποίησης γι' αυτά, μετατρέπονται σε *καθολικά* ή *τελικά άπειρα* και καταγράφονται ως τέτοια. Οι Niu et al. κάνουν χρήση ενός δυαδικού δέντρου τύπου Olken [115] ως βασική δομή δεδομένων. Ωστόσο

ο PARDA είναι συμβατός και με τις υπόλοιπες δομές δεδομένων που παρουσιάστηκαν στην παρούσα ενότητα.

Οι Cui et al. [51] πρότειναν μια παρόμοια στρατηγική παραλληλοποίησης με αυτή του αλγόριθμου PARDA. Ο δικός τους αλγόριθμος, που ονομάζεται HP-RDA, κάνει χρήση του μοντέλου SPMD (single program - multiple data, ένα πρόγραμμα - πολλά δεδομένα) και έχει υλοποιηθεί στην πλατφόρμα για προγραμματισμό μονάδων επεξεργασίας γραφικών CUDA [113]. Η ροή προσπελάσεων τμηματοποιείται και κάθε τμήμα στέλνεται σε διαφορετικό νήμα CUDA για μέγιστη εκμετάλλευση της παραλληλίας. Πραγματοποιείται ανάλυση σε κάθε νήμα ξεχωριστά και στο τελικό στάδιο τα αποτελέσματα από τα επιμέρους τμήματα της ροής προσπελάσεων συγχωνεύονται. Στο στάδιο αυτό υπολογίζονται και οι αποστάσεις επαναχρησιμοποίησης που δεν ήταν δυνατό να υπολογιστούν από τα τμηματοποιημένα κομμάτια. Οι υπολογισμοί αυτοί γίνονται κατά προσέγγιση κάνοντας χρήση ενός πιθανοτικού μοντέλου (probabilistic model) για την επίτευξη όσο το δυνατόν μεγαλύτερης ακρίβειας. Ακόμα μια διαφορά του αλγόριθμου HP-RDA με τον PARDA είναι ότι δεν λειτουργεί σε δυαδικό δέντρο αλλά σε μια κατάλληλα προσαρμοσμένη υβριδική δομή δεδομένων που αποτελείται από έναν πίνακα κατακερματισμού και ένα σύνολο τοπικών πινάκων, έναν για κάθε νήμα CUDA.

Ακόμα ένας αλγόριθμος για την αξιολόγηση της τοπικότητας αναφοράς που υλοποιήθηκε σε CUDA παρουσιάζεται από τους Gupta et al. [75]. Σκοπός της εργασίας αυτής δεν είναι ο υπολογισμός αποστάσεων επαναχρησιμοποίησης δεδομένων. Προτείνεται η χρήση δεσμευμένων πιθανοτήτων (conditional probabilities) για την ποσοτικοποίηση και μέτρηση τόσο της χρονικής όσο και της χωρικής τοπικότητας αναφοράς. Ο παράλληλος αλγόριθμος που παρουσιάζεται έχει παρόμοια λογική με τους PARDA και HP-RDA.

Η ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων έχει επεκταθεί ώστε να είναι εφαρμόσιμη και σε παράλληλα συστήματα μέσω της μοντελοποίησης ιδιωτικών και κοινόχρηστων λανθανουσών μνημών [80]. Για το σκοπό αυτό έχουν εισαχθεί οι έννοιες *ιδιωτική απόσταση επαναχρησιμοποίησης δεδομένων (ΙΑΕΔ)* και *παράλληλη απόσταση επαναχρησιμοποίησης δεδομένων (ΠΑΕΔ).* Η ΙΑΕΔ είναι η απόσταση επαναχρησιμοποίησης που παρατηρείται σε μια ιδιωτική λανθάνουσα μνήμη. Η ΠΑΕΔ σχετίζεται με το σύνολο των προσπελάσεων μνήμης που πραγματοποιούνται από όλες τις διεργασίες σε μία κοινόχρηστη λανθάνουσα μνήμη. Αυτό που δυσκολεύει την ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων σε παράλληλα συ-

στήματα είναι η κοινή χρήση δεδομένων που πραγματοποιείται από τις διάφορες διεργασίες σε συνδυασμό με το ότι ο χρόνος εκτέλεσης κάθε διεργασίας δεν είναι σταθερός και προκαθορισμένος. Για παράδειγμα, έστω δύο διεργασίες *Δ1* και *Δ2* που η *Δ1* προσπελαύνει με σειρά τα στοιχεία *abcba*. Η ΠΑΕΔ από το πρώτο *a* στο δεύτερο θα έχει την τιμή *2+x*. 2 είναι ο αριθμός των μοναδικών στοιχείων (*b* και *c*) και *x* είναι ο αριθμός των μοναδικών στοιχείων που προσπελάστηκαν από την διεργασία *Δ2* στο ίδιο χρονικό πλαίσιο. Οι Jiang et al. [80] κάνουν χρήση ενός στατιστικού μοντέλου για τη διευθέτηση αυτού του θέματος και τον κατά προσέγγιση υπολογισμό των ΠΑΕΔ. Οι Schuff et al. [131, 132] πραγματοποιούν ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων σε παράλληλες εφαρμογές μέσω μιας παράλληλης μεθόδου που πραγματοποιεί και δειγματοληψία. Οι Wu και Young [156, 157, 158] παρουσίασαν ένα μοντέλο που προβλέπει τις αλλαγές στην απόσταση επαναχρησιμοποίησης όταν γίνεται παραλληλοποίηση μιας εργασίας σε διαφορετικούς αριθμούς από διεργασίες. Το μοντέλο αυτό είναι εφαρμόσιμο σε αλγορίθμους που βασίζονται σε βρόχους επανάληψης.

2.2.5 Ανάλυση απόστασης επαναχρησιμοποίησης δεδομένων στο MATLAB

Η πλειοψηφία των υπαρχόντων μεταγλωττιστών της γλώσσας MATLAB στοχεύουν στην παραγωγή βελτιστοποιημένου κώδικα χαμηλότερου επιπέδου για συγκεκριμένες αρχιτεκτονικές. Στην παρούσα διατριβή είναι η πρώτη φορά που ένας μεταγλωττιστής χρησιμοποιείται βοηθητικά για την εξαγωγή μετασχηματισμών που βελτιώνουν την τοπικότητα αναφοράς σε πηγαίο κώδικα MATLAB. Η μόνη εργασία που πραγματοποιείται κάποιου είδους ανάλυσης της απόστασης επαναχρησιμοποίησης δεδομένων σε αλγόριθμους γραμμένους σε μία υψηλού επιπέδου γλώσσα προγραμματισμού πινάκων είναι αυτή των Chauhan και Shei [42]. Στην εργασία αυτή, παρουσιάζουν έναν αλγόριθμο για τον κατά προσέγγιση υπολογισμό αποστάσεων επαναχρησιμοποίησης σε MATLAB κώδικα κάνοντας χρήση κατάλληλα τροποποιημένων γράφων εξαρτήσεων. Υπάρχουν δύο βασικές διαφορές ανάμεσα σε αυτή την προσέγγιση και στη δουλειά που παρουσιάζεται σε αυτή τη διατριβή: (1) Στην εργασία [42] οι συγγραφείς εκτελούν στατική ανάλυση σε κώδικα MATLAB για να συμπεράνουν τις αποστάσεις επαναχρησιμοποίησης, ενώ στο προτεινόμενο εργαλείο μετριούνται οι πραγματικές αποστάσεις επαναχρησιμοποίησης μέσα

από ενορχήστρωση του πηγαίου κώδικα και δυναμική ανάλυση. (2) Οι Chauhan και Shei παρέχουν κατά προσέγγιση εκτιμήσεις για τον ρυθμό αστοχίας των λανθανουσών μνημών που προκαλεί μια υπό εξέταση εφαρμογή ενώ το MemAssist κατευθύνει τον χρήστη στην εφαρμογή συγκεκριμένων μετασχηματισμών στον κώδικά του, οι οποίοι θα βελτιώσουν την τοπικότητα αναφοράς δεδομένων και θα οδηγήσουν στην καλύτερη αξιοποίηση των λανθανουσών μνημών του συστήματος.

2.3 Εργαλεία βελτιστοποίησης της τοπικότητας αναφοράς δεδομένων λανθάνουσας μνήμης

Υπάρχουν αρκετά εργαλεία που στοχεύουν στην αξιολόγηση της συμπεριφοράς της μνήμης. Εκείνα που εφαρμόζουν κάποιας μορφής ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων συχνά χρησιμοποιούνται για την εκτίμηση του ποσοστού αστοχιών σε λανθάνουσες μνήμες καθώς και για την βελτιστοποίηση της τοπικότητας αναφοράς. Μερικά μόνο από αυτά επικεντρώνονται στην παροχή προτάσεων στο χρήστη σχετικά με πιθανούς μετασχηματισμούς στον πηγαίο κώδικα οι οποίοι θα βελτιώσουν την τοπικότητα αναφοράς ενός αλγορίθμου σε υψηλό επίπεδο αφαιρετικότητας [29, 30, 31, 32, 38, 70, 94, 95, 127, 139].

Στη δουλειά που παρουσιάζεται στις εργασίες [29, 30, 31, 32] οι συγγραφείς περιγράφουν το SLO, ένα εργαλείο που βοηθά στην ανάλυση της τοπικότητας αναφοράς. Το εργαλείο αυτό υπολογίζει αποστάσεις επαναχρησιμοποίησης δεδομένων σε εφαρμογές γραμμένες σε γλώσσα C και προτείνει βελτιστοποιήσεις στον κώδικα παρόμοιες με αυτές που προτείνει το MemAssist. Το SLO είναι η πιο άμεσα συγκρίσιμη δουλειά με το MemAssist. Γι' αυτό το λόγο στο κεφάλαιο 3 παρέχεται μια λεπτομερής σύγκριση μεταξύ των δύο εργαλείων βελτιστοποίησης.

Το AutoSCOPE [139] είναι ακόμα ένα σύστημα που παρέχει προτάσεις βελτιστοποίησης. Επεκτείνει το PerfExpert [38], μια υποδομή για τη μέτρηση της απόδοσης και την ανάλυση εφαρμογών. Τα αποτελέσματα της ανάλυσης μέσω του PerfExpert τροφοδοτούνται στο AutoSCOPE και καθοδηγούν τη διαδικασία επιλογής βελτιστοποιήσεων. Επιπλέον, το PerfExpert έχει τη δυνατότητα να κάνει αυτόματα αλλαγές στον πηγαίο κώδικα, εφαρμόζοντας τις βελτιστοποιήσεις που μπόρεσε να εντοπίσει.

Το StructSlim [127] είναι μια εφαρμογή δυναμικής ανάλυσης που παρέχει πληροφορίες και συμβουλές στο χρήστη ώστε να καθοδηγήσει τη διαδικασία του διαχωρισμού δομών (structure splitting) [161]. Από τη στιγμή που στοχεύει στην αναδιοργάνωση των δεδομένων που περιέχονται σε δομές έχει αρκετά διαφορετική προσέγγιση από το MemAssist που εστιάζει στην εφαρμογή μετασχηματισμών σε βρόχους επανάληψης. Ακόμα μια εφαρμογή που αφορά την αναδιοργάνωση δεδομένων παρουσιάστηκε από τους Fu et al. [70]. Σε αυτή την εργασία ακολουθείται μια παρόμοια διαδικασία ανάλυσης με αυτή του MemAssist. Και τα δύο εργαλεία εφαρμόζουν ενορχήστρωση σε επίπεδο πηγαίου κώδικα για την πραγματοποίηση ανάλυσης απόστασης επαναχρησιμοποίησης δεδομένων διατηρώντας παράλληλα πληροφορία σχετικά με τον πηγαίο κώδικα. Το MemAssist χρησιμοποιεί το MEMSCOPT [61] σαν εργαλείο ενορχήστρωσης κώδικα ενώ οι Fu et al. κάνουν χρήση των υποδομών του μεταγλωττιστή SUIF [154] για την ίδια εργασία.

Υπάρχουν επίσης πάρα πολλά εργαλεία οπτικοποίησης λανθανουσών μνημών τα οποία κάνουν χρήση γραφημάτων διάφορων τύπων για την αναπαράσταση της λειτουργίας των μνημών. Τα YACO [121] και CVT [58] περιλαμβάνονται σε αυτή την κατηγορία εργαλείων. Οι εφαρμογές αυτές μπορούν να εντοπίσουν τα στοιχεία δεδομένων που προκαλούν μεγάλο αριθμό αστοχιών λανθάνουσας μνήμης αλλά αφήνουν στο χρήστη την απόφαση για εφαρμογή κατάλληλων μετασχηματισμών στον κώδικα.

2.3.1 Πρόβλεψη ρυθμού αστοχίας της λανθάνουσας μνήμης

Το StatCache [25, 26] είναι ένα εργαλείο που καταγράφει ένα σύνολο προσπελάσεων μνήμης έπειτα από ομοιόμορφη δειγματοληψία και υπολογίζει τις αποστάσεις επαναχρησιμοποίησης δεδομένων γι' αυτές. Αντί να παρέχει έναν πλήρως λειτουργικό προσομοιωτή λανθανουσών μνημών, το StatCache τροφοδοτεί αυτές τις αποστάσεις επαναχρησιμοποίησης σε ένα στατιστικό μοντέλο που υπολογίζει κατά προσέγγιση το ρυθμό αστοχίας των λανθανουσών μνημών. Αν και στην εργασία [25] αναφέρεται ως απόσταση επαναχρησιμοποίησης, το StatCache πραγματοποιεί στην πραγματικότητα μέτρηση της χρονικής απόστασης. Το εργαλείο μπορεί να μοντελοποιήσει λανθάνουσες μνήμες με πλήρως συσχετιστική οργάνωση και πολιτική αντικατάστασης τυχαίας επιλογής. Αργότερα παρουσιάστηκε μια

παραλλαγή του StatCache, το StatStack [66], που είναι ικανό να μοντελοποιήσει λανθάνουσες μνήμες με πλήρως συσχετιστική οργάνωση και πολιτική αντικατάστασης με βάση το χρόνο τελευταίας πρόσβασης (LRU). Η Συγκεκριμένη τεχνολογία ενσωματώθηκε μεταγενέστερα στο εργαλείο ThreadSpotter [8].

Οι Sen και Wood [133] ανέπτυξαν επίσης ένα σύστημα για την πρόβλεψη της απόδοσης (του ρυθμού αστοχίας) μιας λανθάνουσας μνήμης σε πραγματικό χρόνο, βασισμένο στην ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων. Η δική τους υλοποίηση δουλεύει όχι μόνο για μνήμες με πλήρως συσχετιστική οργάνωση αλλά για ένα εύρος συνδυασμών και πολιτικών αντικατάστασης (LRU, PLRU, RANDOM, NMRU). Ακόμα μια εργασία στην οποία χρησιμοποιείται η απόσταση επαναχρησιμοποίησης δεδομένων για την πρόβλεψη των αστοχιών της λανθάνουσας μνήμης είναι αυτή των Zhong et al. [159]. Όπως και στην εργασία των Sen και Wood, το εργαλείο τους δουλεύει για λανθάνουσες μνήμες με διάφορα χαρακτηριστικά. Το μοντέλο που παρουσιάζουν κάνει χρήση ενός *ιστογράμματος των αποστάσεων επαναχρησιμοποίησης* το οποίο, σε αντίθεση με το MemAssist, δεν σχετίζεται με στοιχεία του πηγαίου κώδικα.

Η ομοιότητα του προτεινόμενου εργαλείου με τις εργασίες που αφορούν την πρόβλεψη του ρυθμού αστοχίας, έγκειται στο γεγονός ότι παντού εφαρμόζεται ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων για την επίτευξη των εκάστοτε στόχων. Αυτά τα εργαλεία συνήθως εντοπίζουν προβληματικές περιοχές εντός του πηγαίου κώδικα ή δίνουν μια σφαιρική εικόνα για την τοπικότητα αναφοράς και δεν ασχολούνται με την εξαγωγή συγκεκριμένων βελτιώσεων.

Σε γενικές γραμμές δεν είναι συνηθισμένο να πραγματοποιείται στατική ανάλυση για τον υπολογισμό της απόστασης επαναχρησιμοποίησης δεδομένων και στην πλειοψηφία των σχετικών εργασιών γίνεται κάποιου είδους δυναμική ανάλυση του προγράμματος. Εκτός από την εργασία των Chauhan και Shei [42] που αναφέρθηκε παραπάνω, στην εργασία [109] οι συγγραφείς πραγματοποιούν επίσης στατική ανάλυση για τον κατά προσέγγιση υπολογισμό της απόστασης επαναχρησιμοποίησης. Σκοπός της εργασίας αυτής είναι η πρόβλεψη του ποσοστού αστοχιών της λανθάνουσας μνήμης ενώ οι προτάσεις της συγκεκριμένης εργασίας μπορούν να εφαρμοστούν σε προγράμματα γραμμένα σε γλώσσες όπως οι C/C++ και η Fortran.

2.3.2 Εργαλεία προσομοίωσης ιεραρχίας μνήμης

Μια τυπική κατηγορία εργαλείων ανάλυσης λανθάνουσας μνήμης τα οποία δεν εκτελούν ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων είναι οι προσομοιωτές λανθάνουσας μνήμης. Τέτοια εργαλεία είναι τα Cachegrind [111], Dinero IV [65], XMSIM [62] και πάρα πολλά ακόμα [37, 47, 49, 98, 106, 129, 144, 147, 148]. Συνήθως παρέχουν πληροφορίες σχετικά με την συνολική απόδοση της λανθάνουσας μνήμης για ένα πρόγραμμα. Υπολογίζουν τις αστοχίες και ευστοχίες λανθάνουσας μνήμης που λαμβάνουν χώρα κατά την εκτέλεση του προγράμματος σε μία προκαθορισμένη ιεραρχία μνήμης που ορίζεται από το χρήστη. Μια γενική ιδέα γύρω από την τοπικότητα αναφοράς του προγράμματος μπορεί να σχηματιστεί αλλά δεν παρέχονται συγκεκριμένες προτάσεις βελτιστοποίησης, σε αντίθεση με τις δυνατότητες του προτεινόμενου βελτιστοποιητή. Με βάση τα χαρακτηριστικά και τις δυνατότητες των εργαλείων προσομοίωσης μνήμης έγινε χρήση ενός από αυτά (του Cachegrind) στο κεφάλαιο της διατριβής που αφορά την αξιολόγηση του MemAssist. Η χρήση που έγινε αφορά την αξιολόγηση της απόδοσης (ως προς τη χρήση των λανθανουσών μνημών) των εφαρμογών που βελτιστοποιήθηκαν με το MemAssist για το συγκεκριμένο κεφάλαιο.

2.3.3 Σύγκριση εργαλείων βελτιστοποίησης

Τα κύρια χαρακτηριστικά των υπό σύγκριση εργασιών συνοψίζονται στον πίνακα 2.2. Πολλές ακόμα εργασίες θα μπορούσαν να είχαν συμπεριληφθεί σε αυτή τη λίστα αλλά επιλέχθηκαν πρωτίστως εκείνες που σχετίζονται με την ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων. Στις τέσσερις τελευταίες στήλες του πίνακα χωρίζεται η διαδικασία βελτιστοποίησης σε τέσσερα στάδια και υποδεικνύεται η υποστήριξη που παρέχει για κάθε ένα από αυτά το κάθε εργαλείο που εξετάζεται. Μια παρόμοια κατηγοριοποίηση τεσσάρων σταδίων γίνεται και στην εργασία [139]. Η στήλη *Ανάλυση / Μέτρηση* αφορά την ανάκτηση δεδομένων τα οποία θα επεξεργαστεί αργότερα το εργαλείο. Τα δεδομένα αυτά συνήθως συλλέγονται μέσω μετρήσεων και στατικής ή δυναμικής ανάλυσης του προγράμματος. Ακολουθεί η αξιοποίηση αυτών των δεδομένων στη στήλη *Παροχή γνώσης / Μετρικές*. Τα εν λόγω δεδομένα χρησιμοποιούνται για να εξαχθούν πληροφορίες σχετικά με την απόδοση του προγράμματος μέσω διαφόρων μετρικών ή ακόμα και εντοπίζοντας προβληματικές περιο-

Πίνακας 2.2: Εργαλεία βελτιστοποίησης λανθάνουσας μνήμης.

		Διαθέσιμος κώδικας/εκτελέσιμο	Είσοδος	Ανάλυση / Μέτρηση	Παροχή γνώσης / Μετρικές	Προτάσεις βελτιστοποίησης	Αυτόματες βελτιστοποιήσεις
	SLO [30, 31, 32]	●	C	●	●	●	●
	MemAssist [95]	●	C, MATLAB	●	●	●	◔
	PerfExpert [38]	●	C, C++, Fortran	●	●	●	●
	AutoSCOPE [139]	●		◐	●	●	●
Αναδιοργάνωση δεδομένων	StructSlim [127]	◐	C	●	●	●	●
	Fu et al. [70]	◐	C	●	●	●	●
Πρόβλεψη ρυθμού αστοχίας λανθάνουσας μνήμης	StatCache [25]	◐	C, C++, Fortran	●	●	◐	●
	StatStack [66]	◐		●	●	◐	●
	ThreadSpotter [8]	●		●	●	◐	●
	Sen and Wood [133]	●		●	●	◐	●
	Zhong et al. [159]	●		●	●	◐	●
Οπτικοποίηση μνήμης	YACO [121]	◐	Fortran, Trace	●	●	●	●
	CVT [58]	◐		●	●	●	●
Προσομοιωτές μνήμης	Cachegrind [111]	●	C	●	●	●	●
	Dinero IV [65]	●	C	●	●	●	●
	XMSIM [62]	●	C	●	●	●	●

● = Υπάρχον χαρακτηριστικό, ◐ = Μη υποστηριζόμενο χαρακτηριστικό, ◔ = Υπό εξέλιξη εργασία.

χές εντός του πηγαίου κώδικα. Ορισμένα από τα εργαλεία που παρουσιάζονται σε αυτή τη σύγκριση έχουν τη δυνατότητα να προτείνουν συγκεκριμένες βελτιστοποιήσεις που μπορούν να υλοποιηθούν από το χρήστη ώστε να καλυτερεύσει η απόδοση του εξεταζόμενου προγράμματος. Αυτό το στάδιο υποδεικνύεται στη στήλη *Προτάσεις βελτιστοποίησης*. Η τελευταία στήλη υποδεικνύει αν ένα εργαλείο μπορεί να εφαρμόσει αυτόματα βελτιστοποιήσεις κάνοντας αλλαγές στον πηγαίο κώδικα χωρίς την διαμεσολάβηση του χρήστη.

2.4 Εργαλεία ανάλυσης πηγαίου κώδικα

Στο κεφάλαιο 4 περιγράφεται το σύστημα που χρησιμοποιήθηκε για την ανάπτυξη του MemAssist. Τα δύο βασικά μέρη αυτού του συστήματος είναι: (1) μια γλώσσα προγραμματισμού ειδικού σκοπού για την εφαρμογή ερωτημάτων σε πηγαίο κώδικα που ονομάζεται CastQL και (2) ένας αυτόματος γεννήτορας εμπρόσθιων τμημάτων μεταγλωττιστών και εργαλείων ανάλυσης κώδικα που ονομάζεται FEgen. Στην παρούσα ενότητα γίνεται αναφορά σε σχετικές εργασίες που αφορούν την εφαρμογή ερωτημάτων σε πηγαίο κώδικα, συγκρίνοντάς τες παράλληλα με την CastQL. Αναφέρονται επίσης εργασίες που αφορούν την αυτόματη παραγωγή αφηρημένων συντακτικών δέντρων και συγκρίνονται με την αντίστοιχη προσέγγιση του FEgen.

2.4.1 Σύγκριση με τεχνολογίες εφαρμογής ερωτημάτων σε πηγαίο κώδικα

Σκοπός του συστήματος CastQL/FEgen είναι η δραστική μείωση του χρόνου, του κόπου και του κόστους που απαιτούνται για την ανάπτυξη εργαλείων ανάλυσης κώδικα και εφαρμογής μετασχηματισμών. Για την επίτευξη του συγκεκριμένου στόχου έχουν χρησιμοποιηθεί έννοιες και ιδέες τόσο από εργασίες που αφορούν την ανάλυση πηγαίου κώδικα [12, 13, 27, 46, 50, 52, 54, 76, 102, 150] όσο και από εργασίες που αφορούν την εφαρμογή μετασχηματισμών σε πηγαίο κώδικα [22, 36, 48, 53, 83, 85, 146].

Τα περισσότερα από τα περιβάλλοντα μεταπρογραμματισμού, που είναι προσανατολισμένα προς την εφαρμογή μετασχηματισμών, χρησιμοποιούν γραμματικές χωρίς συμφραζόμενα τύπου SDF για τον προσδιορισμό της γλώσσας εισόδου [36, 83, 146]. Αυτό τους δίνει τη δυνατότητα να λειτουργούν για οποιαδήποτε γλώσσα εισόδου. Τέτοιου

είδους περιβάλλοντα παρέχουν επίσης και μια ή περισσότερες γλώσσες προγραμματισμού ειδικού σκοπού για το χειρισμό του πηγαίου κώδικα αλλά και για την εφαρμογή ερωτημάτων σε αυτόν. Το σύστημα που προτείνεται ενεργεί πάνω σε οποιαδήποτε γλώσσα εισόδου (σε μία κάθε φορά) και κάνει χρήση μιας γλώσσας ειδικού σκοπού για την εφαρμογή ερωτημάτων με εκφραστικό τρόπο στον πηγαίο κώδικα. Το πλεονέκτημα της CastQL έναντι των υπαρχουσών εργασιών είναι ότι, από τη στιγμή που είναι μια εσωτερική/ενσωματωμένη γλώσσα ειδικού σκοπού, διατηρεί τα χαρακτηριστικά και το συντακτικό της γλώσσας υποδοχής. Της γλώσσας γενικού σκοπού δηλαδή, στην οποία έχει γίνει η υλοποίηση της. Αυτό επιτρέπει την ανάπτυξη και εφαρμογή μη προκαθορισμένων διασχίσεων στο συντακτικό δέντρο, όπως γίνεται και σε μια κλασσική υποδομή μεταγλωττιστή [99, 112], αντί να γίνεται χρήση της CastQL. Με τον τρόπο αυτό παρέχεται μια εναλλακτική για την εφαρμογή ερωτημάτων τα οποία δεν μπορούν να εκφραστούν χρησιμοποιώντας την CastQL. Επιπλέον, σε αντίθεση με τις προαναφερθείσες εργασίες, τα εργαλεία που παράγονται από το προτεινόμενο σύστημα έχουν τη δυνατότητα να διανεμηθούν ως αυτόνομες C++ εφαρμογές. Αυτό καθιστά μη αναγκαία την ύπαρξη ενός εξειδικευμένου περιβάλλοντος εκτέλεσης. Τέτοια περιβάλλοντα συνήθως λειτουργούν σε συγκεκριμένα συστήματα. Μόνο ένας μεταγλωττιστής της γλώσσας C++ είναι απαραίτητος για να τρέξουν τα παραγόμενα εργαλεία σε οποιοδήποτε σύστημα.

Όσον αφορά τα εργαλεία που είναι προσανατολισμένα στην ανάλυση, ένα σύνολο γλωσσών για την εφαρμογή ερωτημάτων σε πηγαίο κώδικα είναι διαθέσιμο στη βιβλιογραφία [12, 13, 27, 46, 50, 52, 54, 76, 102, 150]. Τέτοιες εργασίες αφορούν συνήθως γλώσσες λογικού προγραμματισμού (παρόμοιες με την Prolog) [27, 50, 76, 150] ή γλώσσες ερωτημάτων με βάση την σχεσιακή άλγεβρα (παρόμοιες με την SQL) [52, 102]. Παρά την καλή εκφραστικότητα που παρέχουν και άλλα θετικά στοιχεία που διαθέτουν, οι περισσότερες από αυτές περιορίζονται σε μια συγκεκριμένη γλώσσα εισόδου σε αντίθεση με την προσέγγιση που ακολουθείται στην CastQL. Επιπλέον μελέτες σχετικά με την αξιολόγηση των εργαλείων εφαρμογής ερωτημάτων σε κώδικα είναι διαθέσιμες στις εργασίες [11, 125, 145].

Κάποιες λειτουργίες εφαρμογής ερωτημάτων σε πηγαίο κώδικα μπορούν επίσης να βρεθούν στο εσωτερικό αρκετών μεταγλωττιστών. Στην απλούστερη και πιο συχνή περίπτωση παρέχουν στο χρήστη τη δυνατότητα να γράψει δικό του κώδικα για την διάσχιση κάποιας δομής αφηρημένου συντακτικού δέντρου και την εξαγωγή πληροφο-

ρίας από αυτή. Οι μεταγλωττιστές Roslyn [112] και Clang [99] είναι τυπικά παραδείγματα τέτοιων εργαλείων. Παρόμοιες λειτουργίες είναι επίσης διαθέσιμες και στα εργαλεία Gecos [67], ROSE [122], Cetus [91], CIL [110] και SUIF [154]. Η περιορισμένη εκφραστικότητα αυτών των υποδομών μεταγλωττιστών όσον αφορά τις λειτουργίες εφαρμογής ερωτημάτων είναι το κύριο μειονέκτημα τους σε σχέση με τις γλώσσες ειδικού σκοπού που είναι προσανατολισμένες στην εφαρμογή ερωτημάτων. Επιπλέον, όπως και τα περισσότερα συστήματα ανάλυσης κώδικα, δέχονται μόνο ένα προκαθορισμένο σύνολο γλωσσών ως είσοδο.

Το προτεινόμενο σύστημα συνδυάζει την ποικιλία γλωσσών εισόδου (και την ταυτόχρονη δυνατότητα παραμετροποίησης τους) που συνοδεύει τα περισσότερα εργαλεία εφαρμογής μετασχηματισμών με δυνατότητες εφαρμογής ερωτημάτων για την ταχεία ανάπτυξη εργαλείων ανάλυσης και χειρισμού πηγαίου κώδικα. Ακόμα, η αντικειμενοστραφής φύση της υλοποίησης και οι σχεδιαστικοί κανόνες [69, 74] που χρησιμοποιήθηκαν για το σχεδιασμό της CastQL αυξάνουν την επαναχρησιμοποίηση κώδικα, όπως φαίνεται και στη σχετική ενότητα του κεφαλαίου πειραματικής αξιολόγησης στην παρούσα διατριβή.

2.4.2 Αυτόματη παραγωγή αφηρημένων συντακτικών δέντρων

Έχουν γίνει στο παρελθόν αρκετές προσπάθειες για την αυτοματοποίηση της παραγωγής αναπαραστάσεων αφηρημένων συντακτικών δέντρων. Ο Wile [153] προτείνει έναν αλγόριθμο για την μετατροπή συμπαγών συντακτικών δέντρων σε αφηρημένα συντακτικά δέντρα. Ο αλγόριθμος αυτός παράγει το αφηρημένο συντακτικό δέντρο ενώ σαν είσοδο παίρνει μια γραμματική τύπου WBNF. Η συγκεκριμένη γραμματική αποτελεί επέκταση της μορφής συμβολισμού γραμματικών BNF. Οι Arusoaie και Vicol [14] παρουσιάζουν μια γενική μέθοδο για την εξαγωγή κανόνων που χρησιμοποιούνται για την παραγωγή αφηρημένων συντακτικών δέντρων από μια γραμματική χωρίς συμφραζόμενα. Παρέχουν επίσης ένα εργαλείο που παράγει αυτόματα για μια γραμματική, μια επεκταμένη έκδοση της σύμφωνα με την μέθοδο που προτείνουν. Ακόμα μια παρόμοια μέθοδος όπου μια σειρά μετασχηματισμών εφαρμόζεται στο συμπαγές δέντρο για την εξαγωγή του αφηρημένου δέντρου προτείνεται στην εργασία [21].

Ορισμένοι γεννήτορες αναλυτών αυτοματοποιούν την κατασκευή του αφηρημένου συντακτικού δέντρου μέσω της εισαγωγής ειδικών ετικετών στην συμπαγή αναπαράσταση. Τα ANTLR [116] και SableCC [71] είναι κάποια παραδείγματα τέτοιων εργαλείων, αν και η αντίστοιχη λειτουργία δεν είναι πλέον διαθέσιμη στην τελευταία έκδοση του ANTLR. Παράγει πλέον αυτόματα μόνο συμπαγή συντακτικά δέντρα. Άλλες προσεγγίσεις που κάνουν χρήση ετικετών στη συμπαγή αναπαράσταση για την παραγωγή του αφηρημένου συντακτικού δέντρου περιλαμβάνουν τους τύπους γραμματικών TBNF [101] και LBNF [68].

Τα πλεονεκτήματα του προτεινόμενου εργαλείου παραγωγής εμπρόσθιων τμημάτων (FEgen) έναντι των υπαρχουσών μεθόδων είναι ότι στην πραγματικότητα ενσωματώνει και τις δύο προσεγγίσεις που περιγράφηκαν. Μια προτεινόμενη αναπαράσταση αφηρημένου συντακτικού δέντρου εξάγεται αυτόματα από το εργαλείο. Την ίδια στιγμή ο χρήστης μπορεί προαιρετικά να εισάγει τις δικές του αλλαγές μέσω ετικετών στο συμπαγές δέντρο. Με αυτό τον τρόπο αποφεύγεται η επιπλέον εργασία του να οριστεί ολόκληρο το αφηρημένο συντακτικό δέντρο. Παράλληλα όμως διατηρείται η δυνατότητα επέμβασης σε προβληματικές περιοχές του αυτόματα παραγόμενου δέντρου.

Κεφάλαιο 3

Περιγραφή εργαλείου βελτιστοποίησης της τοπικότητας αναφοράς δεδομένων λανθάνουσας μνήμης

Στο παρόν κεφάλαιο παρουσιάζεται το MemAssist[1], ένα εργαλείο που πραγματοποιεί ανάλυση των αποστάσεων επαναχρησιμοποίησης δεδομένων σε συνδυασμό με τη χρήση ενός αλγορίθμου επιλογής βελτιστοποιήσεων, ώστε να συνάγει και να προτείνει στο χρήστη μετασχηματισμούς βρόχων επανάληψης, οι οποίοι ενδέχεται να βελτιώσουν την χρονική τοπικότητα αναφοράς δεδομένων ενός προγράμματος που δέχεται ως είσοδο. Κατά συνέπεια μειώνονται και οι αστοχίες λανθάνουσας μνήμης (cache misses) και βελτιώνεται ο χρόνος εκτέλεσης του εκάστοτε υπό εξέταση προγράμματος. Το MemAssist αναπτύχθηκε αρχικά ως επέκταση για το περιβάλλον ανάπτυξης εφαρμογών Visual Studio[2] και παρέχει μια ενοποιημένη ροή εργασίας κατά την ανάπτυξη και βελτιστοποίηση μιας εφαρμογής. Αργότερα έγινε δημόσια διαθέσιμο ως διαδικτυακή εφαρμογή και επεκτάθηκε ώστε εκτός από κώδικα σε γλώσσα C, που αρχικά δεχόταν ως είσοδο, να υποστηρίζει και κώδικα MATLAB. Για αυτό το σκοπό χρησιμοποιήθηκε ο MATLAB-σε-C μεταγλωττιστής που περιγράφεται στο κεφάλαιο 5. Ο συγκεκριμένος μεταγλωττιστής έχει τη δυνατότητα,

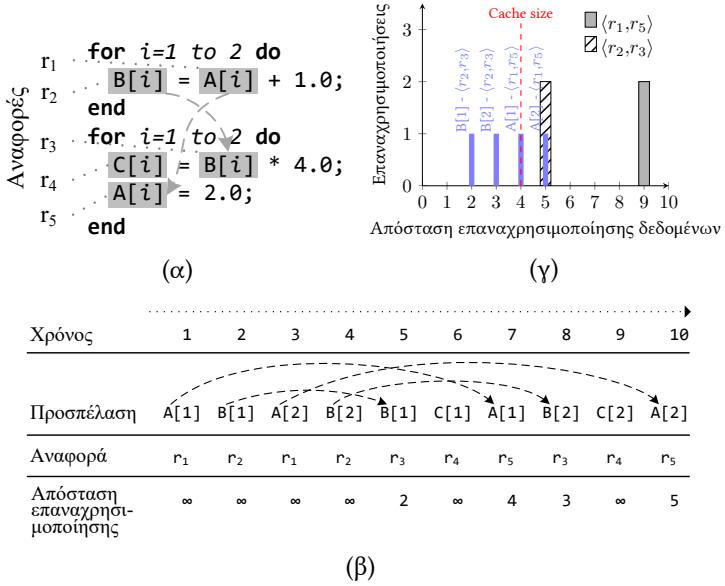
[1]https://www.lezos.gr/tools/memassist/
[2]https://visualstudio.microsoft.com/

μεταξύ άλλων, να συσχετίζει MATLAB μεταβλητές με κομμάτια του κώδικα C που παράγει. Η ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων πραγματοποιείται στον C κώδικα και οι σχετικές προτάσεις εφαρμογής μετασχηματισμών βελτιστοποίησης αντιστοιχίζονται στον αρχικό κώδικα MATLAB.

Το κεφάλαιο αυτό οργανώνεται ως εξής: Στην ενότητα 3.1 παρουσιάζονται οι βασικές αρχές λειτουργίας του εργαλείου, που αφορούν την ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων σε επίπεδο πηγαίου κώδικα. Ακολουθεί η περιγραφή της ροής λειτουργίας του MemAssist στην ενότητα 3.2 και του γραφικού περιβάλλοντος εργασίας που προσφέρει, στην ενότητα 3.3. Έπειτα, στις ενότητες 3.4, 3.5 και 3.6 περιγράφεται ο τρόπος λειτουργίας του εργαλείου εσωτερικά. Πιο συγκεκριμένα, οι ενότητες 3.4 και 3.5 αναφέρονται στη διαδικασία ανάλυσης εφαρμογών (profiling) που ακολουθείται και η ενότητα 3.6 παρουσιάζει τη μεθοδολογία που χρησιμοποιείται για την αυτόματη επιλογή κατάλληλων μετασχηματισμών βρόχων επανάληψης. Εκτός από την παροχή προτάσεων βελτιστοποίησης, το MemAssist παρέχει και κάποιες βοηθητικές μετρικές για την αξιολόγηση της απόδοσης μιας εφαρμογής και την εύρεση σημείων ενδιαφέροντος μέσα στον κώδικα. Οι μετρικές αυτές παρουσιάζονται στην ενότητα 3.7. Τέλος, στην ενότητα 3.8 γίνεται αναλυτική σύγκριση του MemAssist με το παρεμφερές εργαλείο SLO [29, 30, 31, 32].

3.1 Ανάλυση απόστασης επαναχρησιμοποίησης δεδομένων σε επίπεδο πηγαίου κώδικα

Τα στοιχεία των πινάκων και οι βαθμωτές μεταβλητές αποτελούν τους δύο κύριους τύπους *στοιχείων δεδομένων* στον πηγαίο κώδικα που παράγουν *προσπελάσεις μνήμης* κατά την εκτέλεση. Τα σημεία στον πηγαίο κώδικα όπου στην πραγματικότητα δημιουργούνται αυτές οι προσπελάσεις είναι οι αναφορές προς τα στοιχεία δεδομένων, οι οποίες θα αποκαλούνται εφεξής *αναφορές μνήμης*. Οι αναφορές μνήμης που αναφέρονται στην ίδια μεταβλητή είναι πολύ πιθανό να είναι φορείς επαναχρησιμοποίησης δεδομένων. Μια αναφορά μνήμης ορίζεται ως r_y και το σύνολο των αναφορών μνήμης που περιέχονται στον πηγαίο κώδικα ορίζεται ως $M = \{r_1, r_2, r_3, ..., r_m\}$, όπου m είναι ο συνολικός αριθμός των αναφορών. Το σχήμα 3.1α παρουσιάζει ένα κομμάτι κώδικα με σημειωμένες τις αναφορές μνήμης που περιέχει.



Σχήμα 3.1: α) Πηγαίος κώδικας με μαρκαρισμένες της αναφορές μνήμης. β) Ροή προσπελάσεων μνήμης. γ) Ιστόγραμμα αποστάσεων επαναχρησιμοποίησης δεδομένων.

Η σειριακή εκτέλεση ενός προγράμματος μπορεί να ιδωθεί σαν μια ακολουθία από n προσπελάσεις μνήμης, η οποία αποκαλείται *ροή προσπελάσεων μνήμης*. Η ροή αυτή ορίζεται ως $N = \{a_1, a_2, a_3, ..., a_n\}$, ενώ μια απλή προσπέλαση μνήμης ορίζεται ως $a_x = \langle e, i, r \rangle$, όπου e είναι ο πίνακας (array) με τον οποίο σχετίζεται η προσπέλαση, i είναι η θέση στον πίνακα αυτό, r είναι η αναφορά μνήμης προς το e η οποία παρήγαγε την προσπέλαση και x είναι η θέση της προσπέλασης μέσα στη ροή προσπελάσεων μνήμης. Όταν εκτελείται ο κώδικας του σχήματος 3.1α, παράγεται η ροή προσπελάσεων του σχήματος 3.1β. Ο πίνακας με τον οποίο σχετίζεται κάθε προσπέλαση και η αναφορά μνήμης που παρήγαγε την προσπέλαση φαίνονται σε αυτό το σχήμα.

Μια *επαναχρησιμοποίηση δεδομένων* συμβαίνει μεταξύ ενός στοιχείου της ροής προσπελάσεων μνήμης, το οποίο λέγεται *πηγή-επαναχρησιμοποίησης (προσπέλαση-πηγή)* και της πρώτης επανεμφάνισης του, η οποία λέ-

γεται *στόχος-επαναχρησιμοποίησης (προσπέλαση-στόχος)*. Ένα τέτοιο *ζεύγος επαναχρησιμοποίησης* ορίζεται ως $reup_z = \langle a_{x'}, a_x \rangle$, όπου $a_{x'}$ είναι η προσπέλαση-πηγή και a_x είναι ο στόχος. Το *ζεύγος αναφορών μνήμης* που παράγει ζεύγη επαναχρησιμοποίησης ορίζεται ως $refp_w = \langle r_{y'}, r_y \rangle$, όπου $r_{y'}$ είναι η *αναφορά-πηγή* και r_y είναι η *αναφορά-στόχος*. Στο σχήμα 3.1β τα βέλη αναπαριστούν τις επαναχρησιμοποιήσεις δεδομένων, αρχίζοντας από την προσπέλαση-πηγή και καταλήγοντας στην προσπέλαση-στόχο. Τα ζεύγη αναφορών παρουσιάζονται στο σχήμα 3.1α ως βέλη με διακεκομμένες γραμμές, τα οποία αρχίζουν από την αναφορά-πηγή και καταλήγουν στη αναφορά-στόχο.

Η *απόσταση επαναχρησιμοποίησης δεδομένων (reuse distance, RD)* [28, 63] ενός ζεύγους επαναχρησιμοποίησης είναι ο αριθμός των ξεχωριστών στοιχείων δεδομένων που προσπελάστηκαν ανάμεσα από την προσπέλαση-πηγή και την προσπέλαση-στόχο (εξίσωση 3.1). Αποτελεί έναν χρήσιμο τρόπο ποσοτικής μέτρησης της τοπικότητας αναφοράς δεδομένων, αφού μέσω αυτής μπορεί να προσδιοριστεί ο ρυθμός αστοχίας για μια λανθάνουσα μνήμη με πλήρως συσχετιστική οργάνωση και πολιτική αντικατάστασης με βάση τον χρόνο τελευταίας προσπέλασης (LRU). Αυτό μπορεί να επιτευχθεί συγκρίνοντας την απόσταση επαναχρησιμοποίησης κάθε ζεύγους με το μέγεθος της λανθάνουσας μνήμης. Ακόμα, έχει αποδειχθεί ότι η απόσταση επαναχρησιμοποίησης δεδομένων μπορεί να προβλέψει με ακρίβεια τον ρυθμό αστοχίας και για λανθάνουσες μνήμες με διαφορετικούς τύπους χαρτογράφησης [28]. Στην τελευταία σειρά του σχήματος 3.1β φαίνονται οι αποστάσεις επαναχρησιμοποίησης δεδομένων κάτω από την προσπέλαση-στόχο κάθε ζεύγους. Ο υπολογισμός των αποστάσεων επαναχρησιμοποίησης για όλα τα ζεύγη σε μια ροή προσπελάσεων μνήμης είναι μια αρκετά δαπανηρή διαδικασία (ενότητα 2.2). Η χρήση της *χρονικής απόστασης επαναχρησιμοποίησης δεδομένων* μπορεί να χρησιμοποιηθεί σαν εναλλακτική μέθοδος για τον κατά προσέγγιση υπολογισμό τους. Η χρονική απόσταση είναι ο συνολικός αριθμός των προσπελάσεων που διαμεσολαβούν μεταξύ της προσπέλασης-πηγή και της προσπέλασης-στόχου (εξίσωση 3.2). Μπορεί να υπολογιστεί ταχύτερα από την κανονική απόσταση επαναχρησιμοποίησης δεδομένων, αφαιρώντας απλά τον χρόνο κατά τον οποίο πραγματοποιήθηκε η προσπέλαση-πηγή από τον αντίστοιχο χρόνο της προσπέλασης-στόχου (σχήμα 3.1β). Η απόσταση επαναχρησιμοποίησης μπορεί ακόμα να συσχετιστεί και με ζεύγη αναφορών μνήμης. Η *συνολική απόσταση επαναχρησιμοποίησης δεδομένων (total reuse*

distance, TRD) ενός ζεύγους αναφορών είναι το άθροισμα όλων των αποστάσεων των ζευγών επαναχρησιμοποίησης που παρήχθησαν από αυτό το ζεύγος αναφορών (εξίσωση 3.3). Οι συνολικές αποστάσεις επαναχρησιμοποίησης μαζί με τον αριθμό των επαναχρησιμοποιήσεων για κάθε ζεύγος αναφορών τοποθετούνται σε ένα *ιστόγραμμα αποστάσεων επαναχρησιμοποίησης δεδομένων (reuse distance histogram, RDH)* (σχήμα 3.1γ). Το MemAssist κάνει χρήση ενός RDH για να βοηθήσει τον χρήστη να αποφασίσει για την βαρύτητα κάθε προτεινόμενου μετασχηματισμού βρόχων επανάληψης. Οι μετασχηματισμοί αυτοί συνάγονται από το εργαλείο κάνοντας χρήση τις μεθόδου που περιγράφεται στην ενότητα 3.6. Οι βρόχοι επανάληψης που περικλείουν ζεύγη αναφορών με υψηλές τιμές TRD έχουν κατά κανόνα μεγαλύτερη προτεραιότητα για εφαρμογή μετασχηματισμών.

Ορισμός 3.1

$$RD(reup_z) = RD(\langle a_{x'}, a_x \rangle) = \begin{cases} distinct(a_{x'}, a_x) & \textit{if } a_{x'} \textit{ exists} \\ \infty & \textit{otherwize} \end{cases} \tag{3.1}$$

$$TD(reup_z) = TD(\langle a_{x'}, a_x \rangle) = \begin{cases} t(a_x) - t(a_{x'}) & \textit{if } a_{x'} \textit{ exists} \\ \infty & \textit{otherwize} \end{cases} \tag{3.2}$$

όπου $t(a_x)$ *είναι ο χρόνος της προσπέλασης* a_x *και* $distinct(a_{x'}, a_x)$ *είναι ο αριθμός των ξεχωριστών στοιχείων που προσπελάστηκαν μεταξύ* $a_{x'}$ *και* a_x*.*

$$TRD(refp_w) = TRD(\langle r_{y'}, r_y \rangle) = \sum_{i=1}^{k} RD(\langle \langle e, i, r_{y'} \rangle, \langle e, i, r_y \rangle \rangle) \tag{3.3}$$

όπου k *είναι το μέγεθος του πίνακα* e*.*

Παράδειγμα 3.1

$$
\begin{aligned}
N =& \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}\} \\
=& \{\langle A,1,r_1\rangle, \langle B,1,r_2\rangle, \langle A,2,r_1\rangle, \langle B,2,r_2\rangle, \langle B,1,r_3\rangle, \\
& \langle C,1,r_4\rangle, \langle A,1,r_5\rangle, \langle B,2,r_3\rangle, \langle C,2,r_4\rangle, \langle A,2,r_5\rangle\}, \\
reup_1 =& \langle a_1, a_7\rangle = \langle\langle A,1,r_1\rangle, \langle A,1,r_5\rangle\rangle, \quad RD(reup_1) = 4, \\
reup_2 =& \langle a_2, a_5\rangle = \langle\langle B,1,r_2\rangle, \langle B,1,r_3\rangle\rangle, \quad RD(reup_2) = 2, \\
reup_3 =& \langle a_3, a_{10}\rangle = \langle\langle A,2,r_1\rangle, \langle A,2,r_5\rangle\rangle, \quad RD(reup_3) = 5, \\
reup_4 =& \langle a_4, a_8\rangle = \langle\langle B,2,r_2\rangle, \langle B,2,r_3\rangle\rangle, \quad RD(reup_4) = 3, \\
refp_1 =& \langle r_1, r_5\rangle, \quad refp_2 = \langle r_2, r_3\rangle
\end{aligned}
\tag{3.4}
$$

$$
\begin{aligned}
TRD(refp_1) =& TRD(\langle r_1, r_5\rangle) = \sum_{i=1}^{2} RD(\langle\langle A,i,r_1\rangle, \langle A,i,r_5\rangle\rangle) \\
=& RD(reup_1) + RD(reup_3) = 4 + 5 = 9, \\
TRD(refp_2) =& TRD(\langle r_2, r_3\rangle) = \sum_{i=1}^{2} RD(\langle\langle B,i,r_2\rangle, \langle B,i,r_3\rangle\rangle) \\
=& RD(reup_2) + RD(reup_4) = 2 + 3 = 5
\end{aligned}
\tag{3.5}
$$

Εξετάζοντας την ροή προσπελάσεων του σχήματος 3.1β, με βάση τους ορισμούς που αναφέρθηκαν παραπάνω, προκύπτουν τα δεδομένα που φαίνονται στην εξίσωση 3.4. Κάνοντας χρήση αυτών των δεδομένων, οι συνολικές αποστάσεις επαναχρησιμοποίησης για τα δύο ζεύγη αναφορών υπολογίζονται στην εξίσωση 3.5. Το σχήμα 3.1γ δείχνει τον αριθμό των επαναχρησιμοποιήσεων και την συνολική απόσταση επαναχρησιμοποίησης για κάθε ζεύγος επαναχρησιμοποίησης και κάθε ζεύγος αναφορών, τοποθετημένα σε ένα RDH. Σε μια υποθετική λανθάνουσα μνήμη με μέγεθος ίσο με τέσσερα στοιχεία, θα προκαλούνταν αστοχίες από το ζεύγος $\langle r_1, r_5\rangle$. Η λανθάνουσα μνήμη αναπαρίσταται με την διακεκομμένη κάθετη γραμμή στο ιστόγραμμα. Οι αναφορές $r1$ και $r5$ πρέπει να έρθουν πιο κοντά μεταξύ τους ώστε να εξαλειφθούν οι αστοχίες. Η συγχώνευση των δύο βρόχων επανάληψης στον πηγαίο κώδικα (loop fusion), θα μπορούσε, για παράδειγμα, να μειώσει δραστικά τις τιμές των TRD και να εξαλείψει τις αστοχίες της λανθάνουσας μνήμης.

