Κεφάλαιο 6

Ανάλυση και Χαρακτηρισμός των Δυναμικών Πολυμεσικών Εφαρμογών

Όπως γνωρίζουμε, ένας αυξανόμενο όγκος μετακινούμενων εφαρμογών γενικής χρήσης (π.χ. 3D παιχνίδια, video players) στα ενσωματωμένα συστήματα, παρουσιάζει την ανάγκη να αποτυπωθεί εντός μίας φθηνής αλλά και φορητής ενσωματωμένης συσκευής. Όμως τα ενσωματωμένα συστήματα δυσκολεύονται στην εκτέλεση τέτοιων περίπλοκων εφαρμογών, επειδή οι εφαρμογές προέρχονται από επιτραπέζια συστήματα, τα οποία έχουν διαφορετικούς περιορισμούς σε σχέση με τα χαρακτηριστικά της χρήσης μνήμης, και πιο συγκεκριμένα δεν ασχολούνται με την αποδοτική χρήση της δυναμικής μνήμης. Σήμερα, ένας επιτραπέζιος υπολογιστής τυπικά περιέχει τουλάχιστον 4-8 GB μνήμης RAM, εν αντιθέσει στο εύρος των 256-1024 MB που υπάρχουν στις ενσωματωμένες συσκευές χαμηλής κατανάλωσης και υψηλής τεχνολογίας. Επομένως, ένα από τα κύρια βήματα στην διαδικασία δημιουργίας συμβατότητας μεταξύ των πολυμεσικών εφαρμογών (οι οποίες ήταν αρχικά αναπτυγμένες για PC) και ενσωματωμένων συστημάτων, περιλαμβάνει την βελτίωση του υποσυστήματος δυναμικής μνήμης.

Το υπόλοιπο του παρόντος κεφαλαίου είναι οργανωμένο με την εξής σειρά: στην ενότητα 6.1 περιέχεται η ανάλυση του τομέα πολυμεσικών εφαρμογών, συμπεριλαμβανόμενων και των τυπικών χαρακτηριστικών αυτού του τύπου εφαρμογών, που προσφέρει κίνητρα για βελτιστοποιήσεις στις δυναμικές δομές δεδομένων. Έπειτα, στην ενότητα 6.2 γίνεται η παρουσίαση των δυναμικών δομών δεδομένων και των συγκεκριμένων χαρακτηριστικών τους. Στην συνέχεια, στην ενότητα 6.3, εξηγείται λεπτομερώς η ροή της μεθόδου για την ενασχόληση με τα διάφορα στάδια βελτιστοποίησης. Τέλος, στην ενότητα 6.4, εξάγονται κάποια συμπεράσματα.

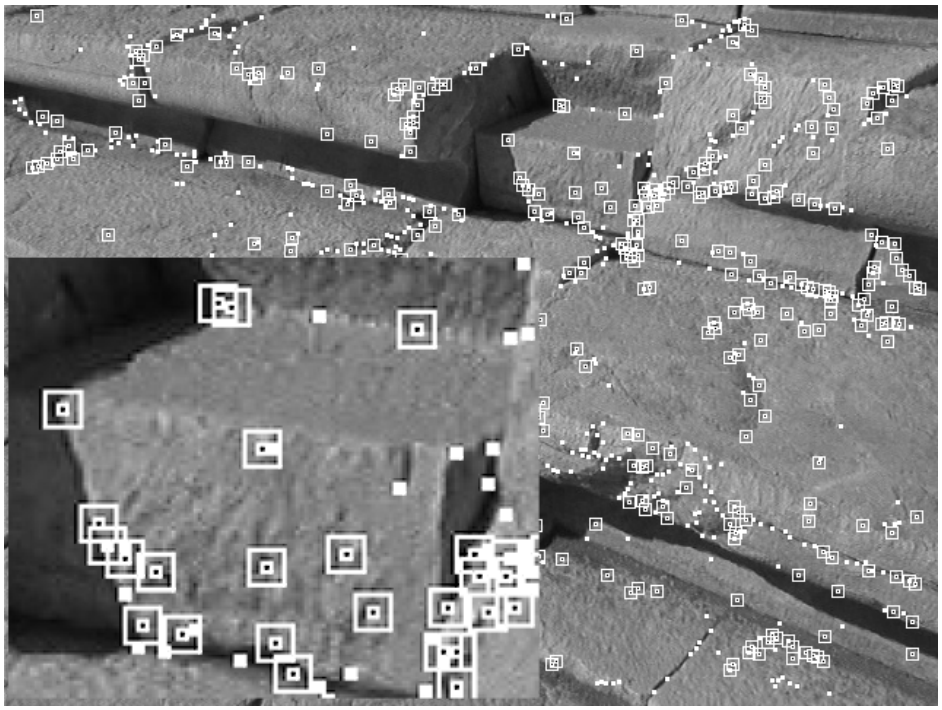
6.1 Χαρακτηριστικά Πολυμεσικών Εφαρμογών

Οι βελτιστοποιήσεις που παρουσιάζονται σε αυτό εδώ το βιβλίο, εκμεταλλεύονται τα χαρακτηριστικά των μοντέρνων πολυμεσικών και επικοινωνιακών ενσωματωμένων εφαρμογών. Ενώ το παραδοσιακό λογισμικό περιείχε μόνο δεδομένα γενικού τύπου ή δεδομένα δομής στοίβας, οι σύγχρονες εφαρμογές απαιτούν την χρήση δεδομένων δομής σωρού (ή αλλιώς δυναμικές δομές δεδομένων). Σε αυτήν την ενότητα, περιγράφονται οι χρήσεις των δυναμικών δομών δεδομένων στις μοντέρνες πολυμεσικές εφαρμογές, και πως χρησιμοποιούνται τα DDTs (dynamic data type, δυναμικός τύπος δεδομένων), για την διαχείριση αυτών των δεδομένων. Το σύνολο των πολυμεσικών και επικοινωνιακών εφαρμογών που αναλύονται περιέχει εφαρμογές 3D αναδόμησης εικόνας [37], [38], εφαρμογές απόδοσης βίντεο, όπως το MPEG-4 Visual Texture Coder (VTC) [39], [40], 3D παιχνίδια [41, 42], την βασισμένη σε URL εφαρμογή πλαισίου εναλλαγών, IPv4 εφαρμογές δρομολογητή [43] και τις εφαρμογές ασφάλειας [44]. Επειδή, αυτές οι εφαρμογές ασχολούνται με δεδομένα που δεν είναι μετρήσιμα σε χρόνο μεταγλώττισης προγράμματος (compile time), απαιτούνται δομές δεδομένων σωρού (ή αλλιώς δυναμικές δομές δεδομένων) για την αποθήκευση των αναγκαίων πληροφοριών για την εφαρμογή. Για παράδειγμα, στη 3D αναδόμηση εικόνας, η οποία μελετάται στην Ενότητα 6.2.1, συγκρίνονται δύο εικόνες για την εύρεση κοινών σημείων. Από την στιγμή που ο αριθμός των κοινών σημείων δεν είναι γνωστός την ώρα της μεταγλώττισης του προγράμματος, τότε απαιτείται μια δυναμική δεδομένων.