3.2 Ροή λειτουργίας του εργαλείου

Το MemAssist δέχεται πάντα πηγαίο κώδικα σαν είσοδο και παράγει προτάσεις για την εφαρμογή μετασχηματισμών βρόχων επανάληψης. Αν αυτοί οι μετασχηματισμοί εφαρμοσθούν από τον χρήστη, η χρονική τοπικότητα αναφοράς δεδομένων του υπό εξέταση προγράμματος θα βελτιστοποιηθεί, οδηγώντας έτσι σε πιθανή μείωση του χρόνου εκτέλεσης. Το εργαλείο στοχεύει στην βελτιστοποίηση κώδικα σε γλώσσα C και MATLAB, ενώ υπάρχουν τρείς διαφορετικές ροές που μπορεί να ακολουθήσει ο χρήστης κατά τη λειτουργία του:

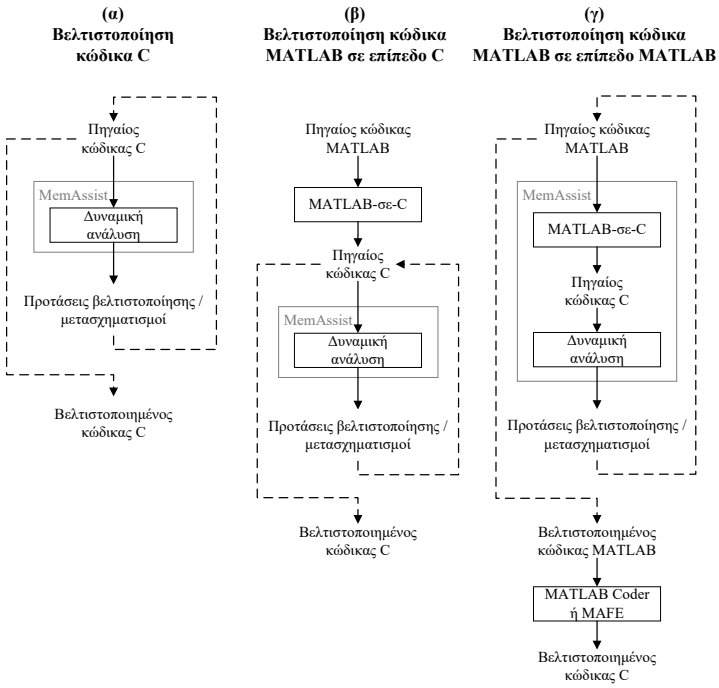
1. Λαμβάνεται κώδικας C από την είσοδο και οι παραγόμενες προτάσεις / μετασχηματισμοί εφαρμόζονται από τον χρήστη απευθείας σε αυτόν (σχήμα 3.2α).

2. Λαμβάνεται κώδικας MATLAB στην είσοδο, ο οποίος αρχικά μεταφράζεται σε αντίστοιχο κώδικα C μέσω του MATLAB-σε-C μεταγλωττιστή MAFE (κεφάλαιο 5). Στη συνέχεια βελτιστοποιείται ο C κώδικας όπως προηγουμένως (σχήμα 3.2β).

3. Παρέχεται κώδικας MATLAB απευθείας στο MemAssist, το οποίο εσωτερικά τον μετατρέπει σε C. Κατά την διαδικασία αυτή, οι MATLAB μεταβλητές συσχετίζονται με κομμάτια του παραγόμενου κώδικα C. Έπειτα πραγματοποιείται δυναμική ανάλυση (profiling) στον C κώδικα και οι σχετικές προτάσεις βελτιστοποίησης αντιστοιχίζονται στον κώδικα MATLAB της εισόδου. Σε αυτή την περίπτωση, οι μετασχηματισμοί εφαρμόζονται από τον χρήστη στον MATLAB κώδικα (σχήμα 3.2γ). Ο κώδικας C που θα παραχθεί από την βελτιστοποιημένη MATLAB έκδοση, κάνοντας χρήση του MATLAB Coder, θα είναι καλύτερος (όσον αφορά την τοπικότητα αναφοράς, την χρήση των λανθανουσών μνημών και την ταχύτητα εκτέλεσης) από αυτόν που θα είχε παραχθεί αν δεν είχαν εφαρμοσθεί οι προτεινόμενες βελτιστοποιήσεις σε επίπεδο MATLAB.

3.3 Γραφικό περιβάλλον διασύνδεσης χρήστη

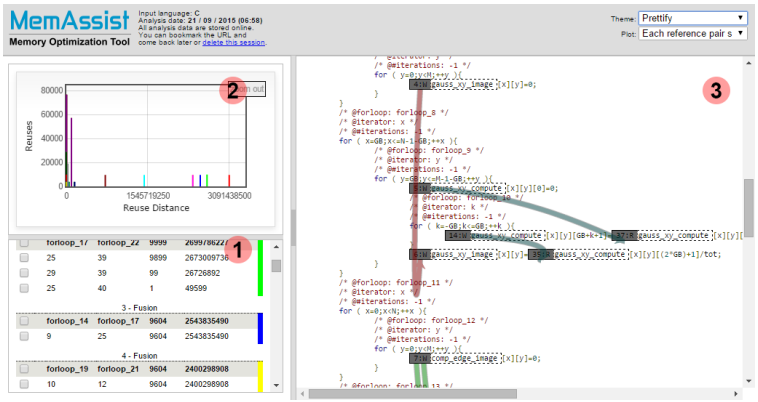
Οι προτάσεις εφαρμογής μετασχηματισμών βρόχων επανάληψης, μαζί με όλες τις υπόλοιπες πληροφορίες που συλλέγονται κατά την



Σχήμα 3.2: Δομή των ροών βελτιστοποίησης του MemAssist.

διαδικασία της ανάλυσης των αποστάσεων επαναχρησιμοποίησης δεδομένων, παρουσιάζονται στον χρήστη μέσα από ένα απλό και περιεκτικό περιβάλλον. Όλα τα ζεύγη αναφορών απαριθμούνται με φθίνουσα σειρά, σύμφωνα με την συνολική απόσταση επαναχρησιμοποίησης τους (σχήμα 3.3, σημείο 1). Ομαδοποιούνται επίσης ανάλογα με τα ζεύγη βρόχων επανάληψης στα οποία περικλείονται και ένα ξεχωριστό χρώμα ανατίθεται σε κάθε ομάδα. Ακόμα, διατίθεται και ένα ιστόγραμμα αποστάσεων επαναχρησιμοποίησης δεδομένων με τα ζεύγη αναφορών (σχήμα 3.3, σημείο 2) καθώς και οπτική αναπαράσταση των αποτελεσμάτων μέσα στον πηγαίο κώδικα (σχήμα 3.3, σημείο 3). Τα ζεύγη αναφορών αναπαριστώνται σαν χρωματιστά βέλη στον πηγαίο κώδικα, τα οποία εκκινούν από την αναφορά-πηγή και κατευθύνονται προς την αναφορά-στόχο. Οι προτάσεις του εργαλείου διατίθενται στον χρήστη σαν μια λίστα (σχήμα 3.3, σημείο 1),

στην οποία οι μετασχηματισμοί κατατάσσονται σύμφωνα με την βαρύτητα τους όσον αφορά τη βελτιστοποίηση της τοπικότητας αναφοράς δεδομένων. Όσο περισσότερες επαναχρησιμοποιήσεις με μεγάλες αποστάσεις επαναχρησιμοποίησης δεδομένων έχει ένα ζεύγος αναφορών, τόσο πιο σημαντικό θεωρείται.

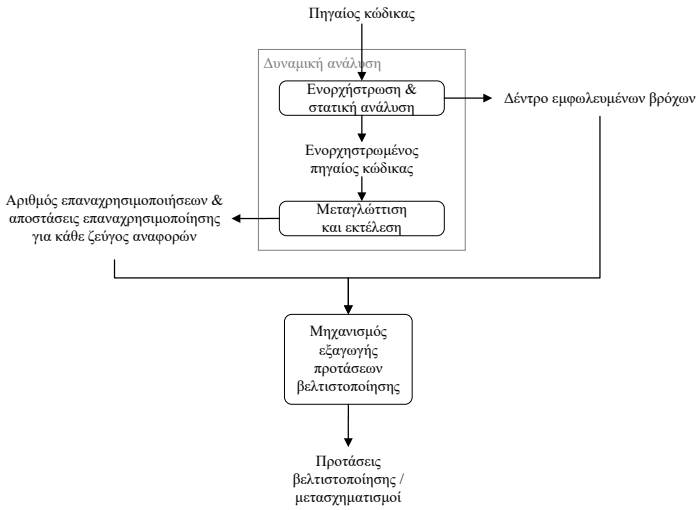


Σχήμα 3.3: Γραφικό περιβάλλον εργασίας του MemAssist.

3.4 Ροή ανάλυσης αλγορίθμων

Η διαδικασία της δυναμικής ανάλυσης (profiling) για τον υπολογισμό αποστάσεων επαναχρησιμοποίησης για τα ζεύγη αναφορών χωρίζεται στα βήματα που φαίνονται στο σχήμα 3.4. Επιλέχθηκε να ακολουθηθεί μια προσέγγιση στην οποία γίνεται ενορχήστρωση στον πηγαίο κώδικα αντί σε κώδικα χαμηλότερων επιπέδων. Αυτό ήταν αναγκαίο ώστε να είναι εφικτός ο συσχετισμός της εξόδου από την ανάλυση με σημεία του πηγαίου κώδικα. Αρχικά, εκτελείται ο μεταγλωττιστής MEMSCOPT [61] με κατάλληλες παραμέτρους ώστε να πραγματοποιήσει στατική ανάλυση και ενορχήστρωση του πηγαίου κώδικα. Σε αυτό το βήμα δημιουργείται ένα σύνολο αρχείων:

1. Από την στατική ανάλυση παράγεται ένα αρχείο XML που περιλαμβάνει πληροφορίες σχετικά με τα μεγέθη και τις εμβέλειες των μεταβλητών του προγράμματος, καθώς και μια αναπαράσταση του *δέντρου εμφώλευσης βρόχων επανάληψης*. Το δέντρο



Σχήμα 3.4: Ροή δυναμικής ανάλυσης και εξαγωγής προτάσεων βελτιστοποίησης του MemAssist.

αυτό είναι μια δομή που θα χρησιμοποιηθεί αργότερα για να εξαχθούν κατάλληλοι μετασχηματισμοί βρόχων επανάληψης.

2. Από την ενορχήστρωση παράγεται μια νέα έκδοση του αρχείου με κώδικα C που δόθηκε ως είσοδος, η οποία είναι εμπλουτισμένη με κάποια αναγκαία επιπλέον κομμάτια κώδικα. Τα περισσότερα από αυτά τα κομμάτια δρουν σαν καταγραφείς προσπελάσεων μνήμης κατά την εκτέλεση. Κάποια άλλα κομμάτια περιέχουν πληροφορίες σχετικά με την ονομασία και τη θέση κάθε βρόχου επανάληψης. Επιπλέον, παράγονται ακόμα δύο νέα αρχεία με κώδικα C (*ProfilerSupport.c* και *ProfilerSupport.h*), τα οποία συνοδεύουν τον ενορχηστρωμένο κώδικα. Σε αυτά τα αρχεία τοποθετούνται οι υλοποιήσεις όλων των συναρτήσεων που κάνουν τους κατάλληλους υπολογισμούς κατά την εκτέλεση.

Το MemAssist μεταγλωττίζει τα τρία αυτά αρχεία μαζί με όλα τα υπόλοιπα αρχεία του προγράμματος και τρέχει το εκτελέσιμο που παράγεται. Ανάμεσα στις λειτουργίες που θα τρέξουν κατά την εκτέ-

λεση, είναι και η παραγωγή κάποιων αρχείων ιχνών (trace files), τα οποία περιέχουν όλα τα δεδομένα που εξήγαγε η ανάλυση. Αυτά τα δεδομένα περιλαμβάνουν τις συνολικές αποστάσεις επαναχρησιμοποίησης δεδομένων και τους αριθμούς επαναχρησιμοποιήσεων για όλα τα ζεύγη αναφορών του υπό εξέταση προγράμματος.

Με βάση την υπάρχουσα εμπειρία από τη χρήση του εργαλείου, η επιβάρυνση που υφίσταται κατά την δυναμική ανάλυση (profiling overhead) μπορεί να χαρακτηριστεί αποδεκτή. Η επιβάρυνση αυξάνεται όταν γίνεται υπολογισμός της κανονικής απόστασης επαναχρησιμοποίησης δεδομένων αντί της χρονικής απόστασης. Αν και μπορεί να γίνει χρήση των αλγόριθμων ή / και των τεχνικών δειγματοληψίας που περιγράφονται στην ενότητα 2.2 για την αποδοτικότερη μέτρηση των αποστάσεων επαναχρησιμοποίησης, η χρήση των χρονικών αποστάσεων είναι προτιμότερη για περιπτώσεις που η ταχύτητα της ανάλυσης αποτελεί προτεραιότητα. Στην περίπτωση του MemAssist παράγονται παρόμοιες προτάσεις βελτιστοποίησης ανεξάρτητα από την μέθοδο που χρησιμοποιείται. Οπότε η μέτρηση των χρονικών αποστάσεων για καθημερινή χρήση του εργαλείου είναι προτιμότερη, από τη στιγμή που η επιβάρυνση είναι σταθερή και σε αποδεκτά επίπεδα.

Η επιβάρυνση που υφίσταται κατά την δυναμική ανάλυση εξαρτάται σχεδόν αποκλειστικά από το μέγεθος του πηγαίου κώδικα του υπό εξέταση προγράμματος. Η ενορχήστρωση κάθε μιας από τις εφαρμογές που βελτιστοποιήθηκαν στο κεφάλαιο 6 αυτής της διατριβής χρειάστηκε περίπου 4 δευτερόλεπτα για να ολοκληρωθεί. Οι εφαρμογές αυτές αποτελούνται από 112 γραμμές κώδικα η μικρότερη μέχρι 331 γραμμές κώδικα η μεγαλύτερη.

3.5 Υπολογισμός απόστασης επαναχρησιμοποίησης δεδομένων

Ο αλγόριθμος 1 περιγράφει συνοπτικά το κεντρικό περιεχόμενο των αρχείων *ProfilerSupport*. Ο αλγόριθμος αποτελείται από δύο βασικές συναρτήσεις:

1. Την συνάρτηση *Initialization*, η οποία καλείται πριν εκτελεστεί ο ενορχηστρωμένος κώδικας.

2. Την συνάρτηση *AccessArrayIndex*, για την οποία τοποθετείται (κατά την ενορχήστρωση) από μια κλήση προς αυτή μετά από

κάθε αναφορά μνήμης μέσα στον πηγαίο κώδικα. Κατά την εκτέλεση του ενορχηστρωμένου προγράμματος, η συνάρτηση αυτή καλείται για κάθε προσπέλαση μνήμης με σκοπό να την καταγράψει.

Κάθε πίνακας (array) του κώδικα εισόδου αναπαρίσταται με έναν ξεχωριστό θετικό ακέραιο αριθμό, ενώ το ίδιο ισχύει και για τις αναφορές μνήμης. Για λόγους απλότητας, όλοι οι πολυδιάστατοι πίνακες μετατρέπονται σε μονοδιάστατους (array flattening) με τον τρόπο που περιγράφεται στον ορισμό 3.2. Στο παράδειγμα 3.2 υπολογίζεται με αυτό τον τρόπο η μονοδιάστατη διεύθυνση που αντιστοιχεί στην πολυδιάστατη διεύθυνση $\{I_1 = 3, I_2 = 2, I_3 = 4\}$ ενός πίνακα με μεγέθη διαστάσεων $\{S_1 = 5, S_2 = 6, S_3 = 7\}$. Οι θέσεις / διευθύνσεις των πινάκων αναπαριστώνται και αυτές από έναν ξεχωριστό θετικό ακέραιο. Οι βαθμωτές μεταβλητές δεν έχουν ληφθεί υπόψη καθώς δεν φέρουν σημαντικό αριθμό επαναχρησιμοποιήσεων. Η μεταβλητή *AC (Access Counter)* του αλγορίθμου 1 είναι ο μετρητής του χρόνου και αυξάνεται κατά 1 σε κάθε κλήση της συνάρτησης *AccessArrayIndex*. Η *LAR (Last Access Reference)* είναι μια δομή στην οποία αποθηκεύεται το ποια αναφορά μνήμης προκάλεσε την τελευταία προσπέλαση σε κάθε στοιχείο δεδομένων. Η *LAT (Last Access Time)* είναι μια παρόμοια δομή στην οποία αποθηκεύεται ο χρόνος κατά τον οποίο έγινε η τελευταία προσπέλαση προς κάθε στοιχείο δεδομένων. Οι *R* και *TRD* είναι η έξοδος του αλγορίθμου. Στην *R* αποθηκεύεται ο αριθμός των επαναχρησιμοποιήσεων κάθε ζεύγους αναφορών και στην *TRD* ο αριθμός της συνολικής απόστασης επαναχρησιμοποίησης για κάθε ζεύγος.

Σε κάθε κλήση της *AccessArrayIndex* καταγράφεται ο τρέχων πίνακας (*array*), η τρέχουσα θέση του πίνακα που προσπελαύνεται (*indexflattened*) και η τρέχουσα αναφορά μνήμης που προκάλεσε την προσπέλαση (*ref*). Πραγματοποιείται αρχικά έλεγχος για το εάν έχει εξεταστεί προηγουμένως η αναφορά μνήμης που προκάλεσε την προηγούμενη προσπέλαση στην τρέχουσα θέση του πίνακα (*LAR[array][indexflattened]*). Εάν η *LAR[array][indexflattened]* έχει εξεταστεί ξανά στο παρελθόν και αυτή δεν είναι η πρώτη προσπέλαση προς το συγκεκριμένο στοιχείο του πίνακα, πρέπει να πραγματοποιηθεί ένας νέος υπολογισμός απόστασης. Ο αριθμός των επαναχρησιμοποιήσεων (*R*) για το τρέχον ζεύγος αναφορών αυξάνεται κατά 1, ενώ η συνολική απόσταση επαναχρησιμοποίησης δεδομένων (*TRD*) του τρέχοντος ζεύγους ισούται με *TRD + MeasureDistinctElements()*. Στην απλή περίπτωση που με-

τριέται η χρονική απόσταση, η συνάρτηση *MeasureDistinctElements* ισούται με *AC - LAT* (αλγόριθμος 1, γραμμή 17).

```
   Data: AC, LAR[Array][IndexFlattened], LAT[Array][IndexFlattened]
   Result: R[Array][RefSource][RefSink],
           TRD[Array][RefSource][RefSink]
   Function Initialization()
1      AC ← 0 ;
2      foreach Array A of all the arrays do
3          foreach Index I of the array A do
4              LAR[A][I] ← 0 − 1 ;
5              LAT[A][I] ← 0 − 1 ;
6          foreach Reference Ri of the array A do
7              foreach Reference Rj of the array A do
8                  R[A][Ri][Rj] ← 0 ;
9                  TRD[A][Ri][Rj] ← 0 ;
10     return;
11  Function AccessArrayIndex(array,indexflattened,ref)
12     AC ± 1 ;
13     if Sampling() == false then
14         return;
15     if LAR[array][indexflattened] > −1 then
16         lastaccessref ← LAR[array][indexflattened] ;
17         R[array][lastaccessref][ref] ± 1 ;
18         rd ← AC − LAT[array][indexflattened] ;
19         TRD[array][lastaccessref][ref] ± rd ;
20     LAR[array][indexflattened] ← ref ;
21     LAT[array][indexflattened] ← AC ;
22     return;
```

Αλγόριθμος 1 Υπολογισμός αποστάσεων επαναχρησιμοποίησης.

Ο παραπάνω αλγόριθμος έχει υλοποιηθεί στο MemAssist με τέτοιο τρόπο ώστε η ροή προσπελάσεων μνήμης να δημιουργείται και να επεκτείνεται παράλληλα με τον υπολογισμό των αποστάσεων επαναχρησιμοποίησης. Η ροή αποθηκεύεται προσωρινά στην μνήμη του συστήματος (online) και όχι σε κάποιο αρχείο (offline). Με αυτό τον τρόπο επιτυγχάνεται μια βελτίωση στην ταχύτητα εκτέλεσης του αλ-

γορίθμου, περιορίζοντας παράλληλα το μέγιστο επιτρεπτό μέγεθος της ροής προσπελάσεων ανάλογα με το μέγεθος της μνήμης του συστήματος.

Ορισμός 3.2

$$Flat(I) = \begin{cases} I_N + \sum_{i=1}^{N-1}(I_{N-1} \times \prod_{\substack{j= \\ (N-i)+1}}^{N}(S_j)) & if\, N > 1 \\ I_N & if\, N = 1 \end{cases} \quad (3.6)$$

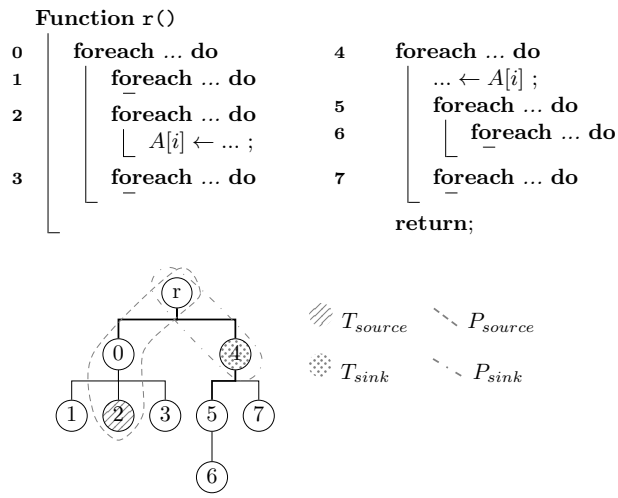
όπου N είναι ο αριθμός των διαστάσεων του πίνακα, S_x είναι το μέγεθος της διάστασης x και I_x είναι η θέση στη διάσταση x, στην οποία γίνεται αναφορά.

Παράδειγμα 3.2

$$I_3 + (I_2 \times S_3) + (I_1 \times S_2 \times S_3) = 4 + (2 \times 7) + (3 \times 6 \times 7) = 144 \quad (3.7)$$

3.6 Μεθοδολογία αυτόματης επιλογής κατάλληλων μετασχηματισμών βρόχων

Το MemAssist προτείνει είτε την *συγχώνευση (fusion)* ενός *ζεύγους βρόχων επανάληψης* είτε την εφαρμογή κάποιου *μετασχηματισμού υπερκόμβων (tiling)*. Και οι δύο αυτοί μετασχηματισμοί έχουν θετική επίδραση στη τοπικότητα αναφοράς δεδομένων. Η ακόλουθη μέθοδος χρησιμοποιήθηκε για την αυτόματη επιλογή κατάλληλων μετασχηματισμών τέτοιου τύπου. Κάθε ζεύγος αναφορών μνήμης αντιστοιχίζεται με το ζεύγος βρόχων επανάληψης από τους οποίους περικλείονται οι δύο αναφορές. Τα δεδομένα που απαιτούνται για να πραγματοποιηθεί αυτή η συσχέτιση μεταξύ αναφορών και βρόχων επανάληψης αποκτούνται με στατική ανάλυση του πηγαίου κώδικα. Η διαδικασία που περιγράφεται στην παρούσα ενότητα εκτελείται πάνω σε μια δομή που λέγεται *δέντρο εμφώλευσης βρόχων επανάληψης*. Ένα τέτοιο δέντρο αναπαριστά την ιεραρχία των βρόχων επανάληψης του προγράμματος εισόδου. Το μπλοκ κώδικα που περιλαμβάνει τους βρόχους επανάληψης (το σώμα της συνάρτησης) αναπαρίσταται από την ρίζα του δέντρου και αναφέρεται ως r. Οι υπόλοιποι κόμβοι αναπαριστούν βρόχους επανάληψης που περιλαμβάνονται σε αυτό το μπλοκ και κάθε ένας από αυτούς χαρακτηρίζεται από έναν



Σχήμα 3.5: Δέντρο εμφώλευσης βρόχων επανάληψης.

ξεχωριστό ακέραιο θετικό αριθμό. Ο βρόχος επανάληψης ο οποίος περικλείει την πηγή ενός ζεύγους αναφορών αποκαλείται *βρόχος-πηγή*, ενώ αυτός που περικλείει τον στόχο λέγεται *βρόχος-στόχος*. Το σχήμα 3.5 δείχνει ένα παράδειγμα ιεραρχίας βρόχων που αποτελείται από 8 βρόχους επανάληψης.

Χρησιμοποιώντας ένα δέντρο εμφώλευσης βρόχων επανάληψης μπορεί να γίνει η επιλογή ανάμεσα από έναν μετασχηματισμό συγχώνευσης (fusion) και έναν μετασχηματισμό υπερκόμβων (tiling):

1. Επιλέγεται μετασχηματισμός υπερκόμβων εάν ο βρόχος-πηγή της επαναχρησιμοποίησης είναι ο ίδιος με τον βρόχο-στόχο ή εάν ο ένας είναι πρόγονος του άλλου στο δέντρο εμφώλευσης, καθώς και στις δύο περιπτώσεις η επαναχρησιμοποίηση συμβαίνει μέσα σε επαναλήψεις της ίδιας φωλιάς βρόχων.

2. Εάν ο ένας από τους κόμβους / βρόχους δεν είναι πρόγονος του άλλου σημαίνει ότι η επαναχρησιμοποίηση συμβαίνει σε επαναλήψεις διαφορετικών βρόχων και φωλιών και προτείνεται η συγχώνευσή τους.

Η μέθοδος αυτή μπορεί να εφαρμοστεί υλοποιώντας μια δομή δεδομένων τύπου δέντρου. Στην υλοποίηση που πραγματοποιήθηκε στο MemAssist, ωστόσο, αποφεύχθηκε η χρήση μιας τέτοιας δομής δεδομένων και των συνεπακόλουθων διασχίσεων (traversals) σε αυτή. Ορίζεται αρχικά ένας αριθμός συνόλων (sets) που ισοδυναμούν με τον αριθμό των βρόχων επανάληψης. Κάθε σύνολο αντιστοιχεί σε έναν κόμβο του δέντρου εμφώλευσης βρόχων και συμβολίζεται με P_x. Τα περιεχόμενα αυτών των συνόλων είναι οι κόμβοι που σχηματίζουν ένα απευθείας μονοπάτι από τον κόμβο x μέχρι την ρίζα του δέντρου. Αυτό μπορεί να οριστεί χρησιμοποιώντας την σημειογραφία δημιουργίας συνόλων (set-builder notation), όπως φαίνεται στις εξισώσεις 3.8 και 3.9. Η τελική απόφαση συνάγεται ανάλογα με το ποια από τις εκφράσεις 3.10 - 3.14 είναι αληθής. Αν η έκφραση 3.10 είναι αληθής τότε κανένας από τους κόμβους δεν είναι πρόγονος του άλλου και προτείνεται η συγχώνευση των δύο βρόχων επανάληψης. Στην περίπτωση που πηγή και στόχος αναφέρονται στον ίδιο κόμβο ή ο ένας κόμβος είναι πρόγονος του άλλου, προτείνεται μετασχηματισμός υπερκόμβων (εκφράσεις 3.11 και 3.12). Αν μια από τις εκφράσεις 3.13 και 3.14 είναι αληθής, σημαίνει ότι μια από τις αναφορές μνήμης τους ζεύγους αναφορών δεν βρίσκεται μέσα σε βρόχο επανάληψης, οπότε δεν προτείνεται κανένας μετασχηματισμός. Στο σχήμα 3.5 φαίνεται ένα παράδειγμα όπου η αναφορά-πηγή βρίσκεται μέσα στον βρόχο επανάληψης 2 και η αναφορά-στόχος βρίσκεται στον βρόχο επανάληψης 4. Κάνοντας χρήση της προτεινόμενης μεθόδου, όλα αυτά αναπαρίστανται όπως δείχνει το παράδειγμα 3.3, ενώ οι πράξεις για την επιλογή του προτεινόμενου μετασχηματισμού φαίνονται στην εξίσωση 3.16.

Ορισμός 3.3

$$T = \{x \mid (x \in \mathbb{Z} \land 0 \leq x < N) \lor x = r\} \tag{3.8}$$

$$P_x = \{y \mid (y \in T \land y = predecessor(x)) \lor y = x\} \tag{3.9}$$

όπου το T περιέχει όλους τους κόμβους του δέντρου, τόσο αυτούς που σχετίζονται με βρόχους επανάληψης, όσο και αυτόν που σχετίζεται με το μπλόκ που περικλείει τους βρόχους (r).

$$(T_{source} \cap P_{sink} = \{\varnothing\}) \land (T_{sink} \cap P_{source} = \{\varnothing\}) \tag{3.10}$$

$$(T_{source} \cap P_{sink} = T_{source}) \land (T_{sink} \cap P_{source} = \{\varnothing\}) \tag{3.11}$$

$$(T_{source} \cap P_{sink} = \{\varnothing\}) \wedge (T_{sink} \cap P_{source} = T_{sink}) \tag{3.12}$$

$$(T_{source} \cap P_{sink} = \{r\}) \wedge (T_{sink} \cap P_{source} = \{\varnothing\}) \tag{3.13}$$

$$(T_{source} \cap P_{sink} = \{\varnothing\}) \wedge (T_{sink} \cap P_{source} = \{r\}) \tag{3.14}$$

όπου το T_{source} είναι ένα σύνολο που περιέχει μόνο τον βρόχο-πηγή και το T_{sink} περιέχει μόνο τον βρόχο-στόχο.

Παράδειγμα 3.3

$$\begin{gathered} N = 8, T = \{r, 0, 1, 2, 3, 4, 5, 6, 7\}, T_{source} = \{2\}, \\ T_{sink} = \{4\}, P_r = \{r\}, P_0 = \{0, r\}, P_1 = \{1, 0, r\}, \\ P_2 = \{2, 0, r\}, P_3 = \{3, 0, r\}, P_4 = \{4, r\}, P_5 = \{5, 4, r\}, \\ P_6 = \{6, 5, 4, r\}, P_7 = \{7, 4, r\}, P_{source} = P_2, P_{sink} = P_4 \end{gathered} \tag{3.15}$$

$$\left.\begin{gathered} T_{source} \cap P_{sink} = \{2\} \cap \{4, r\} = \{\varnothing\} \\ T_{sink} \cap P_{source} = \{4\} \cap \{2, 0, r\} = \{\varnothing\} \end{gathered}\right\} Fusion \tag{3.16}$$

3.7 Επιπρόσθετες μετρικές αξιολόγησης της τοπικότητας αναφοράς

3.7.1 Βάρος βρόχου επανάληψης

Οι μετρικές *βάρος βρόχου επανάληψης (loop weight metric, LWM)* και *επαναλήψεις βρόχου (loop iterations, LI)* παρέχονται από το MemAssist με σκοπό τον χαρακτηρισμό της κρισιμότητας κάθε βρόχου επανάληψης σε μια εφαρμογή. Οι τιμές αυτές μπορούν να ληφθούν υπόψη κατά την βελτιστοποίηση μιας εφαρμογής αλλά δεν είναι άμεσα συσχετισμένες με τις βασικές προτάσεις που παράγει το εργαλείο κάνοντας χρήση της ανάλυσης των αποστάσεων επαναχρησιμοποίησης δεδομένων. Παρέχουν όμως χρήσιμη επιπλέον γνώση. LI είναι απλά ο αριθμός των επαναλήψεων ενός βρόχου, ενώ η μετρική LWM χαρακτηρίζει έναν βρόχο λαμβάνοντας υπόψη τον αριθμό των πινάκων στους οποίους επιδρά, καθώς και τα μεγέθη τους, αλλά και τον αριθμό των προσπελάσεων που πραγματοποιούνται προς τον καθένα. Οι βρόχοι επανάληψης με υψηλές τιμές LWM-LI είναι εξαιρετικά πιθανό να είναι απαιτητικοί και χρονοβόροι, ενώ αποτελούν καλούς υποψήφιους για την εφαρμογή μετασχηματισμών. Η μετρική

LI υποδεικνύει αν ένας βρόχος ξεχωρίζει λόγω των πολλαπλών επαναλήψεων του σε σύγκριση με τους υπόλοιπους και η LWM προσδιορίζει αν εκτελείται έντονη επεξεργασία δεδομένων. Η μέθοδος υπολογισμού της τιμής LWM για την εκτέλεση ενός βρόχου επανάληψης X περιγράφεται στον ορισμό 3.4.

Ορισμός 3.4

$$LWM(X) = \sum_{i=1}^{N} (VA_i \times VS_i) \tag{3.17}$$

όπου X είναι το όνομα του βρόχου, N είναι ο συνολικός αριθμός των μεταβλητών που προσπελαύνονται μέσα στο βρόχο, VA είναι ο αριθμός των προσπελάσεων κάθε μεταβλητής μέσα στο βρόχο και VS είναι το στατικό μέγεθος κάθε μεταβλητής. Οι βαθμωτές μεταβλητές έχουν τιμή VS ίση με 1 και αντιμετωπίζονται σαν πίνακες μεγέθους 1x1.

Το MemAssist παράγει γραφήματα με τις τιμές LWM και LI, στα οποία επισημαίνονται οι κρίσιμοι βρόχοι επανάληψης με έντονη επεξεργασία δεδομένων. Ως κρίσιμοι ορίζονται οι βρόχοι εκείνοι με τιμές LWM-LI που είναι απόμακρες από αυτές των υπόλοιπων βρόχων. Αυτό υλοποιείται υπολογίζοντας τις άνω έκτοπες στατιστικές τιμές (upper outliers) του γραφήματος και μαρκάροντάς τες ως κρίσιμες.

Η επισήμανση των κρίσιμων βρόχων επανάληψης δεν δίνει απαραίτητα εικόνα γύρω από προβλήματα που σχετίζονται με την κακή τοπικότητα αναφοράς δεδομένων. Οι βρόχοι αυτοί θα πρέπει να εξεταστούν περαιτέρω, ακόμα και αν δεν εντοπιστούν προβλήματα σχετικά με την τοπικότητα αναφοράς και η εργασία που εκτελείται εντός τους θα πρέπει, αν είναι εφικτό, να ελαττωθεί. Ένας από τους σχετικούς ελέγχους που μπορεί να πραγματοποιηθεί είναι εάν γίνονται περίπλοκοι πλεονάζοντες επανυπολογισμοί (recomputations) στους βρόχους που υποδεικνύονται. Σε αυτή την περίπτωση μπορεί να γίνει χρήση πινάκων αντιστοίχισης (lookup arrays) με προϋπολογισμένες τιμές για βελτιστοποίηση του βρόχου. Ακόμα μια απλή αλλαγή είναι να τοποθετηθούν έξω από το σώμα του βρόχου όλες οι εντολές επιλογής οι οποίες δεν αλλάζουν καθ' όλη τη διάρκεια εκτέλεσής του. Αυτός ο μετασχηματισμός είναι γνωστός ως *αποδιακλάδωση βρόχου (loop unswitching)* και εφαρμόζεται αυτόματα από τον μεταγλωττιστή GCC (έκδοση 3.4 και πάνω). Περισσότερες λεπτομέρειες σχετικά με παρόμοιους μετασχηματισμούς μπορούν να βρεθούν στο κεφάλαιο 26 του βιβλίου [104].

3.7.2 Συντελεστής επαναχρησιμοποίησης πινάκων

Η υλοποίηση ενός αλγόριθμου σε κάποιο ενσωματωμένο σύστημα μπορεί να επωφεληθεί αρκετά από την εκμετάλλευση των λανθανουσών μνημών ή μνημών scratch-pad και την ύπαρξη επαναχρησιμοποίησης δεδομένων. Για την ανάλυση και κατανόηση της επαναχρησιμοποίησης δεδομένων στα στοιχεία ενός πίνακα προτείνεται η μετρική *συντελεστής επαναχρησιμοποίησης πίνακα (array reuse factor, ARF).* Η μετρική αυτή υπολογίζει για κάθε στοιχείο ενός πίνακα την μέση τιμή των συνεχόμενων προσπελάσεων ανάγνωσης όπου ακολουθούνται από μια προσπέλαση εγγραφής. Είναι γεγονός πως οι μικρότερες μνήμες είναι λιγότερο δαπανηρές όσον αφορά την καθυστέρηση και την κατανάλωση ενέργειας [60]. Τα στοιχεία ενός πίνακα με υψηλές τιμές ARF μπορούν να επωφεληθούν από τεχνικές που έχουν ως στόχο την ανάθεση των τιμών με πολύ συχνή επαναχρησιμοποίηση σε μικρότερες μνήμες [100]. Η μέθοδος υπολογισμού του συντελεστή επαναχρησιμοποίησης για έναν πίνακα A περιγράφεται στον ορισμό 3.5. Η μετρική ARF εστιάζει στις συνεχόμενες αναγνώσεις από τη μνήμη, καθώς οι συνεχόμενες εγγραφές στην ίδια θέση μνήμης δεν έχουν κάποια χρησιμότητα σε κανέναν αλγόριθμο [60]. Στο παράδειγμα 3.4 παρουσιάζεται η διαδικασία υπολογισμού του συντελεστή επαναχρησιμοποίησης για το στοιχείο 0 ενός πίνακα A. Οι προσπελάσεις μνήμης προς το στοιχείο αυτό φαίνονται στο σχήμα 3.6.

Ορισμός 3.5

$$ARF_{A[i]} = \frac{sumr_{A[i]}}{sumw_{A[i]}} \text{ , where } 0 \leq i < n \tag{3.18}$$

$$TotalARF_A = ARF_{A[0]}, \ldots, ARF_{A[n-1]} \tag{3.19}$$

όπου A είναι το όνομα του υπό εξέταση πίνακα, n είναι το μέγεθος του, i είναι το εκάστοτε στοιχείο του, $sumr$ είναι ο αριθμός των προσπελάσεων ανάγνωσης προς ένα στοιχείο του πίνακα και $sumw$ είναι ο αριθμός των συνεχόμενων προσπελάσεων εγγραφής προς ένα στοιχείο του πίνακα.

Παράδειγμα 3.4

$$ARF_{A[0]} = \frac{sumr_{A[0]} = 3 + 3 + 2 + 1 = 9}{sumw_{A[0]} = 4} = 2.25 \tag{3.20}$$

Σχήμα 3.6: Παράδειγμα προσπελάσεων μνήμης του στοιχείου 0 ενός πίνακα *A*.

3.8 Σύγκριση με το εργαλείο SLO

Το SLO [29, 30, 31, 32] είναι ένα εργαλείο βελτιστοποίησης, το οποίο έχει παρόμοιες λειτουργίες με το MemAssist. Από τη στιγμή που το SLO είναι άμεσα συγκρίσιμο με το MemAssist, πραγματοποιήθηκε μια αποτίμηση των διαφορών και των ομοιοτήτων ανάμεσα στα δύο εργαλεία. Ο πίνακας 3.1 συνοψίζει αυτή τη σύγκριση, ενώ ακολουθεί μια πιο λεπτομερής περιγραφή:

1. Και τα δύο εργαλεία χρησιμοποιούνται για την εφαρμογή βελτιστοποιήσεων σε επίπεδο πηγαίου κώδικα και προτείνουν τους ίδιους μετασχηματισμούς βρόχων επανάληψης (την συγχώνευση βρόχων και τον μετασχηματισμό υπερκόμβων).

2. Το SLO αποτελείται από δύο ξεχωριστά μέρη, ένα για την δυναμική ανάλυση του κώδικα εισόδου και ένα για την οπτικοποίηση και παρουσίαση των αποτελεσμάτων / προτάσεων βελτιστοποίησης. Γίνεται χρήση ενός εργαλείου δυναμικής ανάλυσης κώδικα, βασισμένου στον μεταγλωττιστή GCC, η έξοδος του οποίου τροφοδοτείται σε ένα εργαλείο οπτικοποίησης που παρουσιάζει στο χρήστη όλους τους προτεινόμενους μετασχηματισμούς. Το εργαλείο δυναμικής ανάλυσης βασίζεται σε μια αρκετά παλιά έκδοση του GCC και είναι αρκετά περίπλοκη η εγκατάσταση και το στήσιμο του για τον μέσο χρήστη. Αντιθέτως, στο MemAssist όλα τα βήματα είναι ενοποιημένα και η ροή του εργαλείου μπορεί να ακολουθηθεί απρόσκοπτα, μέσω μιας επέκτασης που αναπτύχθηκε για το περιβάλλον του Visual Studio. Επιπλέον, το MemAssist προσφέρει και μια δεύτερη διεπαφή, μέσω μιας διαδικτυακής εφαρμογής, για τους χρήστες που δεν χρησιμοποιούν το Visual Studio και για όσους θέλουν απλά να ελέγξουν τις δυνατότητες του εργαλείου.

3. Και τα δύο εργαλεία μετράνε χρονικές αποστάσεις και αποστάσεις επαναχρησιμοποίησης δεδομένων, ενώ συμπεριλαμβά-

νουν δυνατότητες εφαρμογής δειγματοληψίας για την επιτάχυνση της διαδικασίας μέτρησης.

4. Το MemAssist μπορεί να δεχτεί και MATLAB είσοδο επιπρόσθετα της C, η οποία έχει ήδη χρησιμοποιηθεί εκτενώς σαν προδιαγραφή εισόδου στα περισσότερα παρόμοια συστήματα, συμπεριλαμβανομένου του SLO.

5. Η χρήση του MEMSCOPT σαν εργαλείο ενορχήστρωσης πηγαίου κώδικα από το MemAssist περιορίζει την είσοδο σε κώδικα C που να είναι συμβατός με το στάνταρ C89. Ωστόσο, σχεδιάζεται η ανάπτυξη ενός επιπλέον εργαλείου ενορχήστρωσης, το οποίο θα κάνει χρήση του μεταγλωττιστή Clang ως εμπρόσθιο τμήμα για την γλώσσα C.

6. Το SLO είναι ειδικό αποκλειστικά στην παροχή προτάσεων βελτιστοποίησης μέσω ανάλυσης της απόστασης επαναχρησιμοποίησης δεδομένων, ενώ το MemAssist παρέχει στον χρήστη μια γενικότερη εικόνα σχετικά με πιθανές προοπτικές βελτιστοποίησης, κάνοντας χρήση και κάποιων επιπρόσθετων μετρικών, οι οποίες δεν έχουν αναγκαστικά άμεση σχέση με την απόσταση επαναχρησιμοποίησης δεδομένων.

7. Αυτή τη στιγμή πραγματοποιείται εργασία για την αυτοματοποίηση της εφαρμογής των βελτιστοποιήσεων που προτείνονται από το MemAssist, κατευθείαν μέσα στον πηγαίο κώδικα, χωρίς καθόλου παρέμβαση από το χρήστη. Αυτό μπορεί να επιτευχθεί αξιοποιώντας τις λειτουργίες εφαρμογής μετασχηματισμών βρόχων επανάληψης που παρέχει ο μεταγλωττιστής MEMSCOPT [61].

Στις εργασίες [29] και [30] παρουσιάζεται η προσέγγιση που ακολουθείται από το SLO για να συνάγει σωστούς μετασχηματισμούς βρόχων επανάληψης, οι οποίοι βελτιστοποιούν την τοπικότητα αναφοράς δεδομένων. Η όλη διαδικασία σε αυτή την προσέγγιση πραγματοποιείται στο επίπεδο βασικών μπλοκ (basic blocks) αντί για το επίπεδο των βρόχων επανάληψης. Χρησιμοποιείται ένας *γράφος ροής ελέγχου (control flow graph, CFG)* σε συνδυασμό με μια δομή, παρόμοια με το δέντρο εμφώλευσης βρόχων επανάληψης, η οποία λέγεται *δάσος εμφώλευσης βρόχων επανάληψης*. Το CFG χρησιμοποιείται για την εξαγωγή του ζεύγους των *outermost executed loop headers (OELH)*

Πίνακας 3.1: Σύγκριση μεταξύ των εργαλείων SLO και MemAssist.

	SLO	MemAssist
Χρήση	Ερευνητική	Ερευνητική, Υποστήριξη προγραμματιστών σε πραγματικές συνθήκες ανάπτυξης
Σκοπός	Βελτιστοποιήσεις σε επίπεδο πηγαίου κώδικα	Βελτιστοποιήσεις σε επίπεδο πηγαίου κώδικα
Προτάσεις:		
- Συγχώνευση βρόχων (Fusion)	✔	✔
- Μετασχηματισμός υπερκόμβων (Tiling)	✔	✔
Γραφικό περιβάλλον	✔	✔
Περιβάλλον web	✔	✔
Εμπρόσθιο τμήμα	GCC	MEMSCOPT (C), MAFE (MATLAB)
Εργαλείο ενορχήστρωσης κώδικα	GCC	MEMSCOPT, Clang (⋰)
Μέτρηση	Χρονική απόσταση, Απόσταση επαναχρησιμοποίησης	Χρονική απόσταση, Απόσταση επαναχρησιμοποίησης
Δειγματοληψία	✔	✔ (προαιρετικά)
Τρόπος ανάλυσης συναρτήσεων	Interprocedural	Intraprocedural, Interprocedural (⋰)
Επιτρόσθετες μετρικές	✔	✔ (LWM, LI, ARF)
Αυτόματη εφαρμογή μετασχηματισμών	✔	⋰
Ενσωματωμένος προσομοιωτής ιεραρχίας μνήμης	✔	✔
Εξαγωγή αρχείου ροής προσπελάσεων	✔	✔

✔ = Υπάρχον χαρακτηριστικό, ✔ = Μη υποστηριζόμενο χαρακτηριστικό, ⋰ = Υπό εξέλιξη εργασία.

για τα βασικά μπλοκ στα οποία εμφανίζεται η πηγή και η απόληξη μιας επαναχρησιμοποίησης. Ως OELH ενός βασικού μπλοκ ορίζεται ο πρόγονος του που βρίσκεται πιο κοντά στη ρίζα του δάσους εμφώλευσης βρόχων επανάληψης, ο οποίος εκτελείται ανάμεσα από την προσπέλαση-πηγή (use) και την προσπέλαση-επαναχρησιμοποίηση (reuse). Δεδομένων των OELH πηγής και OELH απόληξης: (1) συνάγεται ένας μετασχηματισμός υπερκόμβων (tiling), αν αναφέρονται και τα δύο στον ίδιο κόμβο, καθώς η πηγή επαναχρησιμοποίησης και η απόληξη επαναχρησιμοποίησης θα συνέβαιναν μεταξύ επαναλήψεων του ίδιου βρόχου και (2) συνάγεται ένας μετασχηματισμός συγχώνευσης βρόχων (fusion), αν αναφέρονται σε διαφορετικούς βρόχους επανάληψης. Το πλεονέκτημα της προσέγγισης που βασίζεται σε σύνολα, η οποία χρησιμοποιείται στο MemAssist, έναντι της αντίστοιχης που προτείνεται στις εργασίες [29] και [30] είναι ότι δεν χρειάζεται ένα CFG και αναλυτικές πληροφορίες για τα βασικά μπλοκ του προγράμματος για να λειτουργήσει. Τα μόνα δεδομένα εισόδου που απαιτούνται αφορούν την ιεραρχία των βρόχων επανάληψης μαζί με πληροφορία σχετικά με το ποιος βρόχος επανάληψης περικλείει κάθε αναφορά μνήμης. Επομένως, η μέθοδος που προτείνεται και την οποία χρησιμοποιεί το MemAssist είναι κατά πολύ ευκολότερο να υλοποιηθεί από την αντίστοιχη του SLO, ενώ συνάγει τους ίδιους μετασχηματισμούς. Όσον αφορά την ακρίβεια, οι δύο μέθοδοι παράγουν παρόμοια αποτελέσματα.

Κεφάλαιο 4

Ανάπτυξη εργαλείων ανάλυσης πηγαίου κώδικα

Πληθώρα ακαδημαϊκών και εμπορικών εργαλείων για την ανάλυση και τον χειρισμό πηγαίου κώδικα έχουν προταθεί κατά καιρούς, ειδικά τις δύο τελευταίες δεκαετίες. Τέτοιου είδους εργαλεία αποτελούν πλέον αναπόσπαστο μέρος της ροής εργασίας των προγραμματιστών και όλα τα κύρια σύγχρονα περιβάλλοντα εργασίας τα ενσωματώνουν σαν επεκτάσεις για διάφορες λειτουργίες. Μπορούν να χρησιμοποιηθούν σε τομείς όπως η κατανόηση προγράμματος (program comprehension), η αναδόμηση κώδικα (code refactoring), η εξαγωγή μετρικών, η αποσφαλμάτωση κ.α.

Ένα εργαλείο ανάλυσης και χειρισμού πηγαίου κώδικα τυπικά περιλαμβάνει ένα εμπρόσθιο τμήμα, για να πραγματοποιεί λεκτική και συντακτική ανάλυση στην/στις γλώσσα/γλώσσες εισόδου και να παράγει κάποιας μορφής *ενδιάμεση αναπαράσταση (intermediate representation, IR).* Η αναπαράσταση αυτή συνήθως έχει τη μορφή κάποιου *αφηρημένου συντακτικού δέντρου (abstract syntax tree, AST).* Πολλά συστήματα μεταγλώττισης παρέχουν εμπρόσθια τμήματα για μια ή πολλαπλές γλώσσες καθώς και έναν μηχανισμό για την πρόσβαση στην IR. Η χρήση κάποιου υπάρχοντος συστήματος θα απλοποιούσε τα πράγματα και το βάρος της ανάπτυξης του εμπρόσθιου τμήματος και της σχεδίασης της ενδιάμεσης αναπαράστασης δε θα

έπεφτε στον προγραμματιστή. Τα εργαλεία Roslyn [112] και Clang [99] είναι ενδεικτικά παραδείγματα διεπαφών που ακολουθούν αυτή την προσέγγιση. Ωστόσο, το μειονέκτημα αυτής της προσέγγισης είναι ο περιορισμός της εισόδου αποκλειστικά στις γλώσσες που υποστηρίζει ο εκάστοτε μεταγλωττιστής.

Τα εργαλεία αυτού του είδους περιλαμβάνουν συνήθως και ένα υποσύστημα για την συλλογή πληροφορίας από την ενδιάμεση αναπαράσταση σχετικά με το πρόγραμμα εισόδου και τις ιδιότητες του. Σε πολλές περιπτώσεις, τα υπάρχοντα συστήματα μεταγλωττιστών δεν χρησιμοποιούν κάποιο μηχανισμό εφαρμογής ερωτημάτων για αυτό το σκοπό. Αντ' αυτού, πρέπει να αναπτυχθούν και να εφαρμοστούν εξατομικευμένα περάσματα στην ενδιάμεση αναπαράσταση. Πρέπει να γραφτεί περίπλοκος κώδικας σε κάποια γλώσσα προγραμματισμού γενικού σκοπού για την ανάπτυξη αυτών των περασμάτων χωρίς τη βοήθεια ενός μηχανισμού εφαρμογής ερωτημάτων. Επιπλέον, η εφαρμογή ερωτημάτων σε μια ενδιάμεση αναπαράσταση μπορεί να θεωρηθεί εξειδικευμένο (domain specific) πρόβλημα. Η μοντελοποίηση του με μια *γλώσσα ειδικού σκοπού (domain specific language, DSL)* θα μπορούσε να προσφέρει καλύτερη εκφραστικότητα και κατά συνέπεια να μειώσει το χρόνο που απαιτείται για την ανάπτυξη ενός εργαλείου ανάλυσης κώδικα. Οι γλώσσες ειδικού σκοπού χρησιμοποιούνται ευρέως για την διευκόλυνση της μοντελοποίησης προβλημάτων σε διάφορους επιστημονικούς τομείς [69].

Δεδομένης της πολυπλοκότητας που εμπεριέχει η ανάπτυξη εργαλείων ανάλυσης και χειρισμού πηγαίου κώδικα, η αυτοματοποίηση της σχετικής διαδικασίας μπορεί να θεωρηθεί πολύ σημαντική, καθώς είναι ικανή να οδηγήσει σε μεγάλη μείωση του χρόνου, του κόπου και του κόστους που απαιτούνται. Αυτό το κεφάλαιο αναφέρεται στην υλοποίηση λογισμικού για την αυτόματη ανάπτυξη εργαλείων ανάλυσης και χειρισμού κώδικα, με στόχο τη μείωση του σχετικού κόστους. Για το σκοπό αυτό προτείνεται ένα σύστημα που αποτελείται από ένα σύνολο αλληλεξαρτώμενων εργαλείων, το οποίο χρησιμοποιήθηκε και για την ανάπτυξη των MEMSCOPT και MemAssist. Τα επιμέρους εργαλεία που το απαρτίζουν περιλαμβάνουν:

- Την γλώσσα προγραμματισμού ειδικού σκοπού *CastQL*. Μια γλώσσα εφαρμογής ερωτημάτων (query language) που λειτουργεί σε συνδυασμό με κάποια απαραίτητα βήματα που πρέπει να ακολουθηθούν κατά την ανάπτυξη ερωτημάτων εξαρτώμενων από τη γλώσσα εισόδου. Τα βήματα αυτά ξεκινούν από

τη διατύπωση του προβλήματος και φτάνουν στην υλοποίηση. Η CastQL λειτουργεί πάνω σε μια AST αναπαράσταση που λέγεται *cAST (contextual abstract syntax tree)*. Πρόκειται για μια εσωτερική/ενσωματωμένη (internal/embedded) γλώσσα προγραμματισμού ειδικού σκοπού [69] που κάνει χρήση της C++ ως γλώσσα υποδοχής. Οι σχεδιαστικοί κανόνες και τα μοτίβα που περιγράφονται στις εργασίες [69] και [74] ακολουθήθηκαν κατά την υλοποίηση της CastQL.

- Έναν αυτόματο γεννήτορα εμπρόσθιων τμημάτων, με το όνομα FEgen, ο οποίος παράγει έναν λεκτικό και συντακτικό αναλυτή για οποιαδήποτε δοθείσα γραμματική κάνει χρήση της μορφής συμβολισμού BNF. Ο κώδικας που περιγράφει το εμπρόσθιο τμήμα και τις προδιαγραφές της cAST αναπαράστασης παράγεται αυτόματα και δεν απαιτείται η συγγραφή C++ κώδικα από τον χρήστη. Το FEgen πρακτικά καθιστά εφικτή τη χρήση της CastQL για οποιαδήποτε γλώσσα εισόδου.

Παρόμοιες τεχνολογίες είναι συνήθως προσανατολισμένες προς την ανάλυση πηγαίου κώδικα [12, 13, 27, 46, 50, 52, 54, 76, 102, 150] ή προς το χειρισμό πηγαίου κώδικα και την εφαρμογή μετασχηματισμών [22, 36, 48, 53, 83, 85, 146]. Το προτεινόμενο σύστημα στοχεύει στην γρήγορη ανάπτυξη εργαλείων και για τις δυο περιοχές. Τα πλεονεκτήματα χρήσης του είναι τα ακόλουθα:

- Δεν υφίσταται περιορισμός σε συγκεκριμένες προκαθορισμένες γλώσσες εισόδου. Οι προδιαγραφές της γλώσσας εισόδου μπορούν να εξατομικευθούν μέσω της μορφής συμβολισμού γραμματικών BNF.

- Η εκφραστικότητα της CastQL απλοποιεί την εφαρμογή ερωτημάτων σε πηγαίο κώδικα. Ακόμα και στην περίπτωση που ένα ερώτημα δεν μπορεί να εκφραστεί σε CastQL, παρέχεται ως εναλλακτική η δυνατότητα εφαρμογής περασμάτων στην cAST, τα οποία έχουν αναπτυχθεί κατευθείαν σε C++.

- Τα εργαλεία που παράγονται από το προτεινόμενο σύστημα μπορούν να διανεμηθούν ως αυτόνομες εφαρμογές γραμμένες σε C++. Έτσι, δεν είναι αναγκαία η χρήση κάποιου εξειδικευμένου περιβάλλοντος για την εκτέλεση τους. Τα εξειδικευμένα αυτά περιβάλλοντα κυριαρχούν στις περισσότερες σχετικές εργασίες.

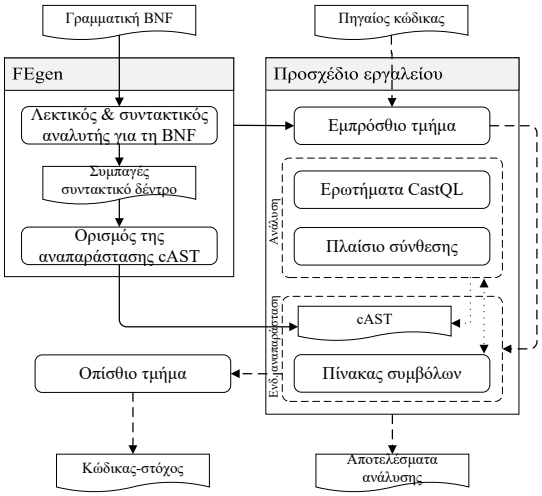
Το προτεινόμενο σύστημα έχει αξιολογηθεί εκτενώς. Πιο συγκεκριμένα, έχει ελεγχθεί η λειτουργία του για δύο πρακτικά σενάρια χρήσης που εφαρμόστηκαν σε ένα σύνολο δοκιμαστικών εφαρμογών. Ο γεννήτορας εμπρόσθιων τμημάτων FEgen έχει χρησιμοποιηθεί για την αυτόματη παραγωγή εμπρόσθιων τμημάτων για είκοσι διαφορετικές γραμματικές γλωσσών εισόδου. Στο κεφάλαιο 6 παρουσιάζονται τα παραπάνω μαζί με δεδομένα που αποδεικνύουν την αποδοτικότητα του προτεινόμενου συστήματος.

Στο παρόν κεφάλαιο γίνεται αναλυτική παρουσίαση του συστήματος CastQL-FEgen. Αρχικά, στην ενότητα 4.1 παρέχεται μια επισκόπηση της προτεινόμενης ροής εργαλείων, ενώ η γλώσσα CastQL περιγράφεται πιο αναλυτικά στην ενότητα 4.3. Έπειτα, στην ενότητα 4.4 γίνεται παρουσίαση της διαδικασίας που ακολουθείται για την ανάπτυξη ορισμένων βασικών λειτουργειών του μεταγλωττιστή MEMSCOPT, κάνοντας χρήση των CastQL και FEgen.

Ακόμα, αναφορά στην CastQL και στο FEgen γίνεται και σε άλλα κεφάλαια της διατριβής. Στην ενότητα 6.2 γίνεται πειραματική αξιολόγηση με ένα σύνολο σεναρίων που κάνουν χρήση της προτεινόμενης προσέγγισης. Τέλος, η ενότητα 2.4 παρουσιάζει μια ανασκόπηση των σχετικών εργασιών, που αφορούν την ανάπτυξη εργαλείων ανάλυσης πηγαίου κώδικα.

4.1 Επισκόπηση της προτεινόμενης ροής εργαλείων

Το προτεινόμενο σύστημα απαρτίζεται από ένα σύνολο διαφορετικών αλληλεξαρτώμενων στοιχείων. Τα κυριότερα από αυτά είναι η γλώσσα CastQL και το εργαλείο FEgen. Στο σχήμα 4.1 φαίνεται η δομή του συστήματος που αποτελείται από δύο κύρια στάδια λειτουργίας: (1) Το στάδιο παραγωγής του εμπρόσθιου τμήματος, όταν δημιουργείται ένα προσχέδιο του παραγόμενου εργαλείου κάνοντας χρήση του FEgen. (2) Το περιβάλλον ανάπτυξης που χρησιμοποιείται μετέπειτα για την επέκταση αυτού του προσχεδίου. Αυτά τα δύο στάδια απεικονίζονται στο σχήμα 4.1 με τους τίτλους *FEgen* και *Προσχέδιο εργαλείου* αντίστοιχα. Τα βέλη με συνεχόμενες γραμμές χρησιμοποιούνται για να υποδείξουν τη ροή μεταξύ των εργασιών που πραγματοποιούνται εσωτερικά στο FEgen. Τα βέλη με διακεκομμένες γραμμές και τελείες υποδεικνύουν συνδιαλλαγές μεταξύ των διαφορετικών κομματιών του παραγόμενου εργαλείου, ενώ εκείνα με απλές διακεκομμένες γραμμές παρουσιάζουν τη ροή που ακολουθούν τα



Σχήμα 4.1: Επισκόπηση της προτεινόμενης ροής εργαλείων CastQL-FEgen.

μέρη που απαρτίζουν το παραγόμενο εργαλείο.

Το FEgen χρησιμοποιείται για την ανάγνωση και ανάλυση μιας γραμματικής BNF και την εξαγωγή του κώδικα που παράγει το συντακτικό δέντρο. Για να το πετύχει αυτό, δημιουργεί μια αναθεωρημένη έκδοση της γραμματικής υπό τη μορφή κλασσικών αρχείων *.y* και *.l* που χρησιμοποιούνται από τον γεννήτορα συντακτικών αναλυτών GNU bison και τον γεννήτορα λεκτικών αναλυτών Flex αντίστοιχα. Κατά το επόμενο βήμα, παρέχονται δύο επιλογές για τον καθορισμό της AST αναπαράστασης:

- Το FEgen μπορεί να εξάγει και να προσδιορίσει αυτόματα μια AST αναπαράσταση από το συμπαγές συντακτικό δέντρο. Πιο συγκεκριμένα, εφαρμόζει ένα πέρασμα που μεταλλάσσει το συντακτικό δέντρο ως ακολούθως: (1) Τα αναδρομικά στοιχεία της BNF γραμματικής μετατρέπονται σε μια λίστα από αντικείμενα. (2) Τα ενδιάμεσα αλυσιδωτά γραμματικά σύμβολα καταργούνται.
- Παρέχεται στο χρήστη ένα γραφικό περιβάλλον για τον ορι-

σμό δικών του στοιχείων στο AST μέσω μαρκαρίσματος του συντακτικού δέντρου με ειδικές εντολές.

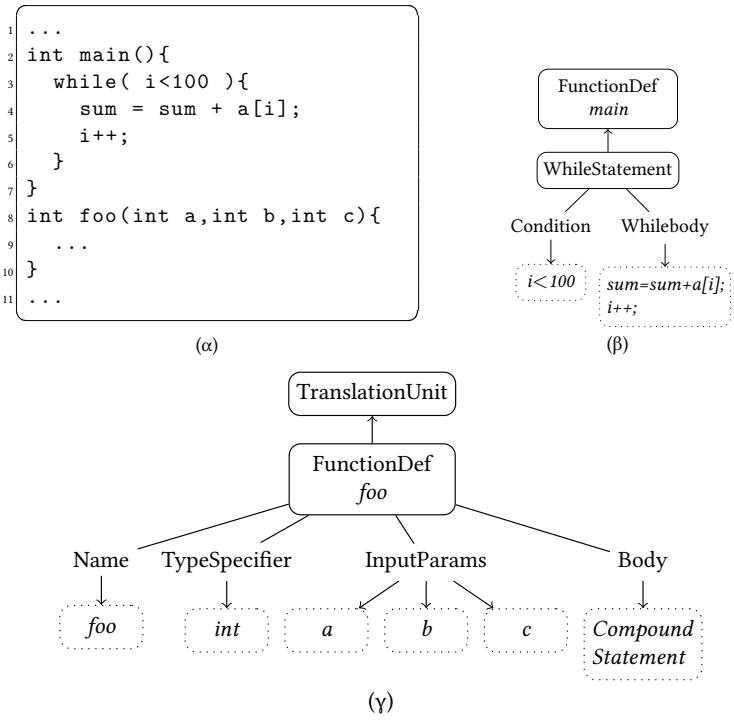
Ο πηγαίος κώδικας που παράγεται από το FEgen περιλαμβάνει το εμπρόσθιο τμήμα του εργαλείου και τις προδιαγραφές της ενδιάμεσης αναπαράστασης. Η ενδιάμεση αναπαράσταση περιλαμβάνει το AST και έναν πίνακα συμβόλων. Το FEgen δεν μπορεί να εξάγει πίνακα συμβόλων για οποιαδήποτε γλώσσα εισόδου. Παρέχει μόνο έναν βασικό πρότυπο πίνακα συμβόλων τον οποίο ο χρήστης μπορεί να επεκτείνει ανάλογα τις ανάγκες του.

Κατά το δεύτερο στάδιο ανάπτυξης του εργαλείου, ο χρήστης εφαρμόζει ερωτήματα και χειρίζεται το AST μέσω της γλώσσας CastQL και μιας βιβλιοθήκης που ονομάζεται *πλαίσιο σύνθεσης (synthesis framework)*. Η σχεδίαση των ερωτημάτων γίνεται τμηματικά. Στην ενότητα 4.3 παρέχεται μια περιγραφή των βασικών στοιχείων και του συντακτικού της CAstQL. Το πλαίσιο σύνθεσης είναι μια βιβλιοθήκη γραμμένη σε C++ που βοηθά στο χειρισμό του AST. Έχει τη δυνατότητα να κατασκευάζει υποδέντρα για το AST, τα οποία μπορούν να χρησιμοποιηθούν για λειτουργίες αναδόμησης και παραγωγής κώδικα.

4.2 Αφηρημένο συντακτικό δέντρο

Το σύστημα CastQL-FEgen χρησιμοποιεί ένα AST που απλοποιεί την πλοήγηση και την αναζήτηση στον κώδικα. Το *contextual abstract syntax tree (cAST)* είναι ένα ετερογενές δέντρο, οι κόμβοι του οποίου αναπαριστούν στοιχεία του προγράμματος και συνδέονται μεταξύ τους με ακμές σύμφωνα με τους γραμματικούς κανόνες της γλώσσας εισόδου. Σε αυτή την ενότητα παρουσιάζεται η δομή ενός cAST χρησιμοποιώντας την C σαν παράδειγμα γλώσσας εισόδου. Μπορεί, ωστόσο, να χρησιμοποιηθεί για την αναπαράσταση οποιασδήποτε γλώσσας είναι δυνατό να περιγραφεί με τον συμβολισμό BNF. Για το υπόλοιπο της ενότητας χρησιμοποιούνται και οι δύο όροι, AST και cAST, όταν γίνεται αναφορά στο αφηρημένο συντακτικό δέντρο.

Κάθε τύπος στοιχείου/κόμβου του cAST μοντελοποιείται με μια κλάση, η οποία αρχικοποιείται όταν γίνεται επίσκεψη στον αντίστοιχο κόμβο του συμπαγούς συντακτικού δέντρου, κατά το πέρασμα που παράγει το AST. Οι κλάσεις αυτές μοιράζονται μια κοινή δημόσια διεπαφή: κληρονομούν όλες μια βασική κλάση που λέγεται *IRElement*. Ο χρήστης μπορεί εύκολα να αναπτύξει δικά του περάσματα στο AST, εκμεταλλευόμενος τα σχεδιαστικά μοτίβα *visitor* και *listener*. Για τη



Σχήμα 4.2: α) Παράδειγμα κώδικα C και τα αντίστοιχα μέρη του cAST για β) μια εντολή while και γ) έναν ορισμό συνάρτησης.

διευκόλυνση της πλοήγησης και της αναζήτησης στο AST, κάθε στοιχείο/κόμβος του cAST διαθέτει μηδέν ή περισσότερες *ομάδες συμφραζομένων* (σύμφωνα με τον τύπο του). Στις ομάδες αυτές κατατάσσονται και συνδέονται οι άμεσοι απόγονοι του, ενώ γίνονται οι ακόλουθες υποθέσεις:

1. Η σειρά των ομάδων κάθε στοιχείου του cAST αντιστοιχεί άμεσα με τη σειρά των γραμματικών κανόνων που συνθέτουν το στοιχείο αυτό.

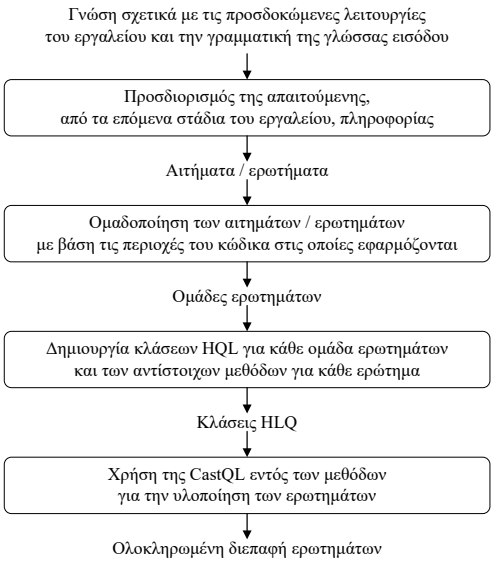
2. Κάθε ομάδα περιλαμβάνει ένα διατεταγμένο σύνολο στοιχείων του cAST, η σειρά διάταξης των οποίων είναι ίδια με τη σειρά εμφάνισής τους στον πηγαίο κώδικα.

Το σχήμα 4.2 δείχνει ένα παράδειγμα κώδικα C και την αντίστοιχη αναπαράσταση του σε cAST. Τα στοιχεία/κόμβοι του cAST αναπαριστώνται με συνεχόμενο μαύρο περίγραμμα και οι ομάδες εμφανίζονται σαν κόμβοι μόνο με κείμενο, χωρίς περίγραμμα. Το σχήμα 4.2β επικεντρώνεται σε μια εντολή while (*WhileStatement*) που περιλαμβάνει τις ομάδες *Condition* και *Whilebody*. Η ομάδα *Condition* συνδέεται με την έκφραση *i<100*, που αναπαριστά τη συνθήκη της εντολής while. Αυτή η έκφραση αναπαρίσταται με παρόμοιο τρόπο, αν ακολουθήσει κανείς το δέντρο από εκεί και κάτω και αναλυθεί περισσότερο αυτό το μέρος του. Στην δεύτερη ομάδα (*Whilebody*) συνδέονται τα στοιχεία που βρίσκονται στο κύριο σώμα της εντολής while. Με το ίδιο σκεπτικό, το σχήμα 4.2γ εστιάζει σε ένα στοιχείο της cAST που αναπαριστά έναν ορισμό συνάρτησης (*FunctionDef*) και περιλαμβάνει τέσσερις ομάδες. Κάνοντας χρήση των ομάδων, σαν έναν επιπλέον βαθμό ελευθερίας, απλοποιείται η πλοήγηση στην ενδιάμεση αναπαράσταση και η δρομολόγηση των περασμάτων γίνεται με μεγαλύτερη ακρίβεια.

4.3 Η γλώσσα ειδικού σκοπού CastQL

Μια στρατηγική πολλαπλών επιπέδων, με ιεραρχικό διαχωρισμό, ακολουθείται για την εφαρμογή ερωτημάτων σε πηγαίο κώδικα. Παρέχονται δύο είδη ερωτημάτων, ανάλογα με το επίπεδο αφαιρετικότητας. Τα *ερωτήματα υψηλού επιπέδου (high level queries, HLQs)* σχετίζονται με το επίπεδο προγράμματος, εξαρτώνται από τη γλώσσα εισόδου και η πληροφορία που ενσωματώνουν είναι άμεσα αξιοποιήσιμη από τα επόμενα στάδια του υπό ανάπτυξη εργαλείου. Τα HLQs συνθέτονται από έναν συνδυασμό *ερωτημάτων χαμηλού επιπέδου (low level queries, LLQs)*, τα οποία διασχίζουν το AST και συλλέγουν κόμβους που πληρούν συγκεκριμένα κριτήρια. Η διαδοχική εφαρμογή ενός συνδυασμού από LLQs επιστρέφει ένα σύνολο από κόμβους του AST που είναι το αποτέλεσμα όλων των LLQs. Αυτό το σύνολο κόμβων ονομάζεται *nodeset*. Η πληροφορία γίνεται έπειτα διαθέσιμη στην υπόλοιπη εφαρμογή μέσω της διεπαφής του HLQ. Τα HLQs είναι γενικά τυπικές κλάσεις της C++ που υλοποιούν μια συγκεκριμένη διεπαφή, ενώ τα LLQs και τα nodesets είναι τα βασικά δομικά στοιχεία της γλώσσας CastQL.

Αυτή η μέθοδος αφήνει το διαμοιρασμό της πληροφορίας σε HLQs στην ευχέρεια του χρήστη. Ωστόσο, στο παράδειγμα πρακτικής εφαρ-



Σχήμα 4.3: Προτεινόμενη ροή υλοποίησης διεπαφών εφαρμογής ερωτημάτων CastQL.

μογής και χρήσης της προτεινόμενης ροής εργαλείων που παρουσιάζεται στην επόμενη ενότητα, δίνονται κάποιες κατευθυντήριες γραμμές για τη σχεδίαση μιας σειράς ερωτημάτων με σκοπό την ανάπτυξη ορισμένων βασικών λειτουργιών του μεταγλωττιστή MEMSCOPT. Το σχήμα 4.3 παρουσιάζει την προτεινόμενη ροή υλοποίησης ενός μηχανισμού εφαρμογής ερωτημάτων.

Το MEMSCOPT υλοποιεί, μεταξύ άλλων, και τον μετασχηματισμό *μετατόπιση βρόχου (loop shift)*. Αυτός ο μετασχηματισμός μετατοπίζει το χώρο επανάληψης ενός βρόχου *for* προς τα εμπρός ή πίσω στο χρόνο μέσω μιας σταθεράς που παρέχεται πριν από τη μεταγλώττιση. Όπως φαίνεται στο σχήμα 4.4, ο συγκεκριμένος μετασχηματισμός επιδρά σε διάφορα μέρη ενός βρόχου επανάληψης, συμπεριλαμβανομένων των σημείων που γίνεται η αρχικοποίηση και ο έλεγχος των *επαγωγικών μεταβλητών (induction variables)* καθώς και σε ορισμένους δείκτες των πινάκων που προσπελαύνονται στο κυρίως σώμα του βρόχου. Μελετώντας την εφαρμογή του μετασχηματισμού

```
for ( x=GB;x<=N-1-GB;++x ){
  for ( y=GB;y<=M-1-GB;++y ){
    for ( k=-GB;k<=GB;++k ){
      temp+=image_in[x+k][y]*Gauss[abs(k)];
    }
    gauss_x_image[x][y]=temp/tot;
  }
}

/*************************************/

for ( x=GB;x<=N-1-GB;++x ){
  for ( y=GB-GB;y<=M-1-GB-GB;++y ){
    for ( k=-GB;k<=GB;++k ){
      temp+=image_in[x+k][y+GB]*Gauss[abs(k)];
    }
    gauss_x_image[x][y+GB]=temp/tot;
  }
}
```

Σχήμα 4.4: Ο μετασχηματισμός μετατόπισης βρόχου (loop shift).

μετατόπισης βρόχου και χρησιμοποιώντας τον σαν παράδειγμα, μπορεί να συμπεραστεί ότι ένα σύστημα χειρισμού πηγαίου κώδικα και εφαρμογής μετασχηματισμών απαιτεί:

1. Τον εντοπισμό μιας επαγωγικής μεταβλητής.
2. Τον εντοπισμό όλων των εκφράσεων στο κυρίως σώμα του βρόχου, στις οποίες εμφανίζεται η μεταβλητή αυτή.
3. Πρόσβαση στις εκφράσεις που βρίσκονται στην περιοχή αρχικοποίησης και στην περιοχή συνθήκης του βρόχου.

Και τα τρία αυτά σενάρια εντοπισμού πληροφορίας σχετίζονται με την περιοχή ενός βρόχου *for* μέσα στον κώδικα. Μια σχεδιαστική επιλογή για τη δημιουργία ενός εργαλείου που εφαρμόζει τέτοιου είδους μετασχηματισμούς θα ήταν να οριστεί μια κλάση HLQ που θα εξυπηρετεί όλα τα αιτήματα εντοπισμού πληροφορίας σχετικής με έναν βρόχο *for*. Αν χρειαστεί μεταγενέστερα να γίνει υλοποίηση και κάποιου άλλου μετασχηματισμού που εμπλέκει βρόχους επανάληψης, μπορεί να επαναχρησιμοποιηθεί ή να επεκταθεί το ίδιο HLQ. Το κριτήριο με βάση το οποίο γίνεται η σχεδίαση των HLQs είναι ο

όσο το δυνατόν καλύτερος καταμερισμός της εντοπιζόμενης πληροφορίας, με βάση τις περιοχές και τα στοιχεία του κώδικα. Οπότε, οι κλάσεις HLQ ορίζονται σε αντιστοιχία με τα στοιχεία του AST που κυριαρχούν στην περιοχή του δέντρου από όπου θα αντληθεί η πληροφορία.

4.3.1 Ερωτήματα υψηλού επιπέδου

Τα HLQs είναι τυπικές κλάσεις της C++ οι οποίες υλοποιούν μια συγκεκριμένη διεπαφή. Οι HLQ κλάσεις περιλαμβάνουν κατάλληλες δημόσιες μεθόδους που εφαρμόζουν ερωτήματα και επιστρέφουν πληροφορία που είναι άμεσα αξιοποιήσιμη από τα επόμενα στάδια του εργαλείου. Για να το πετύχουν αυτό, οι μέθοδοι αυτές χρησιμοποιούν τις δυνατότητες της γλώσσας ειδικού σκοπού CastQL.

Κάθε κλάση HLQ δημιουργείται με σκοπό να εξυπηρετεί ερωτήματα που σχετίζονται με μια συγκεκριμένη περιοχή στον κώδικα και στο AST (π.χ. την περιοχή που σχετίζεται με έναν βρόχο *for*). Ένα αντικείμενο τύπου HLQ, κατά την αρχικοποίησή του, συνδέεται με την ρίζα του υποδέντρου του AST στο οποίο θα πραγματοποιηθεί αναζήτηση. Το σύνολο των αρχικοποιημένων αντικειμένων HLQ χρησιμοποιείται για την εξυπηρέτηση αιτημάτων εντοπισμού πληροφορίας που μπορεί να γίνουν σε επόμενα στάδια του εργαλείου.

4.3.2 Ερωτήματα χαμηλού επιπέδου

Τα αιτήματα εντοπισμού πληροφορίας που εκφράζονται σε υψηλό επίπεδο σε HLQs, υλοποιούνται μέσω ερωτημάτων της CastQL (LLQs) σε χαμηλό επίπεδο, όπου πραγματοποιείται πρόσβαση στο AST και εφαρμογή διασχίσεων σε αυτό. Τα LLQs είναι διακριτές λειτουργίες που εντοπίζουν, συνδυάζουν και επεξεργάζονται πληροφορία από ένα cAST. Όπως έχει αναφερθεί, τα ερωτήματα χαμηλού επιπέδου και κάποιες συλλογές από κόμβους του AST, που λέγονται nodesets, είναι τα βασικά δομικά στοιχεία της γλώσσας ειδικού σκοπού CastQL. Ένα LLQ εφαρμόζεται σε ένα nodeset και πάντα επιστρέφει ένα νέο nodeset σαν αποτέλεσμα. Υπάρχουν διάφοροι τύποι LLQs που είναι διαθέσιμοι, καθώς και διάφορα φίλτρα αναζήτησης, τα οποία μπορούν να εφαρμοστούν σε αυτά. Ακολουθεί μια σύντομη περιγραφή αυτών των φίλτρων και των τύπων των LLQs. Το συντακτικό της γλώσσας παρουσιάζεται στην επόμενη ενότητα μέσω ενός πρακτι-

κού παραδείγματος χρήσης της. Οι βασικοί τύποι ερωτημάτων χαμηλού επιπέδου είναι οι παρακάτω:

- ***node*** - εντοπίζει τα αντικείμενα που αντιστοιχούν σε κάποιον συγκεκριμένο τύπο κόμβου του AST. Μπορούν επίσης να οριστούν και πολλαπλοί τύποι κόμβων προς αναζήτηση.
- ***context*** - εντοπίζει τα αντικείμενα που περιέχονται σε έναν συγκεκριμένο τύπο ομάδας συμφραζομένων στο AST. Μπορούν επίσης να οριστούν και πολλαπλοί τύποι ομάδων προς αναζήτηση.
- ***mixed*** - εντοπίζονται κόμβοι ενός συγκεκριμένου τύπου καθώς και κόμβοι που περιέχονται σε κάποια συγκεκριμένη ομάδα συμφραζομένων.
- ***name*** - εντοπίζει τα αντικείμενα που φέρουν ένα συγκεκριμένο όνομα. Ο πίνακας συμβόλων που παράγεται από το FEgen πρέπει να έχει χρησιμοποιηθεί για να λειτουργήσει ο συγκεκριμένος τύπος LLQ.
- ***complex*** - ένα LLQ αυτού του τύπου δημιουργείται από το συνδυασμό πολλαπλών άλλων, τα οποία εφαρμόζει κατά σειρά στο nodeset εισόδου.
- ***similarity*** - αυτό το LLQ εντοπίζει τους κόμβους του AST που αποτελούν ρίζες υποδέντρων, τα οποία έχουν την ίδια δομή με ένα δοθέν πρότυπο υποδέντρο.
- ***setop*** - επιστρέφει το αποτέλεσμα από την εφαρμογή πράξεων συνόλων στα nodesets που παίρνει σαν είσοδο.
- ***clever*** - αυτό το LLQ έχει την ίδια συμπεριφορά με ένα LLQ τύπου mixed, αλλά μπορεί επιπλέον να διακόψει την διάσχιση όταν περάσει από συγκεκριμένους τύπους κόμβων ή ομάδων.

Τα φίλτρα αναζήτησης είναι ουσιαστικά παράμετροι των LLQs. Ορίζουν το ποιοι από τους κόμβους που έχουν εντοπισθεί θα προωθηθούν στην έξοδο και το πότε θα τερματίσει τη λειτουργία του το LLQ. Οι βασικοί τύποι φίλτρων σε έναν LLQ είναι οι παρακάτω:

- ***shallow*** - απορρίπτει τους κόμβους που ανακαλύπτονται σε τοποθεσίες του AST που είναι διαφορετικές από τους άμεσους απογόνους των σημείων που ξεκινά η αναζήτηση.

- ***deep first found*** - η αναζήτηση μπορεί να φτάσει σε οποιοδήποτε βάθος, αλλά επιστρέφει τον πρώτο κόμβο που είναι ίδιος με τον κόμβο που δίνεται στην είσοδο.
- ***deep random access*** - η αναζήτηση μπορεί να φτάσει σε οποιοδήποτε βάθος, αλλά επιστρέφει τον *n*-οστό κόμβο του AST κατά τη σειρά που εντοπίστηκε.
- ***exhaustive*** - η αναζήτηση μπορεί μα φτάσει σε οποιοδήποτε βάθος του δέντρου και επιστρέφει όλους τους κόμβους που είναι ίδιοι με τον κόμβο που δίνεται στην είσοδο.
- ***depth specific first found*** - η αναζήτηση επιστρέφει τον πρώτο κόμβο που είναι ίδιος με τον κόμβο που δίνεται στην είσοδο και βρίσκεται το πολύ μέχρι ένα δοθέν βάθος.
- ***depth specific random access*** - η αναζήτηση μπορεί να φτάσει μέχρι ένα δοθέν βάθος, αλλά επιστρέφει τον *n*-οστό κόμβο κατά τη σειρά εντοπισμού.
- ***depth specific exhaustive*** - η αναζήτηση επιστρέφει το σύνολο των κόμβων του AST που είναι ίδιοι με τον κόμβο εισόδου και βρίσκονται το πολύ μέχρι ένα δοθέν βάθος.

Ακόμα μία παράμετρος των LLQs αφορά τη σειρά και την κατεύθυνση των διασχίσεων του AST. Η διάσχιση κατά βάθος (depth-first search, DFS) αποτελεί την προκαθορισμένη επιλογή ενώ είναι διαθέσιμες και οι εναλλακτικές της διάσχισης κατά πλάτος (breadth-first search, BFS) και της διάσχισης προς τα επάνω. Στην διάσχιση προς τα επάνω αλλάζει η κατεύθυνση της διάσχισης και σε κάθε βήμα γίνεται επίσκεψη στον γονέα κάθε κόμβου αντί για τους απογόνους του.

4.4 Εφαρμογή των προτεινόμενων εργαλείων

Η γλώσσα προγραμματισμού ειδικού σκοπού CastQL και το εργαλείο αυτόματης παραγωγής εμπρόσθιων τμημάτων FEgen έχουν χρησιμοποιηθεί για την υλοποίηση ενός αριθμού εργαλείων μεταγλώττισης και ανάλυσης προγραμμάτων [61, 88, 89, 90, 141]. Σε αυτή την ενότητα περιγράφεται η χρήση του προτεινόμενου συστήματος για την ανάπτυξη βασικών λειτουργιών του μεταγλωττιστή MEMSCOPT

[61]. Ο μεταγλωττιστής αυτός έχει χρησιμοποιηθεί σαν εργαλείο ενορχήστρωσης πηγαίου κώδικα (source code instrumentor) για τις ανάγκες του MemAssist.

4.4.1 Επισκόπηση του εργαλείου ανάλυσης και μετασχηματισμού κώδικα MEMSCOPT

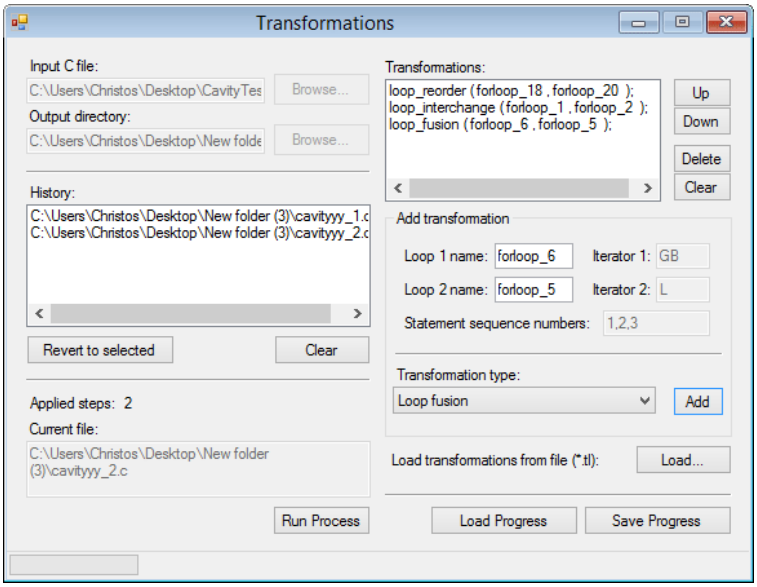
Ο μεταγλωττιστής MEMSCOPT είναι ένα διαδραστικό εργαλείο βελτιστοποίησης πηγαίου κώδικα, το οποίο αναπτύχθηκε αρχικά στα πλαίσια του Ευρωπαϊκού ερευνητικού έργου ENOSYS[1]. Δρα σε προγράμματα γραμμένα στη γλώσσα C, που είναι συμβατά με το στάνταρ C89. Το εργαλείο έχει δυο καταστάσεις λειτουργίας: (1) την *κατάσταση ανάλυσης* και (2) την *κατάσταση μετασχηματισμών*. Στην κατάσταση ανάλυσης γίνεται ενορχήστρωση στον πηγαίο κώδικα με σκοπό την πραγματοποίηση δυναμικής ανάλυσης του προγράμματος. Έχουν υλοποιηθεί διάφοροι αναλυτές στο MEMSCOPT, μεταξύ των οποίων και ο αναλυτής αποστάσεων επαναχρησιμοποίησης δεδομένων που χρησιμοποιείται από το MemAssist. Το εργαλείο έχει τη δυνατότητα να ενσωματώσει τα αποτελέσματα της ανάλυσης κατευθείαν μέσα στον κώδικα μέσω μιας ειδικής σημειογραφίας που λέγεται *SAL (special annotation language)*. Η SAL μπορεί να χρησιμοποιηθεί τόσο βοηθητικά κατά την ανάλυση, όσο και για λόγους τεκμηρίωσης. Οι δηλώσεις της SAL περικλείονται σε σχόλια της C (σχήμα 4.5) και ο χρήστης μπορεί να την εκμεταλλευτεί για να αλληλεπιδράσει με το εργαλείο, παρέχοντας είσοδο που αφορά την ανάλυση. Η SAL περιλαμβάνει δηλωτικές εντολές για την ονοματοδοσία των βρόχων επανάληψης, την ταυτοποίηση των *επαγωγικών μεταβλητών (induction variables)* και την αναπαράσταση διάφορων μετρικών που αφορούν τους βρόχους, όπως για παράδειγμα ο αριθμός των επαναλήψεων που πραγματοποιήθηκαν για κάθε βρόχο κατά την εκτέλεση του προγράμματος.

Για την εφαρμογή βελτιστοποιήσεων στον πηγαίο κώδικα, κάνοντας χρήση του MEMSCOPT, ακολουθείται μια επαναληπτική διαδικασία πολλαπλών βημάτων, σε κάθε ένα από τα οποία εφαρμόζεται ένας ή πολλαπλοί μετασχηματισμοί βρόχων. Οι μετασχηματισμοί που υποστηρίζονται από το MEMSCOPT περιλαμβάνουν τους: *μετατόπιση βρόχου (loop shift)*, *επέκταση βρόχου (loop extend)*, *αντιστροφή βρόχου (loop reversal)*, *συγχώνευση βρόχου (loop fusion)*, *εναλλαγή βρό-*

[1] http://www.enosys-project.eu/

```
...
/* @forloop:forloop_0 */
/* @iterator: k */
/* @#iterations: 5 */
/* @#loopweight: 1 */
for ( k=-GB; k<=GB; ++k ){
  tot+=Gauss[abs(k)];
}
...
```

Σχήμα 4.5: Δηλωτικές εντολές SAL.



Σχήμα 4.6: Το γραφικό περιβάλλον που παρέχει το MEMSCOPT για την εφαρμογή μετασχηματισμών.

χων (loop interchange), διαχωρισμό βρόχων (loop fission), κανονικοποίηση βρόχου (loop normalization), αναδιάταξη βρόχων (loop reorder), αντίστροφη αποδιακλάδωση βρόχου (loop switching), μετακίνηση βρόχου εμπρός ή πίσω (loop scope move forward or backwards). Στο πίνακα Α΄.2 του παραρτήματος Α΄ παρουσιάζονται λεπτομέρειες για αυτούς τους με-

τασχηματισμούς, ενώ στον πίνακα Α΄.1 παρέχεται αναλυτική τεκμηρίωση των παραμέτρων της διεπαφής γραμμής εντολών του MEMSCOPT. Στο σχήμα 4.6 φαίνεται το γραφικό περιβάλλον που παρέχει το MEMSCOPT όταν λειτουργεί σε κατάσταση εφαρμογής μετασχηματισμών. Περιλαμβάνει πεδία για:

1. τον ορισμό του αρχείου εισόδου και του καταλόγου στον οποίο θα τοποθετηθούν τα παραγόμενα αρχεία εξόδου.

2. την επιλογή μετασχηματισμών και την κατάλληλη παραμετροποίησή τους.

3. την εκτέλεση, αποθήκευση και ανάκτηση των βημάτων που έχουν πραγματοποιηθεί μέχρι στιγμής ή την επιστροφή σε συγκεκριμένα βήματα.

4. την επεξεργασία των μετασχηματισμών κάθε βήματος.

Όλα τα βήματα εφαρμογής μετασχηματισμών που εκτελούνται, αποθηκεύονται σε ένα αρχείο XML. Εκεί καταγράφεται το αρχικό αρχείο με τον κώδικα εισόδου και όλες οι λεπτομέρειες κάθε βήματος εφαρμογής μετασχηματισμών που πραγματοποιήθηκε.

4.4.2 Υλοποίηση του εργαλείου MEMSCOPT

Αυτή η υποενότητα αναφέρεται στη διαδικασία ανάπτυξης ορισμένων λειτουργιών του MEMSCOPT, κάνοντας χρήση της προτεινόμενης ροής εργαλείων. Αρχικά, καθορίζεται η βασική πληροφορία που απαιτείται να εντοπιστεί στον πηγαίο κώδικα. Αυτό αφορά, στη συγκεκριμένη περίπτωση, την ταυτοποίηση των βρόχων τύπου *for* του προγράμματος και των σχετικών με αυτούς πληροφοριών. Στον πίνακα 4.1 γίνεται καταγραφή των αιτημάτων που πρέπει να υλοποιηθούν, μαζί με μια περιγραφή της πληροφορίας που πρέπει να επιστρέφει το κάθε ένα και την κλάση HLQ με την οποία σχετίζεται. Για την ακρίβεια, κάθε αίτημα υλοποιείται σαν μέθοδος της κλάσης στην οποία αντιστοιχεί. Αναπτύσσονται οι κλάσεις *HLQTranslationUnit*, *HLQFunctionDef* και *HLQForLoop* και στις μεθόδους τους γίνεται χρήση της CastQL για την ανάπτυξη των απαιτούμενων ερωτημάτων, όπως φαίνεται στα σχήματα 4.7, 4.8, 4.9, 4.10, 4.11 και 4.12.

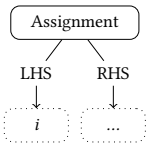
Το HLQTranslationUnit εντοπίζει το σύνολο των ορισμών συναρτήσεων του προγράμματος εισόδου. Η λειτουργία αυτή υλοποιείται

Πίνακας 4.1: Αιτήματα εντοπισμού πληροφορίας.

Αίτημα/Ερώτημα (υψηλού επιπέδου)	Κλάση HLQ
Ορισμοί συναρτήσεων	HLQTranslationUnit
Ανακτά και επιστρέφει τα αντικείμενα του AST που αναπαριστούν ορισμούς συναρτήσεων.	
Βρόχοι της συνάρτησης	HLQFunctionDefinition
Ανακτά και επιστρέφει τα αντικείμενα του AST που αναπαριστούν βρόχους επανάληψης for στο σώμα της τρέχουσας συνάρτησης.	
Εμφωλευμένοι βρόχοι	HLQForLoop
Ανακτά και επιστρέφει τους βρόχους επανάληψης for που είναι εμφωλευμένοι στο σώμα του τρέχοντος βρόχου.	
Αναφορές επαγωγικών μεταβλητών	HLQForLoop
Ανακτά και επιστρέφει τα αντικείμενα του AST που αντιστοιχούν στις αναφορές που γίνονται προς τις επαγωγικές μεταβλητές στο σώμα ενός βρόχου επανάληψης for.	
Μεταβλητή αρχικοποίησης	HLQForLoop
Ανακτά και επιστρέφει το αριστερό μέρος της εντολής εκχώρησης στην οποία γίνεται η αρχικοποίηση μιας επαγωγικής μεταβλητής.	
Έκφραση αρχικοποίησης	HLQForLoop
Ανακτά και επιστρέφει μια έκφραση που εκτελείται κατα την αρχικοποίηση του βρόχου επανάληψης for.	
Τελική έκφραση	HLQForLoop
Ανακτά και επιστρέφει μια έκφραση που χρησιμοποιείται για την ενημέρωση μιας επαγωγικής μεταβλητής σε κάθε επανάληψη του βρόχου.	
Τύπος τελικής έκφρασης	HLQForLoop
Ανακτά και επιστρέφει τον τελεστή που χρησιμοποιείται για την ενημέρωση μιας επαγωγικής μεταβλητής σε κάθε επανάληψη του βρόχου.	
Έκφραση συνθήκης	HLQForLoop
Ανακτά και επιστρέφει μια έκφραση συνθήκης που χρησιμοποιείται για τον τερματισμό εκτέλεσης του βρόχου.	
Εντολές σώματος βρόχου	HLQForLoop
Ανακτά και επιστρέφει τις εντολές που βρίσκονται στο σώμα ενός βρόχου.	
Επαγωγική μεταβλητή	HLQForLoop
Ανακτά και επιστρέφει την κύρια επαγωγική μεταβλητή ενός βρόχου.	

```
1 fundefsPat = LLQ::DEFINE(NODE)
2   .nodetype(FunctionDef)
3   .filter(EXHAUSTIVE)
4 .END();
5 fundefsSet = LLQ::APPLY()
6   .query(fundefsPat)
7   .on(mASTRoot)
8 .END();
```

(α)



(β)

Σχήμα 4.7: α) Κώδικας CastQL για την ανίχνευση των ορισμών συναρτήσεων. β) Πρότυπο AST υποδέντρο για την ταυτοποίηση της αρχικοποίησης της επαγωγικής μεταβλητής.

```
1 pat1 = LLQ::DEFINE(CONTEXT)
2   .contexttype(FunctionDef_Body)
3   .filter(DEEPFIRSTFOUND)
4 .END();
5 pat2 = LLQ::DEFINE(NODE)
6   .nodetype(ForLoop)
7   .filter(DEPTHSPECIFIC_EXHAUSTIVE)
8     .depth(1)
9 .END();
```

```
10 patFunStat = LLQ::DEFINE(COMPLEX)
11   .query(pat1)
12   .query(pat2)
13 .END();
14 seto1 = LLQ::APPLY()
15   .query(patFunStat)
16   .on(mFunDef)
17 .END();
18
```

Σχήμα 4.8: Κώδικας CastQL για την ανίχνευση των εξωτερικών βρόχων (επιπέδου 0) στο σώμα μιας συνάρτησης.

σε χαμηλότερο επίπεδο από τον κώδικα CastQL του σχήματος 4.7α. Δημιουργείται αρχικά ένα LLQ (γραμμές 1-4) που βρίσκει όλους τους κόμβους του AST που αναπαριστούν ορισμούς συναρτήσεων. Εφαρμόζεται επίσης το φίλτρο *exhaustive*, που αναγκάζει το LLQ να επιστρέψει άπαντες τους ορισμούς συναρτήσεων του προγράμματος. Το ερώτημα εφαρμόζεται έπειτα στη ρίζα του AST (γραμμές 5-8). Το HLQTranslationUnit δημιουργεί από ένα αντικείμενο HLQFunctionDef για κάθε ορισμό συνάρτησης που ανακαλύπτει στο πρόγραμμα.

Ο σκοπός του HLQFunctionDef είναι να εντοπίσει τους βρόχους επανάληψης που περιλαμβάνονται σε μια συγκεκριμένη συνάρτηση. Αυτό υλοποιείται με τον κώδικα CastQL του σχήματος 4.8. Δύο ξεχωριστά LLQs (γραμμές 1-4 και 5-9) συνδυάζονται σε ένα LLQ τύπου complex (γραμμές 10-13). Το πρώτο από αυτά τα LLQ δρομολογεί την αναζήτηση στο σώμα μιας συνάρτησης, ενώ το δεύτερο εντοπίζει τους βρόχους επανάληψης που περιέχονται εκεί. Στο δεύτερο ερώτημα εφαρμόζεται το φίλτρο *depth specific exhaustive*, ώστε να αποκλειστούν οι βρόχοι που είναι εμφωλευμένοι σε άλλους (με βά-

```
patIdent = LLQ::DEFINE(NODE)
  .nodetype(Identifier)
.END();
patInitId = LLQ::DEFINE(CONTEXT)
  .contexttype(ForLoop_Init)
.END();
patFinId = LLQ::DEFINE(CONTEXT)
  .contexttype(ForLoop_Finalization)
.END();
patAdjId = LLQ::DEFINE(CONTEXT)
  .contexttype(ForLoop_Adjustment)
.END();
patBodyId = LLQ::DEFINE(CONTEXT)
  .contexttype(ForLoop_Body)
.END();
patForInitIde = LLQ::DEFINE(COMPLEX)
  .query(patinitid)
  .query(patident)
.END();
patForFinIde = LLQ::DEFINE(COMPLEX)
  .query(patfinid)
  .query(patident)
.END();
patForAdjIde = LLQ::DEFINE(COMPLEX)
  .query(patadjid)
  .query(patident)
.END();
patForBodyIde = LLQ::DEFINE(COMPLEX)
  .query(patfbodyid)
  .query(patident)
.END();
iniSet = LLQ::APPLY()
  .query(patForInitIde)
  .on(mForLoop)
.END();
finSet = LLQ::APPLY()
  .query(patForFinIde)
  .on(mForLoop)
.END();
adjSet = LLQ::APPLY()
  .query(patForAdjIde)
  .on(mForLoop)
.END();
fbodySet = LLQ::APPLY()
  .query(patForBodyIde)
  .on(mForLoop)
.END();

interPat = LLQ::DEFINE(SETOP)
  .operation(INTERSECTION)
.END();
inter1 = LLQ::APPLY()
  .query(interPat)
  .on(iniSet)
  .on(finSet)
.END();
inter2 = LLQ::APPLY()
  .query(interPat)
  .on(inter1)
  .on(adjSet)
.END();
inter3 = LLQ::APPLY()
  .query(interPat)
  .on(inter2)
  .on(fbodySet)
.END();
```

Σχήμα 4.9: Κώδικας CastQL για την ανίχνευση των επαγωγικών μεταβλητών ενός βρόχου.

θος μεγαλύτερο ή ίσο του 1) και να επιστραφούν μόνο οι εξωτερικοί (με βάθος 0). Οι βρόχοι που εντοπίζονται τοποθετούνται στο nodeset *seto1* για μετέπειτα αξιοποίηση. Για κάθε βρόχο που εντοπίζεται, δημιουργείται και από ένα αντικείμενο HLQForLoop. Τα αντικείμενα αυτά διεξάγουν με τη σειρά τους νέες αναζητήσεις για τον εντοπισμό εμφωλευμένων βρόχων.

Ένα HLQForLoop εντοπίζει αρχικά τις επαγωγικές μεταβλητές του βρόχου. Μια μεταβλητή πρέπει να εμφανίζεται και στις τέσσερις περιοχές ενός βρόχου (περιοχή αρχικής παράστασης, περιοχή παράστασης συνθήκης, περιοχή τελικής παράστασης, κυρίως σώμα) ώστε να θεωρηθεί επαγωγική. Στο σχήμα 4.9 φαίνεται ο κώδικας CastQL που υλοποιεί την αντίστοιχη αναζήτηση. Το nodeset *inter3* περιλαμβάνει τις μεταβλητές που εμφανίζονται και στις τέσσερις περιοχές. Οι εκφράσεις που ορίζουν το χώρο επανάληψης του βρόχου εντοπίζονται επίσης από το HLQForLoop. Ο κώδικας CastQL του σχήματος

```
1  loopInitPat = LLQ::DEFINE(CONTEXT)
2    .contexttype(ForLoop_Init)
3  .END();
4  assExpCase1 = Synthesis::
5  CreateASTSubTree(
6    Assignment,
7    assExpCase1 = Synthesis::
8    CreateIdent(
9      GetPrimaryIterator()
10   )
11 );
```

```
12 assignInit1 =LLQ::DEFINE(SIMILARITY)
13   .tree(assExpCase1)
14 .END();
15 assignsInit1 = LLQ::DEFINE(COMPLEX)
16   .query(loopInitPat)
17   .query(assignInit1)
18 .END();
19 initAssignments = LLQ::APPLY()
20   .query(assignsInit1)
21   .on(mForLoop)
22 .END();
```

Σχήμα 4.10: Κώδικας CastQL για την ανίχνευση της έκφρασης αρχικοποίησης μιας επαγωγικής μεταβλητής.

```
1  loopAdjReg = LLQ::DEFINE(CONTEXT)
2    .contexttype(ForLoop_Adjustment)
3  .END();
4  adjRegion = LLQ::APPLY()
5    .query(loopAdjReg)
6    .on(mForLoop)
7  .END();
8  loopAdjExp = LLQ::DEFINE(NODE)
9    .nodetype(PostfixIncreament)
10   .nodetype(PrefixIncreament)
11   .nodetype(PostfixDecreament)
12   .nodetype(PrefixDecreament)
13   .nodetype(Assignment)
14   .nodetype(AssignmentAddTo)
15   .nodetype(AssignmentSubtractFrom)
16   .filter(EXHAUSTIVE)
17 .END();
```

```
18 adjExp = LLQ::APPLY()
19   .query(loopAdjExp)
20   .on(adjRegion)
21 .END();
22 iteratorPat =LLQ::DEFINE(SIMILARITY)
23   .comparison(EQUALITY)
24   .tree(
25     Synthesis::CreateIdent(
26       *piterator
27     )
28   )
29   .filter(EXHAUSTIVE)
30 .END();
31 adjId = LLQ::APPLY()
32   .query(iteratorPat)
33   .on(adjExp)
34 .END();
```

Σχήμα 4.11: Κώδικας CastQL για την ανίχνευση της τελικής έκφρασης μιας επαγωγικής μεταβλητής.