6.1.1 Παράδειγμα: Σύστημα 3D Αναδόμησης Εικόνας

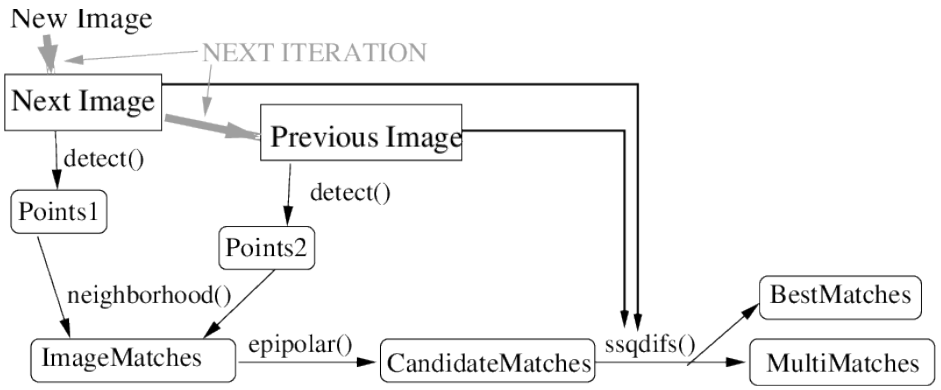
Τα προηγούμενα επιλεγμένα χαρακτηριστικά αναπαριστώνται με την χρήση μιας μοντέρνας εφαρμογής 3D αναδόμησης εικόνας [45]. Συγκεκριμένα, εστιάζουμε σε έναν από τους εσωτερικούς αλγορίθμους, ο οποίος λειτουργεί όπως η τρισδιάστατη αντίληψη των ζωντανών πλασμάτων, όπου η σχετική μετατόπιση μεταξύ ποικίλων δισδιάστατων προβολών χρησιμοποιείται για την αναδόμηση της 3D διάστασης [38]. Αυτή η μονάδα λογισμικού κάνει έντονη την χρήση μιας δυναμικής μνήμης και είναι ένας θεμελιώδης λίθος σε πολλούς σύγχρονους αλγορίθμους τρισδιάστατης όρασης εικόνας: επιλογή χαρακτηριστικού και αντιστοίχηση, οι αλγόριθμοι αυτοί περιέχουν πολλαπλές σειριακές προσπελάσεις και διεργασίες φιλτραρίσματος εισαγόμενων δεδομένων. Ο εξεταζόμενος αλγόριθμος είναι απόσπασμα του πρωτότυπου κώδικα του συστήματος 3D αναδόμησης εικόνας (βλέπε [45] για ολόκληρο τον κώδικα του αλγορίθμου 1,75 εκατομμυρίων σειρών υψηλού επιπέδου C++), και δημιουργεί την μαθηματική αφαίρεση από τα συσχετιζόμενα στιγμιότυπα, η οποία χρησιμοποιείται στον γενικό αλγόριθμο. Ο αλγόριθμος επιλέγει και αντιστοιχίζει στοιχεία

(γωνίες) σε διαφορετικά ακόλουθα στιγμιότυπα, και οι σχετικές αντισταθμίσεις αυτών των χαρακτηριστικών ορίζουν την θέση τους στο χώρο (βλέπε Σχήμα 6.1). Οι διεργασίες που εκτελούνται στις εικόνες φορτώνουν μνήμη, πχ. κάθε εικόνα με ανάλυση 640x489 pixels χρησιμοποιεί περίπου 2.5 MB, και ως αποτελέσματα τυχαιοποιούνται οι προσπελάσεις του αλγορίθμου στις εικόνες. Έτσι, οι κλασικές βελτιστοποιήσεις, όπως οι κατά σειρά πρόσβαση (row-dominated), προσπελάσεων εικόνας δεν μπορούν να εφαρμοσθούν.

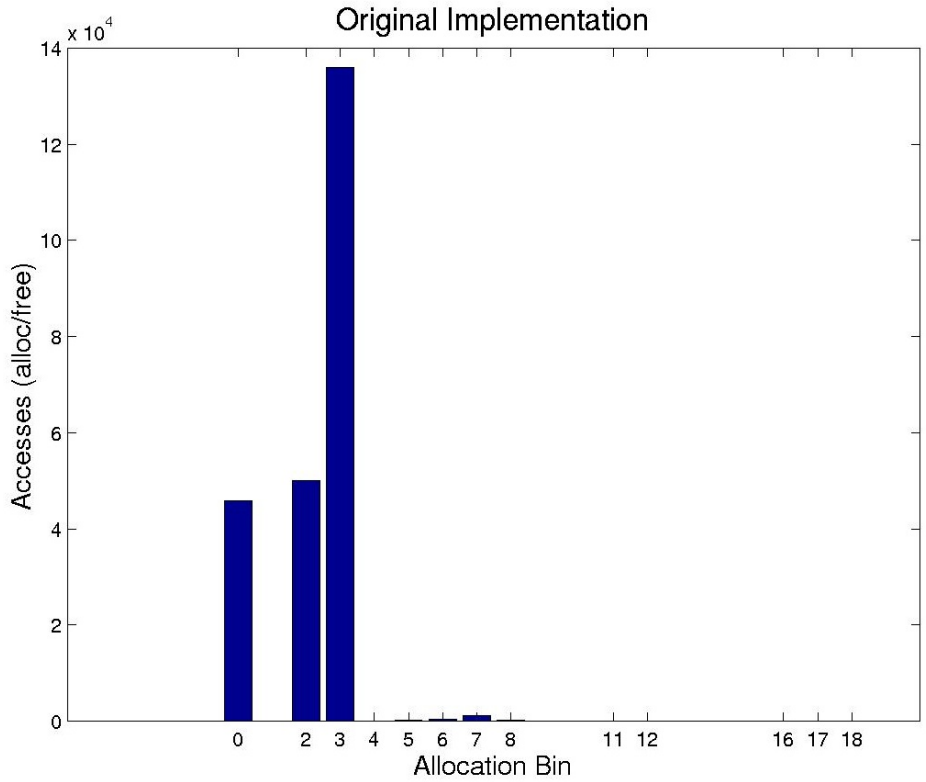


Σχήμα 6.1: Έναρξη του ταιριάσματος γωνιών σε δύο εικόνες των βημάτων του αμφιθεάτρου (αρχαιολογική περιοχή): με βάση την αναζήτηση γειτονικών χωρίων. Ήδη, οι περισσότερες αντιστοιχίες φαίνονται να είναι σωστές (μερικώς λόγω της δευτερεύουσας διαφοράς μεταξύ των εικόνων, οι οποίες μπορούν να φανούν στη δεξιά κάτω γωνία). Το σημείο του κέντρου διευρύνεται.

Για να δικαιολογήσουμε την μεταβαλλόμενη χρήση μνήμης, χρησιμοποιούμε καθορισμένη μνήμη για την λήψη μίας εκτίμησης της γενικής συνεισφοράς του κάθε DDT στην διάχυση ενέργειας, όπως φαίνεται στον Πίνακα 6.1. Αυτό αποδίδει πιο ακριβείς συνεισφορές στις εκτιμήσεις κόστους ενέργειας και αποφεύγει, για παράδειγμα, το DDT με πολύ μικρή ή μεγάλη διαστρέβλωση της χρήσης μνήμης, και αποκρύπτει την συνεισφορά μνήμης από τα υπόλοιπα DDT. Για την εκτίμηση ενέργειας, χρησιμοποιείται το ενσωματωμένο μοντέλο μνήμης SRAM, το οποίο περιγράφεται στο [46], έχοντας για τις εφαρμογές ιεραρχίας μνήμης



Σχήμα 6.2: Ροή εκτέλεσης του DDTs στην τρισδιάστατη εφαρμογή αναδημιουργίας εικόνας.