4.10 ψάχνει στην περιοχή αρχικοποίησης (περιοχή αρχικής παράστασης) για εντολές εκχώρησης που να περιλαμβάνουν αναφορές στην επαγωγική μεταβλητή. Ένα LLQ κατευθύνει την αναζήτηση στην περιοχή αρχικοποίησης (γραμμές 1-3). Έπειτα, με τη βοήθεια του πλαισίου σύνθεσης, δημιουργείται το πρότυπο AST υποδέντρο του σχήματος 4.7β (γραμμές 4-11). Αυτό το υποδέντρο χρησιμοποιείται από το ερώτημα *assignInit1* για την ταυτοποίηση οποιασδήποτε εντολής εκχώρησης αρχικοποιεί την επαγωγική μεταβλητή (γραμμές 11-22). Επιπλέον, το HLQForLoop εντοπίζει τις εκφράσεις συνθήκης και τις τελικές εκφράσεις του βρόχου. Ο εντοπισμός των τελικών εκφράσεων φαίνεται στο σχήμα 4.11, ενώ ο εντοπισμός των εκφράσεων συνθήκης δεν παρουσιάζεται, από τη στιγμή που υλοποιείται με ανάλογο κώδικα. Στο σχήμα 4.11, η αναζήτηση κατευθύνεται προς την περιοχή

```
1  tmp1 = LLQ::DEFINE(CONTEXT)
2    .contexttype(ForLoop_Body)
3  .END();
4  tmp2 = LLQ::DEFINE(NODE)
5    .nodetype(ForLoop_Body)
6    .filter(DEPTHSPECIFIC_EXHAUSTIVE)
7      .depth(1)
8  .END();
```

```
9  pat4Nested4 = LLQ::DEFINE(COMPLEX)
10   .query(tmp1)
11   .query(tmp2)
12 .END();
13 mNestedForLoops = LLQ::APPLY()
14   .query(pat4Nested4)
15   .on(mForLoop)
16 .END();
```

Σχήμα 4.12: Κώδικας CastQL για την ανίχνευση εμφωλευμένων βρόχων επανάληψης.

του βρόχου που περιλαμβάνει τις τελικές εκφράσεις (γραμμές 1-7). Εκεί επιχειρεί να ανακαλύψει εκφράσεις με τους τελεστές *i++*, *i–*, *++i*, *–i*, *i=...*, *i+=...* και *i-=...* (γραμμές 8-21). Έπειτα, επιλέγει την έκφραση στην οποία εμφανίζεται η επαγωγική μεταβλητή και αγνοεί τις υπόλοιπες (γραμμές 22-34). Στο τελευταίο βήμα, το HLQForLoop εντοπίζει τους εμφωλευμένους βρόχους που βρίσκονται στο κυρίως σώμα του εξεταζόμενου βρόχου, χρησιμοποιώντας τον κώδικα CastQL του σχήματος 4.12. Το ερώτημα *tmp1* κατευθύνει την αναζήτηση στην περιοχή του κυρίως σώματος του βρόχου και το *tmp2* χρησιμοποιεί το φίλτρο *depth specific exhaustive* για να ανακαλύψει όλους τους βρόχους σε επίπεδο 1. Οι ανακτηθέντες βρόχοι τοποθετούνται στο nodeset *mNestedForLoops*.

Κεφάλαιο 5

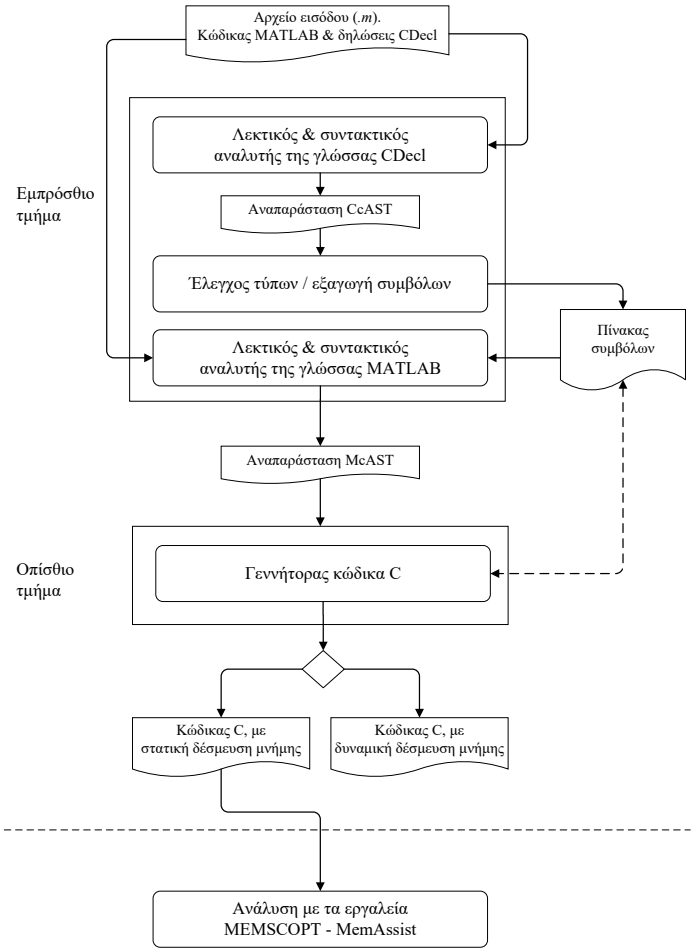
Μεταγλώττιση MATLAB/Scilab-σε-C

Ο μεταγλωττιστής MAFE (MATLAB Front End) είναι ένα εργαλείο που μετατρέπει πηγαίο κώδικα MATLAB ή/και Scilab σε αντίστοιχο πηγαίο κώδικα σε γλώσσα C. Η βασική δομή του εμπρόσθιου τμήματος του μεταγλωττιστή κατασκευάστηκε με τον γεννήτορα εμπρόσθιων τμημάτων FEgen και αποτελείται από δύο μέρη:

1. Έναν λεκτικό και συντακτικό αναλυτή ο οποίος αναγνωρίζει τη σύνταξη της γλώσσας του MATLAB (ή/και του Scilab).

2. Έναν λεκτικό και συντακτικό αναλυτή ο οποίος αναγνωρίζει το συντακτικό της γλώσσας ειδικού σκοπού CDecl. Η γλώσσα αυτή παρέχει δηλωτικές εντολές για τον προσδιορισμό του σχήματος και του τύπου κάθε πίνακα και κάθε συνάρτησης που χρησιμοποιείται στον κώδικα MATLAB. Η CDecl πρακτικά γεφυρώνει την απόσταση μεταξύ του συστήματος τύπων της MATLAB (με δυναμικό έλεγχο τύπων) και του συστήματος τύπων της C, που πραγματοποιεί στατικό έλεγχο τύπων.

Το μέρος του εμπρόσθιου τμήματος που διαβάζει τον κώδικα MATLAB παράγει ένα αφηρημένο συντακτικό δέντρο τύπου cAST, που αναφέρεται ως McAST (MATLAB cAST). Αντίστοιχα, το μέρος του εμπρόσθιου τμήματος που διαβάζει τις δηλωτικές εντολές CDecl παράγει ένα αφηρημένο συντακτικό δέντρο τύπου cAST, που αναφέρε-



Σχήμα 5.1: Ροή λειτουργίας του μεταγλωττιστή MAFE.

ται ως CcAST (CDecl cAST). Το δέντρο CcAST περνάει από ένα στάδιο ελέγχου τύπων, όπως φαίνεται στο σχήμα 5.1, στο οποίο συμπε-

```
...
int A[2][2];        // Δήλωση CDecl
...

//%%

...
A = [5,6 ; 6,2]     % Κώδικας MATLAB
...
```

Σχήμα 5.2: Παράδειγμα αρχείου εισόδου του MAFE.

ραίνονται οι τύποι των συμβόλων του κώδικα MATLAB. Ο πίνακας συμβόλων που προκύπτει από αυτή τη διαδικασία μαζί με το δέντρο McAST είναι οι δύο δομές που απαρτίζουν την εσωτερική αναπαράσταση που χρησιμοποιεί το MAFE. Οι δομές αυτές αξιοποιούνται από το οπίσθιο μέρος του μεταγλωττιστή για να παράξει κώδικα C. Υπάρχει η δυνατότητα παραγωγής κώδικα C που να κάνει στατική ή δυναμική δέσμευση μνήμης για τους πίνακες του προγράμματος. Ο κώδικας με στατική δέσμευση μνήμης μπορεί να τροφοδοτηθεί στο MEMSCOPT με σκοπό την δυναμική ανάλυση του στο πλαίσιο της ροής λειτουργίας του MemAssist. Η παραγωγή κώδικα που κάνει δυναμική δέσμευση μνήμης είναι κατάλληλη για εφαρμογές MATLAB στις οποίες αλλάζουν συνεχώς τα μεγέθη των πινάκων. Ο κώδικας αυτός δεν αξιοποιείται στο πλαίσιο λειτουργίας του MemAssist, παρέχει όμως κάποια οφέλη κατά την εκτέλεση που περιγράφονται στην ενότητα 5.4. Το σχήμα 5.1 δίνει μια γενική εικόνα τις αρχιτεκτονικής του μεταγλωττιστή MAFE.

Το εμπρόσθιο τμήμα του MAFE διαβάζει ένα αρχείο MATLAB *.m* (ή *.sce* για κώδικα Scilab), το οποίο εσωτερικά είναι οργανωμένο σε δύο διακριτές περιοχές, οι οποίες χωρίζονται από μια γραμμή που περιλαμβάνει τους χαρακτήρες *//%%* (σχήμα 5.2). Η πρώτη περιοχή περιλαμβάνει δηλωτικές εντολές CDecl που αναθέτουν τύπους στις μεταβλητές του MATLAB, ενώ η δεύτερη περιοχή είναι διαθέσιμη αποκλειστικά για την τοποθέτηση του ίδιου του κώδικα εισόδου MATLAB. Η έξοδος του εμπρόσθιου τμήματος είναι το δέντρο McAST και ο πίνακας συμβόλων που το συνοδεύει.

Το οπίσθιο τμήμα του MAFE πραγματοποιεί σειριακοποίηση της αναπαράστασης McAST, μετατρέποντας την σε αντίστοιχο κώδικα C. Ο παραγόμενος κώδικας C οργανώνεται σε πολλαπλά αρχεία επι-

κεφαλίδας (*.h*) και υλοποίησης (*.c*). Τα αρχεία αυτά χωρίζονται σε τρείς λογικές κατηγορίες:

1. Η πρώτη περιλαμβάνει τα αρχεία που περιέχουν κώδικα προερχόμενο από την απευθείας μεταγλώττιση της εισόδου.
2. Η δεύτερη περιλαμβάνει αρχεία που περιέχουν δηλώσεις σε δομές δεδομένων που αντιστοιχούν στους τύπους δεδομένων της MATLAB.
3. Η τρίτη περιλαμβάνει αρχεία που περιέχουν τα πρότυπα και τις υλοποιήσεις σε C των εσωτερικών (built-in) συναρτήσεων της MATLAB.

5.1 Εμπρόσθιο τμήμα του μεταγλωττιστή

5.1.1 Ανάγνωση των δηλώσεων CDecl

Στην γλώσσα C πραγματοποιείται στατικός έλεγχος τύπων, σε αντίθεση με τον δυναμικό έλεγχο της MATLAB. Αυτό συνεπάγεται ότι ο μεταγλωττιστής θα πρέπει να έχει γνώση σχετικά με τους τύπους των μεταβλητών ώστε να τοποθετήσει τις κατάλληλες δηλώσεις στον παραγόμενο κώδικα C. Ο MAFE αναθέτει από έναν τύπο σε κάθε μεταβλητή του κώδικα εισόδου MATLAB κάνοντας χρήση της γλώσσας ειδικού σκοπού CDecl, η οποία αναπτύχθηκε αποκλειστικά για την επίλυση αυτού του προβλήματος. Η CDecl είναι εξωτερική (external) γλώσσα ειδικού σκοπού [69], οπότε ένας λεκτικός και συντακτικός αναλυτής έπρεπε να υλοποιηθεί για την αναγνώριση του συντακτικού της. Η ανάγνωση των δηλώσεων της CDecl αποτελεί το πρώτο βήμα που εκτελεί εσωτερικά ο MAFE.

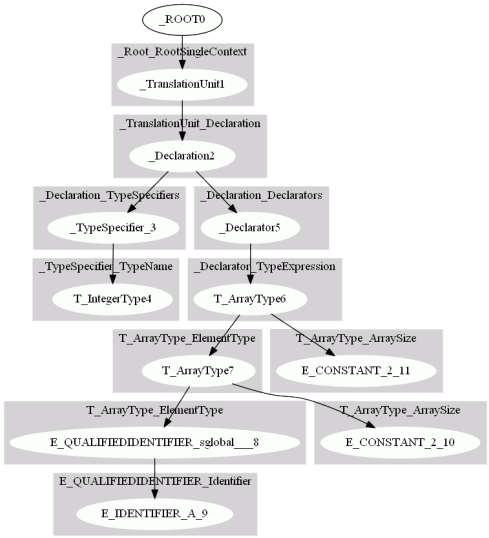
Η CDecl είναι στην ουσία ένα υποσύνολο του στάνταρ C89 της γλώσσας C. Το υποσύνολο αυτό περιλαμβάνει μόνο τις εντολές δηλώσεων της C μαζί με ορισμένα επιπλέον χαρακτηριστικά, τα οποία είναι αναγκαία για την προσαρμογή τους στα πρότυπα της MATLAB. Αυτές οι εντολές περιλαμβάνουν δηλώσεις πινάκων, συμβολοσειρών και συναρτήσεων. Οι βαθμωτές μεταβλητές θεωρούνται πίνακες με ένα μόνο στοιχείο και διαστάσεις 1x1. Σε όλες τις δηλώσεις πινάκων πρέπει να παρέχεται το μέγεθος τους, με εξαίρεση αυτούς που περιλαμβάνονται στο πρότυπο μιας συνάρτησης. Οι πίνακες που δηλώνονται σαν παράμετροι σε μια συνάρτηση πρέπει να φέρουν συγκεκριμένο τύπο στοιχείων, ενώ το μέγεθος τους δεν είναι αναγκαίο

Πίνακας 5.1: Δηλώσεις της γλώσσας CDecl.

Τύπος δήλωσης	Παράδειγμα
Πίνακας καθολικής εμβέλειας.	*int A[10][10];*
Πίνακας τοπικής εμβέλειας (στη συνάρτηση foo).	*int foo::B[10][10];*
Πρότυπο της MATLAB συνάρτησης k = foo(a,b).	*void foo(in int **a, in int **b, out int **k);*

να δηλωθεί. Επιπλέον, οι συναρτήσεις της MATLAB μπορούν να επιστρέφουν πολλαπλές τιμές σε αντίθεση με αυτών της C, οι οποίες επιστρέφουν μόνο μια τιμή. Για αυτό το λόγο εισάγονται τα χαρακτηριστικά (specifiers) *in* και *out* για τον προσδιορισμό των παραμέτρων μιας συνάρτησης ως εισόδου ή εξόδου. Οι μεταβλητές που χρειάζεται να έχουν τοπική εμβέλεια σε μια συνάρτηση πρέπει να δηλωθούν με το όνομα της συνάρτησης ως πρόθεμα ακολουθούμενο από τον χαρακτήρα *::* και έπειτα το όνομα της μεταβλητής. Στον πίνακα 5.1 παρουσιάζονται τρία παραδείγματα δηλώσεων CDecl. Στο πρώτο δηλώνεται ένας πίνακας καθολικής εμβέλειας, τύπου *int*, δύο διαστάσεων και μεγέθους 10x10. Στο δεύτερο δηλώνεται ένας όμοιος πίνακας με τοπική εμβέλεια όμως στη συνάρτηση *foo*. Στο τρίτο παράδειγμα δηλώνεται μια συνάρτηση με όνομα *foo*, η οποία παίρνει δύο παραμέτρους εισόδου και μια εξόδου. Η δήλωση αυτή αντιστοιχεί στην MATLAB συνάρτηση *k = foo(a,b)*.

Μια αναπαράσταση υπό τη μορφή συμπαγούς συντακτικού δέντρου δημιουργείται πριν από το σχηματισμό της αναπαράστασης CcAST. Κάθε τερματικό και μη-τερματικό στοιχείο της γραμματικής της CDecl δημιουργεί συντακτικά αντικείμενα/κόμβους σε αυτό το δέντρο. Γίνεται χρήση LALR συντακτικού αναλυτή για την ανάγνωση της εισόδου, οπότε η σειρά επίσκεψης στα γραμματικά στοιχεία είναι από κάτω προς τα επάνω. Μετά από τη δημιουργία του συμπαγούς συντακτικού δέντρου, πραγματοποιείται διάσχιση του με σκοπό την παραγωγή του αφηρημένου συντακτικού δέντρου CcAST. Στα σχήματα 5.2 και 5.3 παρουσιάζεται ένα απλό παράδειγμα μιας δήλωσης CDecl μαζί με το αντίστοιχο δέντρο CcAST. Όντας δέντρο τύπου cAST, ένα CcAST αποτελείται από *κόμβους* και *ομάδες συμφραζομένων* για κάθε τύπο κόμβου. Στο σχήμα 5.3 οι κόμβοι φαίνονται με λευκό χρώμα και οι ομάδες με γκρίζο χρώμα. Για παράδειγμα, ο



Σχήμα 5.3: Παράδειγμα αφηρημένου συντακτικού δέντρου CcAST για τον CDecl κώδικα του σχήματος 5.2.

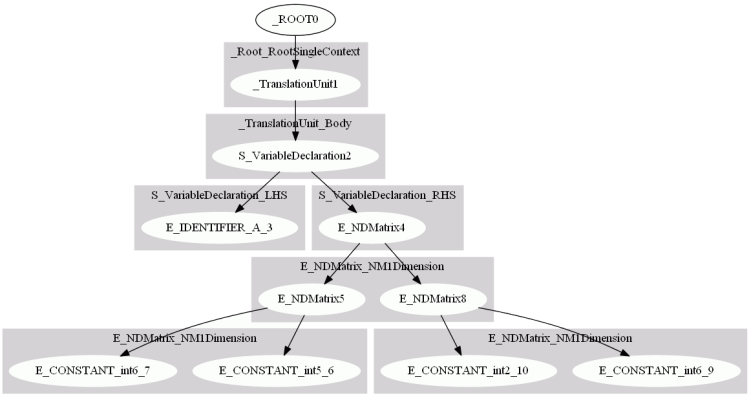
κόμβος *Declaration2*, που είναι τύπου *Declaration*, έχει τις ομάδες συμφραζομένων *_Declaration_Typespecifiers* και *_Declaration_Declarators*, οι οποίες συνδέονται με τους κόμβους *_TypeSpecifier_3* και *_Declarator_5* αντίστοιχα.

5.1.2 Ανάγνωση της γλώσσας MATLAB

Ο μεταγλωττιστής MAFE υποστηρίζει την ανάγνωση πηγαίου κώδικα MATLAB και Scilab. Ο πίνακας 5.2 παρουσιάζει αναλυτικά τα στοιχεία αυτών των γλωσσών που υποστηρίζονται. Όπως και στην περίπτωση της CDecl, δημιουργείται ένα συμπαγές συντακτικό δέντρο και για τον κώδικα MATLAB. Το δέντρο McAST δημιουργείται, επίσης, με παρόμοιο τρόπο με το CcAST. Τα σχήματα 5.2 και 5.4 δείχνουν ένα απλό παράδειγμα της δομής του δέντρου McAST.

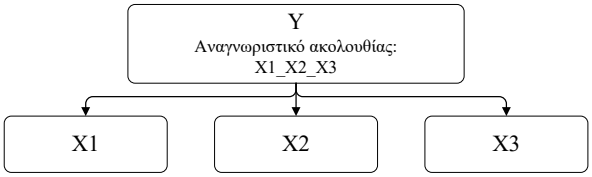
Πίνακας 5.2: Χαρακτηριστικά της γλώσσας MATLAB που υποστηρίζει ο μεταγλωττιστής MAFE.

	Υποστηριζόμενα στοιχεία
Εκφράσεις	Πρόσθεση, αφαίρεση, μοναδιαίο +/-, ομαδοποίηση με παρενθέσεις, διαίρεση βαθμωτών, πολλαπλασιασμός βαθμωτών και πινάκων, αντιμετάθεση πινάκων, λογικές πράξεις, σχεσιακές πράξεις, πράξεις με τον τελεστή εύρους (:), indexing πινάκων, κλήσεις εσωτερικών συναρτήσεων, κλήσεις εξατομικευμένων συναρτήσεων των χρηστών.
Εντολές ελέγχου ροής	if, while, for, break, return.
Τύποι δεδομένων	Βαθμωτά, πίνακες πολλαπλών διαστάσεων, συμβολοσειρές.
Συναρτήσεις	Εσωτερικές συναρτήσεις (min, max, zeros, ones, size, round, floor, sin, cos, atan, sqrt, log, exp, abs, mod, sum, uint32, double, det, transpose, inv), ειδικές συναρτήσεις εργαλειοθηκών (ReadImage, WriteImage, MaskFilter), εξατομικευμένες συναρτήσεις των χρηστών.



Σχήμα 5.4: Παράδειγμα αφηρημένου συντακτικού δέντρου McAST για τον MATLAB κώδικα του σχήματος 5.2.

```
...
Y : X1 X2 X3   { $$ = new Y(X1_X2_X3) }
  | X4 X5 X6   { $$ = new Y(X4_X5_X6) }
  ;
...
```



Σχήμα 5.5: Παράδειγμα κανόνα BNF γραμματικής και παραγόμενου συμπαγούς συντακτικού δέντρου.

5.1.3 Συμπαγές συντακτικό δέντρο

Το συμπαγές συντακτικό δέντρο κάθε ενός από τα δύο μέρη του εμπρόσθιου τμήματος δημιουργείται ενώ γίνεται ανάγνωση της εισόδου μέσω ενός LALR συντακτικού αναλυτή, σύμφωνα με την εκάστοτε γραμματική BNF. Όταν πραγματοποιείται επίσκεψη σε έναν κανόνα της γραμματικής BNF, δημιουργείται ένα κατάλληλο αντικείμενο, το οποίο αντιστοιχεί στο μη-τερματικό στοιχείο του κανόνα. Το αντικείμενο αυτό παίρνει ορισμένες παραμέτρους που προσδιορίζουν ποιος κανόνας είναι υπαίτιος για τη δημιουργία του. Έστω, για παράδειγμα, ο κανόνας *Y* (σχήμα 5.5) που απαρτίζεται από τις ακολουθίες συμβόλων *X1 X2 X3* και *X4 X5 X6*. Μετά από κάθε ακολουθία τοποθετείται κώδικας που δημιουργεί ένα αντικείμενο/κόμβο του συμπαγούς συντακτικού δέντρου με ένα αναγνωριστικό της συγκεκριμένης ακολουθίας ως παράμετρο. Υποθέτοντας ότι ένα αντικείμενο *Y* δημιουργείται από τον πρώτο κανόνα, οι ακμές που φαίνονται στο σχήμα 5.5 θα προστεθούν στο συμπαγές συντακτικό δέντρο.

5.1.4 Αφηρημένο συντακτικό δέντρο

Μια αναπαράσταση τύπου cAST κατασκευάζεται από το συμπαγές συντακτικό δέντρο, αλλά είναι τελείως ανεξάρτητη από αυτό. Κάθε συντακτικό αντικείμενο του συμπαγούς συντακτικού δέντρου και του cAST αναπαρίσταται από μια ξεχωριστή κλάση. Περισσό-

```
X::ParseTreeToASTGenerationPass(current, parent, child){
     // NO PREORDER ACTION
     for(all successors of current){
          current->successors[i]->
              ParseTreeToASTGenerationPass(p,parent,child);
     }
}
```

Σχήμα 5.6: Συνάρτηση *ParseTreeToASTGenerationPass* που δεν δημιουργεί κόμβους στο AST.

```
X::ParseTreeToASTGenerationPass(current, parent, child){
     parent_old = parent;
     new_object = new CS_AST_Y();
     parent->InstallContextElement(new_object);
     parent = new_object;
     for(all successors of current){
          current ->successors[i]->
              ParseTreeToASTGenerationPass(p,parent,child);
     }
     parent = parent_old;
}
```

Σχήμα 5.7: Συνάρτηση *ParseTreeToASTGenerationPass* που δημιουργεί κόμβους στο AST.

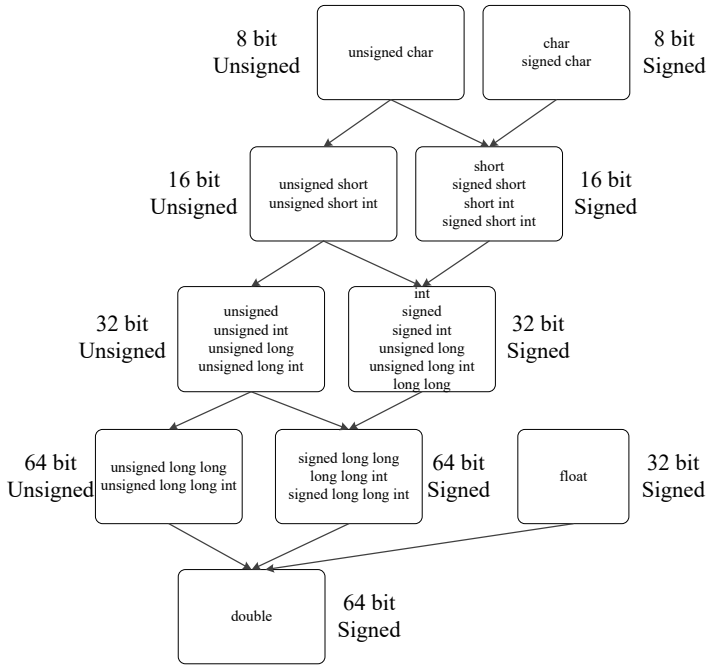
τερες λεπτομέρειες για τα δέντρα τύπου cAST παρέχονται στην ενότητα 4.2. Όλες οι κλάσεις που αναπαριστούν συντακτικά αντικείμενα του συμπαγούς συντακτικού δέντρου υλοποιούν την μέθοδο *ParseTreeToASTGenerationPass*. Αυτό είναι το μέρος στο οποίο δημιουργείται το δέντρο cAST. Υπάρχουν δύο εκδόσεις της *ParseTreeToASTGenerationPass* και το ποια από αυτές θα υλοποιηθεί σε κάθε κλάση εξαρτάται από το συντακτικό αντικείμενο που αναπαριστά. Η έκδοση του σχήματος 5.6 παρακάμπτει τη δημιουργία ενός cAST αντικειμένου και καλεί απλά την *ParseTreeToASTGenerationPass* των παιδιών του τρέχοντος κόμβου. Η έκδοση του σχήματος 5.7 δημιουργεί το κατάλληλο cAST αντικείμενο και καλεί την *ParseTreeToASTGenerationPass* των παιδιών του τρέχοντος κόμβου. Όπως είναι εμφανές, πραγματοποιείται προδιατεταγμένη διάσχιση του συμπαγούς συντακτικού δέντρου για τη δημιουργία του δέντρου cAST.

5.2 Έλεγχος τύπων των μεταβλητών

Ο μεταγλωττιστής MAFE πραγματοποιεί έλεγχο τύπων κατά τη διάρκεια δύο εκ των σταδίων του: (1) Όταν γίνεται η ανάγνωση των δηλώσεων CDecl και (2) κατά την παραγωγή κώδικα C από το οπίσθιο τμήμα.

Με τη γλώσσα ειδικού σκοπού CDecl δηλώνονται οι μεταβλητές του MATLAB προγράμματος. Για την τοποθέτηση των μεταβλητών αυτών στον πίνακα συμβόλων πραγματοποιείται μια διαδικασία μετατροπής του CcAST σε εγγραφές που είναι κατάλληλες για εισαγωγή στον πίνακα συμβόλων. Αυτό γίνεται με την εφαρμογή μιας ειδικής διάσχισης του CcAST (*Έλεγχος τύπων/εξαγωγή συμβόλων* στο σχήμα 5.1). Ο πίνακας συμβόλων περιέχει πληροφορία σχετικά με τους τύπους των μεταβλητών του χρήστη αλλά και αυτών που παράγονται αυτόματα. Είναι οργανωμένος σύμφωνα με την εμβέλεια των μεταβλητών, ενώ υπάρχουν δύο τύποι εμβελειών: Η καθολική και η τοπική εμβέλεια. Μια μεταβλητή που ορίζεται τοπικά σε μια συνάρτηση έχει τοπική εμβέλεια και είναι ορατή μόνο μέσα στη συνάρτηση. Μια καθολική μεταβλητή είναι ορατή σε όλες τις συναρτήσεις εκτός από αυτές στις οποίες έχει οριστεί εντός τους μια μεταβλητή με το ίδιο όνομα. Ακόμα, κάθε μεταβλητή μπορεί να οριστεί μόνο μια φορά για κάθε εμβέλεια. Ο πίνακας συμβόλων (1) δημιουργείται αρχικά κατά την ανάγνωση των δηλώσεων CDecl και έπειτα (2) προσπελαύνεται και επεκτείνεται από το οπίσθιο τμήμα του μεταγλωττιστή που είναι υπεύθυνο για την παραγωγή κώδικα C. Πρωτού γίνει προσπέλαση του πίνακα συμβόλων από το οπίσθιο τμήμα θα πρέπει ήδη να έχουν εντοπισθεί για όλες τις μεταβλητές που είναι παράλληλα και πίνακες: (1) ο τύπος των στοιχείων τους και (2) το μέγεθος κάθε διάστασης τους.

Κατά την παραγωγή κώδικα C ενδέχεται να υπάρχουν εκχωρήσεις τιμών από εκφράσεις όπου οι τελεστέοι έχουν διαφορετικούς τύπους μεταξύ τους. Σε αυτή την περίπτωση πρέπει με κάποιο τρόπο να συναχθεί ο σωστός τύπος της μεταβλητής που βρίσκεται στο αριστερό μέρος της εκχώρησης. Στον κώδικα *C=A+B*, αν τα *A* και *B* είναι ίδιου τύπου, το *C* θα έχει και αυτό τον ίδιο τύπο. Αν, όμως, το *A* είναι ένας 16 bit μη προσημασμένος ακέραιος (unsigned short) και το *B* είναι ένας 8 bit προσημασμένος ακέραιος (char), τότε ο τύπος του *C* θα πρέπει να συναχθεί δυναμικά με κάποιο τρόπο. Για την επίλυση αυτού του προβλήματος γίνεται χρήση ενός αλγόριθμου που δέχεται δύο τύπους δεδομένων σαν είσοδο και επιλέγει τον λιγότερο δαπα-



Σχήμα 5.8: Αλγόριθμος επιλογής τύπου δεδομένων.

νηρό από άποψη χώρου, οι τιμές του οποίου να μπορούν να εκφράσουν και τους δύο τύπους εισόδου. Ο αλγόριθμος αυτός ουσιαστικά αναζητά τον πρώτο κοινό απόγονο των δύο δοθέντων τύπων στον γράφο του σχήματος 5.8. Το μοναδικό μειονέκτημα της συγκεκριμένης μεθόδου είναι ότι αν το αποτέλεσμα είναι τύπου *double* ενδέχεται να υπάρξει απώλεια πληροφορίας, καθώς ο τύπος *double* δεν καλύπτει όλες τις πιθανές τιμές των τύπων ακεραίων.

5.3 Παραγωγή κώδικα C

Το οπίσθιο τμήμα του μεταγλωττιστή είναι υπεύθυνο για την μετατροπή της McAST αναπαράστασης σε αντίστοιχο κώδικα C. Ο παραγόμενος κώδικας είναι οργανωμένος στα παρακάτω αρχεία:

- ***M2C.c***. Περιέχει τον κώδικα C που προέρχεται από την απευ-

θείας μεταγλώττιση της εισόδου.

- ***MTypedef.h***. Περιέχει δηλώσεις δομών που αφορούν τις υλοποιήσεις δομών δεδομένων της MATLAB στη C.
- ***MTypedef.c***. Περιέχει τις υλοποιήσεις του αρχείου Mtypedef.h.
- ***MInternalFunctions.h***. Περιέχει τα πρότυπα των εσωτερικών (built-in) συναρτήσεων της MATLAB όπως υλοποιούνται στη C. Οι υποστηριζόμενες συναρτήσεις φαίνονται στον πίνακα 5.2.
- ***MInternalFunctions.c***. Περιέχει τις υλοποιήσεις των συναρτήσεων που δηλώνονται στο αρχείο MInternalFunctions.h.

5.3.1 Δυναμική δέσμευση μνήμης

Οι δομές δεδομένων που ορίζονται αν παραχθεί κώδικας που κάνει στατική δέσμευση μνήμης είναι απλοί στατικοί πίνακες της C. Όμως, στον κώδικα που κάνει δυναμική δέσμευση μνήμης απαιτείται ο ορισμός ειδικών δομών για την αναπαράσταση των πινάκων της MATLAB στην C. Ο ορισμός μιας τέτοιας δομής, όπως παράγεται στο αρχείο MTypedef.h, παρουσιάζεται στο σχήμα 5.9. Η δομή αυτή αντιστοιχεί σε έναν MATLAB πίνακα δυο διαστάσεων με 8 bit ακέραια στοιχεία (int8). Άλλες δομές ορίζονται για διάφορους τύπους δεδομένων, όπως: 8 bit μη προσημασμένος ακέραιος, 8 bit προσημασμένος ακέραιος, 16 bit μη προσημασμένος ακέραιος, 16 bit προσημασμένος ακέραιος, 32 bit μη προσημασμένος ακέραιος, 32 bit προσημασμένος ακέραιος, 64 bit μη προσημασμένος ακέραιος, 64 bit προσημασμένος

```
typedef struct array_int8_2D {
   int dnumdims;
   int snumdims;
   int numelems;
   int snumelems;
   int *ssize;
   int *dsize;
   char **rdata;
} ARRAY_INT8_2D;
```

Σχήμα 5.9: Παραγόμενη δομή αναπαράστασης ενός πίνακα MATLAB.

```
ARRAY_INT8_2D *AllocateArray_ARRAY_INT8_2D(int N0, int N1){
ARRAY_INT8_2D *x;
int i0;
int i,offset;
   x = (ARRAY_INT8_2D *)malloc(sizeof(ARRAY_INT8_2D));
   x->dnumdims=2;
   x->snumdims=2;
   x->rdata=(char **)malloc(N0* sizeof(char *));
   x->rdata[0]=(char*)malloc(N0*N1* sizeof(char));
   offset=0;
   for ( i0 = 0; i0 < N0; i0++,offset+=N1){
      x->rdata[i0]=x->rdata[0]+ offset;
   }
   x->ssize = (int *)malloc( sizeof(int)*2 );
   x->dsize = (int *)malloc( sizeof(int)*2 );
   x->ssize[0] =N0;
   x->dsize[0] =1;
   x->ssize[1] =N1;
   x->dsize[1] =1;
   x->numelems=1;
   x->snumelems=1;
   for ( i =0; i< 2; i++){
      x->snumelems*=x->ssize[i];
      x->numelems*=x->dsize[i];
   }
return x;
}
```

Σχήμα 5.10: Παράδειγμα παραγόμενης συνάρτησης *AllocateArray*.

ακέραιος, 32 bit αριθμός κινητής υποδιαστολής (float) και 64 bit αριθμός κινητής υποδιαστολής (double). Μια τέτοια δομή περιλαμβάνει τα ακόλουθα μέλη:

- ***rdata***. Κρατάει τα δεδομένα του πίνακα.
- ***snumdims***. Ο μέγιστος στατικός αριθμός διαστάσεων του πίνακα όπως έχει οριστεί στην αντίστοιχη δήλωση CDecl.
- ***dnumdims***. Ο δυναμικός αριθμός διαστάσεων του πίνακα. Αρχικά έχει ίδια τιμή με το snumdims αλλά μπορεί να αλλάξει κατά την εκτέλεση.
- ***ssize***. Το μέγιστο στατικό μέγεθος κάθε διάστασης του πίνακα, όπως ορίζεται στην αντίστοιχη δήλωση CDecl.

- ***dsize***. Το δυναμικό μέγεθος κάθε διάστασης του πίνακα. Αρχικά έχει τιμές ίδιες με το ssize αλλά μπορεί να αλλάξει κατά την εκτέλεση.

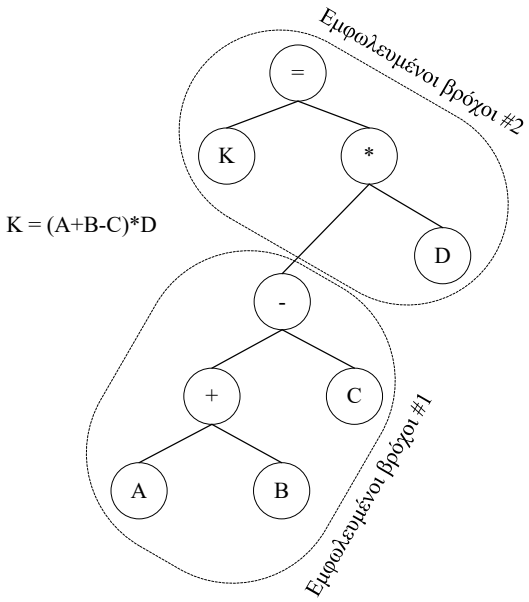
- ***snumelems***. Ο μέγιστος αριθμός στοιχείων που μπορεί να χωρέσει ο πίνακας.

- ***numelems***. Ο τρέχων αριθμός στοιχείων που περιλαμβάνει ο πίνακας.

Κάθε δομή συνοδεύεται από τον ορισμό δύο συναρτήσεων, οι οποίες δεσμεύουν και απελευθερώνουν μνήμη αντίστοιχα, για αυτές και τα μέλη τους. Για παράδειγμα, αν χρειαστεί να οριστεί μια δομή με όνομα ARRAY_INT8_2D, θα δημιουργηθούν επίσης και οι συναρτήσεις AllocateArray_ARRAY_INT8_2D και FreeArray_ARRAY_INT8_2D. Οι συναρτήσεις αυτές δηλώνονται στο αρχείο MTypedef.h και ορίζονται στο αρχείο MTypedef.c, ενώ η υλοποίηση της πρώτης φαίνεται στο σχήμα 5.10.

5.3.2 Παραγωγή εκφράσεων

Οι εκφράσεις του MATLAB που περιλαμβάνουν αναφορές σε πίνακες υλοποιούνται με βρόχους επανάληψης στην C. Για να γίνει ο διαχωρισμός μιας σύνθετης έκφρασης σε φωλιές βρόχων, γίνεται χρήση μιας δενδροειδούς δομής που λέγεται *δέντρο έκφρασης*. Εφαρμόζεται αρχικά μια ειδική διάσχιση στο McAST, η οποία δημιουργεί από ένα δέντρο έκφρασης για κάθε σύνθετη έκφραση του προγράμματος. Έπειτα, αναλύεται κάθε δέντρο ξεχωριστά και γίνεται ο διαχωρισμός σε φωλιές βρόχων επανάληψης. Η διαδικασία αυτή αφορά την εφαρμογή μιας μεταδιατεταγμένης διάσχισης, από κάτω προς τα επάνω, σε κάθε δέντρο. Κατά την διάσχιση ενός δέντρου, ανακαλύπτονται με σειρά οι τελεστές της έκφρασης. Μόλις ο αλγόριθμος φτάσει σε έναν τελεστή που δεν είναι συμβατός με τον αμέσως προηγούμενο, δημιουργείται καινούρια φωλιά βρόχων επανάληψης. Η σειρά των φωλιών στον παραγόμενο κώδικα είναι ίδια με την σειρά κατά την οποία δημιουργήθηκαν. Τα σχήματα 5.11 και 5.12 δείχνουν ένα παράδειγμα δέντρου έκφρασης και τον αντίστοιχο παραγόμενο κώδικα C για την MATLAB έκφραση *K=(A+B-C)*D*.



Σχήμα 5.11: Διαχωρισμός της έκφρασης *K=(A+B-C)*D* σε εμφωλευμένους βρόχους επανάληψης.

5.3.3 Έλεγχοι κατά την εκτέλεση

Στον παραγόμενο κώδικα C που πραγματοποιεί δυναμική δέσμευση μνήμης, τα σχήματα των μεταβλητών έχουν την δυνατότητα να αλλάζουν δυναμικά κατά την εκτέλεση. Για το λόγο αυτό, είναι αναγκαία η πραγματοποίηση ελέγχων στους τύπους των τελεστών και τους τελεστέους τους, προτού τρέξει κάθε έκφραση. Για παράδειγμα, μια πρόσθεση πινάκων απαιτεί όλοι οι τελεστέοι να έχουν ίδιες διαστάσεις, ενώ στον πολλαπλασιασμό πινάκων ο αριθμός στηλών του ενός πίνακα πρέπει να ισούται με τον αριθμό γραμμών του άλλου. Οι έλεγχοι αυτοί πραγματοποιούνται σε συναρτήσεις οι οποίες καλούνται πριν από την εκτέλεση της εκάστοτε έκφρασης.

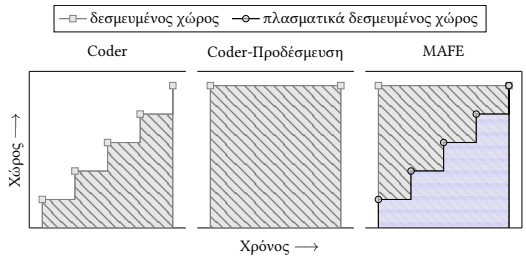
```
// Βρόχοι #1
for(int i=0; i<A.size_rows; i++){
     for(int j=0; j<A.size_cols; j++){
          temp[i][j] = A[i][j]+B[i][j]-C[i][j];
     }
}

// Βρόχοι #2
for(int i=0; i<temp.size_rows; i++){
     for(int j=0; j<D.size_cols; j++){
          sum = 0;
          for(int k=0; k<D.size_rows; k++){
               sum = temp[i][k] * D[k][j];
          }
          K[i][j] = sum;
     }
}
```

Σχήμα 5.12: Παραγόμενος κώδικας C για την MATLAB έκφραση *K=(A+B-C)*D*.

5.4 Ιδιαιτερότητες και πλεονεκτήματα του προτεινόμενου μεταγλωττιστή

Ο κώδικας C με δυναμική δέσμευση μνήμης που παράγεται από το MAFE στην πραγματικότητα ακολουθεί μια υβριδική στατική-δυναμική προσέγγιση. Οι πίνακες είναι πάντα προδεσμευμένοι, σύμφωνα με τα μεγέθη που έχουν δηλωθεί μέσω εντολών CDecl. Την ίδια στιγμή για κάθε έναν από αυτούς ορίζεται ένα *μέγιστο* και ένα *πλασματικό/τρέχον* μέγεθος. Το μέγιστο μέγεθος κάθε πίνακα είναι αυτό που δηλώθηκε στην CDecl, ενώ το πλασματικό μέγεθος είναι αρχικά μηδενικό για όλες τις διαστάσεις του. Όταν αλλάζει το μέγεθος ενός πίνακα, για παράδειγμα μέσω μιας απλής εντολής εκχώρησης, το πλασματικό μέγεθος του αλλάζει σύμφωνα με τη νέα τιμή που του εκχωρήθηκε. Το MAFE πραγματοποιεί επίσης ελέγχους κατά την εκτέλεση, σχετικούς με τα μεγέθη των πινάκων και παράγει σφάλμα στην περίπτωση που ξεπεραστεί το μέγιστο μέγεθος κάποιου από αυτούς. Αυτή η προσέγγιση έχει κάποιες διαφορές σε σχέση με τον δυναμικό κώδικα που παράγει το MATLAB Coder της MathWorks. Το Coder συνάγει όλα τα μεγέθη των πινάκων κατά την μεταγλώττιση. Δεν υποστηρίζει την επέκταση του μεγέθους ενός πίνακα όταν δεν είναι σε θέση να συνά-



Σχήμα 5.13: Η επέκταση και οι επαναδεσμεύσεις μνήμης που πραγματοποιούνται για έναν πίνακα που αυξάνεται συνεχώς το μέγεθος του.

γει το μέγιστο μέγεθός του γιατί θα μπορούσε να βρεθεί σε μια κατάσταση κατά την οποία να επαναδεσμεύεται χώρος διαρκώς, κάτι το οποίο θα χειροτέρευε δραστικά την απόδοσή του (σχήμα 5.13). Αυτό θεωρητικά θα μπορούσε να λυθεί με την προδέσμευση του πίνακα, αλλά από τη στιγμή που δεν υπάρχουν έλεγχοι κατά την εκτέλεση, το πρόβλημα θα εξακολουθούσε να υπάρχει.

Κεφάλαιο 6

Πειραματική αξιολόγηση

Σε αυτό το κεφάλαιο αξιολογούνται τα εργαλεία και οι τεχνικές που προτείνονται στην παρούσα διατριβή. Αρχικά, στην ενότητα 6.1 γίνεται χρήση του MemAssist για την βελτιστοποίηση ορισμένων εφαρμογών. Παρουσιάζονται μετρήσεις από τέσσερα διαφορετικά συστήματα, που αφορούν το χρόνο εκτέλεσης αυτών των εφαρμογών και τη βελτίωσή του. Ακόμα, στην ίδια ενότητα γίνεται χρήση του προσομοιωτή λανθανουσών μνημών Cachegrind [111] για την αξιολόγηση της βελτίωσης στην χρήση των λανθανουσών μνημών που μπορεί να επιτευχθεί με το MemAssist. Έπειτα, στην ενότητα 6.2 αξιολογείται η προσέγγιση που προτείνεται για την ανάπτυξη εργαλείων ανάλυσης και χειρισμού πηγαίου κώδικα. Η γλώσσα ειδικού σκοπού CastQL και το εργαλείο παραγωγής εμπρόσθιων τμημάτων FEgen δοκιμάζονται σε ορισμένα πρακτικά σενάρια χρήσης.

6.1 Αξιολόγηση του εργαλείου MemAssist

6.1.1 Πειραματικός σχεδιασμός

6.1.1.1 Εφαρμογές

Έξι εφαρμογές, οι οποίες κυριαρχούνται από βρόχους επανάληψης, βελτιστοποιήθηκαν με τη βοήθεια του εργαλείου MemAssist, με σκοπό την επιβεβαίωση της αποτελεσματικότητάς του. Όλες οι εφαρ-

μογές προέρχονται από τον τομέα της επεξεργασίας εικόνας και είναι κατάλληλες για την επίδειξη των δυνατοτήτων του εργαλείου, λόγω της έντονης επαναχρησιμοποίησης δεδομένων που πραγματοποιείται κατά τα διάφορα αλγοριθμικά στάδιά τους. Στον πίνακα 6.1 περιγράφονται τα χαρακτηριστικά των εφαρμογών αυτών, όσον αφορά την επεξεργασία και επαναχρησιμοποίηση που πραγματοποιούν στα δεδομένα εισόδου. Για κάθε μια παρουσιάζονται πληροφορίες για τον *αριθμό των γραμμών κώδικα (lines of code, LOC)* που την απαρτίζουν, τον *αριθμό των βρόχων επανάληψης*, τις *αναφορές μνήμης* που περιλαμβάνει, καθώς και τον αριθμό των *ζευγαριών από αναφορές μνήμης*. Να σημειωθεί εδώ, ότι λαμβάνονται υπόψη μόνο τα ζεύγη στα οποία υπάρχει επαναχρησιμοποίηση δεδομένων. Έχει γίνει χρήση πολλαπλών αρχείων εισόδου για κάθε εφαρμογή με σκοπό την επίδειξη του πώς προσαρμόζεται η βελτιστοποίηση σε διάφορα μεγέθη εισόδου. Ο πίνακας 6.1 παρουσιάζει επίσης τα *μεγέθη* αυτών των εισόδων και τον αριθμό των *στοιχείων δεδομένων*, των *προσπελάσεων μνήμης*, των *επαναχρησιμοποιήσεων δεδομένων* και την *μέση απόσταση επαναχρησιμοποίησης δεδομένων*. Τα δεδομένα αυτά έχουν συλλεχθεί κάνοντας χρήση των δυνατοτήτων δυναμικής ανάλυσης του MemAssist. Πραγματοποιήθηκε δυναμική ανάλυση σε κάθε εφαρμογή για κάθε μια από τις αντίστοιχες εισόδους. Η μέση απόσταση επαναχρησιμοποίησης δεδομένων αναφέρεται στη μέση τιμή των αποστάσεων όλων των ζευγαριών αναφορών μνήμης κάθε εφαρμογής. Για όλες τις εφαρμογές έχει υλοποιηθεί και βελτιστοποιηθεί κώδικας τόσο σε C όσο και σε MATLAB. Ο πίνακας 6.1 αναφέρεται μόνο στις C εκδόσεις τους. Οι αριθμοί αυτοί όμως είναι σχεδόν πανομοιότυποι και για τους MATLAB κώδικες. Στο σχήμα 6.1 δίνεται μια σχηματική αναπαράσταση των εφαρμογών χρησιμοποιώντας διαγράμματα ροής, ενώ ακολουθεί μια σύντομη περιγραφή για κάθε μια από αυτές.

- Cavity Detector (***cavity***) [41]. Ιατρική διαγνωστική εφαρμογή για τον εντοπισμό εγκεφαλικών όγκων.
- Image Segmentation (***segm***) [59]. Μια υλοποίηση του αλγόριθμου ομαδοποίησης K-μέσων (k-means) για την τμηματοποίηση (segmentation) εικόνας. Η υλοποίηση αυτή κάνει χρήση της μεθόδου αφαιρετικής ομαδοποίησης (subtractive clustering) για τον εντοπισμό των βέλτιστων αρχικών κέντρων για κάθε ομάδα.
- Edge Detect (***edge***). Αλγόριθμος ανίχνευσης ακμών (edge detection) από τη σουίτα UTDSP [128], ο οποίος κάνει χρήση της

διαδικασίας συνέλιξης (convolution) και τελεστών Sobel.