Σχήμα 6.3: Μελέτη των δοχείων κατανομής μεγέθους blocks του διαχειριστή DM Kingsley που ζητείται από την αρχική τρισδιάστατη εφαρμογή αναδημιουργίας.

υπόψη έναν κόμβο τεχνολογίας 0.13 μm. Αυτό το μοντέλο βασίζεται σε παράγοντες αποτυπωμάτων μνήμης (π.χ. μέγεθος, διαρροές ή εσωτερική δομή) και παράγοντες που προέρχονται από προσπελάσεις μνήμης (π.χ. αριθμός προσπελάσεων ή το είδος τεχνολογίας του κόμβου που χρησιμοποιήθηκε). Να σημειωθεί ότι μπορεί να χρησιμοποιηθεί οποιοδήποτε μοντέλο εκτίμησης ενέργειας, αν αντικαταστήσουμε αυτή την μονάδα στα εργαλεία.

Πίνακας 6.1: Το αρχικό DDT στην τρισδιάστατη εφαρμογή αναδημιουργίας εικόνας.

Μεταβλητή	Προσβάσεις DDT	Αποτύπωμα μνήμης(B)	Ενέργεια 0,13μm tech.(μJ)
Αντιστοιχίες εικόνας	1.20*106	5.14x102	0.18x103
Αντιστοιχίες Υποψηφίων	8.44x105	2.75x105	3.03x103
CM Στατικό Αντίγραφο	6.24x104	1.08x105	4.48x104
Πολλαπλές Αντιστοιχίες	1.84x104	3.62x102	0.02x101
Καλύτερες Αντιστοιχίες	1.66x104	3.07x102	0.02x101
Σύνολο	2.14x106	3.86x105	4.80x104

Η ανάλυση των πληροφοριών που καταγράφουν τα ιδιαίτερα χαρακτηριστικά (Πίνακας 6.1) μας δείχνει πώς τα DDTs επηρεάζουν το υποσύστημα της δυναμικής μνήμης. Τα νούμερα της ενέργειας υπολογίζονται με βάση μία μνήμη που είναι αρκετά μεγάλη, ώστε να χωράει ένα DDT. Τα DDTs και οι διάφορες δυναμικές συμπεριφορές μπορούν να προέλθουν από την εσωτερική υποδιαίρεση ενός γενικού αλγορίθμου σε διάφορους υποαλγορίθμους, που έχουν περιγραφεί προηγουμένως, οι οποίοι περιέχουν διαφορετικές φάσεις διάσχισης και φιλτραρίσματος. Πρώτον, το CandidateMatches είναι το μεγαλύτερο DDT σε αυτό το σύστημα. Δεύτερον, το ImageMatches έχει συχνές προσπελάσεις. Τέλος, το CMCopyStatic είναι μία εφαρμογή δυναμικού πίνακα, η οποία έχει πολύ γρηγορότερη πρόσβαση σε συγκεκριμένα δεδομένα αποθήκευσης (και κρατάει παρά μόνο ένα αντίγραφο των περιεχόμενων του CandidateMatches), και καταναλώνει ένα σημαντικό ποσοστό της ενέργειας που χρησιμοποιείται από το σύστημα. Ο λόγος που έχει τόσο υψηλό κόστος ενέργειας, είναι ότι συγκεκριμένες προσπελάσεις προσθέτουν καινούργια στοιχεία, που επιβαρύνουν το σύστημα με το κόστος μιας μηδενικής μνήμης.

Όπως μας έδειξε αυτή η ανάλυση, τα DDTs στις πολυμεσικές εφαρμογές αλληλεπιδρούν ως τοπικές μεταβλητές (με πολύ μικρό προσδόκιμο ζωής). Αυτές αποθηκεύουν τα ενδιάμεσα δεδομένα που παράγονται ανάμεσα στους διάφορους υποαλγορίθμους. Οι υποαλγόριθμοι χρησιμοποιούνται στο τέλος για την παροχή δεδομένων εξόδου της τρέχοντος μονάδας, στις επόμενες μονάδες λογισμικού των συνολικών εφαρμογών, λόγω της μοναδικότητας στην μηχανική

λογισμικού, όπως είναι ο ρόλος του BestMatches και του MultiMatches. Ένα τελευταίο χαρακτηριστικό των πολυμεσικών και επικοινωνιακών εφαρμογών είναι ο επαναλαμβανόμενος κύκλος των μοτίβων δυναμικής εκχώρησης και αποεκχώρησης τους. Πιο συγκεκριμένα, στη μονάδα 3D αναδόμησης εικόνας, αυτή η συμπεριφορά που αναλύθηκε προηγουμένως είναι παρόμοια για κάθε ζευγάρι διαδοχικών στιγμιοτύπων, και σε κάθε καινούργια έκδοση του αλγορίθμου, το παλιότερο στιγμιότυπο αντικαθίσταται από το επόμενο σύνολο στιγμιοτύπων της εισόδου και έτσι οι αλγόριθμοι εφαρμόζονται με τον ίδιο ακριβώς τρόπο.

6.1.2 Προοπτική για Βελτιστοποιήσεις

Σ' αυτό το βιβλίο μελετάμε ένα σημαντικό υποσύνολο του τομέα των πολυμεσικών εφαρμογών, πιο συγκεκριμένα την επεξεργασία εικόνας (ανάλυση, ερμήνευση εικόνας και αναδόμηση εικόνας), επεξεργασία βίντεο (αναδόμηση βίντεο πολλαπλών οπτικών γωνιών), όπως και την 3D αναδόμηση και απόδοση εικόνας. Γενικά, το πιο στάνταρ κομμάτι της επεξεργασίας εικόνας (φιλτράρισμα και προγραμματισμός) των εφαρμογών χειρίζεται τα στατικά δεδομένα της στοίβας [47]. Όμως, οι πιο προηγμένοι αλγόριθμοι όρασης υπολογιστή και κατανόησης εικόνας πρέπει να διαχειριστούν δυναμικούς τύπους μέσα στο σωρό [37], [48]. Οι τύποι δυναμικών δεδομένων που χρησιμοποιούνται, είτε βασίζονται σε βιβλιοθήκες, όπως η STL (Standard Template Library, Βιβλιοθήκη Στάνταρ Προτύπων) [49], [50] ή σε δικό τους κώδικα. Έτσι, μπορούμε με αυτήν την προσέγγιση να συμπεράνουμε από αυτό ότι η πλειονότητα αυτών των DDTs θα ήταν κατάλληλη για διαχείριση.

Επιπλέον, πολλοί αλγόριθμοι επεξεργασίας εικόνας και βίντεο λειτουργούν με δομές δεδομένων που είναι πολύ πολυπλοκότερες από τους δισδιάστατους πίνακες και είναι δομημένοι με δυναμικό τρόπο (π.χ. με μεγέθη που εξαρτώνται από τις καταγραφές δεδομένων) [51]. Για παράδειγμα, οι αλγόριθμοι καταμερισμού λειτουργούν με δομές δεδομένων που μπορούν να οργανωθούν ως διασυνδεδεμένες λίστες. Παρόλα αυτά, οι πιο αποδοτικοί αλγόριθμοι έχουν ως στόχο την πρόσβαση σε τέτοια δεδομένα με σύνηθες τρόπο. Ο κυρίως τομέας εφαρμογών είναι ο 3D καταμερισμός εικόνας.

Επίσης, βλέπουμε πιο λεπτομερώς τις εφαρμογές επικοινωνιακών πρωτοκόλλων για τα μετακινούμενα ενσωματωμένα συστήματα. Πιο συγκεκριμένα, στο [52] βλέπουμε πως τα πρωτόκολλα επικοινωνίας περιέχουν δομές δεδομένων δυναμικής φύσεως. Ακόμα πιο συγκεκριμένα, οι αλγόριθμοι δρομολόγησης πακέτου έχουν να κάνουν με μία ποικιλία σειρών πακέτων, οι οποίες είναι εκ φύσεως δυναμικές, αφού είναι γνωστό εκ των προτέρων πόσα πακέτα θα μπουν σε κάθε σειρά δρομολόγησης.

Για αυτούς τους δύο συγκεκριμένους τομείς, το εύρος των πιθανών εφαρμογών είναι αρχικά πολύ μεγάλο. Όμως, για τα περισσότερα ενσωματωμένα

συστήματα ο αριθμός και οι τύποι των εφαρμογών πολυμεσικών και επικοινωνιακών πρωτοκόλλων, για να συμπεριληφθούν στον τελικό σχεδιασμό (τουλάχιστον σε ένα μεγάλο βαθμό) πρέπει να είναι γνωστός στο χρόνο σχεδιασμού. Έτσι, είναι δυνατό να αναλύσουμε τους τύπους των δυναμικά κατανεμημένων αντικειμένων που υπάρχουν σε κάθε εφαρμογή (π.χ. σημεία, τρίγωνα, 3D πρόσωπα, αναγνώριση ή συχνά μεγέθη πακέτων κλπ.) και για την σχεδίαση της πιο βολικής εφαρμογής DDT για κάθε μεταβλητή, όπως και της πιο βολικής DDM για την προκείμενη εφαρμογή.

Επιπλέον, η εμπειρία μας έχει δείξει, ότι σε κάθε εφαρμογή, το εύρος μεγεθών που χρησιμοποιείται από δυναμικά στοιχεία είναι πολύ περιορισμένο και μπορεί να καθοριστεί στο χρόνο σχεδιασμού: πρώτον, με μία στατική ανάλυση του πηγαίου κώδικα της εφαρμογής, και δεύτερον, με μία ανάλυση καταγραφής ιδιαίτερων χαρακτηριστικών της συγκεκριμένης εφαρμογής με ένα μειωμένο ποσό ομάδας δεδομένων εισόδου (π.χ. γενικά λιγότερο από 10 παραλλαγές). Να σημειωθεί πως το μέγεθος των δυναμικών στοιχείων είναι γνωστό εκ των προτέρων, αλλά όχι ο ίδιος ο αριθμός αυτών που θα κατανεμηθούν, και μπορεί να ποικίλει σε σημαντικό βαθμό από την μία αντιπροσωπευτική είσοδο δεδομένων στην άλλη. Έτσι, η χρήση της δυναμικά κατανεμημένης μνήμης δικαιολογείται και χρησιμοποιείται εκτενώς σε αυτούς τους τομείς εφαρμογών. Για παράδειγμα, με σκοπό να αναλύσουμε έναν MPEG-4 video player, αλλά και διάφορες ρυθμίσεις συστήματος, όπως ανάλυση οθόνης και ανάλυση οπτικοποίησης, θα πρέπει να μελετηθούν σύμφωνα με την ενσωματωμένη συσκευή για την οποία προορίζονται: smartphone, PDA (Φορητός Ψηφιακός Βοηθός), μία φορητή gaming συσκευή κλπ. Αυτό το σετ ρυθμίσεων πρέπει να εξερευνηθεί για ένα αντιπροσωπευτικό σετ στιγμιοτύπων εισόδου (πχ. με ένα διαφορετικό αριθμό αποδιδόμενων αντικειμένων και υφών), ενώ έχουμε υπόψη μας την κατανομή πιθανότητας των διάφορων τύπων εισόδου για τις τελικές συνθήκες λειτουργίας της συγκεκριμένης ενσωματωμένης συσκευής, αφού αυτά τα χαρακτηριστικά θα επηρεάσουν την τελική δυναμική δομή δεδομένων για την τελική χρήση της εφαρμογής.

Επίσης, είναι απαραίτητο να αναγνωρίσουμε τα κυρίαρχα δυναμικά κατανεμημένα στοιχεία για κάθε εφαρμογή. Στην πραγματικότητα, κάθε εφαρμογή περιέχει ένα μεγάλο αριθμό κατανεμημένων στοιχείων, αλλά στις περισσότερες εφαρμογές πολυμεσικών και επικοινωνιακών πρωτοκόλλων λίγες μεταβλητές (π.χ. μεταξύ 15 και 20), τείνουν να δικαιολογούν ένα μεγάλο ποσοστό των συνολικών προσπελάσεων μνήμης ή των αποτυπωμάτων μνήμης που χρησιμοποιούνται για δυναμική αποθήκευση μνήμης: γενικά μεταξύ 50 και 70 του ποσοστού προσπελάσεων στη μνήμη.

6.2 Διαχείριση Δυναμικών Δεδομένων

Όπως βλέπουμε στην Ενότητα 6.1, στις μοντέρνες εφαρμογές πολυμέσων και επικοινωνιακών πρωτοκόλλων, τα δεδομένα αποθηκεύονται σε οντότητες οι οποίες ονομάζονται DDTs ή απλώς δοχεία, όπως διανύσματα, λίστες ή δέντρα, που μπορούν δυναμικά να προσαρμοστούν στην ποσότητα μνήμης που χρησιμοποιείται από κάθε εφαρμογή [53]. Αυτά τα δοχεία υλοποιούνται στο επίπεδο αρχιτεκτονικής λογισμικού και είναι υπεύθυνα για την συγκράτηση και την οργάνωση των δεδομένων μέσα στην μνήμη, και επίσης υπηρετούν τα αιτήματα της εφαρμογής στον χρόνο εκτέλεσης. Αυτές οι υπηρεσίες απαιτούν την συμπερίληψη αφηρημένων τελεστών τύπων δεδομένων για αποθήκευση, ανάκτηση και παραποίηση των τιμών των δεδομένων, που δεν συνδέεται με μία εφαρμογή συγκεκριμένου δοχείου και που μοιράζεται μία κοινή διεπαφή (όπως στην STL [49]).