- DCT/JPEG Preview (***dct***). Μια υλοποίηση της μεθόδου απωλεστικής συμπίεσης JPEG [151], η οποία εφαρμόζει τον *διακριτό μετασχηματισμό συνημιτόνου (discrete cosine transform, DCT)* και τα βήματα της κβαντοποίησης, για έναν δοθέντα πίνακα κβαντοποίησης και έπειτα αντιστρέφει τη διαδικασία. Σκοπός της συγκεκριμένης εφαρμογής είναι η παροχή μιας προεπισκόπησης της συμπιεσμένης εικόνας για την αξιολόγηση της ποιότητας της για έναν πίνακα κβαντοποίησης.
- 2D DWT/IDWT (***dwt***,***dwtlegall***). Ο *διακριτός μετασχηματισμός κυματιδίων (2D discrete wavelet transform, DWT)* όπως υλοποιείται στο πρότυπο συμπίεσης εικόνας JPEG 2000 [137]. Η εικόνα εισόδου διασπάται με τον DWT και έπειτα ανασυντίθεται με τη διαδικασία του αντίστροφου DWT (inverse DWT, IDWT). Ο αλγόριθμος έχει δοκιμαστεί με τα DWT φίλτρα CDF 9/7 και LeGall 5/3.

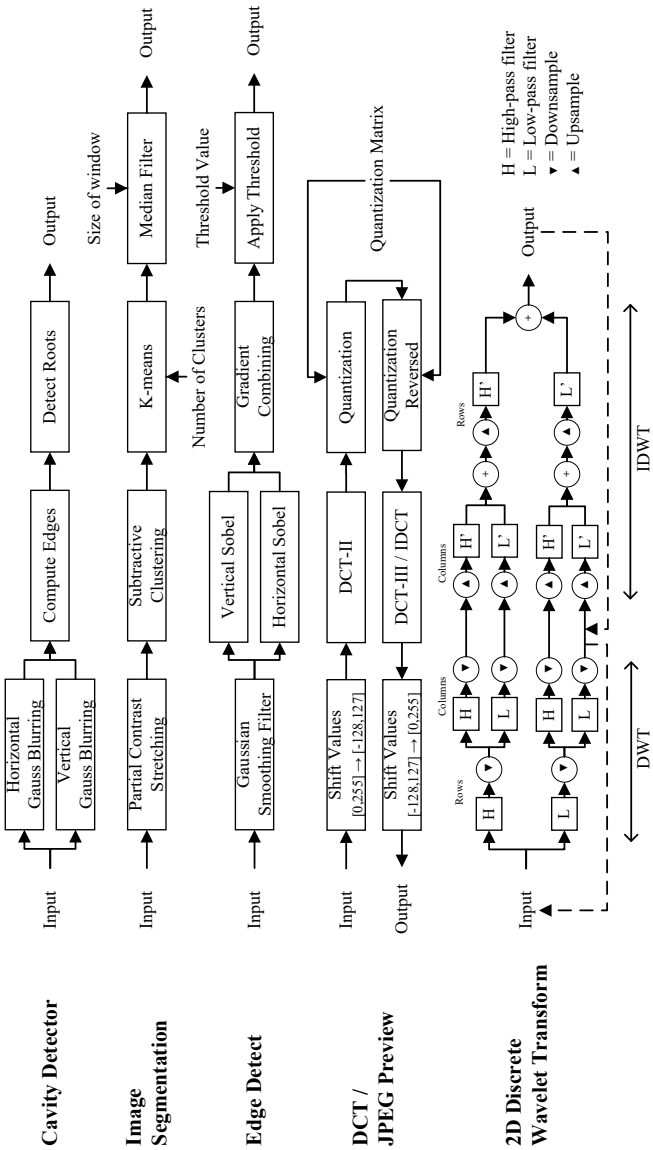
Πίνακας 6.1: Χαρακτηριστικά των υπό εξέταση εφαρμογών.

	Πηγή	Γραμμές κώδικα	Βρόχοι επανάληψης	Αναφορές μνήμης	Ζεύγη αναφορών μνήμης	Μέγεθος εισόδου	Στοιχεία δεδομένων	Προσπελάσεις μνήμης	Επαναχρησιμοποιήσεις δεδομένων	Μέση απόσταση επαναχρησιμοποίησης
Cavity Detector	[41]	112	23	52	61	32^2	32.79K	123.58K	94.08K	2.61M
						64^2	131.10K	526.60K	402.19K	44.21M
						128^2	524.31K	2.17M	1.66M	377.87M
						256^2	2.09M	8.85M	6.78M	543.61M
						512^2	8.38M	35.77M	27.43M	527.05M
						1024^2	33.55B	144.11M	110.66M	627.01M
DCT / JPEG Preview	[151]	162	42	98	102	32^2	5.31K	163.90K	158.59K	5.07M
						64^2	20.67K	655.42K	634.75K	78.78M
						128^2	81.11K	2.62M	2.53M	155.93M
						256^2	327.87K	10.48M	10.15M	265.84M
						512^2	1.31M	41.94M	40.63M	263.33M
						1024^2	5.24M	167.77M	162.52M	421.72M
DWT/IDWT CDF 9/7	[137]	331	44	86	121	32^2	17.44K	171.52K	154.08K	9.77M
						64^2	69.66K	686.08K	616.41K	155.65M
						128^2	278.56K	2.74M	2.46M	958.59M
						256^2	1.11M	10.97M	9.86M	1.13B
						512^2	4.45M	43.90M	39.45M	1.40B
						1024^2	17.82M	175.63M	157.81M	1.39B

$K = 10^3$, $M = 10^6$, $B = 10^9$.

	Πηγή	Γραμμές κώδικα	Βρόχοι επανάληψης	Αναφορές μνήμης	Ζεύγη αναφορών μνήμης	Μέγεθος εισόδου	Στοιχεία δεδομένων	Προσπελάσεις μνήμης	Επαναχρησιμοποιήσεις δεδομένων	Μέση απόσταση επαναχρησιμοποίησης
DWT/IDWT LeGall 5/3	[137]	331	44	86	121	32^2	17.44K	99.84K	82.41K	5.55M
						64^2	69.66K	399.36K	329.71K	88.60M
						128^2	278.56K	1.59M	1.31M	635.40M
						256^2	1.11M	6.38M	5.27M	1.14B
						512^2	4.45M	25.55M	21.10M	1.38B
						1024^2	17.82M	102.23M	84.41M	1.42B
Edge Detect	[128]	121	22	44	65	32^2	3.08K	56.47K	53.39K	2.41M
						64^2	12.29K	239.64K	227.34K	40.90M
						128^2	49.16K	986.90K	937.74K	277.41M
						256^2	196.61K	4.00M	3.80M	367.99M
						512^2	786.44K	16.13M	15.35M	362.12M
						1024^2	3.14M	64.77M	61.63M	432.61M
Image Segmentation	[59]	199	34	50	80	32^2	6.71K	2.48M	2.48M	44.58M
						64^2	25.14K	10.79M	10.76M	188.79M
						128^2	98.87K	44.58M	44.48M	415.64M
						256^2	393.78K	180.84M	180.44M	470.96M
						512^2	1.57M	726.69M	725.12M	695.98M
						1024^2	6.29M	2.90B	2.89B	685.58M

$K = 10^3$, $M = 10^6$, $B = 10^9$.



Σχήμα 6.1: Διαγράμματα ροής των υπό εξέταση εφαρμογών.

6.1.1.2 Πλατφόρμες

Έχουν πραγματοποιηθεί μετρήσεις στον προσομοιωτή λανθανουσών μνημών Cachegrind, καθώς και σε ένα σύνολο πραγματικών συστημάτων. Μια λίστα με τα χαρακτηριστικά αυτών των συστημάτων παρουσιάζεται στον πίνακα 6.2, ενώ οι ιεραρχίες μνήμης τους φαίνονται στον πίνακα 6.3. Το *a53* είναι ένα κινητό τηλέφωνο με λειτουργικό σύστημα Android (5.0.2), τα *pi3* και *mips* είναι υπολογιστές μονής πλακέτας (single-board computers, SBC) και το *i5* είναι ένας φορητός υπολογιστής με λειτουργικό σύστημα Windows 10. Τα πλεονεκτήματα από την εφαρμογή βελτιστοποιήσεων στον πηγαίο κώδικα ελέγχονται επίσης σε συνδυασμό με τις βελτιστοποιήσεις που πραγματοποιούνται αυτόματα κατά τη μεταγλώττιση από τους τρείς γνωστότερους μεταγλωττιστές της C: (1) Τον μεταγλωττιστή Visual C++ (*msvc*), (2) τον GCC (*gcc*) και (3) το Clang/LLVM (*clang*). Πραγματοποιήθηκαν δύο σετ μετρήσεων: ένα με απενεργοποιημένη την εφαρμογή αυτόματων βελτιστοποιήσεων και ένα με όλες τις βελτιστοποιήσεις ενεργοποιημένες, για όλους τους μεταγλωττιστές. Πιο συγκεκριμένα, ενεργοποιήθηκε ο διακόπτης *-O3* στον *gcc* και το *clang* και ο */O2* στον *msvc*. Ο πίνακας 6.2 περιλαμβάνει πληροφορίες και για το ποιοι μεταγλωττιστές χρησιμοποιήθηκαν σε κάθε πλατφόρμα. Στο *a53* έγινε χρήση των εργαλείων cross-compile που παρέχει το Android Native Development Kit (NDK)[1]. Στο *pi3* οι μετρήσεις που αφορούν τους μεταγλωττιστές *clang* και *gcc* έτρεξαν στο λειτουργικό σύστημα Raspbian, ενώ για τον *msvc* χρησιμοποιήθηκε το λειτουργικό σύστημα Windows 10 IoT Core.

6.1.1.3 Αξιόπιστη μέτρηση του χρόνου εκτέλεσης

Η καταγραφή ενός ακριβούς αριθμού δεν είναι συνήθως εφικτή κατά την μέτρηση του χρόνου εκτέλεσης ενός προγράμματος και στην συμβατική/αφελή μέτρηση μπορεί να υπάρχει μεγάλο στατιστικό σφάλμα. Ένας ελαφρά διαφορετικός χρόνος εκτέλεσης θα μετριέται πάντα, σε κάθε εκτέλεση του προγράμματος, ανεξάρτητα από τη μέθοδο που χρησιμοποιείται για την ανάκτηση των χρονοσημάνσεων (timestamps). Υπάρχουν διάφοροι παράγοντες που ευθύνονται για αυτό. Πρώτα και κύρια, το γεγονός ότι το υπό εξέταση πρόγραμμα δεν είναι το μόνο που εκτελείται τη στιγμή της μέτρησης. Ο επεξεργαστής χρησιμοποιείται επίσης συνεχόμενα από άλλες εφαρμογές, συμπεριλαμ-

[1]https://developer.android.com/ndk/guides/standalone_toolchain.html

Πίνακας 6.2: Επισκόπηση των συστημάτων στα οποία πραγματοποιήθηκαν μετρήσεις.

Συντομογραφία	Πλακέτα/Συσκευή	Τύπος	Επεξεργαστής	ISA	Μεταγλωττιστές
a53	Asus Zenfone 2 Laser ZE500KL	Έξυπνο τηλέφωνο	ARM Cortex-A53	ARMv8	gcc
pi3	Raspberry Pi 3 Model B	Υπολογιστής μονής πλακέτας	ARM Cortex-A53	ARMv8	clang, gcc, msvc
mips	MIPS Creator Ci20	Υπολογιστής μονής πλακέτας	XBurst	MIPS32r2	gcc
i5	Acer Aspire 5750	Φορητός υπολογιστής	Intel Core i5-2450M	x86	clang, gcc, msvc

Πίνακας 6.3: Ιεραρχίες μνήμης των συστημάτων.

	a53	pi3	mips	i5
L0-D cache	4 KB	⛔	⛔	⛔
L0-I cache	4 KB	⛔	⛔	⛔
L1-D cache	16 KB	16 KB	32 KB	32 KB
L1-I cache	16 KB	16 KB	32 KB	32 KB
L2 cache	2 MB	512 KB	512 KB	256 KB
L3 cache	⛔	⛔	⛔	3 MB
RAM size	2 GB	1 GB	1 GB	8 GB

Πίνακας 6.4: Ιεραρχίες μνήμης Cachegrind.

		conf1	conf2	conf3
L1-D	Μέγεθος	64 KB	32 KB	8 KB
	Χαρτογράφηση	2-way	8-way	direct
L1-I	Μέγεθος	64 KB	32 KB	8 KB
	Χαρτογράφηση	2-way	8-way	direct
L2	Μέγεθος	256 KB	4 MB	32 KB
	Χαρτογράφηση	8-way	16-way	4-way

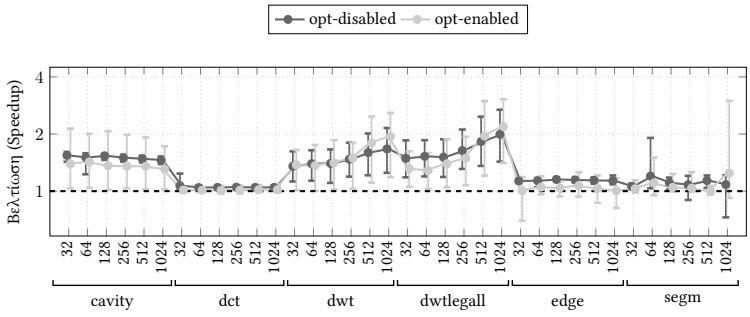
Μέγεθος γραμμής = 64bytes (conf1, conf2) και 16bytes (conf3)

βανομένων διαφόρων διεργασιών που τρέχουν στο παρασκήνιο κλπ., οι οποίες μπορεί να διαστρεβλώσουν την μέτρηση. Μια συνήθης προσέγγιση για την λήψη μιας πιο αξιόπιστης μέτρησης είναι να εκτελείται το πρόγραμμα πολλές φορές, να μετριέται ο χρόνος εκτέλεσης για κάθε μια από αυτές και να γίνεται χρήση της μέσης τιμής ή της διαμέσου αυτών των τιμών. Με αυτό τον τρόπο, περισσότερες εκτελέσεις θα συνεπάγονται και πιο αξιόπιστη τελική μέτρηση. Ωστόσο, ένας αριθμός πιο αποδοτικών τεχνικών έχουν προταθεί, οι οποίες παρέχουν μεγαλύτερη ακρίβεια από την απλή χρήση της μέσης τιμής ή της διαμέσου [107, 108].

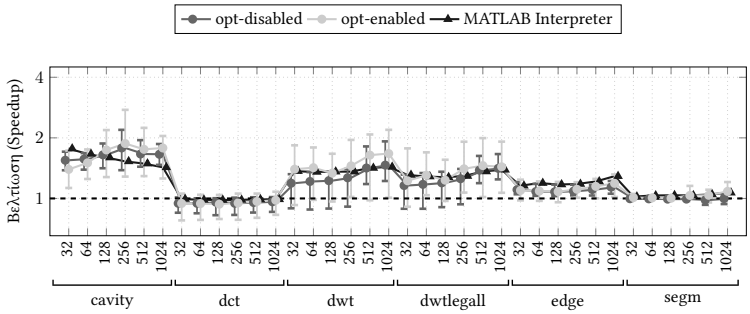
Για τις μετρήσεις της παρούσας διατριβής που αφορούν χρόνους εκτέλεσης έγινε χρήση του αλγόριθμου που παρουσιάζεται από τον Mueller στο άρθρο [108]. Το πρόγραμμα εκτελείται αρχικά *n* φορές και καταγράφεται ο χρόνος εκτέλεσης για κάθε μια. Έπειτα, για αυτές τις *n* τιμές, υπολογίζεται η διάμεσος και η διάμεση απόλυτη απόκλιση (median absolute deviation, MAD). Οι δύο αυτές τιμές χρησιμοποιούνται αρχικά για την απομάκρυνση των έκτοπων τιμών από το δείγμα. Όλες οι μετρήσεις που αποκλίνουν από τη διάμεσο πάνω από $X \times MAD$ απορρίπτονται και για τις εναπομένουσες τιμές υπολογίζεται ξανά η διάμεση απόλυτη απόκλιση. Ο μέσος όρος των τιμών αυτών είναι η αναμενόμενη τιμή και η διάμεση απόλυτη απόκλιση χρησιμοποιείται σαν μέτρο μεταβλητότητας. Ο υπολογισμός της αβεβαιότητας για την αναμενόμενη τιμή γίνεται με την πράξη $\frac{MAD}{\sqrt{N}}$, όπου *N* είναι ο αριθμός των εναπομενουσών μετρήσεων. Αν το αποτέλεσμα αυτής της πράξης είναι μικρότερο ή ίσο μιας προκαθορισμένης τιμής τότε ολοκληρώνεται η διαδικασία και ο μέσος όρος καταγράφεται ως ο τελικός χρόνος εκτέλεσης. Στην αντίθετη περίπτωση, το πρόγραμμα θα εκτελεστεί ακόμα μια φορά, έτσι ώστε να προστεθεί ακόμα μια τιμή στο δείγμα και θα επαναληφθεί η διαδικασία από την αρχή. Αυτό επαναλαμβάνεται έως ότου επιτευχθεί η ζητούμενη αβεβαιότητα.

6.1.2 Χρόνος εκτέλεσης

Οι πραγματικοί χρόνοι εκτέλεσης για τις *αρχικές* και τις *βελτιστοποιημένες* εκδόσεις όλων των εφαρμογών μετρήθηκαν σε όλες τις πλατφόρμες. Οι βελτιώσεις στην ταχύτητα εκτέλεσης που επιτεύχθηκαν παρουσιάζονται στα σχήματα 6.2, 6.3, 6.4, 6.5 και 6.6. Για κάθε μέτρηση, το *opt-enabled* υποδηλώνει ότι οι αυτόματες βελτιστοποιήσεις των μεταγλωττιστών έχουν ενεργοποιηθεί και το *opt-disabled* σημαί-

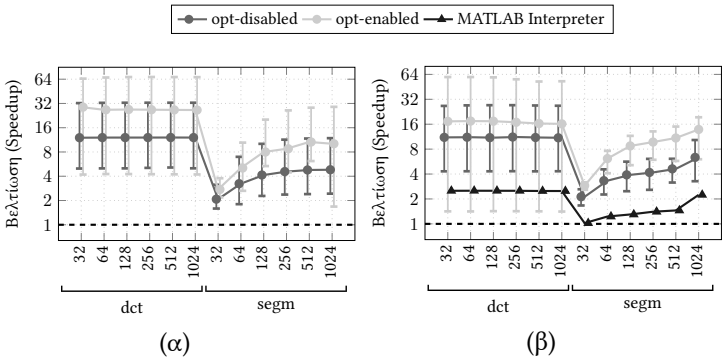


Σχήμα 6.2: Βελτιώσεις των χρόνων εκτέλεσης για την C έκδοση κάθε εφαρμογής έχοντας εφαρμόσει βελτιστοποιήσεις που αφορούν την τοπικότητα αναφοράς δεδομένων.



Σχήμα 6.3: Βελτιώσεις των χρόνων εκτέλεσης για τις MATLAB εκδόσεις κάθε εφαρμογής (MATLAB Interpreter και MATLAB Coder) έχοντας εφαρμόσει βελτιστοποιήσεις που αφορούν την τοπικότητα αναφοράς δεδομένων.

νει ότι έχουν απενεργοποιηθεί. Τα σχήματα αυτά, δείχνουν ότι επιτυγχάνεται μεγαλύτερη βελτιστοποίηση για τις εφαρμογές με κακή χρονική τοπικότητα αναφοράς δεδομένων. Οι εφαρμογές *segm* και *dct*, για παράδειγμα, ξεχωρίζουν καθώς σε αυτές δεν υπάρχει σχεδόν καθόλου βελτίωση της ταχύτητας εκτέλεσης σε αντίθεση με τις υπόλοιπες. Αυτές οι δύο εφαρμογές είτε έχουν ήδη πολύ καλή χρονική τοπικότητα, είτε δεν είναι εφικτό να βελτιστοποιηθούν κάνοντας χρήση των μετασχηματισμών που προτείνει το MemAssist. Πιο

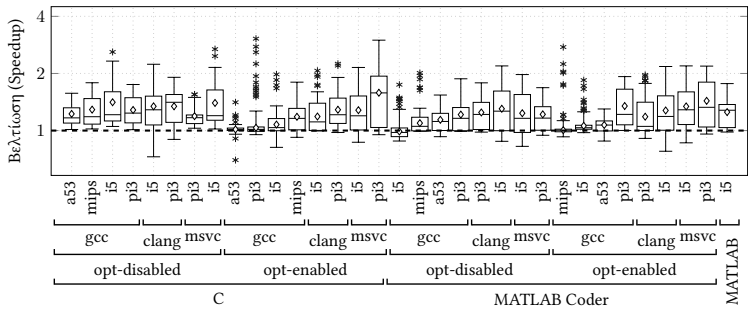


Σχήμα 6.4: Βελτιώσεις των χρόνων εκτέλεσης έχοντας εφαρμόσει βελτιστοποιήσεις που αφορούν την μείωση των περιττών επανυπολογισμών για: α) την C έκδοση κάθε εφαρμογής και β) τις MATLAB εκδόσεις κάθε εφαρμογής (MATLAB Interpreter και MATLAB Coder).

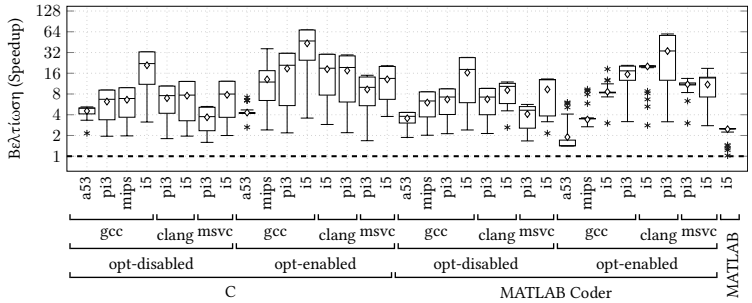
συγκεκριμένα, η εφαρμογή *dct* χωρίζει την εικόνα εισόδου σε μικρά κομμάτια των 8x8 εικονοστοιχείων και δρα ξεχωριστά στο κάθε ένα από αυτά. Κατά κάποιο τρόπο, το tiling είναι ένα έμφυτο χαρακτηριστικό του αλγορίθμου. Συνεπώς, η χρονική τοπικότητα του βρίσκεται ήδη σε αποδεκτά επίπεδα. Από την άλλη, μεγαλύτερη βελτιστοποίηση παρατηρείται σε εφαρμογές που εφαρμόζουν πολλαπλά φίλτρα συνέλιξης (convolution filters) επάνω σε εικόνες, ή γενικότερα που προσπελαύνουν τα ίδια στοιχεία μνήμης επαναλαμβανόμενα. Οι μεγάλες αποστάσεις ανάμεσα σε πολλαπλές προσπελάσεις στο ίδιο στοιχείο και η επακόλουθη κακή τοπικότητα αναφοράς δεδομένων είναι αυτά που φέρνει στην επιφάνεια το MemAssist ώστε να γίνει βελτιστοποίηση ενός αλγορίθμου. Οι *cavity*, *edge*, *dwt* και *dwtlegall* είναι τυπικές εφαρμογές που φέρουν τέτοια χαρακτηριστικά και για αυτές επετεύχθη μέχρι και τρεις φορές βελτίωση (3x) της ταχύτητας εκτέλεσης. Από τη στιγμή που δεν εντοπίστηκαν σημαντικά προβλήματα σχετιζόμενα με την τοπικότητα αναφοράς δεδομένων για τις εφαρμογές *dct* και *segm*, έγινε χρήση του MemAssist για την εύρεση άλλων πιθανών ευκαιριών βελτιστοποίησης. Χρησιμοποιήθηκαν οι μετρικές LWM και LI για την ανίχνευση των βρόχων επανάληψης που πραγματοποιούν έντονη επεξεργασία δεδομένων, ώστε να γίνει εστίαση σε αυτούς και να επιλυθούν πιθανά προβλήματα. Χρησιμοποιήθηκαν πί-

νακες αντιστοίχισης (lookup arrays) και στους δύο αλγόριθμους για τη μείωση των προβλημάτων που σχετίζονται με περιττούς επανυπολογισμούς τιμών, οι οποίοι γίνονται μέσα σε βρόχους επανάληψης. Οι επανυπολογισμοί αυτοί συνήθως περιλαμβάνουν κλήσεις σε δαπανηρές μαθηματικές συναρτήσεις, οι οποίες επιβαρύνουν σε μεγάλο βαθμό το συνολικό χρόνο εκτέλεσης του προγράμματος. Από αυτές τις βελτιστοποιήσεις προέκυψαν βελτιώσεις στην ταχύτητα εκτέλεσης των εφαρμογών έως και 68x (σχήμα 6.4). Μπορεί να παρατηρηθεί από τις μετρήσεις ότι οι βελτιώσεις της ταχύτητας εκτέλεσης παραμένουν στα ίδια επίπεδα ανεξάρτητα από το αν είναι ενεργοποιημένες ή απενεργοποιημένες οι αυτόματες βελτιστοποιήσεις των μεταγλωττιστών. Επομένως, οι μετασχηματισμοί που έχουν προταθεί από το MemAssist δεν αποτρέπουν πιθανές ευκαιρίες για την εφαρμογή περαιτέρω αυτόματων βελτιστοποιήσεων από τους μεταγλωττιστές. Αυτό καθιστά το MemAssist ιδανικό για χρήση σε συνδυασμό με τις βελτιστοποιήσεις που πραγματοποιούνται κατά τη μεταγλώττιση.

Τα σχήματα 6.5 και 6.6 παρουσιάζουν τις επιτευχθείσες βελτιώσεις της ταχύτητας εκτέλεσης ομαδοποιημένες σε θηκογράμματα (box plots). Κάθε θηκόγραμμα αναπαριστά την κατανομή των αποτελεσμάτων για όλες τις εισόδους, κάτω από έναν συγκεκριμένο συνδυασμό πλατφόρμας και μεταγλωττιστή. Κάθε κατανομή αποτελείται από 36 αποτελέσματα στο σχήμα 6.5 (6 εφαρμογές * 6 εισόδους η κάθε μια) και από 12 αποτελέσματα στο σχήμα 6.6 (2 εφαρμογές * 6 εισόδους η κάθε μια). Στο σχήμα 6.5 φαίνεται ότι η μεταγλώττιση με τον *msvc* προσφέρει καλύτερα αποτελέσματα από τους υπόλοιπους μεταγλωττιστές, όταν είναι ενεργοποιημένες οι αυτόματες βελτιστοποιήσεις. Αυτό δείχνει ότι οι βελτιστοποιήσεις που έχουν εφαρμοσθεί στον πηγαίο κώδικα έχουν μικρότερο αντίκτυπο στις αυτόματες βελτιστοποιήσεις που εφαρμόζει ο *msvc* και μεγαλύτερο σε αυτές που εφαρμόζει ο *gcc*. Δεν φαίνεται να υπάρχει κάποιο αντίστοιχο μοτίβο όσον αφορά τις επιπλέον βελτιστοποιήσεις που εφαρμόστηκαν στις εφαρμογές *dct* και *segm* (σχήμα 6.6). Συγκρίνοντας τις βελτιώσεις της ταχύτητας εκτέλεσης που επετεύχθησαν σε κάθε πλατφόρμα, φαίνεται πως σε μια συσκευή με μικρές λανθάνουσες μνήμες (*pi3*) υπάρχουν περισσότερα οφέλη από την εφαρμογή βελτιστοποιήσεων, σχετικών με την τοπικότητα αναφοράς δεδομένων. Από την άλλη, μια συσκευή με μεγάλες λανθάνουσες μνήμες (*i5*) δίνει καλύτερα αποτελέσματα για τις βελτιστοποιήσεις που αφορούν την μείωση των περιττών επανυπολογισμών.



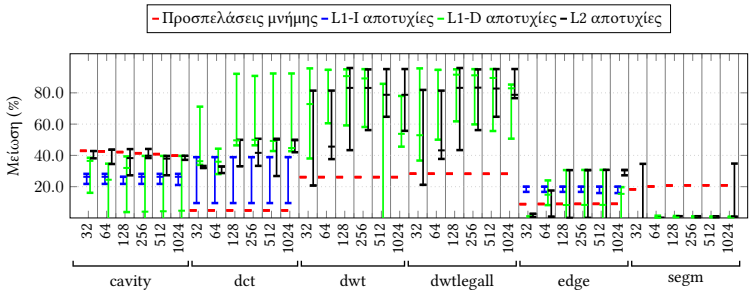
Σχήμα 6.5: Βελτιώσεις των χρόνων εκτέλεσης έχοντας εφαρμόσει βελτιστοποιήσεις που αφορούν την τοπικότητα αναφοράς δεδομένων.



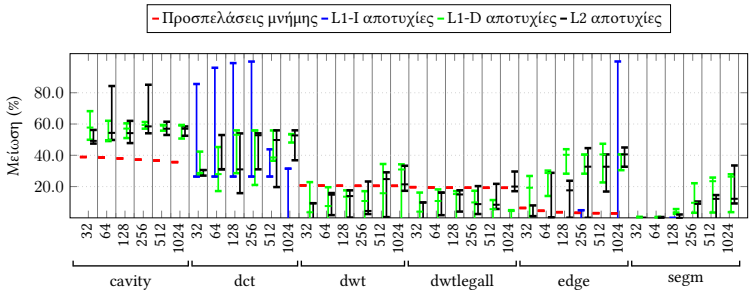
Σχήμα 6.6: Βελτιώσεις των χρόνων εκτέλεσης έχοντας εφαρμόσει βελτιστοποιήσεις που αφορούν την μείωση των περιττών επανυπολογισμών.

6.1.3 Αξιολόγηση της απόδοσης των λανθανουσών μνημών

Η αξιολόγηση της απόδοσης των λανθανουσών μνημών πραγματοποιήθηκε κάνοντας χρήση του προσομοιωτή Cachegrind [111]. Οι ρεαλιστικές ιεραρχίες μνήμης που παρουσιάζονται στον πίνακα 6.4 χρησιμοποιήθηκαν για όλα τα πειράματα προσομοίωσης. Η πρώτη γραμμή αυτού του πίνακα υποδεικνύει την σύντμηση που χρησιμοποιείται όταν γίνεται αναφορά στο κείμενο σε κάθε ιεραρχία μνήμης, ενώ τα σχήματα 6.7 και 6.8 παρουσιάζουν τα αποτελέσματα. Κάθε κάθετη γραμμή σε αυτό το γράφημα μπορεί να έχει μέχρι και τρία



Σχήμα 6.7: Βελτιώσεις στην απόδοση των λανθανουσών μνημών για την C έκδοση κάθε εφαρμογής.



Σχήμα 6.8: Βελτιώσεις στην απόδοση των λανθανουσών μνημών για την MATLAB Coder έκδοση κάθε εφαρμογής.

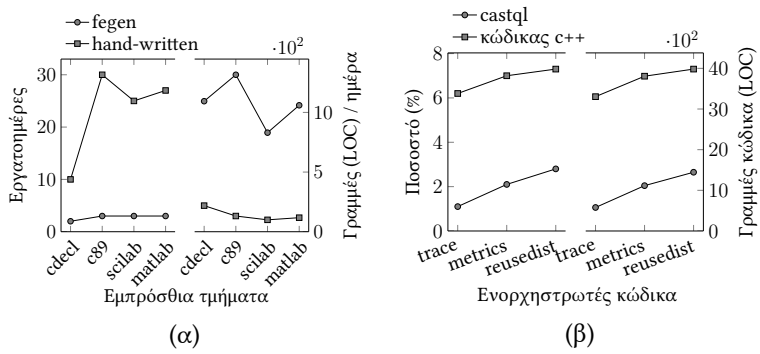
σημάδια, ένα για κάθε ιεραρχία μνήμης. Όπου υπάρχει μόνο ένα ή δύο σημάδια, σημαίνει πως οι μετρήσεις για τις δύο ή και για τις τρείς ιεραρχίες μνήμης συμπίπτουν. Η συγκεκριμένη ομάδα μετρήσεων επιβεβαιώνει ότι οι βελτιώσεις στην ταχύτητα εκτέλεσης που έχουν επιτευχθεί οφείλονται στη βελτιστοποίηση της τοπικότητας αναφοράς δεδομένων των εφαρμογών και στην επακόλουθη μείωση των αστοχιών λανθάνουσας μνήμης. Τα αποτελέσματα, ειδικά αυτά που αφορούν τις προσπελάσεις μνήμης, είναι ανάλογα με τις αντίστοιχες βελτιώσεις στην ταχύτητα εκτέλεσης για τις εφαρμογές *cavity*, *dwt*, *dwtlegall* και *edge*. Για την εφαρμογή *dct* επετεύχθη μια μείωση στις αστοχίες λανθάνουσας μνήμης, η οποία όμως δεν συνοδεύεται και από μια επακόλουθη μείωση των προσπελάσεων μνήμης. Για

την εφαρμογή segm δεν παρατηρείται σχεδόν καθόλου βελτιστοποίηση όσον αφορά τις αστοχίες λανθάνουσας μνήμης. Ωστόσο, υπάρχει μείωση των προσπελάσεων μνήμης, αλλά μόνο για την C έκδοση της εφαρμογής. Γενικά, οι προσπελάσεις μνήμης μειώθηκαν έως και 42%, οι αστοχίες λανθάνουσας μνήμης L2 και L1 (δεδομένων) μειώθηκαν έως και 95% και οι αστοχίες λανθάνουσας μνήμης L1 (εντολών) μειώθηκαν έως και 99%. Οι αστοχίες λανθάνουσας μνήμης δεν μειώνονται πάντα μαζί και στα δύο επίπεδα και μπορούν να παρατηρηθούν διάφορες διακυμάνσεις. Η σημαντική παρατήρηση ωστόσο, είναι ότι οι προσπελάσεις μνήμης μειώνονται σε όλες τις περιπτώσεις.

6.2 Αξιολόγηση της γλώσσας CastQL και του εργαλείου FEgen

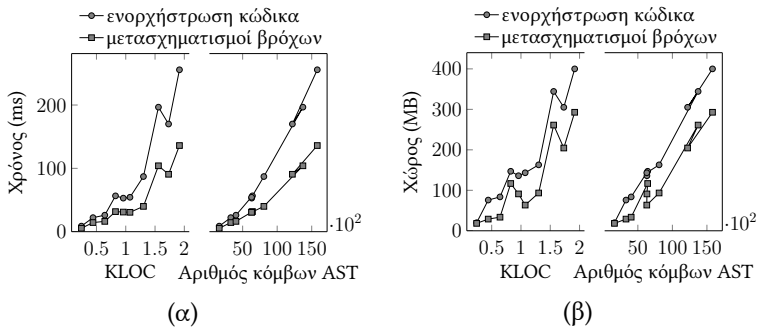
Η προσέγγιση που παρουσιάζεται στο κεφάλαιο 4, για την ανάπτυξη εργαλείων ανάλυσης και χειρισμού πηγαίου κώδικα, έχει αξιολογηθεί εκτενώς από τρείς διαφορετικές οπτικές: (1) Αρχικά, εξετάζονται τα οφέλη από τη χρήση του συστήματος CastQL-FEgen όσον αφορά την αύξηση της παραγωγικότητας του χρήστη/προγραμματιστή. (2) Έπειτα, παρουσιάζονται και σχολιάζονται πειραματικά αποτελέσματα που σχετίζονται με την απόδοση του παραγόμενου κώδικα/εργαλείου και τις απαιτήσεις του σε χρόνο και χώρο. (3) Ακόμα, αξιολογούνται οι ικανότητες παραγωγής κώδικα του FEgen και παρέχονται πειραματικά αποτελέσματα σχετικά με το μέγεθος του παραγόμενου κώδικα σε σύγκριση με τη γραμματική εισόδου. Στα σημεία που χρειάστηκε να γίνει ανάπτυξη λογισμικού από προγραμματιστή και όχι αυτόματα, για χάρη των πειραμάτων, αυτή έγινε από άτομο που κατέχει βασικές γνώσεις ανάπτυξης μεταγλωττιστών.

Στα σχήματα 6.9α και 6.9β φαίνονται τα αποτελέσματα από την πρώτη ομάδα πειραμάτων, όπου τα FEgen και CastQL αξιολογούνται ως προς την αύξηση της παραγωγικότητας του προγραμματιστή. Στην περίπτωση του FEgen καθορίστηκαν οι ημέρες εργασίας που απαιτούνται για την ανάπτυξη εμπρόσθιων τμημάτων για τέσσερις γλώσσες προγραμματισμού (σχήμα 6.9α). Για κάθε γλώσσα αναπτύχθηκε ένα εμπρόσθιο τμήμα χειροκίνητα από τον προγραμματιστή και ένα παρήχθη αυτόματα από το FEgen. Οι εργατοημέρες που απαιτούνται για την ανάπτυξη των εμπρόσθιων τμημάτων με το FEgen αποτελούν μόλις το 12% των αντίστοιχων εργατοημερών που απαιτούνται για την ανάπτυξή τους χωρίς τη χρήση κάποιου εργα-



Σχήμα 6.9: α) Απαιτούμενη εργασία για την ανάπτυξη εμπρόσθιων τμημάτων. β) Σύγκριση χρήσης της CastQL έναντι απλής υλοποίησης σε C++ όσον αφορά το μέγεθος του τελικού κώδικα.

λείου. Όσον αφορά το μέγεθος του κώδικα, ο αριθμός των *γραμμών κώδικα (lines of code, LOC)* που παράγεται καθημερινά αυξάνεται κατά 7.5 φορές όταν γίνεται χρήση του FEgen. Η αξιολόγηση της CastQL έγινε κατά τη χρήση της για την ανάπτυξη τριών λειτουργιών ενορχήστρωσης πηγαίου κώδικα, οι οποίες ενσωματώθηκαν στον μεταγλωττιστή MEMSCOPT [61]. Οι τρείς αυτοί ενορχηστρωτές κώδικα χρησιμοποιούνται για την πραγματοποίηση δυναμικής ανάλυσης. Πιο συγκεκριμένα: (1) για την εξαγωγή της ροής προσπελάσεων μνήμης σε αρχείο, (2) για τον υπολογισμό των μετρικών *βάρος βρόχου επανάληψης (loop weight metric, LWM)* και *συντελεστής επαναχρησιμοποίησης πινάκων (array reuse factor, ARF)* [93] και (3) για τη μέτρηση αποστάσεων επαναχρησιμοποίησης δεδομένων [94, 95]. Για κάθε μια από τις τρείς αυτές λειτουργίες αναπτύχθηκε μια έκδοση βασισμένη στην CastQL και μια βασισμένη σε κλασσικό κώδικα C++. Στο σχήμα 6.9β συγκρίνονται οι δύο αυτές εκδόσεις. Το γράφημα στα αριστερά παρουσιάζει το ποσοστό των γραμμών κώδικα που αντιστοιχούν στο μηχανισμό εφαρμογής ερωτημάτων σε σχέση με τον αριθμό των γραμμών κώδικα ολόκληρης της εφαρμογής. Το γράφημα στα δεξιά παρουσιάζει τις γραμμές που καταλαμβάνει κάθε μια από τις δύο εκδόσεις. Χρησιμοποιώντας την CastQL γράφεται πολύ λιγότερος κώδικας. Παρατηρείται μείωση των LOC στο 28% των αντίστοιχων γραμμών που πρέπει να γραφτούν σε C++. Η CastQL αυξάνει την επα-



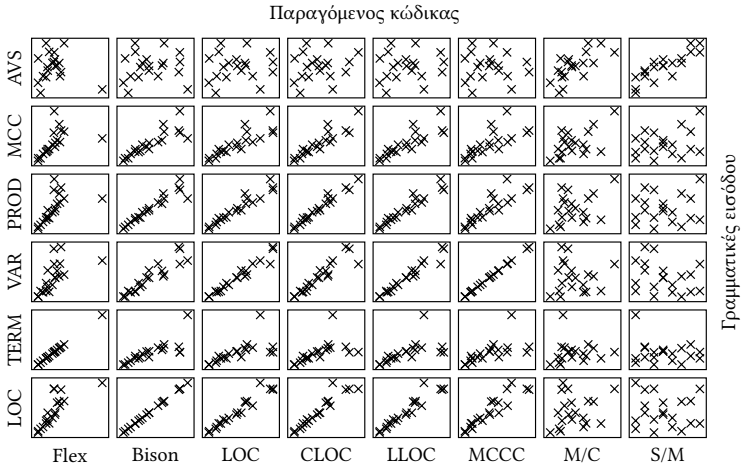
Σχήμα 6.10: α) Χρόνος εκτέλεσης και β) κατανάλωση μνήμης των δύο εξεταζόμενων σεναρίων για διάφορα μεγέθη εισόδων.

ναχρησιμοποίηση κώδικα ενώ ταυτόχρονα διατηρεί την υψηλή απόδοση που προσφέρει η C++, από τη στιγμή που είναι ενσωματωμένη σε αυτή.

Για την αξιολόγηση της απόδοσης του προτεινόμενου συστήματος, σχετικά με τις απαιτήσεις του παραγόμενου κώδικα/προγράμματος σε χρόνο και χώρο, έγιναν δοκιμές με δύο σενάρια πρακτικής εφαρμογής του. Σε κάθε σενάριο εφαρμόζεται μια σειρά λειτουργιών ανάλυσης και χειρισμού πηγαίου κώδικα, οι οποίες περιέχονται στο MEMSCOPT, σε ένα σύνολο δοκιμαστικών αλγόριθμων διάφορων μεγεθών. Το πρώτο σενάριο περιλαμβάνει την χρήση των τριών ενορχηστρωτών που αναφέρθηκαν προηγουμένως, ενώ το δεύτερο σενάριο αφορά την εφαρμογή μιας σειράς μετασχηματισμών βρόχων επανάληψης. Οι μετασχηματισμοί αυτοί περιλαμβάνουν την *επέκταση (loop extend)* των ορίων όλων των βρόχων κατά δέκα επιπλέον επαναλήψεις και την *εναλλαγή (loop interchange)* όλων των εμφωλευμένων βρόχων. Για τη συγγραφή των αρχείων εισόδου χρησιμοποιήθηκε κώδικας από έξι διαφορετικές δοκιμαστικές εφαρμογές, που όλες τους υπάγονται στο πεδίο της επεξεργασίας εικόνας και σήματος. Πιο συγκεκριμένα, πρόκειται για την ιατρική εφαρμογή cavity detector [41] και για πέντε αλγόριθμους από τη σουίτα UTDSP [128]. Το MEMSCOPT εκτελέστηκε σε έναν υπολογιστή με επεξεργαστή DualCore Intel Core i5, στα 2.50GHz, με 7.85 GB αξιοποιήσιμης μνήμης RAM. Το μέγεθος κάθε αρχείου εισόδου μετριέται σε *χιλιάδες γραμμών κώδικα (thousands of lines of code, KLOC)* και σε *συνολικό αριθμό κόμβων* που απαιτούνται

για την αναπαράσταση του σε μορφή cAST. Τα σχήματα 6.10α και 6.10β παρουσιάζουν τους χρόνους εκτέλεσης και τη χρήση της μνήμης για τα σενάρια που εξετάζονται. Οι απαιτήσεις σε χρόνο και χώρο είναι γραμμικά συσχετιζόμενες με τους αριθμούς γραμμών κώδικα των εισόδων και με τους αριθμούς των κόμβων που απαρτίζουν τα αντίστοιχα AST. Για την ακρίβεια, υπάρχει λίγο μεγαλύτερη συσχέτιση με τον αριθμό των κόμβων του AST, από τη στιγμή που η CastQL λειτουργεί κατευθείαν πάνω σε αυτό. Η απόδοση του προτεινόμενου συστήματος εξαρτάται πρωτίστως από το μέγεθος του AST και λιγότερο από άλλες παραμέτρους, όπως ο αριθμός των γραμμών κώδικα. Σε κάθε περίπτωση, η διαφορά αυτή είναι μηδαμινή για τα υπό εξέταση σενάρια, όπου ο αριθμός των AST κόμβων υπολογίζεται ότι έχει σταθερή αναλογία σε σχέση με τις γραμμές κώδικα για όλα τα αρχεία εισόδου. Η αναλογία αυτή, όπως είναι φανερό, εξαρτάται αποκλειστικά από το πόσο περίπλοκες είναι οι γραμμές του πηγαίου κώδικα. Μπορεί να ποικίλει για διαφορετικά στυλ μορφοποίησης κώδικα (coding styles) επηρεάζοντας και τη σχετική αναλογία.

Η αξιολόγηση του εργαλείου αυτόματης παραγωγής εμπρόσθιων τμημάτων (FEgen) πραγματοποιείται συγκρίνοντας ένα σύνολο μετρικών, που σχετίζονται με το μέγεθος και την πολυπλοκότητα του παραγόμενου κώδικα, με τις γλώσσες εισόδου. Εξετάστηκαν είκοσι γραμματικές BNF με σκοπό την απόδειξη της μεγάλης εφαρμοσιμότητας του εργαλείου. Πιο συγκεκριμένα, περάστηκαν από το FEgen οι γραμματικές των γλωσσών: C89, CDecl, Scilab, MATLAB, Ruby, PHP, C#, C++, Java, Delphi, XML, SQL, ALGOL60, Pascal, Fortran, Ada95, COBOL, BibTex, Yacc και SML. Ορισμένες από αυτές έχουν αναπτυχθεί για τις ανάγκες της παρούσας αξιολόγησης, ενώ οι υπόλοιπες έχουν ανακτηθεί από άλλες δημόσια διαθέσιμες πηγές και έχουν τροποποιηθεί κατάλληλα ώστε να γίνουν συμβατές με το FEgen. Ο παραγόμενος κώδικας C++ είναι κατά μέσο όρο 11 φορές μεγαλύτερος από τη γραμματική εισόδου, όσον αφορά τις γραμμές κώδικα (LOC). 475 γραμμές γραμματικής BNF αντιστοιχούν κατά μέσο όρο σε 5220 γραμμές C++ κώδικα. Οι παραγόμενες αυτές γραμμές αφορούν μόνο το μέρος του εργαλείου που φτάνει μέχρι τη δημιουργία του συμπαγούς συντακτικού δέντρου. Η διαδικασία μέχρι αυτό το σημείο είναι πλήρως αυτοματοποιημένη και ο χρήστης δεν χρειάζεται να γράψει καθόλου C++ κώδικα. Αφηρημένα συντακτικά δέντρα έχουν οριστεί μόνο για τις γλώσσες C89, MATLAB, Scilab και CDecl. Τα εμπρόσθια τμήματα των γλωσσών αυτών είναι κατά μέσο όρο 30% μεγαλύτερα, σε αριθμό γραμμών κώδικα, από εκείνα των γλωσσών για τις οποίες



Σχήμα 6.11: Συσχέτιση μεταξύ των γραμματικών εισόδου και των παραγόμενων αναλυτών.

δεν έχουν οριστεί αφηρημένα συντακτικά δέντρα. Τα διαγράμματα διασποράς του σχήματος 6.11 παρέχουν μια αναλυτικότερη εικόνα, όσον αφορά τη σύγκριση του κώδικα C++ που παράγεται από το FEgen με τις γραμματικές εισόδου τύπου BNF. Οι μετρικές που αναφέρονται σε αυτό το σχήμα περιλαμβάνουν, για τις γραμματικές εισόδου:

- τον αριθμό των τερματικών συμβόλων (TERM).
- τον αριθμό των μη-τερματικών συμβόλων (VAR).
- τον συνολικό αριθμό των κανόνων παραγωγής (PROD).
- την κυκλωματική πολυπλοκότητα του McCabe (MCC) [118].
- το μέσο μέγεθος του δεξιού μέρους των κανόνων παραγωγής (AVS) [118].

και για τον παραγόμενο κώδικα:

- τον αριθμό γραμμών του παραγόμενου αρχείου *.l* (Flex).
- τον αριθμό γραμμών του παραγόμενου αρχείου *.y* (GNU bison).

- τον αριθμό γραμμών σχολίων του παραγόμενου κώδικα C++ (CLOC).
- τον αριθμό λογικών γραμμών του παραγόμενου κώδικα C++ (LLOC).
- την κυκλωματική πολυπλοκότητα McCabe [118] του παραγόμενου κώδικα C++ (MCCC).
- τον μέσο αριθμό μεθόδων ανά παραγόμενη κλάση της C++ (M/C).
- τον μέσο αριθμό εντολών ανά μέθοδο (S/M).

Για τον καθορισμό της συσχέτισης μεταξύ των εξεταζόμενων μετρικών έγινε χρήση του *συντελεστή συσχέτισης Spearman*. Όπως ήταν αναμενόμενο, τα περισσότερα από τα συγκρινόμενα ζεύγη είναι εξαιρετικά συσχετισμένα. Πιο συγκεκριμένα, ο αριθμός γραμμών της γραμματικής εισόδου και του παραγόμενου αρχείου .y (GNU bison) έχουν εξαιρετικά μεγάλη συσχέτιση (ο συντελεστής συσχέτισης μεταξύ τους είναι 0.99). Αυτό, βέβαια, είναι λογικό από τη στιγμή που τα δύο αρχεία έχουν σχεδόν το ίδιο περιεχόμενο. Ακόμα μια ενδιαφέρουσα παρατήρηση είναι ότι οι τιμές της μετρικής MCC των γραμματικών εισόδου δεν είναι άκρως συσχετισμένες με τις τιμές MCC του παραγόμενου κώδικα (ο συντελεστής συσχέτισης μεταξύ τους είναι 0.72). Αυτό δείχνει πως η πολυπλοκότητα της γραμματικής εισόδου δεν επηρεάζει άμεσα την πολυπλοκότητα του παραγόμενου κώδικα. Από την άλλη, ο αριθμός των μη-τερματικών συμβόλων παίζει πολύ σημαντικό ρόλο στην πολυπλοκότητα του κώδικα, μιας και ο συντελεστής συσχέτισης μεταξύ των μετρικών VAR και MCCC είναι 1.

Κεφάλαιο 7

Συμπεράσματα

Σε αυτή την διατριβή παρουσιάστηκε το MemAssist, ένα εργαλείο που αναλύει εφαρμογές γραμμένες σε C ή MATLAB και προτείνει βελτιστοποιήσεις, οι οποίες μπορούν να υλοποιηθούν από τον χρήστη σε επίπεδο πηγαίου κώδικα, ώστε να βελτιωθεί η χρονική τοπικότητα αναφοράς δεδομένων των εφαρμογών. Εξετάστηκαν ζητήματα που αφορούν την εσωτερική λειτουργία του προτεινόμενου εργαλείου, ενώ προτάθηκαν και κάποιες βοηθητικές μετρικές για την αξιολόγηση εφαρμογών. Βελτιστοποιήθηκαν έξι εφαρμογές ακολουθώντας τις προτάσεις / ενδείξεις του MemAssist και τα αποτελέσματα δείχνουν ότι επετεύχθη μείωση των προσπελάσεων της κύριας μνήμης έως και 42%, με τη συνακόλουθη βελτίωση του χρόνου εκτέλεσης (speedup) έως και 3x για βελτιστοποιήσεις που αφορούν την τοπικότητα αναφοράς δεδομένων και έως 68x για βελτιστοποιήσεις που αφορούν την μείωση των περιττών επανυπολογισμών.

7.1 Συμβολή της διατριβής

Οι συνεισφορές της διατριβής συνοψίζονται παρακάτω:

- Παρουσιάστηκαν συγκεντρωτικά και με συνεκτικό τρόπο οι αλγόριθμοι υπολογισμού αποστάσεων επαναχρησιμοποίησης δεδομένων που έχουν προταθεί κατά καιρούς.

- Έγινε παρουσίαση του εργαλείου MemAssist, το οποίο διατίθεται σαν επέκταση (extension) για το Microsoft Visual Studio, παρέχοντας ένα ενοποιημένο περιβάλλον για την ανάπτυξη και την βελτιστοποίηση εφαρμογών, αλλά είναι επίσης διαθέσιμο και σαν διαδικτυακή εφαρμογή.
- Επειδή το MemAssist πραγματοποιεί ανάλυση της απόστασης επαναχρησιμοποίησης δεδομένων μόνο για κώδικα C, αναπτύχθηκε ένας MATLAB-σε-C μεταγλωττιστής, ο οποίος ονομάζεται MAFE. Ο μεταγλωττιστής αυτός έχει τη δυνατότητα να συσχετίζει τις μεταβλητές του MATLAB κώδικα που δέχεται με κομμάτια του κώδικα C που παράγει. Έπειτα, πραγματοποιείται η διαδικασία της ανάλυσης των αποστάσεων επαναχρησιμοποίησης δεδομένων στον C κώδικα και οι προτάσεις που παράγονται αντιστοιχίζονται στον MATLAB κώδικα της εισόδου.
- Προτάθηκε μια μέθοδος για την αυτόματη επιλογή κατάλληλων μετασχηματισμών βρόχων επανάληψης.
- Έγινε αναλυτική σύγκριση του MemAssist με έναν παρεμφερή βελτιστοποιητή.
- Παρουσιάστηκαν οι μετρικές: (1) βάρος βρόχου επανάληψης (LWM/LI), για την εύρεση σημείων ενδιαφέροντος μέσα στον πηγαίο κώδικα, στα οποία ενδέχεται να πραγματοποιείται έντονη επεξεργασία δεδομένων και (2) συντελεστής επαναχρησιμοποίησης πίνακα (ARF).
- Πραγματοποιήθηκε βελτιστοποίηση έξι εφαρμογών επεξεργασίας εικόνας, κάνοντας χρήση του MemAssist, αποδεικνύοντας πειραματικά την αποτελεσματικότητά του.
- Παρουσιάστηκε η γλώσσα προγραμματισμού ειδικού σκοπού CastQL, για την εφαρμογή ερωτημάτων σε πηγαίο κώδικα (query language). Η CastQL λειτουργεί πάνω σε μια AST αναπαράσταση που λέγεται cAST (contextual abstract syntax tree), ενώ πρόκειται για μια εσωτερική/ενσωματωμένη (internal/embedded) γλώσσα προγραμματισμού ειδικού σκοπού [69] που κάνει χρήση της C++ ως γλώσσα υποδοχής.
- Αναπτύχθηκε ένας αυτόματος γεννήτορας εμπρόσθιων τμημάτων μεταγλωττιστών, με το όνομα FEgen, ο οποίος παράγει

έναν λεκτικό και συντακτικό αναλυτή για οποιαδήποτε δοθείσα γραμματική κάνει χρήση της μορφής συμβολισμού BNF.

7.2 Μελλοντικές κατευθύνσεις

Η δουλειά που παρουσιάζεται στην παρούσα διατριβή αποτελείται από διάφορα επιμέρους εργαλεία και τεχνικές στα οποία μελλοντικά θα μπορούσαν να υλοποιηθούν οι ακόλουθες επεκτάσεις:

- Η υλοποίηση παράλληλων αλγορίθμων για την ανάλυση αποστάσεων επαναχρησιμοποίησης δεδομένων. Προς το παρόν γίνεται χρήση σειριακών αλγορίθμων και τεχνικών δειγματοληψίας.
- Η εισαγωγή και χρήση εννοιών απόστασης επαναχρησιμοποίησης δεδομένων για την ανάλυση της τοπικότητας αναφοράς παράλληλων εφαρμογών.
- Η αναζήτηση άλλων μεθόδων για την μείωση της επιβάρυνσης που υφίσταται κατά την δυναμική ανάλυση (profiling overhead).
- Μπορεί να αφιερωθεί κάποιος χρόνος για την ανάπτυξη μεθόδων για την αυτοματοποίηση της εφαρμογής των μετασχηματισμών που προτείνονται από το MemAssist. Μια πρώτη ιδέα για να επιτευχθεί αυτό είναι να γίνει χρήση των δυνατοτήτων εφαρμογής source-to-source μετασχηματισμών που παρέχει ο μεταγλωττιστής MEMSCOPT.
- Θα μπορούσαν να ενσωματωθούν δυνατότητες προσομοίωσης ιεραρχιών μνήμης στο MemAssist, ώστε ο χρήστης να σχηματίζει καλύτερη εικόνα γύρω από την τοπικότητα αναφοράς δεδομένων της υπό εξέταση εφαρμογής, παρέχοντάς του πληροφορίες σχετικά με τον ρυθμό αστοχίας των λανθανουσών μνημών.
- Ο εμπλουτισμός της CastQL με επιπλέον τύπους ερωτημάτων χαμηλού επιπέδου (LLQs) και στρατηγικών αναζήτησης.
- Τα εργαλεία CastQL και FEgen προς το παρόν δεν παρέχουν κάποιο ολοκληρωμένο περιβάλλον εργασίας, το οποίο θα βελτίωνε τη συνεργασία μεταξύ τους. Η υλοποίηση μίας επέκτασης για το Visual Studio, όπως στην περίπτωση του MemAssist,

θα έκανε τη λειτουργία τους απρόσκοπτη και θα παρείχε ένα φιλικό προς το χρήστη περιβάλλον εργασίας.

Παράρτημα Α΄

Παράμετροι εργαλείων

Σε αυτό το παράρτημα παρουσιάζονται οι παράμετροι που δέχονται οι διεπαφές γραμμής εντολών των εργαλείων που αναπτύχθηκαν στα πλαίσια της παρούσας διατριβής. Στον πίνακα Α΄.1 φαίνονται όλες οι παράμετροι γραμμής εντολών του MEMSCOPT, ενώ ο πίνακας Α΄.2 παρέχει μια λίστα με τους μετασχηματισμούς βρόχων επανάληψης που υποστηρίζει το εργαλείο όταν εκτελείται σε κατάσταση μετασχηματισμών. Οι πίνακες Α΄.3 και Α΄.4 δείχνουν τις παραμέτρους γραμμής εντολών του μεταγλωττιστή MAFE και του γεννήτορα εμπρόσθιων τμημάτων FEgen αντίστοιχα. Το MEMSCOPT παρέχει και γραφική διεπαφή χρήστη που υποστηρίζει τη πλειονότητα των διαθέσιμων λειτουργιών. Το MemAssist είναι διαθέσιμο ως επέκταση για το Visual Studio και ως διαδικτυακή εφαρμογή, ενώ δεν παρέχει διεπαφή γραμμής εντολών.

Πίνακας Α΄.1: Παράμετροι γραμμής εντολών του MEMSCOPT.

Τρόπος χρήσης: *MEMSCOPT <αρχείο_εισόδου> <κατάσταση> [παράμετροι]*

Παράμετρος	Περιγραφή
<αρχείο_εισόδου>	Το αρχείο εισόδου *.c*.
<κατάσταση>	Παίρνει μια από τις τιμές: *analysis* ή *transformations*.
–t	Εξάγει τα περιεχόμενα του πίνακα συμβόλων σε αρχείο, σε μορφή κειμένου.
–noSAL	Δεν τοποθετεί σχόλια SAL στους βρόχους του παραγόμενου κώδικα C.
–noC	Απενεργοποιεί το οπίσθιο τμήμα παραγωγής C κώδικα και δεν παράγεται αρχείο εξόδου *.c*.
–genPT	Ενεργοποιεί την εξαγωγή του συμπαγούς συντακτικού δέντρου σε γράφο Graphviz.
–genAST	Ενεργοποιεί την εξαγωγή του αφηρημένου συντακτικού δέντρου σε γράφο Graphviz.
–genXML	Ενεργοποιεί την εξαγωγή του αφηρημένου συντακτικού δέντρου σε μορφή *.xml*.
–version	Εμφανίζει πληροφορίες για την έκδοση της εφαρμογής.
–help	Εμφανίζει πληροφορίες για τον τρόπο χρήσης της εφαρμογής.
Κατάσταση ανάλυσης:	
–A:όνομα_συνάρτησης	Το όνομα της προς ανάλυση συνάρτησης από το αρχείο εισόδου.
–P:διαδρομή_αρχείου	Το όνομα / η διαδρομή ενός αρχείου *.xml* που περιλαμβάνει επιπλέον παραμέτρους για την ανάλυση.
–a:όνομα_αρχείου	Το όνομα του ενορχηστρωμένου αρχείου *.c* που θα παραχθεί.
–T:διαδρομή_φακέλου	Η διαδρομή του φακέλου στον οποίο θα αποθηκευθούν τα αποτελέσματα της δυναμικής ανάλυσης.
–lwmarf	Ενεργοποιεί τις μετρικές LWM και ARF.
–din	Ενεργοποιεί την εξαγωγή της ροής προσπελάσεων μνήμης σε αρχείο.
–cnt	Μετράει το σύνολο των προσπελάσεων μνήμης.
–rd	Πραγματοποιεί ανάλυση των αποστάσεων επαναχρησιμοποίησης δεδομένων.
Κατάσταση μετασχηματισμών:	
–tl:διαδρομή_αρχείου	Ένα αρχείο *.tl* που περιλαμβάνει μετασχηματισμούς προς εφαρμογή (πίνακας Α΄.2).
–o:διαδρομή_αρχείου	Το όνομα / η διαδρομή του αρχείου *.c* που θα παραχθεί.

Πίνακας Α΄.2: Λίστα διαθέσιμων μετασχηματισμών στο MEMSCOPT.

Μετασχηματισμός	Σύνταξη
Μετατόπιση βρόχου (loop shift)	loop_shift(loopX,±N);
Μετατοπίζει το χώρο επαναλήψεων του βρόχου loopX κατά N βήματα προς την θετική (+) ή αρνητική (-) πλευρά. Το N μπορεί να είναι ένας αριθμός ή μια συμβολική σταθερά.	
Επέκταση βρόχου (loop extend)	loop_extend(loopX,+M,+L);
Επεκτείνει τα όρια του βρόχου loopX κατά M βήματα προς την αρνητική κατεύθυνση και κατά L βήματα προς τη θετική κατεύθυνση. Τα M και L μπορούν να είναι αριθμοί ή συμβολικές σταθερές.	
Αντιστροφή βρόχου (loop reversal)	loop_reverse(loopX);
Αντιστρέφει την κατεύθυνση/φορά των επαναλήψεων του βρόχου.	
Συγχώνευση βρόχου (loop fusion)	loop_fusion(loopX,loopY);
Συγχωνεύει/ενώνει τους βρόχους loopX και loopY σε έναν βρόχο που περιλαμβάνει τις εντολές που βρίσκονται στα κυρίως σώματα και των δύο.	
Εναλλαγή βρόχων (loop interchange)	loop_interchange(loopX,loopY);
Εναλλάσσει τον βρόχο loopX με τον loopY. Προϋποθέτει ότι ο ένας βρόχος είναι εμφωλευμένος στον άλλο.	

Μετασχηματισμός	Σύνταξη
Διαχωρισμός βρόχων (loop fission)	loop_fission(loopX,%instr_set%);
Διαχωρίζει τον βρόχο loopX σε δύο ξεχωριστούς βρόχους. %instr_set% είναι οι εντολές που θα μεταφερθούν στον δεύτερο βρόχο μετά από τον διαχωρισμό. Οι υπόλοιπες τοποθετούνται στον πρώτο βρόχο.	
Κανονικοποίηση βρόχου (loop normalization)	loop_normalization(loopX);
Κάνει τις κατάλληλες μετατροπές στον βρόχο loopX ώστε να ξεκινάει από το 0 και να έχει σταθερό βήμα.	
Αναδιάταξη βρόχων (loop reorder)	loop_reorder(loopX,loopY);
Αντιστρέφει τη σειρά εκτέλεσης των δύο βρόχων.	
Αντίστροφη αποδιακλάδωση βρόχου (loop switching)	loop_switching(loopX);
Μετακινεί μια εντολή συνθήκης μέσα στο κυρίως σώμα ενός βρόχου, ενώ βρίσκεται αρχικά εκτός του.	
Μετακίνηση βρόχου εμπρός (loop scope move forward)	loop_scope_mv_fwd(loopX);
Αναδιατάσσει τη σειρά εκτέλεσης του βρόχου με την επόμενη από αυτόν εντολή.	
Μετακίνηση βρόχου πίσω (loop scope move backwards)	loop_scope_mv_bwd(loopX);
Αναδιατάσσει τη σειρά εκτέλεσης του βρόχου με την προηγούμενη από αυτόν εντολή.	

Πίνακας Α΄.3: Παράμετροι γραμμής εντολών του MAFE.

Τρόπος χρήσης: *MAFE <αρχείο_εισόδου> [παράμετροι]*

Παράμετρος	Περιγραφή
<αρχείο_εισόδου>	Το αρχείο εισόδου *.m*.
–t	Εξάγει τα περιεχόμενα του πίνακα συμβόλων σε αρχείο, σε μορφή κειμένου.
–genPT	Ενεργοποιεί την εξαγωγή του συμπαγούς συντακτικού δέντρου σε γράφο GraphViz.
–genAST	Ενεργοποιεί την εξαγωγή του αφηρημένου συντακτικού δέντρου σε γράφο GraphViz.
–S:τιμή	Παίρνει μια από τις τιμές: *static* (για την παραγωγή κώδικα C που κάνει στατική δέσμευση μνήμης για τους πίνακες) ή *dynamic* (για την παραγωγή κώδικα C με δυναμική δέσμευση μνήμης). Αν παραλειφθεί αυτή η παράμετρος, παράγεται δυναμικός κώδικας σαν προεπιλογή.
–P:διαδρομή_φακέλου	Η διαδρομή του φακέλου στον οποίο θα αποθηκευθούν τα παραγόμενα αρχεία *.c* και *.h*.
–version	Εμφανίζει πληροφορίες για την έκδοση της εφαρμογής.
–help	Εμφανίζει πληροφορίες για τον τρόπο χρήσης της εφαρμογής.

Πίνακας Α΄.4: Παράμετροι γραμμής εντολών του FEgen.

Τρόπος χρήσης:
FEgen <γραμματική_εισόδου> <αρχείο_παραμετροποίησης> <διαδρομή_εξόδου>

Παράμετρος	Περιγραφή
<γραμματική_εισόδου>	Το αρχείο εισόδου γραμματικής BNF.
<αρχείο_παραμετροποίησης>	Το όνομα / η διαδρομή ενός αρχείου *.xml* που περιλαμβάνει παραμέτρους που αφορούν την ονομασία των στοιχείων που απαρτίζουν τον παραγόμενο κώδικα (ονόματα κλάσεων, αρχείων κτλ).
<διαδρομή_εξόδου>	Η διαδρομή του φακέλου στον οποίο θα αποθηκευθούν τα παραγόμενα αρχεία *.cpp*, *.h*, *.y* και *.l*.
–version	Εμφανίζει πληροφορίες για την έκδοση της εφαρμογής.
–help	Εμφανίζει πληροφορίες για τον τρόπο χρήσης της εφαρμογής.

Παράρτημα Β΄

Παραδείγματα υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων

Σε αυτό το παράρτημα παρουσιάζονται μια σειρά από παραδείγματα εκτέλεσης των αλγόριθμων υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων που αναφέρονται στο κεφάλαιο 2. Όλα τα παραδείγματα, εκτός από το τελευταίο κατά σειρά (σχήμα Β΄.7), παρουσιάζουν τα βήματα κάθε αλγόριθμου για τον υπολογισμό των αποστάσεων της ροής προσπελάσεων μνήμης που φαίνεται στο σχήμα Β΄.1. Στο τελευταίο παράδειγμα, που δείχνει τον αλγόριθμο με ισοζυγισμένο δυαδικό δέντρο τρυπών (σχήμα Β΄.7), γίνεται υπολογισμός των αποστάσεων για ένα στιγμιότυπο μιας ξεχωριστής ροής προσπελάσεων μνήμης. Η ροή αυτή φαίνεται στο επάνω μέρος του σχήματος Β΄.7. Η ροή στο σχήμα Β΄.1 αποτελείται από 9 μόνο προσπελάσεις μνήμης. Ο αριθμός αυτός είναι πολύ μικρός για την σωστή επίδειξη του αλγόριθμου τρυπών.

Στο σχήμα Β΄.2 παρουσιάζεται το παράδειγμα υπολογισμού των αποστάσεων με τον αφελή αλγόριθμο. Το κουτί με πράσινο χρώμα αναπαριστά την τρέχουσα προσπέλαση ενώ εκείνο με κόκκινο χρώμα αναπαριστά την αμέσως προηγούμενη προσπέλαση προς το ίδιο στοιχείο. Η προηγούμενη αυτή προσπέλαση ανακτάται σε κάθε βήμα από τον πίνακα κατακερματισμού. Η διαδικασία αυτή υποδεικνύεται με

το μαύρο βέλος. Το γκρίζο κουτί στον πίνακα κατακερματισμού δείχνει την αλλαγή του χρόνου τελευταίας προσπέλασης για το τρέχον στοιχείο κάθε βήματος. Τα πράσινα και κόκκινα βέλη αναπαριστούν διασχίσεις των προσπελάσεων από το χρόνο t_{cur} προς τα αριστερά μέχρι το χρόνο t_{prev}. Κάθε διάσχιση αναζητά ένα στοιχείο. Με πράσινο φαίνονται οι διασχίσεις που ανακαλύπτουν το στοιχείο που ψάχνουν και με κόκκινο φαίνονται εκείνες που φτάνουν στο χρόνο t_{prev} χωρίς να βρουν το αναζητούμενο στοιχείο.

Στο σχήμα Β΄.3 παρουσιάζεται το παράδειγμα υπολογισμού των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση στοίβας. Με πράσινο αναπαριστώνται τα καινούρια στοιχεία που δεν υπάρχουν στη στοίβα και εισάγονται κατευθείαν στην κεφαλή της. Με κόκκινο περίγραμμα φαίνονται τα στοιχεία που υπάρχουν ήδη στη στοίβα ενώ τα κόκκινα βέλη συμβολίζουν τη μετακίνηση τους από τη θέση που εντοπίστηκαν προς την κεφαλή.

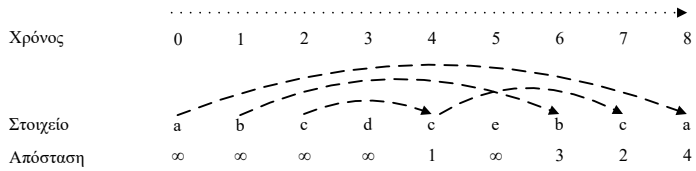
Στο σχήμα Β΄.4 παρουσιάζεται το παράδειγμα υπολογισμού των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση πίνακα δυαδικών ψηφίων. Οι προσπελάσεις που γίνονται στους χρόνους t_{cur} και t_{prev} καθώς και ο πίνακας κατακερματισμού παρουσιάζονται με ακριβώς τον ίδιο τρόπο όπως και στο σχήμα Β΄.2. Για τις προσπελάσεις που αλλάζουν τιμή, οι παλιές τιμές φαίνονται διαγραμμένες και με κόκκινο χρώμα.

Στο σχήμα Β΄.5 παρουσιάζεται το παράδειγμα υπολογισμού των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση στατικά δεσμευμένου m-αδικού δέντρου. Τα χοντρά μαύρα βέλη δείχνουν την διάσχιση από το φύλλο που αναπαριστά την προσπέλαση σε χρόνο t_{prev} μέχρι τη ρίζα του δέντρου. Οι χοντρές μαύρες γραμμές που εφάπτονται σε αυτά τα βέλη δείχνουν τον υπολογισμό της απόστασης επαναχρησιμοποίησης, αθροίζοντας τις τιμές των υποδέντρων που βρίσκονται στα δεξιά κάθε βέλους/διάσχισης. Για τους κόμβους που αλλάζουν τιμή, οι παλιές τιμές φαίνονται διαγραμμένες και με κόκκινο χρώμα.

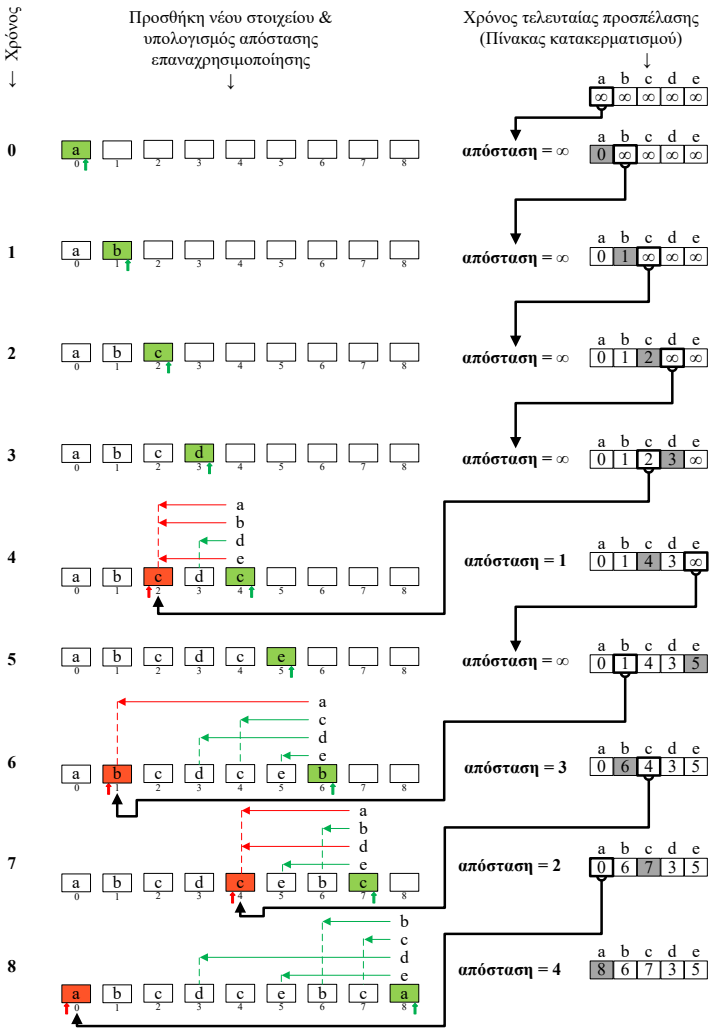
Στο σχήμα Β΄.6 παρουσιάζεται το παράδειγμα υπολογισμού των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση ισοζυγισμένου δυαδικού δέντρου AVL. Τα χοντρά μαύρα βέλη δείχνουν τη διάσχιση από τη ρίζα του δέντρου προς τον κόμβο που αναπαριστά την προσπέλαση που έγινε σε χρόνο t_{prev}. Οι χοντρές μαύρες γραμμές που εφάπτονται σε κάθε βέλος δείχνουν τα βάρη που προστίθενται για τον υπολογισμό της απόστασης επαναχρησιμοποίησης. Με κόκκινο περίγραμμα φαίνονται οι προς διαγραφή κόμβοι ενώ για τα βάρη

που αλλάζουν τιμές, οι παλιές τιμές φαίνονται διαγραμμένες και με κόκκινο χρώμα.

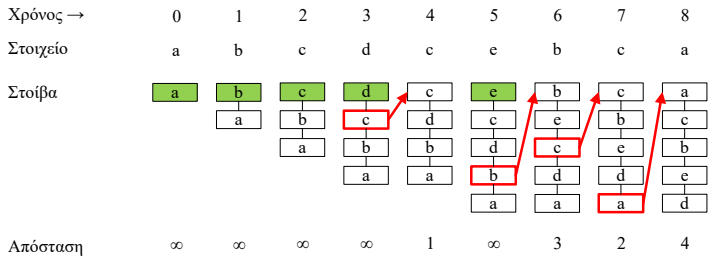
Στο σχήμα Β΄.7 παρουσιάζεται το παράδειγμα υπολογισμού των αποστάσεων επαναχρησιμοποίησης δεδομένων με χρήση δέντρου τρυπών. Από τα τέσσερα βήματα/χρόνους που φαίνονται στο σχήμα, το πρώτο (2500) εμπίπτει στην πρώτη περίπτωση του αλγόριθμου όπου η προς εισαγωγή τρύπα είναι γειτονική σε μία μόνο κενή περιοχή. Σε αυτή την περίπτωση απλά προστίθεται η τρύπα 101 στον κατάλληλο κόμβο. Το δεύτερο (2501) και τρίτο (2502) βήμα εμπίπτουν στη δεύτερη περίπτωση του αλγόριθμου όπου η προς εισαγωγή τρύπα είναι γειτονική σε δύο κενές περιοχές. Σε αυτή την περίπτωση γίνεται διαγραφή του ενός από τους κόμβους που αναπαριστούν αυτές τις δύο περιοχές. Οι κόμβοι που διαγράφονται εμφανίζονται με κόκκινο περίγραμμα. Το τέταρτο βήμα (2503) εμπίπτει στην τρίτη περίπτωση του αλγόριθμου όπου η προς εισαγωγή τρύπα δεν γειτνιάζει με καμία κενή περιοχή. Σε αυτή την περίπτωση πρέπει να γίνει εισαγωγή νέου κόμβου στο δέντρο. Ο κόμβος αυτός αναπαρίσταται με πράσινο χρώμα.



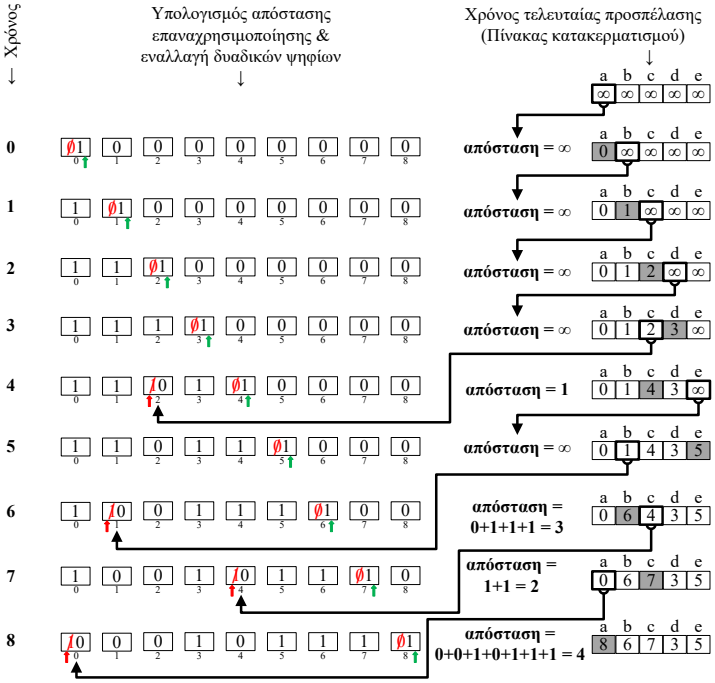
Σχήμα Β΄.1: Παράδειγμα ροής προσπελάσεων μνήμης.



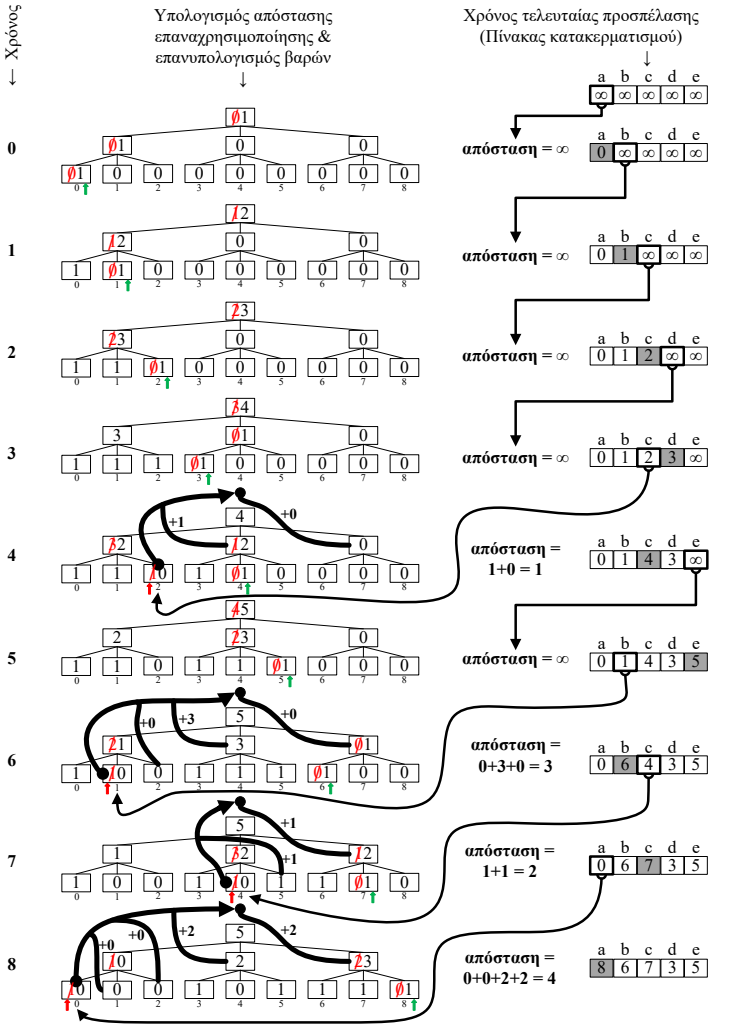
Σχήμα Β΄.2: Παράδειγμα υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων με αφελή αλγόριθμο.



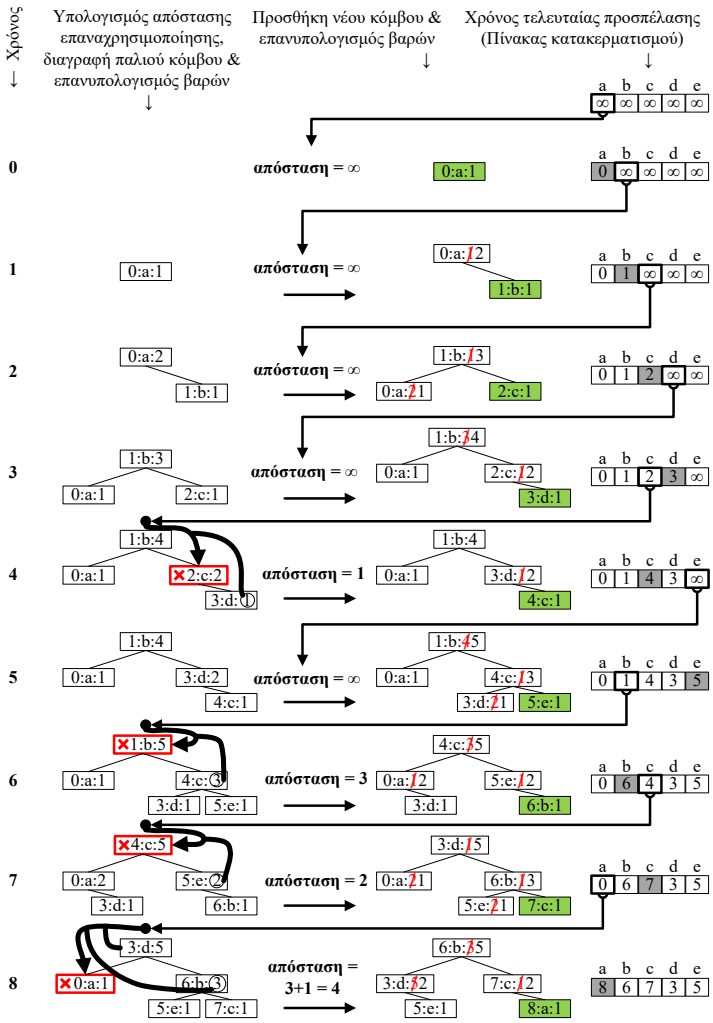
Σχήμα Β΄.3: Παράδειγμα υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων με χρήση στοίβας.



Σχήμα Β΄.4: Παράδειγμα υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων με χρήση πίνακα δυαδικών ψηφίων.

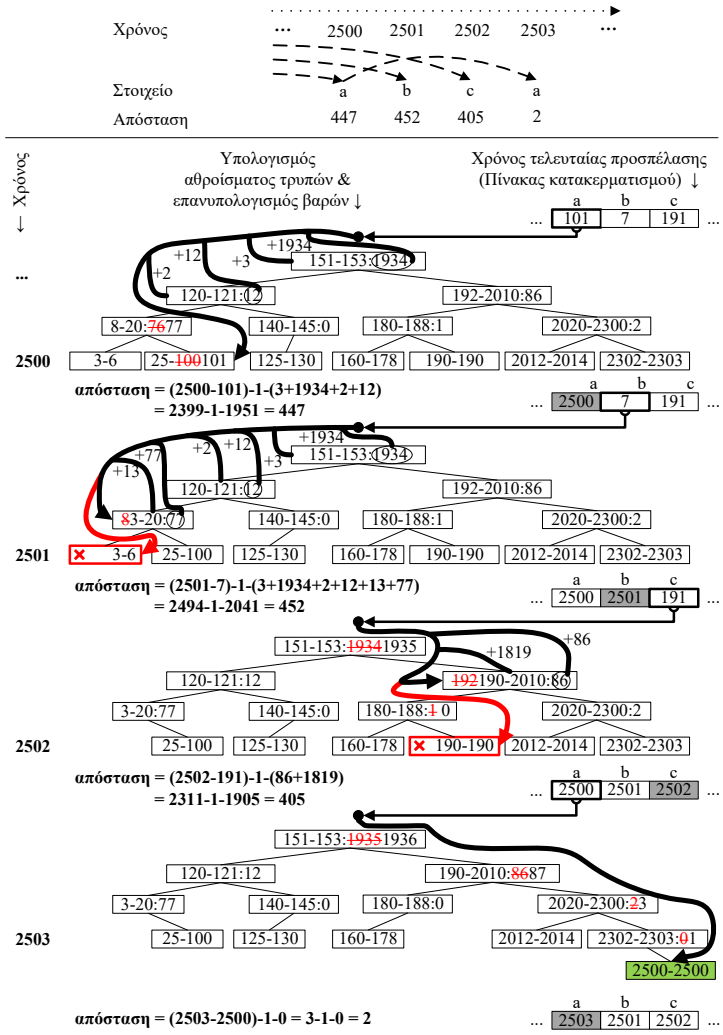


Σχήμα Β΄.5: Παράδειγμα υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων με χρήση πίνακα δυαδικών ψηφίων / στατικά δεσμευμένου m-αδικού δέντρου.



Σχήμα Β΄.6: Παράδειγμα υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων με χρήση ισοζυγισμένου δυαδικού δέντρου AVL.

Β΄. Παραδείγματα υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων



Σχήμα Β΄.7: Παράδειγμα υπολογισμού της απόστασης επαναχρησιμοποίησης δεδομένων με χρήση ισοζυγισμένου δυαδικού δέντρου τρυπών.

Παράρτημα Γ΄

Μετρήσεις

Τα σχήματα με τα πειραματικά αποτελέσματα τα οποία παρουσιάζονται στην ενότητα 6.1, δείχνουν μέσους όρους, μέγιστες και ελάχιστες τιμές. Σε αυτό το παράρτημα παρατίθενται αναλυτικά οι μετρήσεις που χρησιμοποιήθηκαν για αυτά τα σχήματα. Πιο συγκεκριμένα:

- από το σχήμα Γ΄.1 έως το σχήμα Γ΄.33 φαίνονται οι μετρήσεις σχετικά με τις βελτιώσεις των χρόνων εκτέλεσης έχοντας εφαρμόσει βελτιστοποιήσεις που αφορούν την τοπικότητα αναφοράς δεδομένων.

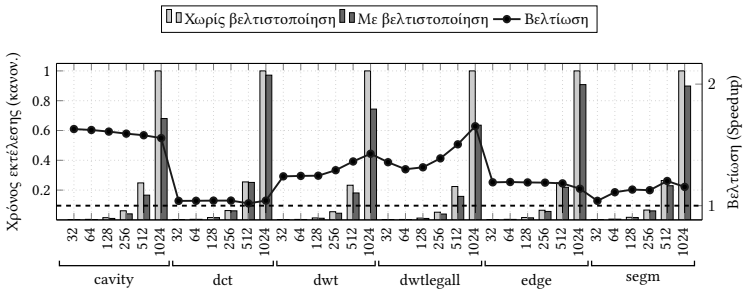
- από το σχήμα Γ΄.34 έως το σχήμα Γ΄.50 φαίνονται οι μετρήσεις σχετικά με τις βελτιώσεις των χρόνων εκτέλεσης έχοντας εφαρμόσει βελτιστοποιήσεις που αφορούν την μείωση των περιττών επανυπολογισμών.

- από το σχήμα Γ΄.51 έως το σχήμα Γ΄.74 φαίνονται οι μετρήσεις που πραγματοποιήθηκαν στον προσομοιωτή Cachegrind έχοντας εφαρμόσει βελτιστοποιήσεις που αφορούν την τοπικότητα αναφοράς δεδομένων.

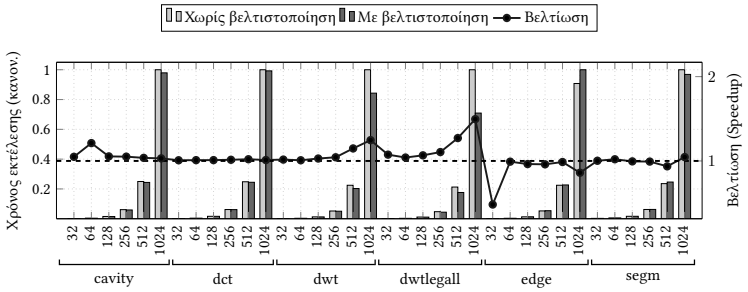
- από το σχήμα Γ΄.75 έως το σχήμα Γ΄.86 φαίνονται οι μετρήσεις που πραγματοποιήθηκαν στον προσομοιωτή Cachegrind έχο-

ντας εφαρμόσει βελτιστοποιήσεις που αφορούν την μείωση των περιττών επανυπολογισμών.

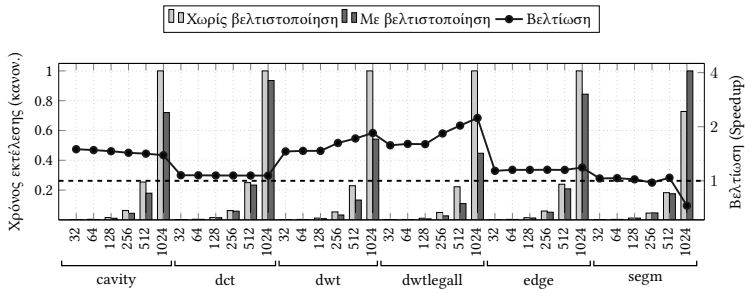
Σε όλες τις τιμές των παραπάνω σχημάτων που αφορούν χρόνους εκτέλεσης, αριθμό προσπελάσεων μνήμης και αριθμό αστοχιών λανθάνουσας μνήμης έχει πραγματοποιηθεί κανονικοποίηση στο εύρος [0-1]. Σε όλα τα σχήματα αυτού του παραρτήματος η εφαρμογή βελτιστοποιήσεων που αφορούν την τοπικότητα αναφοράς δεδομένων αναφέρεται ως *trans_locality*, ενώ η εφαρμογή βελτιστοποιήσεων που αφορούν την μείωση των περιττών επανυπολογισμών αναφέρεται ως *trans_recomputations*.



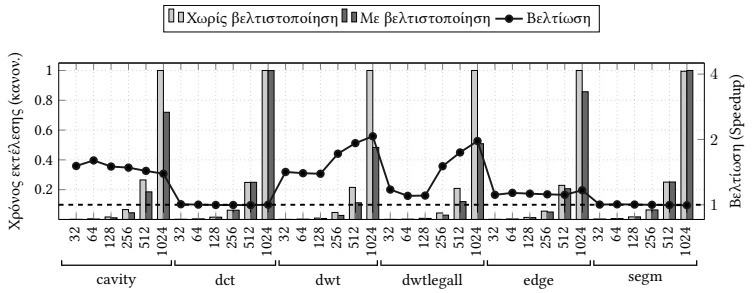
Σχήμα Γ΄.1: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *a53* / *gcc* / *noopt*.



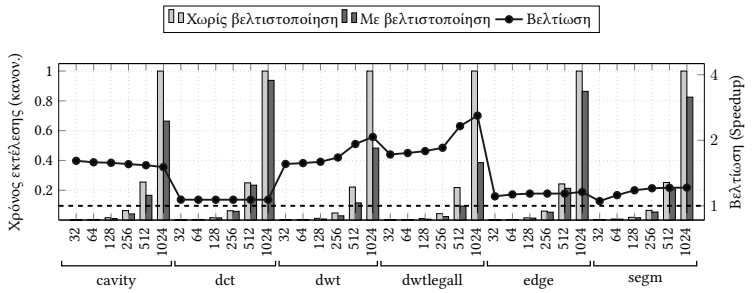
Σχήμα Γ΄.2: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *a53* / *gcc* / *o3*.



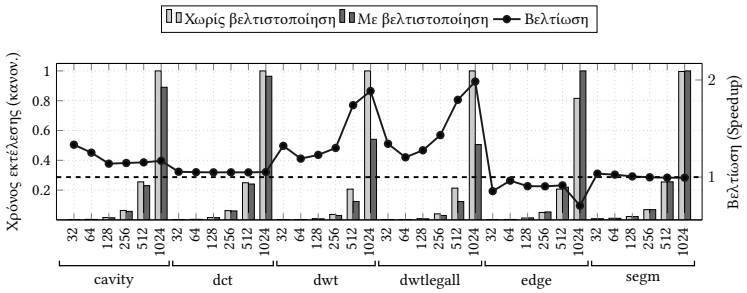
Σχήμα Γ΄.3: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *i5* / *clang* / *noopt*.



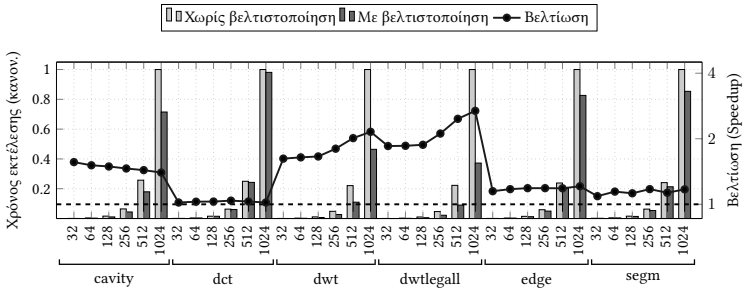
Σχήμα Γ΄.4: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *i5* / *clang* / *o3*.



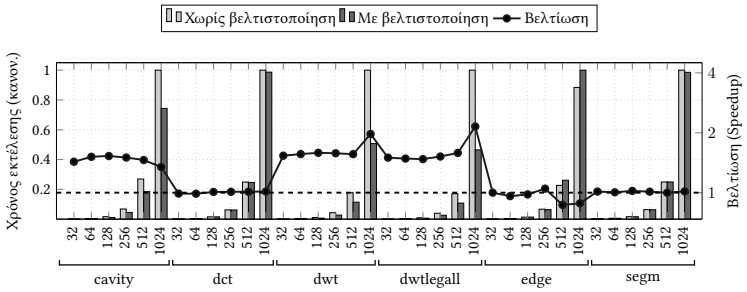
Σχήμα Γ΄.5: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *i5* / *gcc* / *noopt*.



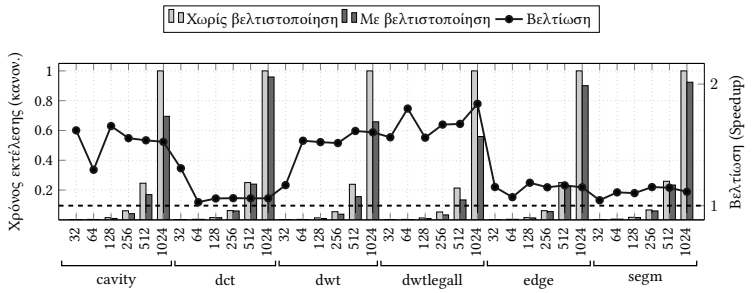
Σχήμα Γ΄.6: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *i5* / *gcc* / *o3*.



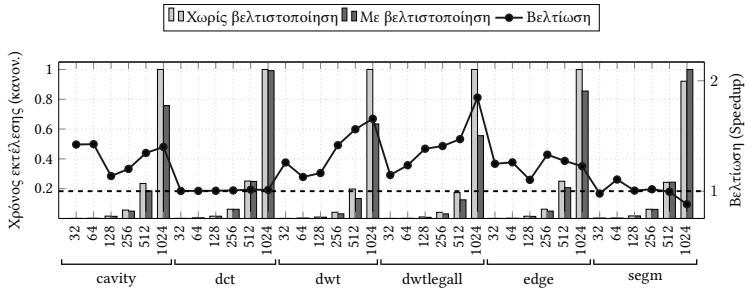
Σχήμα Γ΄.7: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *i5* / *msvc* / *noopt*.



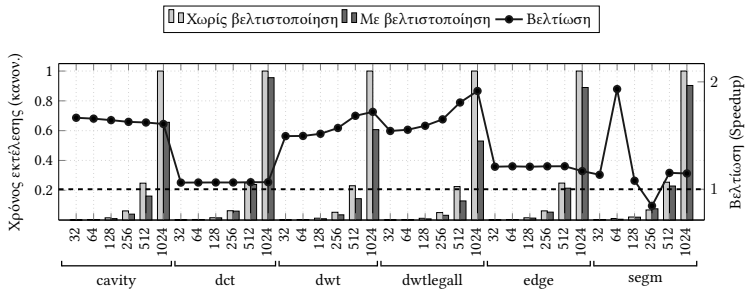
Σχήμα Γ΄.8: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *i5* / *msvc* / *o3*.



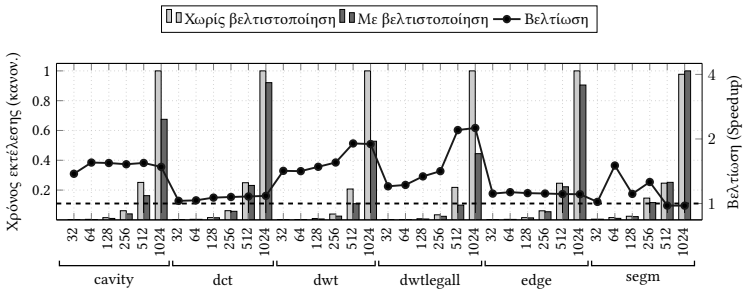
Σχήμα Γ΄.9: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *mips* / *gcc* / *noopt*.



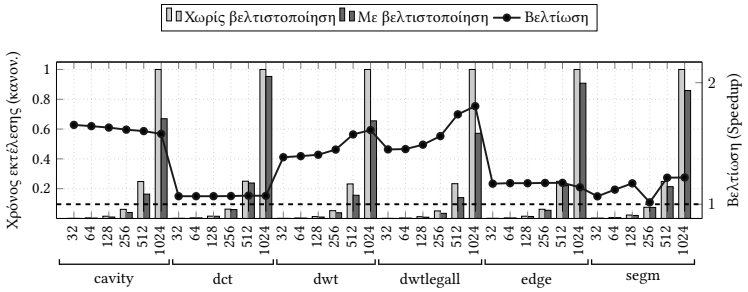
Σχήμα Γ΄.10: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *mips* / *gcc* / *o3*.



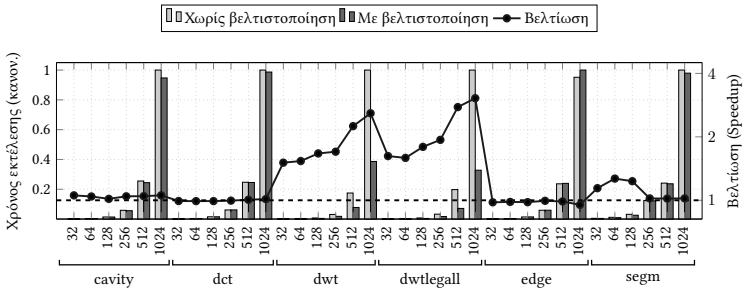
Σχήμα Γ΄.11: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *pi3* / *clang* / *noopt*.



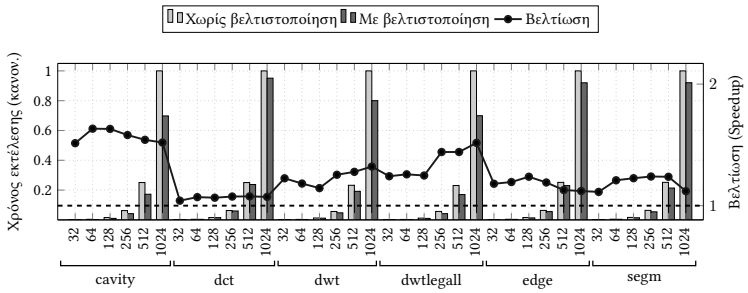
Σχήμα Γ΄.12: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *pi3* / *clang* / *o3*.



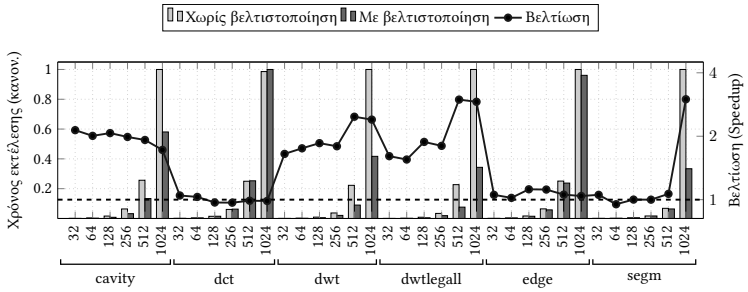
Σχήμα Γ΄.13: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *pi3* / *gcc* / *noopt*.



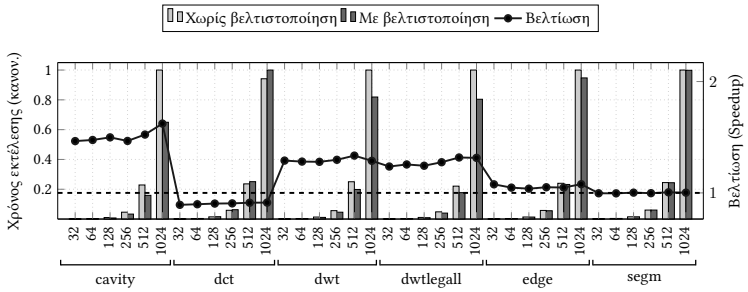
Σχήμα Γ΄.14: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *pi3* / *gcc* / *o3*.



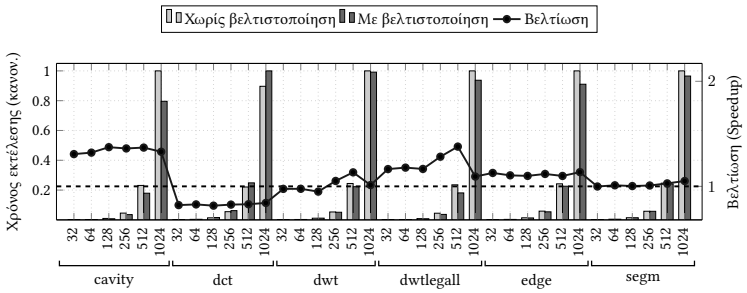
Σχήμα Γ′.15: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *pi3* / *msvc* / *noopt*.