Τα δυναμικά δεδομένα συχνά αναφέρονται ως δεδομένα σωρού, ενώ τα στατικά δεδομένα, συνήθως ή κατοικούν στο τομέα γενικών δεδομένων ή στην στοίβα. Η διαφορά μεταξύ δυναμικών τύπων δεδομένων και δυναμικών δομών δεδομένων, είναι ότι το πρώτο μας παρέχει μία διεπαφή με μία καθορισμένη ομάδα διεργασιών, ενώ το δεύτερο μας παρέχει μία συγκεκριμένη εφαρμογή που υπακούει στην διεπαφή. Για παράδειγμα, η STL παρέχει ακολουθίες ως δυναμικούς τύπους δεδομένων. Με αυτή την λογική τα διανύσματα και οι λίστες, είναι δύο συγκεκριμένες δομές δεδομένων με διαφορετικούς συμβιβασμούς πολυπλοκότητας που εφαρμόζουν τον τύπο ακολουθίας δεδομένων.

Για να κατανοήσουμε την διαφορά μεταξύ δυναμικών τύπων δεδομένων και παραδοσιακών στατικών τύπων δεδομένων, είναι απαραίτητο πρώτα να κατανοήσουμε πως τα διαφορετικά στοιχεία που αποθηκεύονται σε αυτούς τους τύπους δεδομένων αποτυπώνονται στην πραγματική ιεραρχία της μνήμης. Η αποτύπωση ενός στοιχείου στην πραγματική του θέση στην μνήμη για τύπους στατικών και δυναμικών δεδομένων, αποτελούνται από δύο στρώσεις. Η πρώτη στρώση, που από εδώ και στο εξής θα αναφέρεται ως αποτύπωση στοιχείων, αποτυπώνει την τοποθεσία ενός στοιχείου σε μία συγκεκριμένη δομή δεδομένων, σε μία θέση στα block μνήμης που απασχολούνται από την δομή δεδομένων. Η δεύτερη στρώση, που από εδώ και στο εξής θα αναφέρεται ως αποτύπωση block, αποτυπώνει τα block μνήμης μιας δομής δεδομένων σε δεξαμενές δεδομένων, που μπορούν αντιπροσωπεύσουν την ιεραρχία της φυσικής μνήμης. Η παρουσία της μονάδας διαχείρισης μνήμης προσθέτει μία τρίτη στρώση, υλικού (hardware), που αποτυπώνει διευθύνσεις εικονικής μνήμης σε διευθύνσεις φυσικής μνήμης.

Η διαφορά μεταξύ δυναμικών και στατικών τύπων δεδομένων είναι πως υλοποιούνται αυτές οι αποτυπώσεις. Οι διαφορές περιγράφονται στην στρώση αποτύπωσης των στοιχείων. Μετά περιγράφονται και στην στρώση αποτύπω-

σης των blocks. Στην στρώση αποτύπωσης στοιχείων, η αποτύπωση είναι αμετάβλητη για τους τύπους στατικών δεδομένων. Η πιο τυπική δομή στατικών δεδομένων είναι ο πίνακας, ο οποίος ένα πολύ καλό παράδειγμα δομής. Η αντιστάθμιση μνήμης ενός στοιχείου στο block μνήμης που χρησιμοποιείται από τον πίνακα, είναι μία γραμμική αποτύπωση του ευρετηρίου του (που γίνεται με το να πολλαπλασιάζεται με το μέγεθος του στοιχείου). Αυτή η αποτύπωση είναι στατική και καθορίζεται από την δυαδική διεπαφή της εφαρμογής της πλατφόρμας (ABI) [54].

Για τους τύπους των δυναμικών δεδομένων, από την άλλη, η αποτύπωση του στοιχείου στην θέση των block μνήμης καθορίζεται από την εφαρμογή της συγκεκριμένης δομής μνήμης. Επιπλέον, μέσω των λειτουργιών που παρέχονται από τους δυναμικούς τύπους δεδομένων, αυτή η αποτύπωση μπορεί να αλλάξει κατά την διάρκεια του χρόνου εκτέλεσης. Για παράδειγμα, ένας vector, που είναι ότι πιο πλησιέστερο σε έναν πίνακα, αποτυπώνει τα στοιχεία σε ένα block μνήμης που χρησιμοποιεί, αρκετά όμοια με έναν πίνακα. Όμως, ένας vector επιτρέπει την εισαγωγή στοιχείων οπουδήποτε με χρήση ευρετηρίου. Αυτή η εισαγωγή επηρεάζει όλα αυτά τα στοιχεία μετά από το πρώτο στοιχείο που έχει εισαχθεί, και αλλάζει την τοποθεσία τους. Το γεγονός ότι οι τύποι δυναμικών δεδομένων επιτρέπουν την εισαγωγή, όπως και την αφαίρεση στοιχείων, έχει συνέπειες για την στρώση αποτύπωσης των blocks. Είναι απαραίτητο να αντικαταστήσουμε τα μικρότερα block μνήμης με όσο το δυνατόν μεγαλύτερα στοιχεία γίνεται να προστεθούν στην δομή δυναμικών δεδομένων, τα οποία αυξάνονται. Έτσι, για τους τύπους δυναμικών δεδομένων, η στρώση αποτύπωσης block είναι απαραίτητα δυναμική.

Για τους στατικούς τύπους δεδομένων, σε περίπτωση που κατανεμηθούν σε χρόνο εκτέλεσης, είναι δυνατό να έχουμε δυναμική αποτύπωση μνήμης. Όμως, επειδή δεν υπάρχει δυναμικότητα στην αποτύπωση στοιχείων, αυτή η κατανομή δεν χρειάζεται τόσο συχνά, αφού είναι απαραίτητη μόνο για την δημιουργία εντελώς καινούργιων δομών δεδομένων, αποφεύγοντας την δυναμικότητα στην αποτύπωση στοιχείων.

Η πραγματική εφαρμογή δυναμικών δομών δεδομένων κωδικοποιεί μόνο την στρώση αποτύπωσης στοιχείων. Ο δυναμισμός στην στρώση αποτύπωσης block διαχειρίζεται από τον διαχειριστή δυναμικής μνήμης (DDM), ο οποίος όχι μόνο έχει να αναλάβει την εξυπηρέτηση των αιτημάτων κατανομής μνήμης, αλλά και να επιβεβαιώσει ότι αυτό γίνεται με τον βέλτιστο τρόπο.

Με τον συνδυασμό της αποτύπωσης της στρώσης ευρετηρίου και της αποτύπωσης block έχουμε ως αποτέλεσμα την αποθήκευση στοιχείων των DDTs σε δεξαμενές μνήμης. Μετά από αυτή την αποτύπωση, είναι δυνατό να αποτυπώσουμε αυτές τις δεξαμενές μνήμης, μαζί με την δεξαμενή που είναι αφιερωμένη στα γενικά δεδομένα και στα δεδομένα στοίβας, σε μία ιεραρχία πραγματικής μνήμης. Αυτό το βήμα συνδυάζει τους δυναμικούς και στατικούς τύπους δεδο-

μένων και είναι συμβατό με την αποτύπωση της στρώσης ευρετηρίου και την αποτύπωση block. Έτσι, τα δυναμικά δεδομένα πρέπει να είναι τα πρώτα που θα επεξεργαστούμε, πριν να μπορέσουν να συνδυαστούν σε θέματα δεξαμενών μνήμης με την στατική δεξαμενή στοίβας που έχει απομείνει.

Όπως είδαμε δύο στοιχεία είναι απαραίτητα για να γίνει δυνατός ο σχεδιασμός εφαρμογών που χρησιμοποιούν δυναμικά δεδομένα. Πρώτον, η εφαρμογή τελεστών ενός DDT μπορεί να έχει σημαντικές συνέπειες στα μοτίβα αποθήκευσης και προσπέλασης. Δεύτερον, η εφαρμογή του DDM μπορεί να επηρεάσει σε ποιο σημείο της ιεραρχίας μνήμης βρίσκονται αυτές οι προσπελάσεις και η αποθήκευση τους. Αυτά περιγράφονται παρακάτω.

6.2.1 Ευκαιρίες Βελτιστοποίησης Δομών Δυναμικών Δεδομένων

Ο σχεδιασμός των DDTs υποτάσσεται σε μία συγκεκριμένη διεπαφή που αποτελείται από μία ομάδα τελεστών. Οι υπάρχουσες και δημοφιλείς βιβλιοθήκες έχουν ένα ικανοποιητικό σύνολο τελεστών, που παρέχουν την ζητούμενη ελαστικότητα, χωρίς να περιορίζουν σε μεγάλο βαθμό την εφαρμογή των DDTs (π.χ. STL [49] και συλλογές Java). Πριν όμως εξετάσουμε αυτούς τους τελεστές, είναι σημαντικό να αντιληφθούμε ότι υπάρχει μια ποικιλία διάφορων DDTs. Επίσης, η επιλογή του ιδανικού DDT, συνήθως βασίζεται στην εφαρμογή για την οποία τον χρειαζόμαστε. Έτσι, υπάρχει ένα ευρύ σύνολο βιβλιοθηκών που μας δίνει διαφορετικές κλάσεις DDTs. Μερικές βιβλιοθήκες, όπως η STL [49], είναι πιο τυποποιημένες και έρχονται μαζί με τους περισσότερους μεταγλωττιστές της C++. Άλλες βιβλιοθήκες είναι πιο κατακερματισμένες, και είναι έτσι στάνταρ ad-hoc ή δημιουργημένες για συγκεκριμένους σκοπούς. Αυτές οι δύο κατηγορίες βιβλιοθηκών εξετάζονται λεπτομερώς.

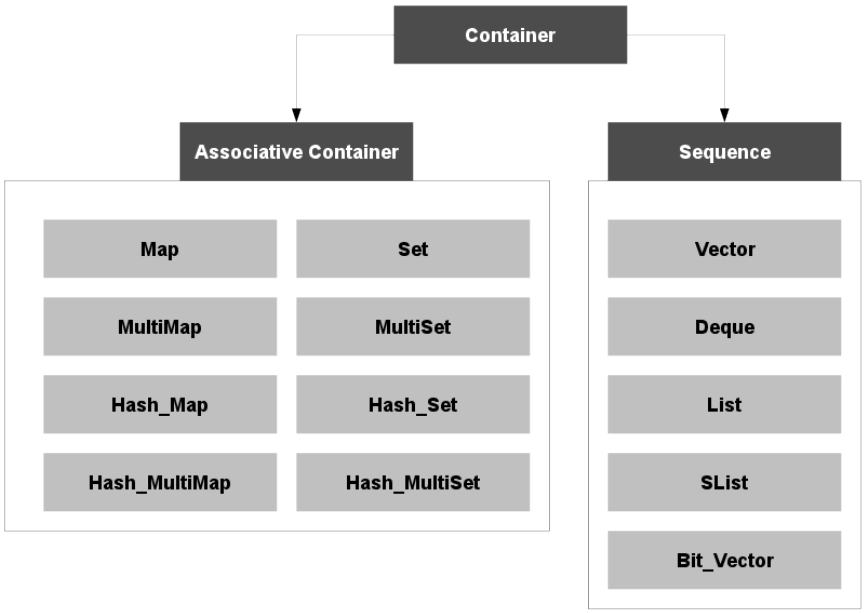
Αυτή η ταξινόμηση, όπως ορίζεται από την STL, περιέχει τις ακόλουθες έννοιες υψηλού επιπέδου. Αυτή η ιεραρχία, μαζί με τις εφαρμογές παρουσιάζονται και στο Σχήμα 6.4. Τα λιγότερο διαφανή στοιχεία είναι οι διάφορες κλάσεις των DDTs, ενώ τα πιο διαφανή συγκεκριμενοποιούν κάποιες εφαρμογές, που παρέχονται από την βιβλιοθήκη STL.

- Το Δοχείο: είναι ένα αντικείμενο που αποθηκεύει άλλα αντικείμενα (τα στοιχεία του), και έχει μεθόδους για την προσπέλαση των στοιχείων. Συγκεκριμένα, κάθε τύπος που είναι μοντέλο Δοχείο έχει ένα συσχετισμένο τύπο επανάληψης, που μπορεί να χρησιμοποιηθεί για την επαναλαμβανόμενη διάσχιση τον στοιχείων του.

- Ακολουθίες: Ένα δοχείο μεταβλητού μεγέθους, του οποίου τα στοιχεία είναι στοιχισμένα σε μία αυστηρά γραμμική ακολουθία. Υποστηρίζουν την εισαγωγή και αφαίρεση στοιχείων.

- Συσχετιζόμενα Δοχεία: Ένα δοχείο μεταβλητού μεγέθους, που υποστηρίζει την αποδοτική ανάκτηση στοιχείων, η οποία είναι βασισμένη σε κλειδιά. Υποστηρίζει την εισαγωγή και αφαίρεση στοιχείων, αλλά διαφέρει από τις ακολουθίες στο ότι δεν παρέχει μηχανισμό για την εισαγωγή στοιχείου σε μία συγκεκριμένη θέση.

Πέρα από αυτές τις γενικές κατηγορίες, η STL επίσης παρέχει κάποιες συγκεκριμένες δομές δεδομένων, κυρίως για να παρέχει λειτουργικότητα στα παραπάνω είδη δοχείων, ή για τις συγκεκριμένες ανάγκες της διαχείρισης ακολουθιών χαρακτήρων (strings). Λόγω του ότι βασίζονται στα ήδη υπάρχοντα δοχεία, ή απευθύνονται σε κάποια συγκεκριμένη ανάγκη (διαχείριση string), που δεν είναι πολύ σχετική με τον τομέα τον multimedia. Αυτά δεν περιγράφονται περαιτέρω σ' αυτό εδώ το βιβλίο.



Σχήμα 6.4: Ιεραρχία των διάφορων DDTs όπως έχουν ταξινομηθεί από την STL (σε πιο σκούρο χρώμα), μαζί με τις εφαρμογές (ανοιχτό χρώμα) που συσχετίζονται με την STL.

- Σχοινιά: Αυτά είναι σχεδιασμένα για να είναι αναρριχήσιμες οι εφαρμογές μονοδιάστατων πινάκων, και είναι σχεδιασμένες ειδικά για διεργασίες που περιέχουν ολόκληρη την ακολουθία χαρακτήρων.