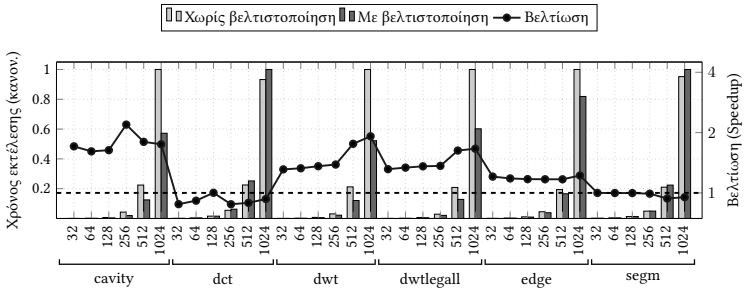
Σχήμα Γ′.16: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *C* / *pi3* / *msvc* / *o3*.



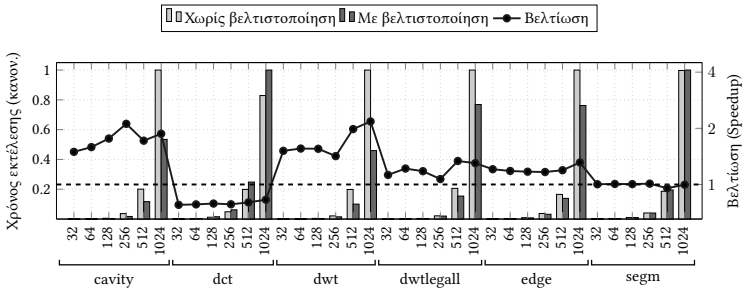
Σχήμα Γ′.17: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *a53* / *gcc* / *noopt*.



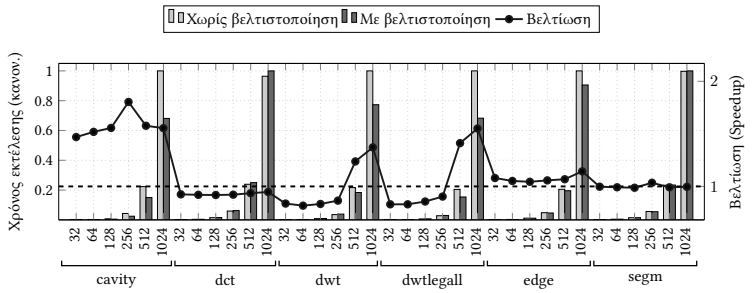
Σχήμα Γ΄.18: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *a53* / *gcc* / *o3*.



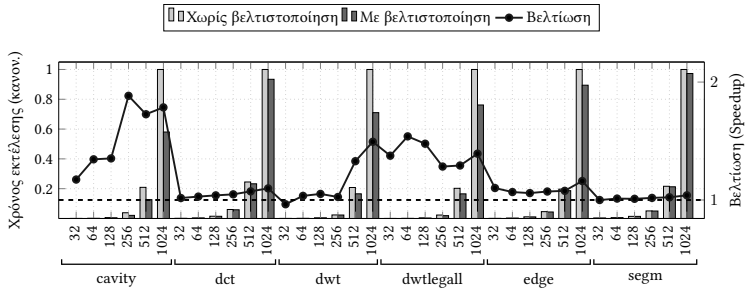
Σχήμα Γ΄.19: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *i5* / *clang* / *noopt*.



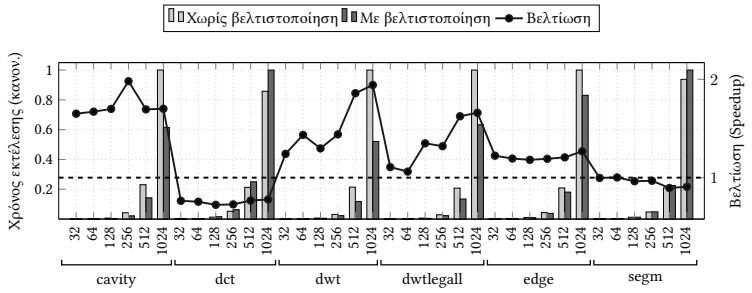
Σχήμα Γ΄.20: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *i5* / *clang* / *o3*.



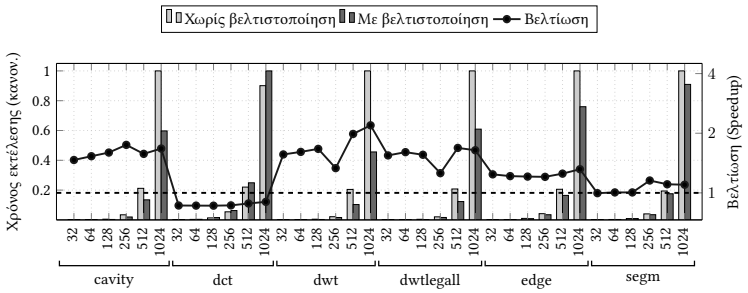
Σχήμα Γ΄.21: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *i5* / *gcc* / *noopt*.



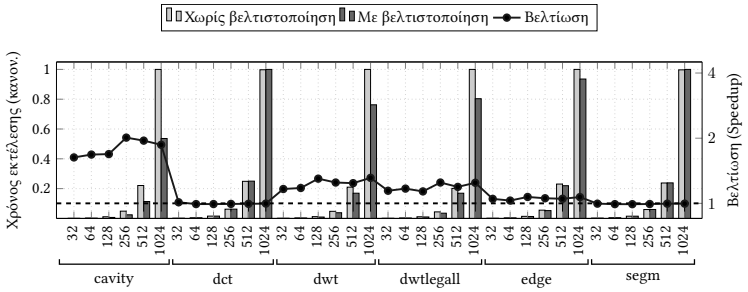
Σχήμα Γ΄.22: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *i5* / *gcc* / *o3*.



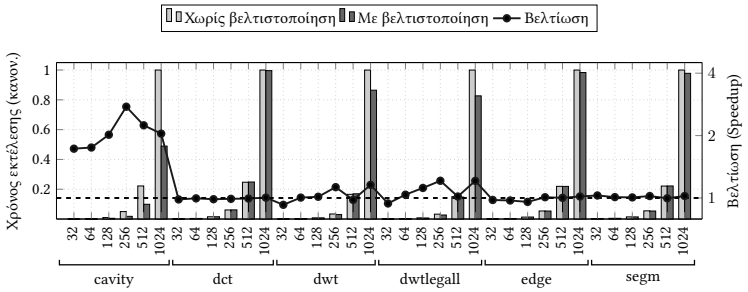
Σχήμα Γ΄.23: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *i5* / *msvc* / *noopt*.



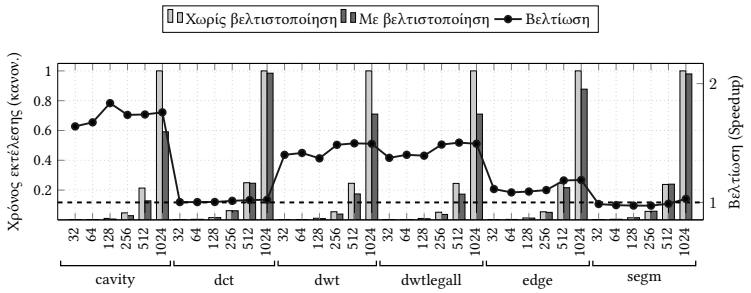
Σχήμα Γ΄.24: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *i5* / *msvc* / *o3*.



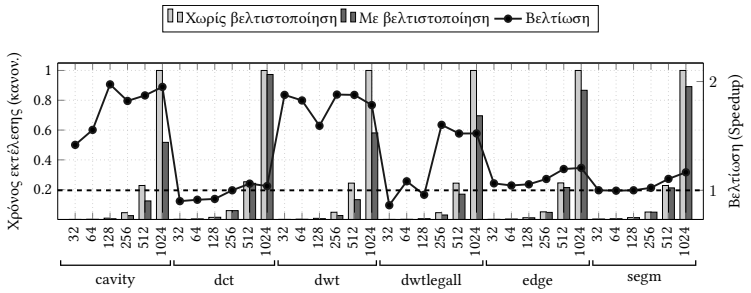
Σχήμα Γ΄.25: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *mips* / *gcc* / *noopt*.



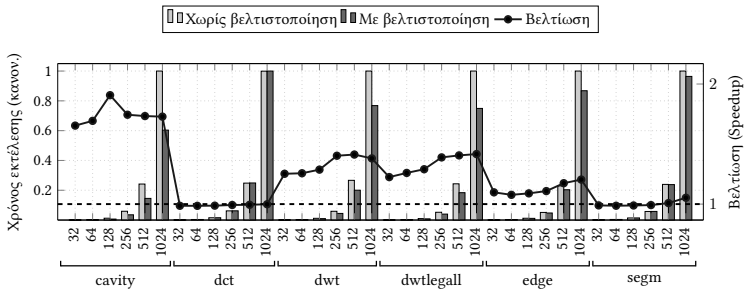
Σχήμα Γ΄.26: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *mips* / *gcc* / *o3*.



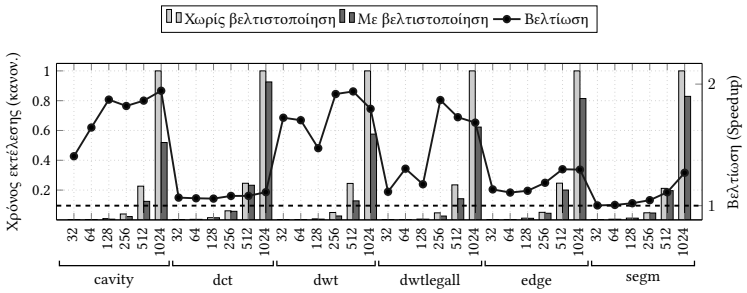
Σχήμα Γ΄.27: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *pi3* / *clang* / *noopt*.



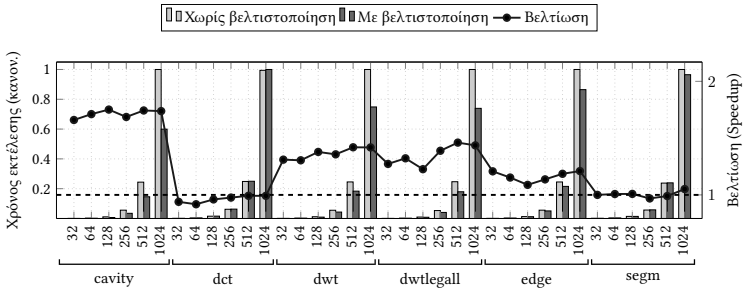
Σχήμα Γ΄.28: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *pi3* / *clang* / *o3*.



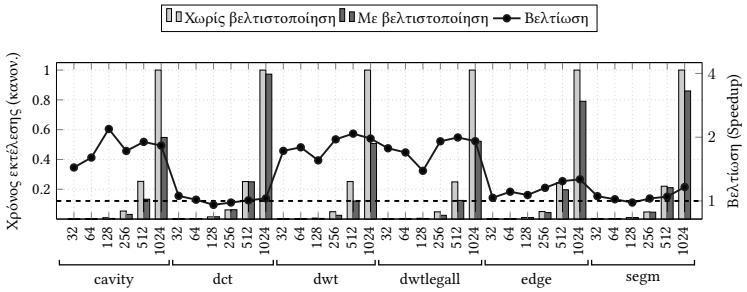
Σχήμα Γ΄.29: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *pi3* / *gcc* / *noopt*.



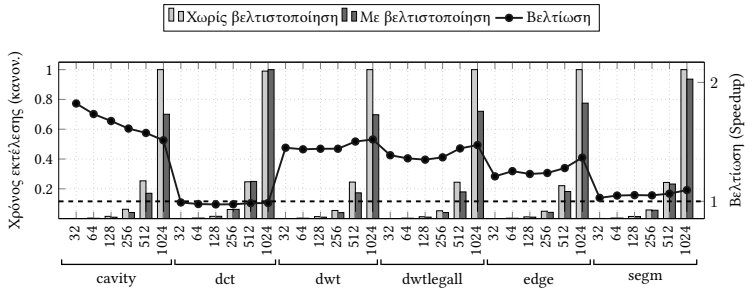
Σχήμα Γ΄.30: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *pi3* / *gcc* / *o3*.



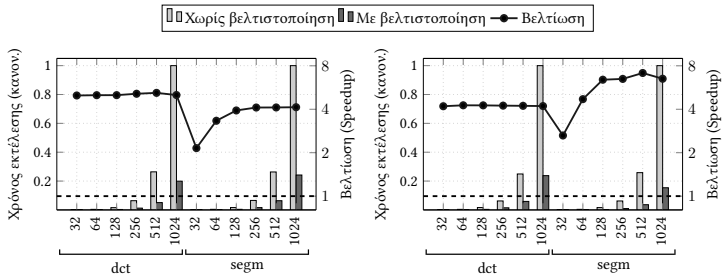
Σχήμα Γ΄.31: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *pi3* / *msvc* / *noopt*.



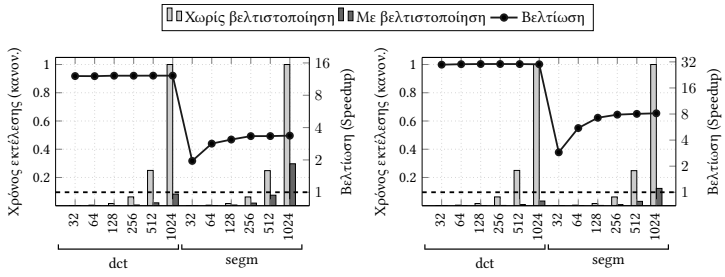
Σχήμα Γ΄.32: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *pi3* / *msvc* / *o3*.



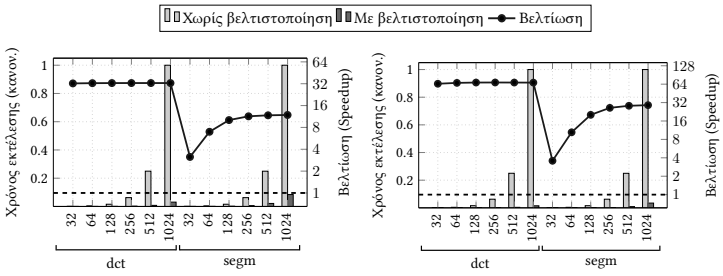
Σχήμα Γ΄.33: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_locality* / *MATLAB Interpreter* / *i5* / *interpreter* / *noopt*.



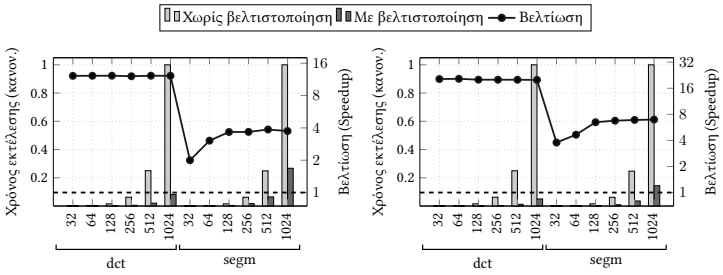
Σχήμα Γ΄.34: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *C* / *a53* / *gcc* / *noopt* (αριστερά) και *trans_recomputations* / *C* / *a53* / *gcc* / *o3* (δεξιά).



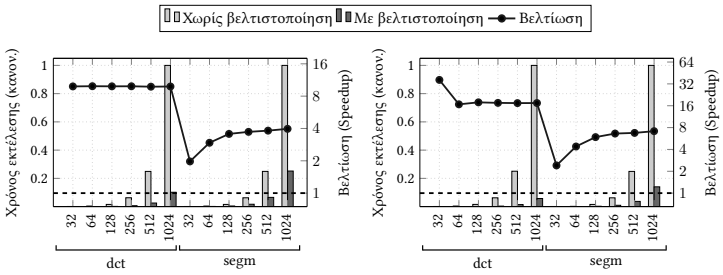
Σχήμα Γ΄.35: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *C* / *i5* / *clang* / *noopt* (αριστερά) και *trans_recomputations* / *C* / *i5* / *clang* / *o3* (δεξιά).



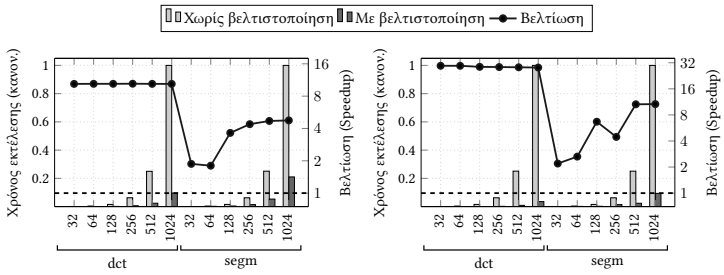
Σχήμα Γ΄.36: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *C* / *i5* / *gcc* / *noopt* (αριστερά) και *trans_recomputations* / *C* / *i5* / *gcc* / *o3* (δεξιά).



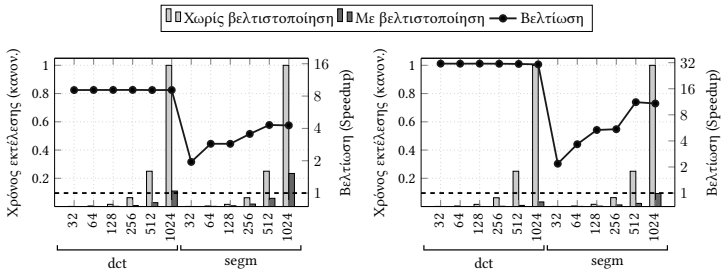
Σχήμα Γ΄.37: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *C* / *i5* / *msvc* / *noopt* (αριστερά) και *trans_recomputations* / *C* / *i5* / *msvc* / *o3* (δεξιά).



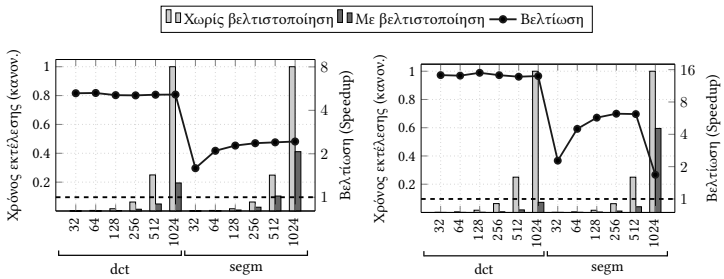
Σχήμα Γ΄.38: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *C* / *mips* / *gcc* / *noopt* (αριστερά) και *trans_recomputations* / *C* / *mips* / *gcc* / *o3* (δεξιά).



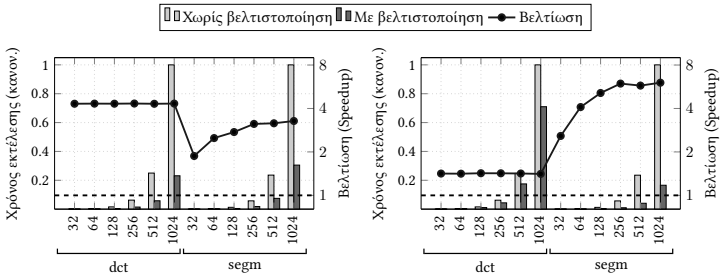
Σχήμα Γ΄.39: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *C* / *pi3* / *clang* / *noopt* (αριστερά) και *trans_recomputations* / *C* / *pi3* / *clang* / *o3* (δεξιά).



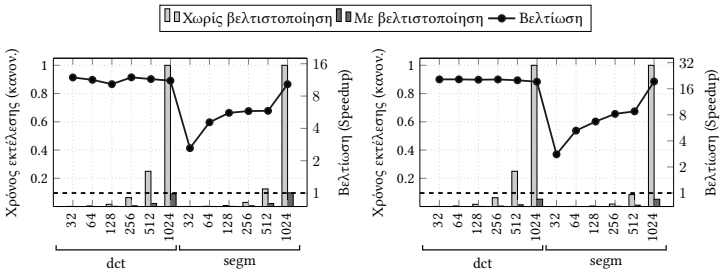
Σχήμα Γ΄.40: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *C* / *pi3* / *gcc* / *noopt* (αριστερά) και *trans_recomputations* / *C* / *pi3* / *gcc* / *o3* (δεξιά).



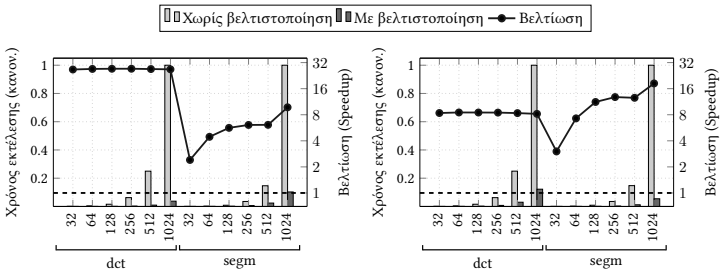
Σχήμα Γ΄.41: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *C* / *pi3* / *msvc* / *noopt* (αριστερά) και *trans_recomputations* / *C* / *pi3* / *msvc* / *o3* (δεξιά).



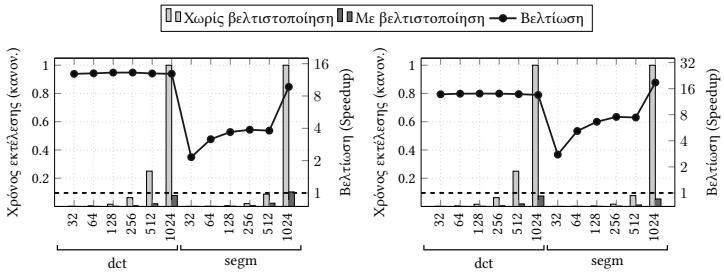
Σχήμα Γ΄.42: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *a53* / *gcc* / *noopt* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *a53* / *gcc* / *o3* (δεξιά).



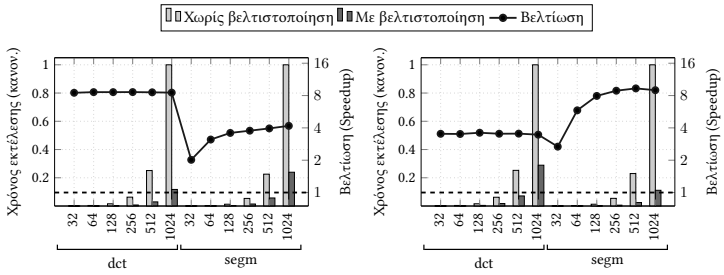
Σχήμα Γ΄.43: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *i5* / *clang* / *noopt* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *i5* / *clang* / *o3* (δεξιά).



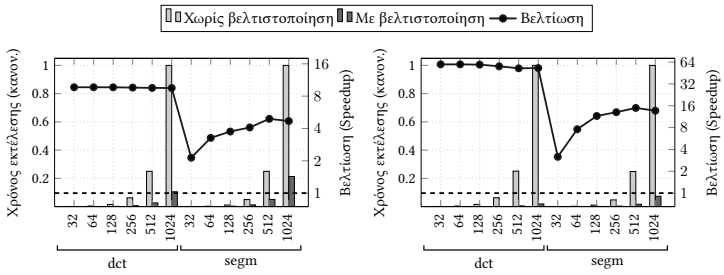
Σχήμα Γ΄.44: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *i5* / *gcc* / *noopt* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *i5* / *gcc* / *o3* (δεξιά).



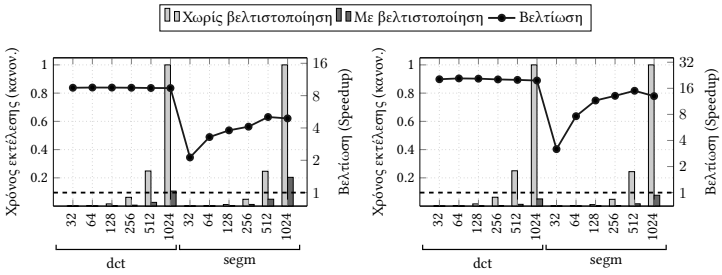
Σχήμα Γ΄.45: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *i5* / *msvc* / *noopt* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *i5* / *msvc* / *o3* (δεξιά).



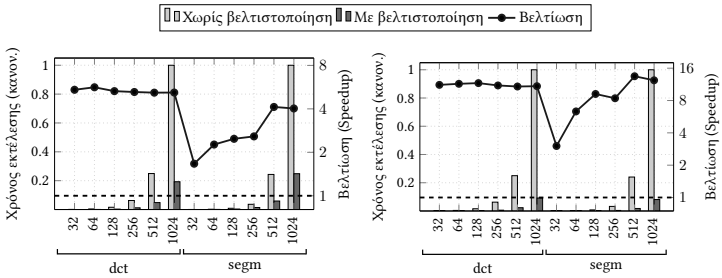
Σχήμα Γ΄.46: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *mips* / *gcc* / *noopt* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *mips* / *gcc* / *o3* (δεξιά).



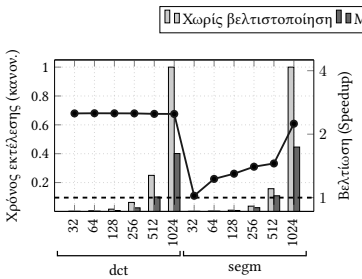
Σχήμα Γ΄.47: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *pi3* / *clang* / *noopt* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *pi3* / *clang* / *o3* (δεξιά).



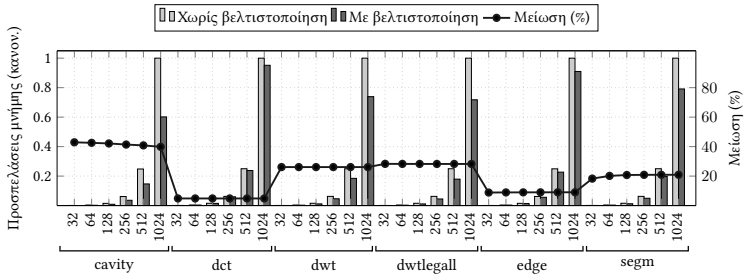
Σχήμα Γ΄.48: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *pi3* / *gcc* / *noopt* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *pi3* / *gcc* / *o3* (δεξιά).



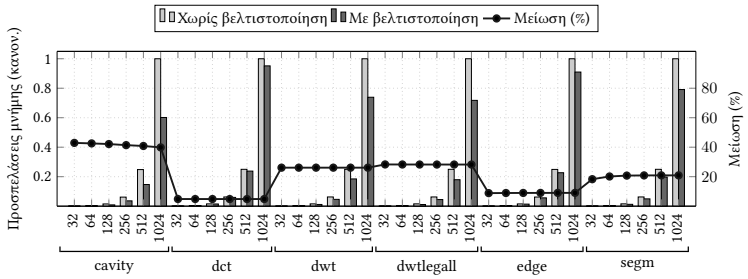
Σχήμα Γ΄.49: Βελτίωση χρόνου εκτέλεσης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *pi3* / *msvc* / *noopt* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *pi3* / *msvc* / *o3* (δεξιά).



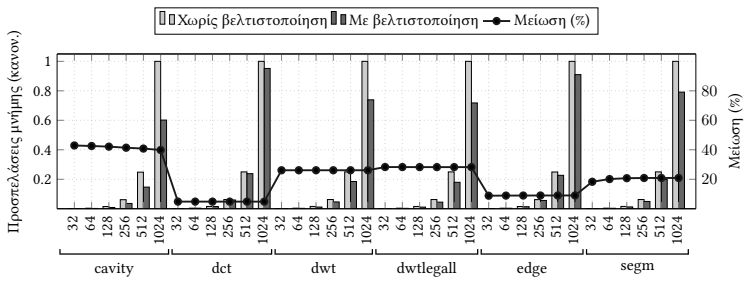
Σχήμα Γ΄.50: Βελτίωση χρόνου εκτέλεσης για τον συνδυασμό *trans_recomputations* / *MATLAB Interpreter* / *i5*.



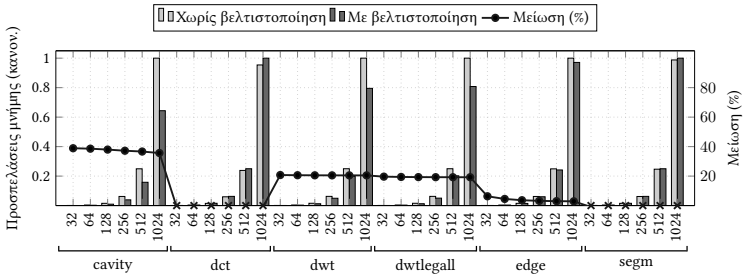
Σχήμα Γ′.51: Μείωση προσπελάσεων μνήμης για τον συνδυασμό *trans_locality* / *C* / *conf1*.



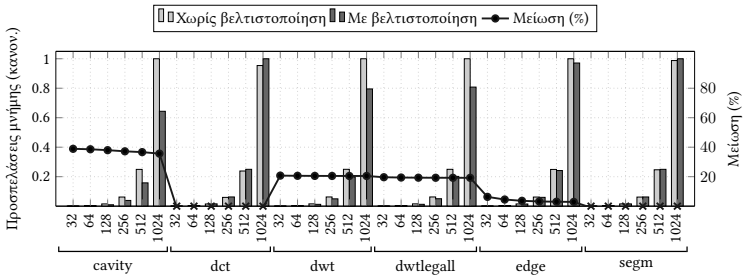
Σχήμα Γ′.52: Μείωση προσπελάσεων μνήμης για τον συνδυασμό *trans_locality* / *C* / *conf2*.



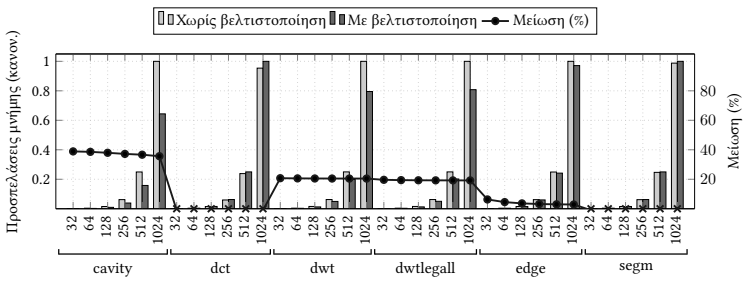
Σχήμα Γ′.53: Μείωση προσπελάσεων μνήμης για τον συνδυασμό *trans_locality* / *C* / *conf3*.



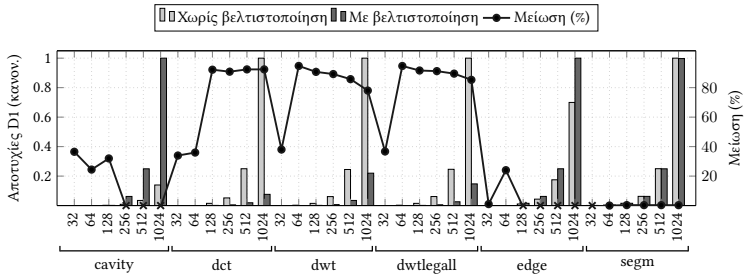
Σχήμα Γ΄.54: Μείωση προσπελάσεων μνήμης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf1*.



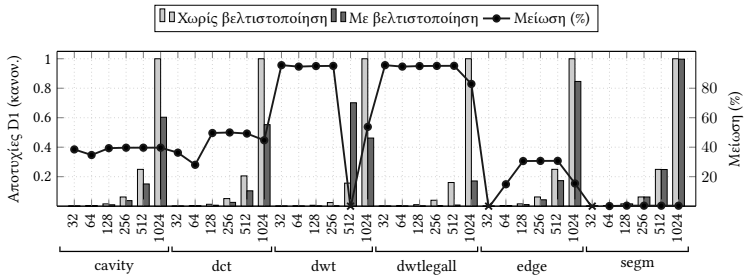
Σχήμα Γ΄.55: Μείωση προσπελάσεων μνήμης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf2*.



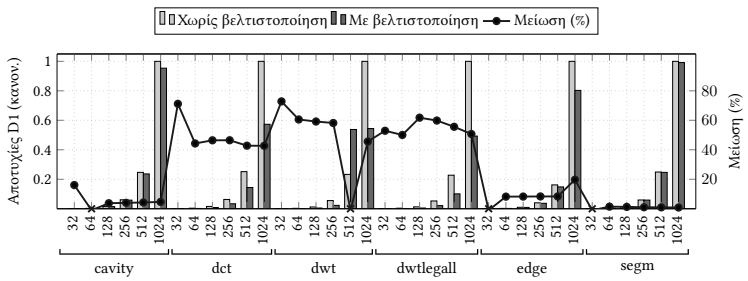
Σχήμα Γ΄.56: Μείωση προσπελάσεων μνήμης για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf3*.



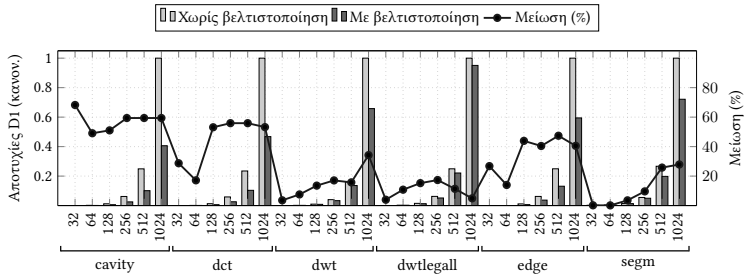
Σχήμα Γ΄.57: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τον συνδυασμό *trans_locality* / *C* / *conf1*.



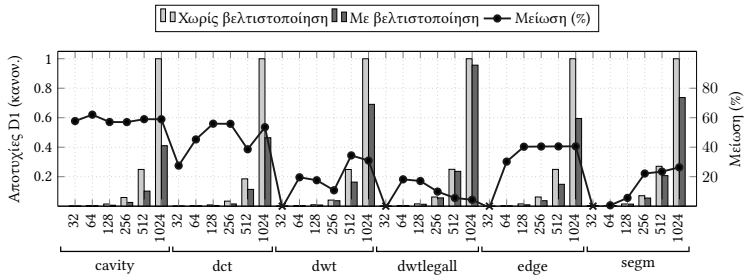
Σχήμα Γ΄.58: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τον συνδυασμό *trans_locality* / *C* / *conf2*.



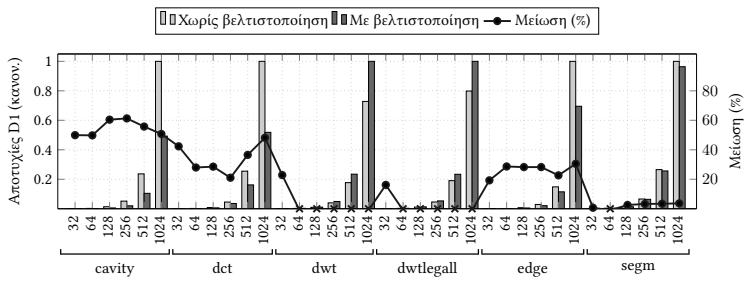
Σχήμα Γ΄.59: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τον συνδυασμό *trans_locality* / *C* / *conf3*.



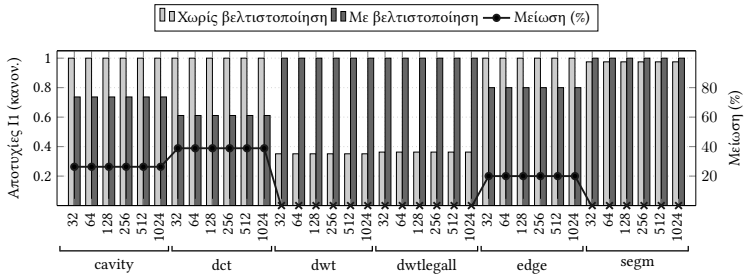
Σχήμα Γ΄.60: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf1*.



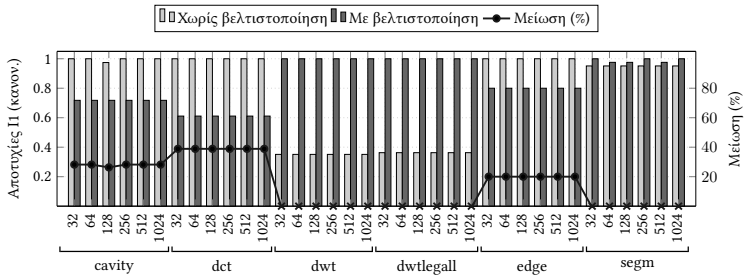
Σχήμα Γ΄.61: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf2*.



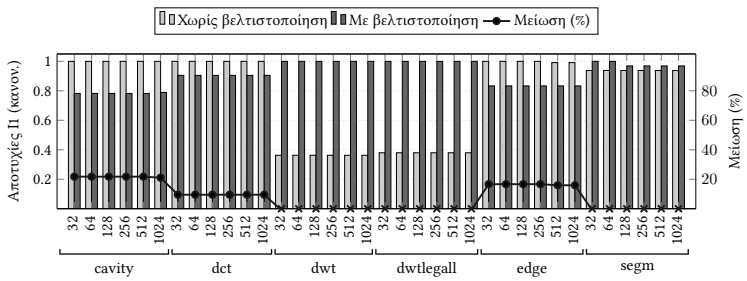
Σχήμα Γ΄.62: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf3*.



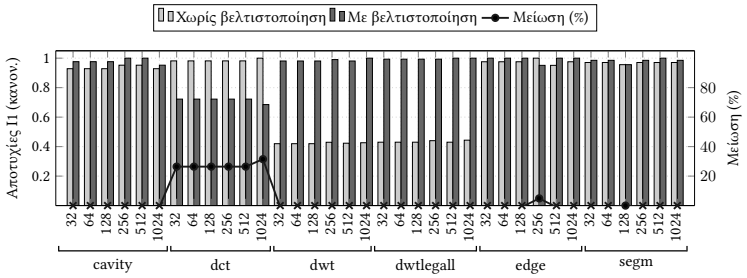
Σχήμα Γ΄.63: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τον συνδυασμό *trans_locality* / *C* / *conf1*.



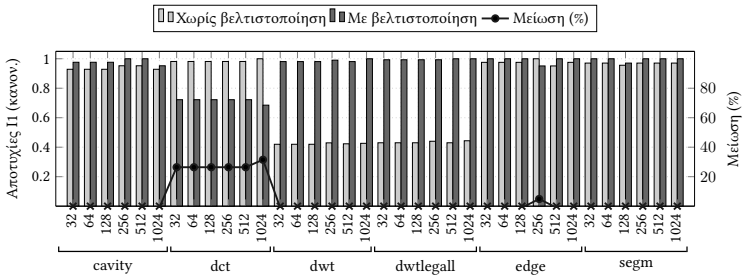
Σχήμα Γ΄.64: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τον συνδυασμό *trans_locality* / *C* / *conf2*.



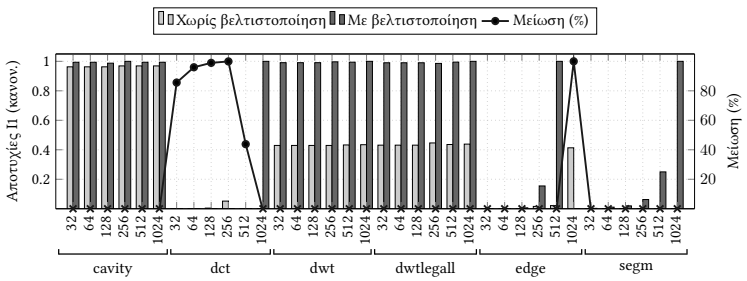
Σχήμα Γ΄.65: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τον συνδυασμό *trans_locality* / *C* / *conf3*.



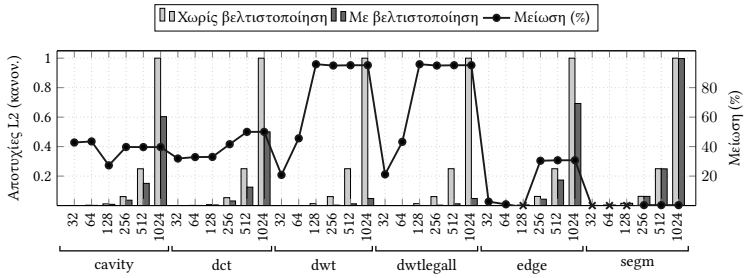
Σχήμα Γ΄.66: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf1*.



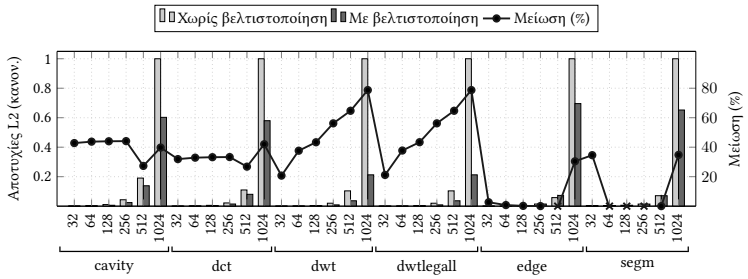
Σχήμα Γ΄.67: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf2*.



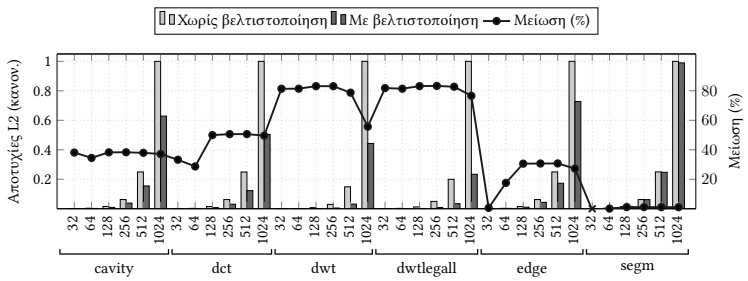
Σχήμα Γ΄.68: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf3*.



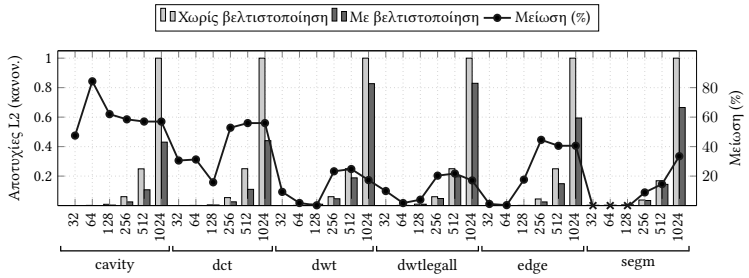
Σχήμα Γ΄.69: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τον συνδυασμό *trans_locality* / *C* / *conf1*.



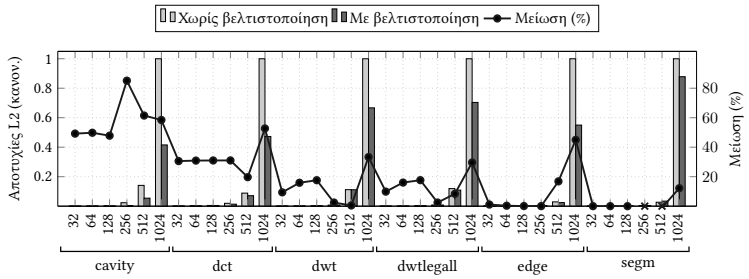
Σχήμα Γ΄.70: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τον συνδυασμό *trans_locality* / *C* / *conf2*.



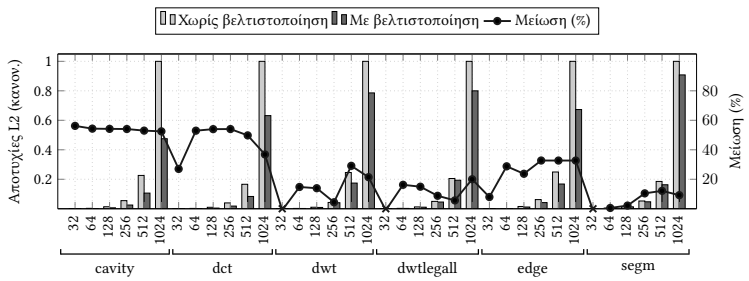
Σχήμα Γ΄.71: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τον συνδυασμό *trans_locality* / *C* / *conf3*.



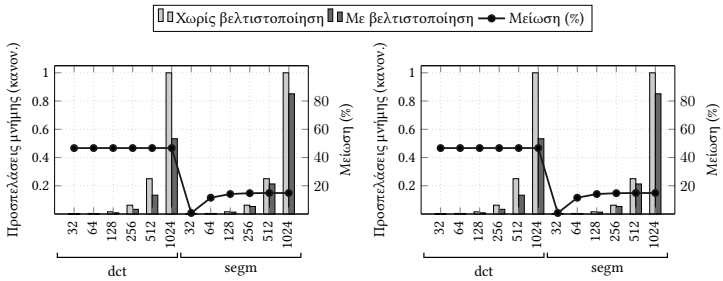
Σχήμα Γ΄.72: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf1*.



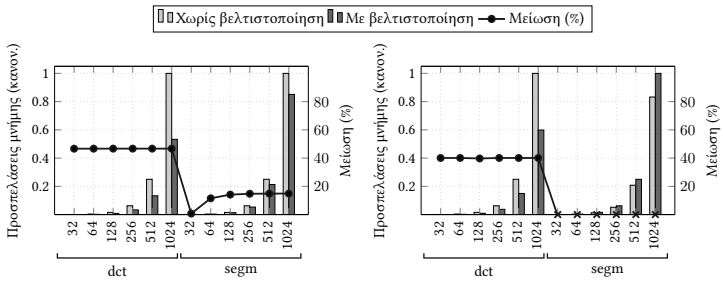
Σχήμα Γ΄.73: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf2*.



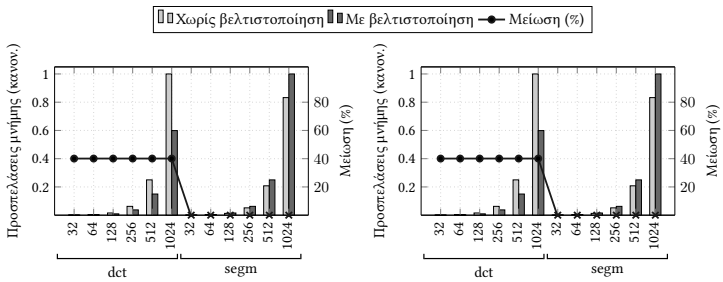
Σχήμα Γ΄.74: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τον συνδυασμό *trans_locality* / *MATLAB Coder* / *conf3*.



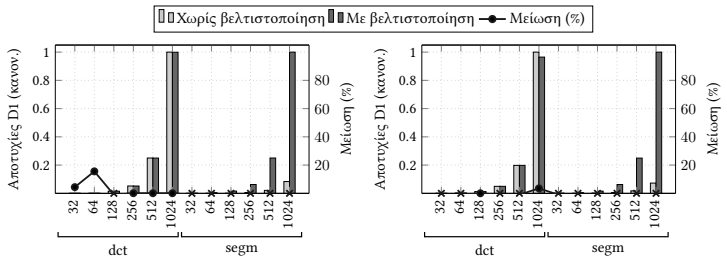
Σχήμα Γ΄.75: Μείωση προσπελάσεων μνήμης για τους συνδυασμούς: *trans_recomputations* / *C* / *conf1* (αριστερά) και *trans_recomputations* / *C* / *conf2* (δεξιά).



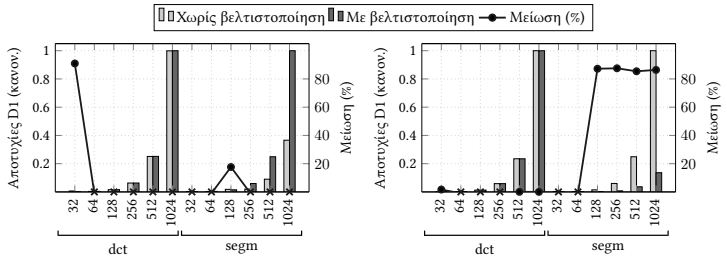
Σχήμα Γ΄.76: Μείωση προσπελάσεων μνήμης για τους συνδυασμούς: *trans_recomputations* / *C* / *conf3* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *conf1* (δεξιά).



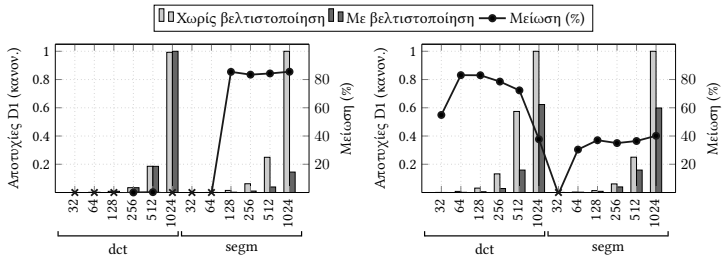
Σχήμα Γ΄.77: Μείωση προσπελάσεων μνήμης για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *conf2* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *conf3* (δεξιά).



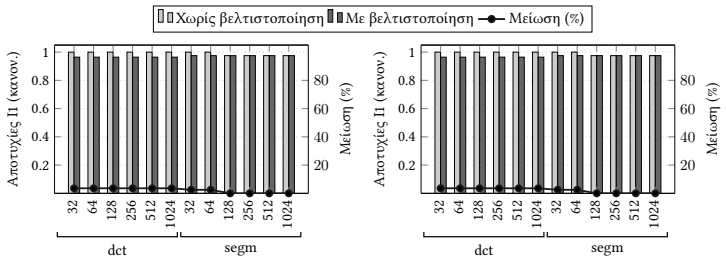
Σχήμα Γ΄.78: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τους συνδυασμούς: *trans_recomputations* / *C* / *conf1* (αριστερά) και *trans_recomputations* / *C* / *conf2* (δεξιά).



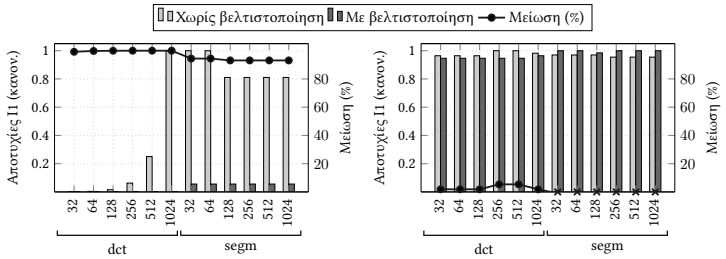
Σχήμα Γ΄.79: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τους συνδυασμούς: *trans_recomputations* / *C* / *conf3* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *conf1* (δεξιά).



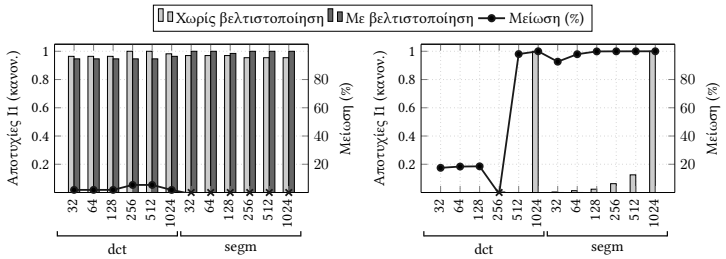
Σχήμα Γ΄.80: Μείωση αστοχιών λανθάνουσας μνήμης D1 για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *conf2* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *conf3* (δεξιά).



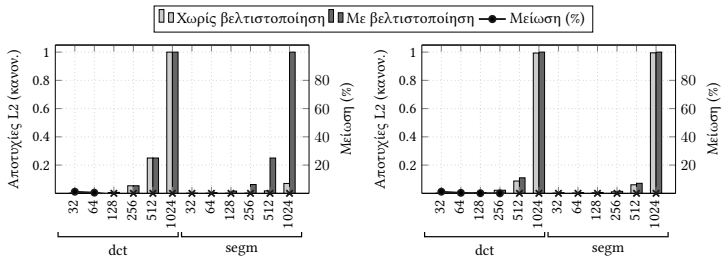
Σχήμα Γ΄.81: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τους συνδυασμούς: *trans_recomputations* / *C* / *conf1* (αριστερά) και *trans_recomputations* / *C* / *conf2* (δεξιά).



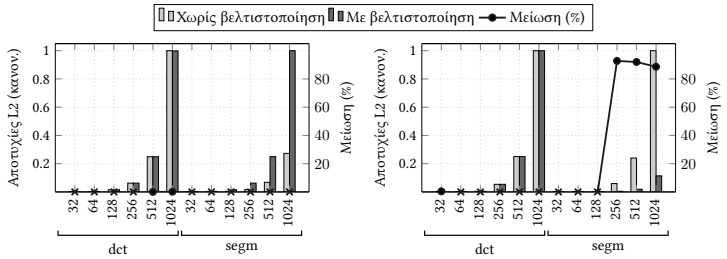
Σχήμα Γ΄.82: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τους συνδυασμούς: *trans_recomputations* / *C* / *conf3* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *conf1* (δεξιά).



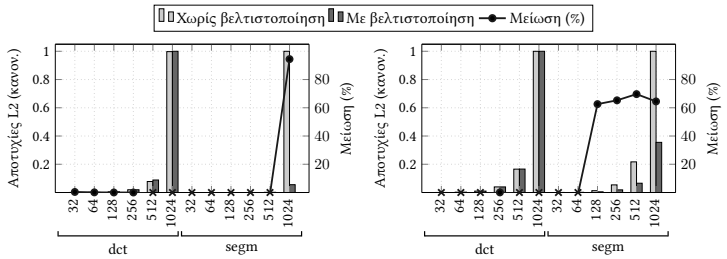
Σχήμα Γ΄.83: Μείωση αστοχιών λανθάνουσας μνήμης I1 για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *conf2* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *conf3* (δεξιά).



Σχήμα Γ΄.84: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τους συνδυασμούς: *trans_recomputations* / *C* / *conf1* (αριστερά) και *trans_recomputations* / *C* / *conf2* (δεξιά).



Σχήμα Γ΄.85: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τους συνδυασμούς: *trans_recomputations* / *C* / *conf3* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *conf1* (δεξιά).



Σχήμα Γ΄.86: Μείωση αστοχιών λανθάνουσας μνήμης L2 για τους συνδυασμούς: *trans_recomputations* / *MATLAB Coder* / *conf2* (αριστερά) και *trans_recomputations* / *MATLAB Coder* / *conf3* (δεξιά).

Παράρτημα Δ΄

Δημοσιεύσεις

Στις παρακάτω επιστημονικές δημοσιεύσεις παρουσιάζονται τμήματα της παρούσας διατριβής ή έχει πραγματοποιηθεί εργασία σχετική με τη διατριβή:

- **A MATLAB vectorizing compiler targeting Application Specific Instruction Set Processors**. Ioannis Latifis, Karthick Parashar, Grigoris Dimitroulakos, Hans Cappelle, Christakis Lezos, Konstantinos Masselos, Francky Catthoor. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Volume 22 Issue 2, March 2017.

- **Compiler-directed data locality optimization in MATLAB**. Christakis Lezos, Ioannis Latifis, Grigoris Dimitroulakos, Konstantinos Masselos. *Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*, Sankt Goar, Germany, May 23-25th, 2016.

- **Automatic generation of code analysis tools: The CastQL approach**. Christakis Lezos, Grigoris Dimitroulakos, Ioannis Latifis, Konstantinos Masselos. *Proceedings of the 1st International Workshop on Real World Domain Specific Languages (RWDSL)*, held in conjunction with the *2016 International Symposium on*

Code Generation and Optimization (CGO), Barcelona, Spain, March 12-18, 2016.

- **MAFE: An environment for MATLAB-to-C compilation supporting static and dynamic memory allocation and multi-level user interactive code optimization. [Poster abstract]** Christakis Lezos, Grigoris Dimitroulakos, Ioannis Latifis, Konstantinos Masselos. *Proceedings of the 2016 International Symposium on Code Generation and Optimization (CGO)*, Barcelona, Spain, March 12-18, 2016.

- **MATLAB-to-C compilation targeting Application Specific Instruction Set Processors**. Ioannis Latifis, Karthick Parashar, Grigoris Dimitroulakos, Hans Cappelle, Christakis Lezos, Konstantinos Masselos, Francky Catthoor. *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, March 14-18, 2016.

- **Reuse distance analysis for locality optimization in loop-dominated applications**. Christakis Lezos, Grigoris Dimitroulakos, Konstantinos Masselos. *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, March 9-13, 2015.

- **Dynamic source code analysis for memory hierarchy optimization in multimedia applications**. Christakis Lezos, Grigoris Dimitroulakos, Angeliki Freskou, Konstantinos Masselos. *Proceedings of the 2013 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Cagliari, Italy, October 8-10, 2013.

- **MEMSCOPT: A source-to-source compiler for dynamic code analysis and loop transformations**. Grigoris Dimitroulakos, Christakis Lezos, Konstantinos Masselos. *Proceedings of the 2012 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Karlsruhe, Germany, October 23-25, 2012.

- **XMSIM: A tool for early memory hierarchy evaluation**. Grigoris Dimitroulakos, Theodoros Lioris, Christakis Lezos, Konstantinos Masselos. *Proceedings of the 2012 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Karlsruhe, Germany, October 23-25, 2012.

Βιβλιογραφία

[1] Embedded Coder, . URL https://www.mathworks.com/products/embedded-coder.html.

[2] HDL Coder, . URL https://www.mathworks.com/products/hdl-coder.html.

[3] Intel Math Kernel Library (Intel MKL), . URL https://software.intel.com/en-us/mkl.

[4] LAPACK -Linear Algebra PACKage, . URL http://www.netlib.org/lapack/.

[5] MATLAB Coder, . URL https://www.mathworks.com/products/matlab-coder.html.

[6] OpenBLAS : An optimized BLAS library, . URL http://www.openblas.net/.

[7] Simulink Coder, . URL https://www.mathworks.com/products/simulink-coder.html.

[8] ThreadSpotter - ParaTools, Inc., . URL http://www.paratools.com/threadspotter.

[9] R. Allen. Compiling high-level languages to DSPs: automating the implementation path. *IEEE Signal Processing Magazine*, 22(3):47–56, May 2005. ISSN 1053-5888. doi: 10.1109/MSP.2005.1425897.

[10] George Almási, Călin Caşcaval, and David A. Padua. Calculating Stack Distances Efficiently. In *Proceedings of the 2002 Workshop on Memory System Performance*, MSP '02, pages 37–43, New York, NY, USA, 2002. ACM. doi: 10.1145/773146.773043. URL http://doi.acm.org/10.1145/773146.773043.

[11] T.L. Alves, J. Hage, and P. Rademaker. A Comparative Study of Code Query Technologies. In *2011 11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 145–154, September 2011. doi: 10.1109/SCAM.2011.14.

[12] Paul Anderson and Tim Teitelbaum. Software inspection using CodeSurfer. In *Proceedings of the 1st Workshop on Inspection in Software*

Engineering (WISE), Paris, France, July 2001.

[13] G. Antoniol, M. Di Penta, and E. Merlo. YAAB (Yet another AST browser): using OCL to navigate ASTs. In *11th IEEE International Workshop on Program Comprehension, 2003*, pages 13–22, May 2003. doi: 10.1109/WPC.2003.1199185.

[14] A. Arusoaie and D.I. Vicol. Automating Abstract Syntax Tree Construction for Context Free Grammars. In *2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 152–159, September 2012. doi: 10.1109/SYNASC.2012.8.

[15] Toheed Aslam, Jesse Doherty, Anton Dubrau, and Laurie Hendren. AspectMatlab: An Aspect-oriented Scientific Programming Language. In *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, AOSD '10, pages 181–192, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-958-9. doi: 10.1145/1739230.1739252. URL http://doi.acm.org/10.1145/1739230.1739252.

[16] P. Banerjee. An Overview of a Compiler for Mapping MATLAB Programs Onto FPGAs. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ASP-DAC '03, pages 477–482, New York, NY, USA, 2003. ACM. ISBN 978-0-7803-7660-1. doi: 10.1145/1119772.1119870. URL http://doi.acm.org/10.1145/1119772.1119870.

[17] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Chang, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, and M. Walkden. MATCH: A MATLAB Compiler for Configurable Computing Systems. Technical report, 1999.

[18] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden, and D. Zaretsky. A MATLAB compiler for distributed, heterogeneous, reconfigurable computing systems. In *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No.PR00871)*, pages 39–48, 2000. doi: 10.1109/FPGA.2000.903391.

[19] P. Banerjee, M. Haldar, A. Nayak, V. Kim, V. Saxena, S. Parkes, D. Bagchi, S. Pal, N. Tripathi, D. Zaretsky, R. Anderson, and J. R. Uribe. Overview of a compiler for synthesizing MATLAB programs onto FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(3):312–324, March 2004. ISSN 1063-8210. doi: 10.1109/TVLSI.2004.824301.

[20] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting Distinct Elements in a Data Stream. In José D. P. Rolim and Salil Vadhan, editors, *Randomization and Approximation Techniques in Computer Science*, number 2483 in Lecture Notes in Computer Science, pages 1–10. Springer Berlin Heidelberg, January 2002. ISBN 978-3-540-44147-2 978-3-540-45726-8. URL http://link.springer.com/chapter/10.1007/3-540-45726-7_1.

[21] Shaon Barman. *Aster: Automatic abstract syntax*. PhD thesis, University of Texas at Austin, May 2009. URL http://repositories.lib.utexas.edu/handle/2152/13381.

[22] Ira D. Baxter, Christopher Pidgeon, and Michael Mehlich. DMS®: Program Transformations for Practical Scalable Software Evolution. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 625–634, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 978-0-7695-2163-3. URL http://dl.acm.org/citation.cfm?id=998675.999466.

[23] M. Benincasa, R. Besler, D. Brassaw, and R. L. Kohler. Rapid development of real-time systems using RTExpressTM. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 594–599, March 1998. doi: 10.1109/IPPS.1998.669986.

[24] B. T. Bennett and V.J. Kruskal. LRU Stack Processing. *IBM Journal of Research and Development*, 19(4):353–357, July 1975. ISSN 0018-8646. doi: 10.1147/rd.194.0353.

[25] E. Berg and E. Hagersten. StatCache: a probabilistic approach to efficient and accurate data locality analysis. In *Performance Analysis of Systems and Software, 2004 IEEE International Symposium on - ISPASS*, pages 20–27, 2004. doi: 10.1109/ISPASS.2004.1291352.

[26] Erik Berg and Erik Hagersten. Fast Data-locality Profiling of Native Execution. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '05, pages 169–180, New York, NY, USA, 2005. ACM. ISBN 1-59593-022-1. doi: 10.1145/1064212.1064232. URL http://doi.acm.org/10.1145/1064212.1064232.

[27] Dirk Beyer. Relational Programming with CrocoPat. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 807–810, New York, NY, USA, 2006. ACM. ISBN 1-59593-375-1. doi: 10.1145/1134285.1134420. URL http://doi.acm.org/10.1145/1134285.1134420.

[28] Kristof Beyls and Erik D'Hollander. Reuse Distance as a Metric for Cache Behavior. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 617–622, Anaheim, California, USA, 2001. URL http://hdl.handle.net/1854/LU-144051.

[29] Kristof Beyls and Erik H. D'Hollander. Discovery of Locality-Improving Refactorings by Reuse Path Analysis. In Michael Gerndt and Dieter Kranzlmüller, editors, *High Performance Computing and Communications*, number 4208 in Lecture Notes in Computer Science, pages 220–229. Springer Berlin Heidelberg, January 2006. ISBN 978-3-540-39368-

9 978-3-540-39372-6. URL http://link.springer.com/chapter/10.1007/11847366_23.

[30] Kristof Beyls and Erik H. D'Hollander. Intermediately Executed Code is the Key to Find Refactorings That Improve Temporal Data Locality. In *Proceedings of the 3rd Conference on Computing Frontiers*, CF '06, pages 373–382, New York, NY, USA, 2006. ACM. ISBN 1-59593-302-6. doi: 10.1145/1128022.1128071. URL http://doi.acm.org/10.1145/1128022.1128071.

[31] Kristof Beyls and Erik H. D'Hollander. Refactoring Intermediately Executed Code to Reduce Cache Capacity Misses. *The Journal of Instruction-Level Parallelism*, 10, June 2008.

[32] Kristof Beyls and Erik H. D'Hollander. Refactoring for Data Locality. *Computer*, 42(2):62–71, February 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.57.

[33] Joao Bispo, Luis Reis, and Joao M. P. Cardoso. Multi-Target C Code Generation from MATLAB. In *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY'14, pages 95:95–95:100, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2937-8. doi: 10.1145/2627373.2627389. URL http://doi.acm.org/10.1145/2627373.2627389.

[34] João Bispo and João M. P. Cardoso. A MATLAB subset to C compiler targeting embedded systems. *Software: Practice and Experience*, 47(2):249–272, February 2017. ISSN 1097-024X. doi: 10.1002/spe.2408. URL http://onlinelibrary.wiley.com/doi/10.1002/spe.2408/abstract.

[35] João Bispo, Luís Reis, and João M. P. Cardoso. Techniques for Efficient MATLAB-to-C Compilation. In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY 2015, pages 7–12, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3584-3. doi: 10.1145/2774959.2774961. URL http://doi.acm.org/10.1145/2774959.2774961.

[36] Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Visser. Stratego/XT 0.17. A language and toolset for program transformation. *Science of Computer Programming*, 72(1–2):52–70, June 2008. ISSN 0167-6423. doi: 10.1016/j.scico.2007.11.003. URL http://www.sciencedirect.com/science/article/pii/S0167642308000452.

[37] Doug Burger and Todd M. Austin. The SimpleScalar Tool Set. URL http://www.simplescalar.com/.

[38] Martin Burtscher, Byoung-Do Kim, Jeff Diamond, John McCalpin, Lars Koesterke, and James Browne. PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Comput-*

ing, Networking, Storage and Analysis, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-1-4244-7559-9. doi: 10.1109/SC.2010.41. URL https://doi.org/10.1109/SC.2010.41.

[39] João M.P. Cardoso, Tiago Carvalho, José G.F. Coutinho, Wayne Luk, Ricardo Nobre, Pedro Diniz, and Zlatko Petrov. LARA: An Aspect-oriented Programming Language for Embedded Systems. In *Proceedings of the 11th Annual International Conference on Aspect-oriented Software Development*, AOSD '12, pages 179–190, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1092-5. doi: 10.1145/2162049.2162071. URL http://doi.acm.org/10.1145/2162049.2162071.

[40] Steve Carr, Kathryn S. McKinley, and Chau-Wen Tseng. Compiler Optimizations for Improving Data Locality. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VI, pages 252–262, New York, NY, USA, 1994. ACM. ISBN 0-89791-660-3. doi: 10.1145/195473.195557. URL http://doi.acm.org/10.1145/195473.195557.

[41] F. Catthoor, K. Danckaert, S. Wuytack, and N. D. Dutt. Code transformations for data transfer and storage exploration preprocessing in multimedia processors. *IEEE Design Test of Computers*, 18(3):70–82, May 2001. ISSN 0740-7475. doi: 10.1109/54.922804.

[42] Arun Chauhan and Chun-Yu Shei. Static Reuse Distances for Locality-based Optimizations in MATLAB. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, pages 295–304, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0018-6. doi: 10.1145/1810085.1810125. URL http://doi.acm.org/10.1145/1810085.1810125.

[43] Stéphane Chauveau and François Bodin. Menhir: An Environment for High Performance Matlab. *Scientific Programming*, 7(3-4):303–312, 1999. doi: 10.1155/1999/525690. URL https://www.hindawi.com/journals/sp/1999/525690/abs/.

[44] Aiyou Chen and Jin Cao. Distinct Counting with a Self-Learning Bitmap. In *IEEE 25th International Conference on Data Engineering, 2009. ICDE '09*, pages 1171–1174, March 2009. doi: 10.1109/ICDE.2009.193.

[45] Maxime Chevalier-Boisvert, Laurie Hendren, and Clark Verbrugge. Optimizing Matlab through Just-In-Time Specialization. In *Compiler Construction*, Lecture Notes in Computer Science, pages 46–65. Springer, Berlin, Heidelberg, March 2010. ISBN 978-3-642-11969-9 978-3-642-11970-5. doi: 10.1007/978-3-642-11970-5_4. URL https://link.springer.com/chapter/10.1007/978-3-642-11970-5_4.

[46] Tal Cohen, Joseph (Yossi) Gil, and Itay Maman. JTL: The Java Tools Language. In *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 89–108, New York, NY, USA, 2006. ACM. ISBN 1-

59593-348-4. doi: 10.1145/1167473.1167481. URL http://doi.acm.org/10.1145/1167473.1167481.

[47] E. S. Cordeiro, I. G. A. Stefani, T. C. A. P. Soares, and C. A. P. S. Martins. DCMSIM: didactic cache memory simulator. In *33rd Annual Frontiers in Education, 2003. FIE 2003.*, volume 2, pages F1C–14–19 Vol.2, November 2003. doi: 10.1109/FIE.2003.1264669.

[48] James R. Cordy. The TXL source transformation language. *Science of Computer Programming*, 61(3):190–210, August 2006. ISSN 0167-6423. doi: 10.1016/j.scico.2006.04.002. URL http://www.sciencedirect.com/science/article/pii/S0167642306000669.

[49] L. M. N. Coutinho, J. L. D. Mendes, and C. A. P. S. Martins. MSCSim -Multilevel and Split Cache Simulator. In *Proceedings. Frontiers in Education. 36th Annual Conference*, pages 7–12, October 2006. doi: 10.1109/FIE.2006.322536.

[50] Roger F. Crew. ASTLOG: A Language for Examining Abstract Syntax Trees. In *Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997*, DSL'97, pages 18–18, Berkeley, CA, USA, 1997. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1267950.1267968.

[51] Huimin Cui, Qing Yi, Jingling Xue, Lei Wang, Yang Yang, and Xiaobing Feng. A Highly Parallel Reuse Distance Analysis Algorithm on GPUs. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 1080–1092, May 2012. doi: 10.1109/IPDPS.2012.100.

[52] O. de Moor, M. Verbaere, and E. Hajiyev. Keynote Address: .QL for Source Code Analysis. In *Seventh IEEE International Working Conference on Source Code Analysis and Manipulation, 2007. SCAM 2007*, pages 3–16, September 2007. doi: 10.1109/SCAM.2007.31.

[53] C. De Roover and R. Stevens. Building development tools interactively using the EKEKO meta-programming library. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 429–433, February 2014. doi: 10.1109/CSMR-WCRE.2014.6747211.

[54] Coen De Roover, Carlos Noguera, Andy Kellens, and Vivane Jonckers. The SOUL Tool Suite for Querying Programs in Symbiosis with Eclipse. In *Proceedings of the 9th International Conference on Principles and Practice of Programming in Java*, PPPJ '11, pages 71–80, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0935-6. doi: 10.1145/2093157.2093168. URL http://doi.acm.org/10.1145/2093157.2093168.

[55] L. De Rose, K. Gallivan, E. Gallopoulos, B. Marsolf, and D. Padua. FALCON: A MATLAB interactive restructuring compiler. In *Languages and Compilers for Parallel Computing*, Lecture Notes in Computer Sci-

ence, pages 269–288. Springer, Berlin, Heidelberg, August 1995. ISBN 978-3-540-60765-6 978-3-540-49446-1. doi: 10.1007/BFb0014205. URL https://link.springer.com/chapter/10.1007/BFb0014205.

[56] Luiz De Rose and David Padua. A MATLAB to Fortran 90 Translator and Its Effectiveness. In *Proceedings of the 10th International Conference on Supercomputing*, ICS '96, pages 309–316, New York, NY, USA, 1996. ACM. ISBN 978-0-89791-803-9. doi: 10.1145/237578.237627. URL http://doi.acm.org/10.1145/237578.237627.

[57] Luiz De Rose and David Padua. Techniques for the Translation of MATLAB Programs into Fortran 90. *ACM Trans. Program. Lang. Syst.*, 21(2): 286–323, March 1999. ISSN 0164-0925. doi: 10.1145/316686.316693. URL http://doi.acm.org/10.1145/316686.316693.

[58] E. van der Deijl, G. Kanbier, O. Temam, and E. D. Granston. A cache visualization tool. *Computer*, 30(7):71–78, July 1997. ISSN 0018-9162. doi: 10.1109/2.596631.

[59] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image Segmentation Using K - means Clustering Algorithm and Subtractive Clustering Algorithm. *Procedia Computer Science*, 54:764–771, January 2015. ISSN 1877-0509. doi: 10.1016/j.procs.2015.06.090. URL http://www.sciencedirect.com/science/article/pii/S1877050915014143.

[60] J.P. Diguet, W.F. Catthoor, and H. de Man. Formalized methodology for data reuse exploration in hierarchical memory mappings. In , *1997 International Symposium on Low Power Electronics and Design, 1997. Proceedings*, pages 30–35, August 1997.

[61] Grigoris Dimitroulakos, Christakis Lezos, and Konstantinos Masselos. MEMSCOPT: A source-to-source compiler for dynamic code analysis and loop transformations. In *Proceedings of the 2012 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 385–386, Karlsruhe, Germany, October 2012.

[62] Grigoris Dimitroulakos, Theodoros Lioris, Christakis Lezos, and Konstantinos Masselos. XMSIM: A tool for early memory hierarchy evaluation. In *Proceedings of the 2012 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 405–406, Karlsruhe, Germany, October 2012.

[63] Chen Ding and Yutao Zhong. Reuse Distance Analysis. Technical Report UR-CS-TR-741, University of Rochester, Rochester, New York, February 2001.

[64] Chen Ding and Yutao Zhong. Predicting Whole-program Locality Through Reuse Distance Analysis. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*,

PLDI '03, pages 245–257, New York, NY, USA, 2003. ACM. ISBN 1-58113-662-5. doi: 10.1145/781131.781159. URL http://doi.acm.org/10.1145/781131.781159.

[65] Jan Edler and Mark D. Hill. Dinero IV Trace-Driven Uniprocessor Cache Simulator. URL http://pages.cs.wisc.edu/~markhill/DineroIV/.

[66] D. Eklov and E. Hagersten. StatStack: Efficient modeling of LRU caches. In *2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*, pages 55–65, March 2010. doi: 10.1109/ISPASS.2010.5452069.

[67] Antoine Floc'h, Tomofumi Yuki, Ali El-Moussawi, Antoine Morvan, Kevin Martin, Maxime Naullet, Mythri Alle, Ludovic L'Hours, Nicolas Simon, Steven Derrien, Francois Charot, Christophe Wolinski, and Olivier Sentieys. GeCoS: A framework for prototyping custom hardware design flows. In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, volume 0, pages 100–105, Los Alamitos, CA, USA, 2013. IEEE Computer Society. doi: 10.1109/SCAM.2013.6648190.

[68] Markus Forsberg. *Three Tools for Language Processing: BNF Converter, Functional Morphology, and Extract*. Ph.D. Thesis, Chalmers University of Technology and Goteborg University, Goteborg, Sweden, 2007.

[69] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, Upper Saddle River, NJ, 1 edition edition, October 2010. ISBN 978-0-321-71294-3.

[70] Xiong Fu, Yu Zhang, and Yiyun Chen. Data-Layout Optimization Using Reuse Distance Distribution. In Xiaobo Zhou, Oleg Sokolsky, Lu Yan, Eun-Sun Jung, Zili Shao, Yi Mu, Dong Chun Lee, Dae Young Kim, Young-Sik Jeong, and Cheng-Zhong Xu, editors, *Emerging Directions in Embedded and Ubiquitous Computing*, number 4097 in Lecture Notes in Computer Science, pages 858–867. Springer Berlin Heidelberg, January 2006. ISBN 978-3-540-36850-2 978-3-540-36851-9. URL http://link.springer.com/chapter/10.1007/11807964_86.

[71] E.M. Gagnon and L.J. Hendren. SableCC, an object-oriented compiler framework. In *Technology of Object-Oriented Languages, 1998. TOOLS 26. Proceedings*, pages 140–154, August 1998. doi: 10.1109/TOOLS.1998.711009.

[72] Phillip B. Gibbons. Distinct-values estimation over data streams. In *In Data Stream Management: Processing High-Speed Data Streams*. Springer, January 2007.

[73] G. Goulas, P. Alefragis, N.S. Voros, C. Valouxis, C. Gogos, N. Kavvadias, G. Dimitroulakos, K. Masselos, D. Goehringer, S. Derrien, D. Menard, O. Sentieys, M. Huebner, T. Stripf, O. Oey, J. Becker, G. Rauwerda,

K. Sunesen, D. Kritharidis, and N. Mitas. From Scilab to multicore embedded systems: Algorithms and methodologies. In *2012 International Conference on Embedded Computer Systems (SAMOS)*, pages 268–275, July 2012. doi: 10.1109/SAMOS.2012.6404184.

[74] Sebastian Gunther. Development of Internal Domain-specific Languages: Design Principles and Design Patterns. In *Proceedings of the 18th Conference on Pattern Languages of Programs*, PLoP '11, pages 1:1–1:25, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1283-7. doi: 10.1145/2578903.2579139. URL http://doi.acm.org/10.1145/2578903.2579139.

[75] S. Gupta, Ping Xiang, Yi Yang, and Huiyang Zhou. Locality Principle Revisited: A Probability-Based Quantitative Approach. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 995–1009, 2012. doi: 10.1109/IPDPS.2012.93.

[76] Elnar Hajiyev, Mathieu Verbaere, and Oege de Moor. codeQuest: Scalable Source Code Queries with Datalog. In Dave Thomas, editor, *ECOOP 2006 – Object-Oriented Programming*, number 4067 in Lecture Notes in Computer Science, pages 2–27. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-35726-1 978-3-540-35727-8. URL http://link.springer.com/chapter/10.1007/11785477_2.

[77] Laurie Hendren, Jesse Doherty, Anton Dubrau, Rahul Garg, Nurudeen Lameed, Soroush Radpour, Amina Aslam, Toheed Aslam, Andrew Casey, Maxime Chevalier Boisvert, Jun Li, Clark Verbrugge, and Olivier Savary Belanger. McLAB: Enabling Programming Language, Compiler and Software Engineering Research for Matlab. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '11, pages 195–196, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0942-4. doi: 10.1145/2048147.2048203. URL http://doi.acm.org/10.1145/2048147.2048203.

[78] Martin Hirzel and Trishul Chilimbi. Bursty Tracing: A Framework for Low-Overhead Temporal Profiling. In *4th ACM Workshop on Feedback-Directed and Dynamic Optimization*, pages 117–126, 2001.

[79] Bruce Jacob, Spencer Ng, and David Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, Burlington, MA, 1 edition edition, September 2007. ISBN 978-0-12-379751-3.

[80] Yunlian Jiang, Eddy Z. Zhang, Kai Tian, and Xipeng Shen. Is Reuse Distance Applicable to Data Locality Analysis on Chip Multiprocessors? In Rajiv Gupta, editor, *Compiler Construction*, number 6011 in Lecture Notes in Computer Science, pages 264–282. Springer Berlin Heidelberg, January 2010. ISBN 978-3-642-11969-9 978-3-642-11970-5. URL http://link.springer.com/chapter/10.1007/978-3-642-11970-5_15.

[81] Pramod G. Joisha and Prithviraj Banerjee. The MAGICA Type Inference Engine for MATLAB. In *Compiler Construction*, Lecture Notes in Computer Science, pages 121–125. Springer, Berlin, Heidelberg, April 2003. ISBN 978-3-540-00904-7 978-3-540-36579-2. doi: 10.1007/3-540-36579-6_9. URL https://link.springer.com/chapter/10.1007/3-540-36579-6_9.

[82] Pramod G. Joisha and Prithviraj Banerjee. A translator system for the MATLAB language. *Software: Practice and Experience*, 37(5):535–578, April 2007. ISSN 1097-024X. doi: 10.1002/spe.781. URL http://onlinelibrary.wiley.com/doi/10.1002/spe.781/abstract.

[83] Lennart C.L. Kats and Eelco Visser. The Spoofax Language Workbench: Rules for Declarative Specification of Languages and IDEs. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '10, pages 444–463, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0203-6. doi: 10.1145/1869459.1869497. URL http://doi.acm.org/10.1145/1869459.1869497.

[84] Hideyuki Kawabata, Mutsumi Suzuki, and Toshiaki Kitamura. A MATLAB-Based Code Generator for Sparse Matrix Computations. In *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 280–295. Springer, Berlin, Heidelberg, November 2004. ISBN 978-3-540-23724-2 978-3-540-30477-7. doi: 10.1007/978-3-540-30477-7_19. URL https://link.springer.com/chapter/10.1007/978-3-540-30477-7_19.

[85] P. Klint, T. van der Storm, and J. Vinju. RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation. In *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, 2009. SCAM '09*, pages 168–177, September 2009. doi: 10.1109/SCAM.2009.28.

[86] Donald E. Knuth. Structured Programming with go to Statements. *ACM Computing Surveys (CSUR)*, 6(4):261–301, December 1974. ISSN 0360-0300. doi: 10.1145/356635.356640. URL http://dl.acm.org/citation.cfm?id=356635.356640.

[87] Markus Kowarschik and Christian Weiß. An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. In Ulrich Meyer, Peter Sanders, and Jop Sibeyn, editors, *Algorithms for Memory Hierarchies*, number 2625 in Lecture Notes in Computer Science, pages 213–232. Springer Berlin Heidelberg, January 2003. ISBN 978-3-540-00883-5 978-3-540-36574-7. URL http://link.springer.com/chapter/10.1007/3-540-36574-5_10.

[88] I. Latifis, K. Parashar, G. Dimitroulakos, H. Cappelle, C. Lezos, K. Masselos, and F. Catthoor. MATLAB-to-C compilation targeting Application Specific Instruction Set Processors. In *2016 Design, Automation Test in*

Europe Conference Exhibition (DATE), pages 1453–1456, March 2016.

[89] Ioannis Latifis. *A MATLAB to C vectoring compiler exploiting custom instructions of targeted processors*. Διδακτορική Διατριβή, Πανεπιστήμιο Πελοποννήσου. Τμήμα Πληροφορικής και Τηλεπικοινωνιών, 2017. URL http://hdl.handle.net/10442/hedi/41138.

[90] Ioannis Latifis, Karthick Parashar, Grigoris Dimitroulakos, Hans Cappelle, Christakis Lezos, Konstantinos Masselos, and Francky Catthoor. A MATLAB vectorizing compiler targeting Application-Specific Instruction Set Processors. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(2):32:1–32:28, January 2017. ISSN 1084-4309. doi: 10.1145/2996182. URL http://doi.acm.org/10.1145/2996182.

[91] Sang-Ik Lee, Troy A. Johnson, and Rudolf Eigenmann. Cetus – An Extensible Compiler Infrastructure for Source-to-Source Transformation. In Lawrence Rauchwerger, editor, *Languages and Compilers for Parallel Computing*, number 2958 in Lecture Notes in Computer Science, pages 539–553. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-21199-0 978-3-540-24644-2. URL http://link.springer.com/chapter/10.1007/978-3-540-24644-2_35.

[92] Shun-tak Leung and John Zahorjan. Optimizing Data Locality by Array Restructuring. Technical Report 95-09-01, University of Washington, Seattle, WA, USA, September 1995.

[93] Christakis Lezos, Grigoris Dimitroulakos, Angeliki Freskou, and Konstantinos Masselos. Dynamic source code analysis for memory hierarchy optimization in multimedia applications. In *Proceedings of the 2013 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 343–344, Cagliari, Italy, October 2013.

[94] Christakis Lezos, Grigoris Dimitroulakos, and Konstantinos Masselos. Reuse distance analysis for locality optimization in loop-dominated applications. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1237–1240, Grenoble, France, March 2015.

[95] Christakis Lezos, Ioannis Latifis, Grigoris Dimitroulakos, and Konstantinos Masselos. Compiler-directed data locality optimization in MATLAB. In *Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems*, SCOPES '16, pages 6–9, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4320-6. doi: 10.1145/2906363.2906378. URL http://doi.acm.org/10.1145/2906363.2906378.

[96] Kim-Hung Li. Reservoir-sampling Algorithms of Time Complexity O(N(1 + Log(N/N))). *ACM Trans. Math. Softw.*, 20(4):481–493, December 1994. ISSN 0098-3500. doi: 10.1145/198429.198435. URL http://doi.acm.org/10.1145/198429.198435.

[97] X. Li and L. Hendren. Mc2for: A tool for automatically translating MATLAB to FORTRAN 95. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 234–243, February 2014. doi: 10.1109/CSMR-WCRE.2014.6747175.

[98] Theodoros Lioris, Grigoris Dimitroulakos, and Konstantinos Masselos. An early memory hierarchy evaluation simulator for multimedia applications. *Microprocessors and Microsystems*, 38(1):31–41, February 2014. ISSN 0141-9331. doi: 10.1016/j.micpro.2013.10.006. URL http://www.sciencedirect.com/science/article/pii/S0141933113001701.

[99] Bruno Cardoso Lopes and Rafael Auler. *Getting Started with LLVM Core Libraries*. Packt Publishing, August 2014. ISBN 978-1-78216-692-4.

[100] A. Macii, E. Macii, and M. Poncino. Improving the efficiency of memory partitioning by address clustering. In *Automation and Test in Europe Conference and Exhibition 2003 Design*, pages 18–23, 2003. doi: 10.1109/DATE.2003.1253581.

[101] Paul B Mann. A Translational BNF Grammar Notation (TBNF). *SIGPLAN Not.*, 41(4):16–23, April 2006. ISSN 0362-1340. doi: 10.1145/1147214.1147218. URL http://doi.acm.org/10.1145/1147214.1147218.

[102] Michael Martin, Benjamin Livshits, and Monica S. Lam. Finding Application Errors and Security Flaws Using PQL: A Program Query Language. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 365–383, New York, NY, USA, 2005. ACM. ISBN 1-59593-031-0. doi: 10.1145/1094811.1094840. URL http://doi.acm.org/10.1145/1094811.1094840.

[103] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation Techniques for Storage Hierarchies. *IBM Syst. J.*, 9(2):78–117, June 1970. ISSN 0018-8670. doi: 10.1147/sj.92.0078. URL http://dx.doi.org/10.1147/sj.92.0078.

[104] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, Redmond, Wash, 2nd edition edition, June 2004. ISBN 978-0-7356-1967-8.

[105] Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng. Improving Data Locality with Loop Transformations. *ACM Trans. Program. Lang. Syst.*, 18(4):424–453, July 1996. ISSN 0164-0925. doi: 10.1145/233561.233564. URL http://doi.acm.org/10.1145/233561.233564.

[106] Shaily Mittal and N Nitin. Memory Map: A Multiprocessor Cache Simulator. *Journal of Electrical and Computer Engineering*, 2012. doi: 10.1155/2012/365091. URL https://www.hindawi.com/journals/jece/2012/365091/.

[107] Carlos Moreno and Sebastian Fischmeister. Accurate Measurement of Small Execution Times – Getting Around Measurement Errors. *IEEE Embedded Systems Letters*, PP(99), 2017. ISSN 1943-0663. doi: 10.1109/LES.2017.2654160.

[108] Steffen Mueller. Your benchmarks suck!, September 2010. URL http://blogs.perl.org/users/steffen_mueller/2010/09/your-benchmarks-suck.html.

[109] Sri Hari Krishna Narayanan and Paul Hovland. Calculating Reuse Distance from Source Code. 2016.

[110] George C. Necula, Scott McPeak, Shree P. Rahul, and Westley Weimer. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. In R. Nigel Horspool, editor, *Compiler Construction*, number 2304 in Lecture Notes in Computer Science, pages 213–228. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43369-9 978-3-540-45937-8. URL http://link.springer.com/chapter/10.1007/3-540-45937-5_16.

[111] Nicholas Nethercote. *Dynamic binary analysis and instrumentation*. Ph.D. Dissertation, University of Cambridge, 2004.

[112] Karen Ng, Matt Warren, Peter Golde, and Anders Hejlsberg. The Roslyn Project: Exposing the C# and VB compiler's code analysis. White Paper, Microsoft Corporation, September 2012.

[113] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable Parallel Programming with CUDA. *Queue*, 6(2):40–53, March 2008. ISSN 1542-7730. doi: 10.1145/1365490.1365500. URL http://doi.acm.org/10.1145/1365490.1365500.

[114] Qingpeng Niu, J. Dinan, Qingda Lu, and P. Sadayappan. PARDA: A Fast Parallel Reuse Distance Analysis Algorithm. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 1284–1294, May 2012. doi: 10.1109/IPDPS.2012.117.

[115] F. Olken. Efficient Methods for Calculating the Success Function of Fixed-Space Replacement Policies. Technical Report LBL-12370, Lawrence Berkeley Lab., CA (USA), May 1981. URL http://www.osti.gov/scitech/biblio/6051879.

[116] Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, Dallas, Texas, second edition edition, January 2013. ISBN 978-1-934356-99-9.

[117] G. Y. Paulsen, J. Feinberg, X. Cai, B. Nordmoen, and H. P. Dahle. Matlab2cpp: A Matlab-to-C++ code translator. In *2016 11th System of Systems Engineering Conference (SoSE)*, pages 1–5, June 2016. doi: 10.1109/SYSOSE.2016.7542966.

[118] James F. Power and Brian A. Malloy. A metrics suite for grammar-

based software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6):405–426, November 2004. ISSN 1532-0618. doi: 10.1002/smr.293. URL http://onlinelibrary.wiley.com/doi/10.1002/smr.293/abstract.

[119] Ashwin Prasad, Jayvant Anantpur, and R. Govindarajan. Automatic Compilation of MATLAB Programs for Synergistic Execution on Heterogeneous Processors. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 152–163, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0663-8. doi: 10.1145/1993498.1993517. URL http://doi.acm.org/10.1145/1993498.1993517.

[120] Changwoo Pyo, Kyung-Woo Lee, Hye-Kyung Han, and Gyungho Lee. Reference distance as a metric for data locality. In *High Performance Computing on the Information Superhighway, 1997. HPC Asia '97*, pages 151–156, April 1997. doi: 10.1109/HPC.1997.592139.

[121] Boris Quaing, Jie Tao, and Wolfgang Karl. YACO: A User Conducted Visualization Tool for Supporting Cache Optimization. In Laurence T. Yang, Omer F. Rana, Beniamino Di Martino, and Jack Dongarra, editors, *High Performance Computing and Communications*, number 3726 in Lecture Notes in Computer Science, pages 694–703. Springer Berlin Heidelberg, September 2005. ISBN 978-3-540-29031-5 978-3-540-32079-1. URL http://link.springer.com/chapter/10.1007/11557654_80.

[122] Dan Quinlan. Rose: compiler support for object-oriented frameworks. *Parallel Processing Letters*, 10(02n03):215–226, June 2000. ISSN 0129-6264. doi: 10.1142/S0129626400000214. URL http://www.worldscientific.com/doi/abs/10.1142/S0129626400000214.

[123] M. J. Quinn, A. Malishevsky, and N. Seelam. Otter: bridging the gap between MATLAB and ScaLAPACK. In *Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No.98TB100244)*, pages 114–121, July 1998. doi: 10.1109/HPDC.1998.709963.

[124] M. J. Quinn, A. Malishevsky, N. Seelam, and Y. Zhao. Preliminary results from a parallel MATLAB compiler. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 81–87, March 1998. doi: 10.1109/IPPS.1998.669894.

[125] Peter Rademaker. Binary relational querying for structural source code analysis. Master's thesis, Universiteit van Utrecht, Utrecht, the Netherlands, 2008.

[126] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image

Processing Pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 519–530, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2014-6. doi: 10.1145/2491956.2462176. URL http://doi.acm.org/10.1145/2491956.2462176.

[127] Probir Roy and Xu Liu. StructSlim: A Lightweight Profiler to Guide Structure Splitting. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, CGO '16, pages 36–46, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3778-6. doi: 10.1145/2854038.2854053. URL http://doi.acm.org/10.1145/2854038.2854053.

[128] Mazen A. R. Saghir. *Application-Specific Instruction-Set Architectures for Embedded DSP Applications*. Ph.D. Dissertation, University of Toronto, Toronto, Canada, 1998.

[129] J. Sahuquillo, N. Tomas, S. Petit, and A. Pont. Spim-Cache: A Pedagogical Tool for Teaching Cache Memories Through Code-Based Exercises. *IEEE Transactions on Education*, 50(3):244–250, August 2007. ISSN 0018-9359. doi: 10.1109/TE.2007.900021.

[130] Conrad Sanderson. Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. Technical Report, NICTA, October 2010. URL https://espace.library.uq.edu.au/view/UQ:224609.

[131] Derek L. Schuff, Milind Kulkarni, and Vijay S. Pai. Accelerating Multicore Reuse Distance Analysis with Sampling and Parallelization. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 53–64, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0178-7. doi: 10.1145/1854273.1854286. URL http://doi.acm.org/10.1145/1854273.1854286.

[132] D.L. Schuff, B.S. Parsons, and V.S. Pai. Multicore-aware reuse distance analysis. In *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–8, 2010. doi: 10.1109/IPDPSW.2010.5470780.

[133] Rathijit Sen and David A. Wood. Reuse-based Online Models for Caches. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, pages 279–292, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1900-3. doi: 10.1145/2465529.2465756. URL http://doi.acm.org/10.1145/2465529.2465756.

[134] C. Y. Shei, A. Yoga, M. Ramesh, and A. Chauhan. MATLAB Parallelization through Scalarization. In *2011 15th Workshop on Interaction between Compilers and Computer Architectures*, pages 44–53, February 2011. doi: 10.1109/INTERACT.2011.18.

[135] Xipeng Shen, Jonathan Shaw, and Brian Meeker. Accurate Approximation of Locality from Time Distance Histograms. Technical Report TR902, University of Rochester, Computer Science Department, August 2006. URL https://urresearch.rochester.edu/institutionalPublicationPublicView.action?institutionalItemId=3141&versionNumber=1.

[136] Xipeng Shen, Jonathan Shaw, Brian Meeker, and Chen Ding. Locality Approximation Using Time. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, pages 55–61, New York, NY, USA, 2007. ACM. ISBN 1-59593-575-4. doi: 10.1145/1190216.1190227. URL http://doi.acm.org/10.1145/1190216.1190227.

[137] A. Skodras, C. Christopoulos, and T. Ebrahimi. The JPEG 2000 still image compression standard. *IEEE Signal Processing Magazine*, 18(5):36–58, September 2001. ISSN 1053-5888. doi: 10.1109/79.952804.

[138] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting Binary Search Trees. *J. ACM*, 32(3):652–686, July 1985. ISSN 0004-5411. doi: 10.1145/3828.3835. URL http://doi.acm.org/10.1145/3828.3835.

[139] Olalekan A. Sopeju, Martin Burtscher, Ashay Rane, and James Browne. AutoSCOPE: Automatic Suggestions for Code Optimizations Using PerfExpert. pages 19–25, Las Vegas, Nevada, USA, July 2011.

[140] T. Stripf, O. Oey, T. Bruckschloegl, R. Koenig, M. Huebner, J. Becker, G. Goulas, P. Alefragis, N.S. Voros, G. Rauwerda, K. Sunesen, S. Derrien, D. Menard, O. Sentieys, N. Kavvadias, G. Dimitroulakos, K. Masselos, D. Goehringer, T. Perschke, D. Kritharidis, and N. Mitas. A flexible approach for compiling scilab to reconfigurable multi-core embedded systems. In *2012 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, July 2012. doi: 10.1109/ReCoSoC.2012.6322879.

[141] Timo Stripf, Oliver Oey, Thomas Bruckschloegl, Juergen Becker, Gerard Rauwerda, Kim Sunesen, George Goulas, Panayiotis Alefragis, Nikolaos S. Voros, Steven Derrien, Olivier Sentieys, Nikolaos Kavvadias, Grigoris Dimitroulakos, Kostas Masselos, Dimitrios Kritharidis, Nikolaos Mitas, and Thomas Perschke. Compiling Scilab to high performance embedded multicore systems. *Microprocessors and Microsystems*, 37(8, Part C):1033–1049, November 2013. ISSN 0141-9331. doi: 10.1016/j.micpro.2013.07.004. URL http://www.sciencedirect.com/science/article/pii/S014193311300094X.

[142] Rabin A. Sugumar and Santosh G. Abraham. Efficient Simulation of Caches Under Optimal Replacement with Applications to Miss Characterization. In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '93,

pages 24–35, New York, NY, USA, 1993. ACM. ISBN 978-0-89791-580-9. doi: 10.1145/166955.166974. URL http://doi.acm.org/10.1145/166955.166974.

[143] Rabin Andrew Sugumar. *Multi-configuration simulation algorithms for the evaluation of computer architecture designs*. Thesis, 1993. URL http://deepblue.lib.umich.edu/handle/2027.42/103857. Ph.D.

[144] Richard A. Uhlig and Trevor N. Mudge. Trace-driven Memory Simulation: A Survey. *ACM Comput. Surv.*, 29(2):128–170, June 1997. ISSN 0360-0300. doi: 10.1145/254180.254184. URL http://doi.acm.org/10.1145/254180.254184.

[145] Raoul-Gabriel Urma and Alan Mycroft. Programming Language Evolution via Source Code Query Languages. Tucson, AZ, USA, October 2012.

[146] M. G. J. van den Brand, A. van Deursen, J. Heering, H. A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P. A. Olivier, J. Scheerder, J. J. Vinju, E. Visser, and J. Visser. The Asf+Sdf Meta-Environment: A Component-Based Language Development Environment. *Electronic Notes in Theoretical Computer Science*, 44(2):3–8, June 2001. ISSN 1571-0661. doi: 10.1016/S1571-0661(04)80917-4. URL http://www.sciencedirect.com/science/article/pii/S1571066104809174.

[147] Miguel A. Vega-rodriguez, R. Jorge Gil-ramos, Juan A. Gomez-pulido, and Juan M. Sanchez-perez. A versatile simulator for cache memories on DSM systems. In *Proceedings of the 19th European Conference on Modeling and Simulation*, Riga, Latvia, June 2005. ISBN 1-84233-112-4 (Set) / 1-84233-113-2 (CD).

[148] Miguel Angel Vega Rodriguez, Juan Manuel Sanchez Perez, and Juan Antonio Gomez Pulido. An educational tool for testing caches on symmetric multiprocessors. *Microprocessors and Microsystems*, 25(4):187–194, June 2001. ISSN 0141-9331. doi: 10.1016/S0141-9331(01)00111-9. URL http://www.sciencedirect.com/science/article/pii/S0141933101001119.

[149] Jeffrey S. Vitter. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, March 1985. ISSN 0098-3500. doi: 10.1145/3147.3165. URL http://doi.acm.org/10.1145/3147.3165.

[150] Kris De Volder. JQuery: A Generic Code Browser with a Declarative Configuration Language. In Pascal Van Hentenryck, editor, *Practical Aspects of Declarative Languages*, number 3819 in Lecture Notes in Computer Science, pages 88–102. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-30947-5 978-3-540-31685-5. URL http://link.springer.com/chapter/10.1007/11603023_7.

[151] G. K. Wallace. The JPEG still picture compression standard. *IEEE Trans-*

actions on Consumer Electronics, 38(1):xviii–xxxiv, February 1992. ISSN 0098-3063. doi: 10.1109/30.125072.

[152] Kyu-Young Whang, Brad T. Vander-Zanden, and Howard M. Taylor. A Linear-time Probabilistic Counting Algorithm for Database Applications. *ACM Trans. Database Syst.*, 15(2):208–229, June 1990. ISSN 0362-5915. doi: 10.1145/78922.78925. URL http://doi.acm.org/10.1145/78922.78925.

[153] David S. Wile. Abstract Syntax from Concrete Syntax. In *Proceedings of the 19th International Conference on Software Engineering*, ICSE '97, pages 472–480, New York, NY, USA, 1997. ACM. ISBN 0-89791-914-9. doi: 10.1145/253228.253388. URL http://doi.acm.org/10.1145/253228.253388.

[154] Robert P. Wilson, Robert S. French, Christopher S. Wilson, Saman P. Amarasinghe, Jennifer M. Anderson, Steve W. K. Tjiang, Shih-Wei Liao, Chau-Wen Tseng, Mary W. Hall, Monica S. Lam, and John L. Hennessy. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers. *SIGPLAN Not.*, 29(12):31–37, December 1994. ISSN 0362-1340. doi: 10.1145/193209.193217. URL http://doi.acm.org/10.1145/193209.193217.

[155] Michael E. Wolf and Monica S. Lam. A Data Locality Optimizing Algorithm. In *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, PLDI '91, pages 30–44, New York, NY, USA, 1991. ACM. ISBN 0-89791-428-7. doi: 10.1145/113445.113449. URL http://doi.acm.org/10.1145/113445.113449.

[156] Meng-Ju Wu and D. Yeung. Coherent Profiles: Enabling Efficient Reuse Distance Analysis of Multicore Scaling for Loop-based Parallel Programs. In *2011 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 264–275, October 2011. doi: 10.1109/PACT.2011.58.

[157] Meng-Ju Wu and Donald Yeung. Identifying Optimal Multicore Cache Hierarchies for Loop-based Parallel Programs via Reuse Distance Analysis. In *Proceedings of the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, MSPC '12, pages 2–11, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1219-6. doi: 10.1145/2247684.2247687. URL http://doi.acm.org/10.1145/2247684.2247687.

[158] Meng-Ju Wu and Donald Yeung. Efficient Reuse Distance Analysis of Multicore Scaling for Loop-Based Parallel Programs. *ACM Trans. Comput. Syst.*, 31(1):1:1–1:37, February 2013. ISSN 0734-2071. doi: 10.1145/2427631.2427632. URL http://doi.acm.org/10.1145/2427631.2427632.

[159] Y. Zhong, S.G. Dropsho, Xipeng Shen, A. Studer, and Chen Ding. Miss Rate Prediction Across Program Inputs and Cache Configurations. *IEEE Transactions on Computers*, 56(3):328–343, March 2007. ISSN 0018-9340.

doi: 10.1109/TC.2007.50.

[160] Yutao Zhong and Wentao Chang. Sampling-based Program Locality Approximation. In *Proceedings of the 7th International Symposium on Memory Management*, ISMM '08, pages 91–100, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-134-7. doi: 10.1145/1375634.1375648. URL http://doi.acm.org/10.1145/1375634.1375648.

[161] Yutao Zhong, Maksim Orlovich, Xipeng Shen, and Chen Ding. Array Regrouping and Structure Splitting Using Whole-program Reference Affinity. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, PLDI '04, pages 255–266, New York, NY, USA, 2004. ACM. ISBN 978-1-58113-807-8. doi: 10.1145/996841.996872. URL http://doi.acm.org/10.1145/996841.996872.

[162] Yutao Zhong, Xipeng Shen, and Chen Ding. Program Locality Analysis Using Reuse Distance. *ACM Trans. Program. Lang. Syst.*, 31(6):20:1–20:39, August 2009. ISSN 0164-0925. doi: 10.1145/1552309.1552310. URL http://doi.acm.org/10.1145/1552309.1552310.