- Bitset: Παρόμοιο μ' ένα vector τιμών Boole, περιέχει μια συλλογή bits και παρέχει συνεχή πρόσβαση σε κάθε bit. Αντίθετα από τον vector, το bitset δεν μπορεί να αλλάξει. Έτσι αυτό το κάνει παρόμοιο και μ' έναν πίνακα. Τα bitset δεν αναλύονται σ' αυτό το βιβλίο, αφού δεν ορίζονται ως δοχείο STL, αφού έχουν ελλιπείς επαναλήψεις για διάσχιση.

- Προσαρμογείς Δοχείων: Τεχνικά μιλώντας οι προαναφερθέντες τομείς δεν είναι δοχεία, αλλά προσαρμογείς που μπορούν να τοποθετηθούν πάνω από άλλα δοχεία για να περιορίσουν την διεπαφή.

Σχετικό είναι το γεγονός ότι οι προσπελάσεις στα δοχείο επιτυγχάνονται μέσω των επαναληπτών. Ενώ τα στατικά δεδομένα, όπως και οι πίνακες, χρησιμοποιούν δείκτες για την προσπέλαση και την πλοήγηση στα στοιχεία, αυτό είναι δυνατό μόνο και μόνο γιατί οι θέσεις μνήμης διαδοχικών χαρακτήρων είναι συνεχόμενες. Οι επαναλήπτες είναι μία γενική μορφή των δεικτών. Οι επαναλήπτες είναι τα κεντρικά στοιχεία του γενικού προγραμματισμού, γιατί είναι μια διεπαφή μεταξύ των δοχείων και των αλγόριθμων: οι αλγόριθμοι τυπικά θεωρούν τους επαναλήπτες ως ορίσματα, έτσι ένα δοχείο χρειάζεται μόνο να παρέχει ένα τρόπο προσπέλασης στοιχείων με την χρήση επαναληπτών [49]. Οι επαναλήπτες κατηγοριοποιούνται από την STL, όπως φαίνεται στο Σχήμα 6.5.

- Απλός Επαναλήπτης: Ένα αντικείμενο που μπορεί να υποστεί διακοπή αναφοράς για να κατευθύνει σε ένα άλλο αντικείμενο. Το πιο απλό παράδειγμα είναι στην περίπτωση των στατικών τύπων δεδομένων, στο οποίο υπάρχει ένα δείκτης που μας οδηγεί σε έναν πίνακα.

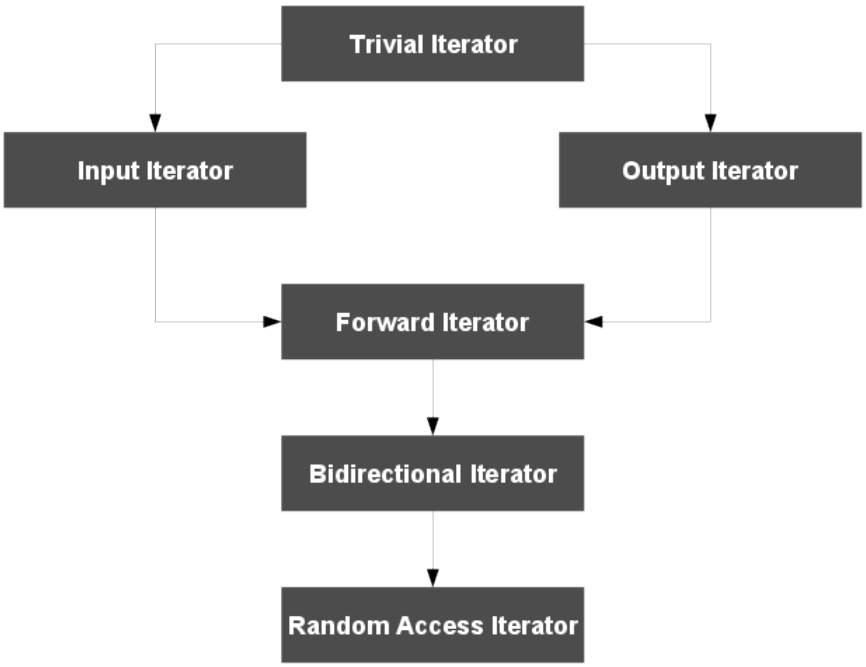
- Επαναλήπτης Εισόδου: Ένας επαναλήπτης που μπορεί να υποστεί διακοπή αναφοράς για να κατευθύνει σε ένα άλλο αντικείμενο, και μπορεί να αυξηθεί για να μπορεί να αποκτήσει τον επόμενο επαναλήπτη σε μία ακολουθία.

- Επαναλήπτης Εξόδου: Ένας επαναλήπτης που παρέχει έναν μηχανισμό για την αποθήκευση (αλλά όχι απαραίτητα την προσπέλαση) ακολουθιών τιμών.

- Προωθητικός Επαναλήπτης: Ένας επαναλήπτης που ανταποκρίνεται στην διαισθητική έννοια μίας γραμμικής ακολουθίας τιμών.

- Διευθυντικός Επαναλήπτης: Ένας επαναλήπτης που μπορεί να αυξηθεί και να μειωθεί.

- Επαναλήπτης Τυχαίας Προσπέλασης: Ένας επαναλήπτης που παρέχει αύξηση και μείωση και επίσης παρέχει μεθόδους συνεχούς χρόνου για την εμπρός και πίσω μετακίνηση, σε βήματα αυθαίρετου μεγέθους. Το καλύτερο παράδειγμα είναι ένας δείκτης προς κάποιον πίνακα.



Σχήμα 6.5: Ιεραρχία των διάφορων κλάσεων επαναληπτών που χρησιμοποιούνται για την διάσχιση και την προσπέλαση των διάφορων DDTs, όπως ορίζεται από την STL

Ο τελευταίος τρόπος με τον οποίο μπορούν να κατηγοριοποιηθούν, είναι ανάλογα με το πώς χρησιμοποιούνται στην εφαρμογή. Εδώ, διακρίνουμε τους διάφορους τρόπους με τους οποίους μπορεί να χρησιμοποιηθεί ένα DDT από μία εφαρμογή. Τα μοτίβα χρήσης βασίζονται στο πως χρησιμοποιείται ο επαναλήπτης προς το DDT. Οι περισσότερες λειτουργίες που ορίζονται στα DDTs απαιτούν έναν επαναλήπτη, αυτό όμως δεν ισχύει για όλες. Για τις ακολουθίες, θεωρούμε την λειτουργία append (τοποθέτησης στο τέλος) όπως βασίζεται σε ένα προωθητικό επαναλήπτη, ως την τοποθεσία όπου προστίθεται ένα στοιχείο που είναι ακριβώς ορισμένο. Από την άλλη, η ανάθεση σε ένα ευρετήριο για την ακολουθία ή την ανάθεση μίας τιμής σε ένα κλειδί στο συσχετιζόμενο δοχείο, θεωρείται όπως βασίζεται σε έναν τυχαίο επαναλήπτη. Έτσι είναι δυνατό να ξεχωρίσουμε δύο διαφορετικά μοτίβα χρήσης:

- Ένας στατικός αριθμός επαναληπτών προελαύνει το DDT. Αυτή είναι η περίπτωση όπου ένα DDT χρησιμοποιείται ως μια ενδιάμεση μεταβλητή και χτίζεται σε μία σειρά φάσεων. Σημειώνουμε πως ο πραγματικός αριθμός σε χρόνο εκτέλεσης δεν πρέπει να είναι στατικός, αλλά περιστασιακά στατικός, με την έννοια ότι υπάρχει ένα ακριβώς ορισμένο σύνολο φωλιών βρόχων που θα μπορούσαν να παραποιήσουν το DDT. Επίσης να σημειωθεί ότι αυτή η προϋπόθεση δεν είναι τόσο περιοριστική, αφού οι ροές ελέγχου είναι κατά κύριο λόγο στατικές. Έτσι, όσο δεν υπάρχει βρόχος υψηλού επιπέδου που συνεχώς δημιουργεί έναν επαναλήπτη για την προσπέλαση του DDT, τόσο το μοτίβο χρήσης θεωρείται πως ανήκει σε αυτήν την κατηγορία.

- Ένας δυναμικός αριθμός επαναληπτών προελαύνει το DDT. Αυτή είναι η περίπτωση όπου το DDT χρησιμοποιείται ως κύρια κατάσταση, και τροποποιείται ξανά και ξανά σε ένα βρόχο υψηλού επιπέδου. Ένα παράδειγμα αυτού είναι ένα ιστόγραμμα που υπολογίζεται στην διάρκεια μίας ολόκληρης εισόδου, και έτσι ένας επαναλήπτης τυχαίας προσπέλασης χρησιμοποιείται για να μεταποιήσει το DDT κάθε τιμής.

Ενώ τα DDTs λειτουργούν ως κύριες καταστάσεις, μπορούν να βελτιστοποιηθούν για να έχουν αποδοτικές λειτουργίες. Για τα DDTs που λειτουργούν ως ενδιάμεσες μεταβλητές, είναι δυνατό, δεδομένων των κατάλληλων βελτιστοποιήσεων, να αφαιρεθούν εντελώς. Αυτό περιγράφεται λεπτομερώς στην Ενότητα 6.3.3. Για όλους τους τύπους δυναμικών δεδομένων είναι δυνατό να χρησιμοποιήσουμε πληροφορίες σχετικές με την εφαρμογή, για να βελτιστοποιήσουμε λειτουργίες, αλλά και την αποτύπωση επιπέδου blocks στην ιεραρχία της μνήμης. Αυτό περιγράφεται στην Ενότητα 6.3.

Ad-hoc και Ειδικές Βιβλιοθήκες

Πέρα από τις Boost C++ βιβλιοθήκες [55], που παρέχουν μία επιπλέον ποικιλία λειτουργικότητας που λείπει από την C++ και την STL, υπάρχουν αρκετές βιβλιοθήκες δέντρων που είναι διαθέσιμες στο κοινό εδώ και αρκετά χρόνια [49]. Ενώ η Boost είναι ένα στάνταρ ad-hoc, οι άλλες βιβλιοθήκες είναι προσωπικές συνεισφορές από διάφορους συγγραφείς, χωρίς να έχουν οριστεί. Αυτές οι βιβλιοθήκες εστιάζονται στις κλάσεις των DDTs (όπως και σε άλλες λειτουργικότητες) που δεν υπάρχουν στην STL. Κυρίως εστιάζονται σε γραφήματα και δέντρα, συγκεκριμένα σε δομές δεδομένων που έχουν περισσότερη δομή στην διεπαφή τους. Τα δέντρα περιέχουν την έννοια γονέα και παιδιού, όπως και αδελφών, ενώ οι ακολουθίες έχουν μόνο αδερφικά στοιχεία. Τα γραφήματα προσθέτουν ακόμα περισσότερη ελαστικότητα στους συνδέσμους.

Είναι σημαντικό να σημειωθεί πως η συζήτηση που γίνεται εδώ για τα δέντρα, προορίζεται για δέντρα, των οποίων η πραγματική δομή τους έχει σημασιολογικό συσχετισμό με την εφαρμογή. Κάποια από τα συσχετιζόμενα δοχεία, για παράδειγμα το map, χρησιμοποιούν τα εσωτερικά δέντρα, αλλά από άποψη διεπαφής παρουσιάζονται ως συσχετιζόμενα δοχεία που αποτυπώνουν κλειδιά σε τιμές. Τα δέντρα που παρουσιάζονται εδώ, χρησιμοποιούνται από εφαρμογές που στηρίζονται πάνω σε μία συγκεκριμένη δομή στην σχέση παιδιού- γονέα, πέρα από αυτές που χρησιμοποιούνται καθαρά για λόγους επιδόσεων. Έτσι, τέτοιες βιβλιοθήκες έχουν πολύ λιγότερη ελαστικότητα, αναφορικά με τις εσωτερικές δομές δεδομένων που χρησιμοποιούνται για την αντιπροσώπευση αυτών των δέντρων. Από την άλλη, επειδή υπάρχουν σημασιολογίες στην φυσική διάταξη των στοιχείων, αυτές οι πληροφορίες μπορούν να εκμεταλλευτούν για χάρη διάφορων βελτιστοποιήσεων.

Παρόμοιες παρατηρήσεις ισχύουν εδώ, όπως και στην τελευταία ενότητα που είναι αφιερωμένη στην δυνατότητα για βελτιστοποιήσεις. Κατ' αρχήν, οι ενδιάμεσες εκδόσεις αναπαραστάσεων των δεδομένων ενός δέντρου μπορούν να είναι περιττές, και έτσι είναι δυνατόν να αφαιρεθούν. Αλλά λόγω των διαφορετικών διεπαφών και της μεγαλύτερης ποικιλίας στον τρόπο που ορίζονται αυτές οι βιβλιοθήκες δέντρων, δεν υπάρχει σε αυτό το βιβλίο η ίδια η διαδικασία βελτιστοποίησης. Οι ακολουθίες STL αναφέρονται στην Ενότητα 6.3.3.

6.2.2 Διαχείριση Δυναμικής Μνήμης

Ο ρόλος του υποσυστήματος DDM είναι να τροφοδοτεί την εφαρμογή ή τα DDTs της εφαρμογής με block μνήμης, όπου μπορούν να αποθηκευτούν τα πραγματικά δεδομένα. Αυτά τα block ζητούνται μέσω της εκχώρησης και επιστρέφονται από την εφαρμογή μέσω της αποεκχώρησης. Αυτό συμβαίνει μέσω των λειτουργιών που είναι επ ακριβώς ορισμένες [56], και περιγράφονται παρακάτω. Από τη στιγμή που το υποσύστημα DMM είναι ορισμένο σε αυτές τις δύο μόνο λειτουργίες, δεν περιορίζεται ως προς την εφαρμογή του σε οποιαδήποτε διεργασία.

Είναι δυνατόν να δοθούν μόνο μερικές εγγυήσεις: ότι η εκχώρηση ενός block μνήμης θα επιστρέψει ένα block τουλάχιστον του αιτούμενου μεγέθους. Δηλαδή ένα block που είναι κατανεμημένο δεν συμπίπτει με ένα άλλο κατανεμημένο block, έτσι το το block μνήμης δεν μεταφέρεται στην μνήμη, και κατά προτίμηση η μνήμη αποεκχώρησης θα επαναχρησιμοποιηθεί για μελλοντικές κατανομές. Οι δύο κύριες λειτουργίες που παρέχονται από το DMM είναι:

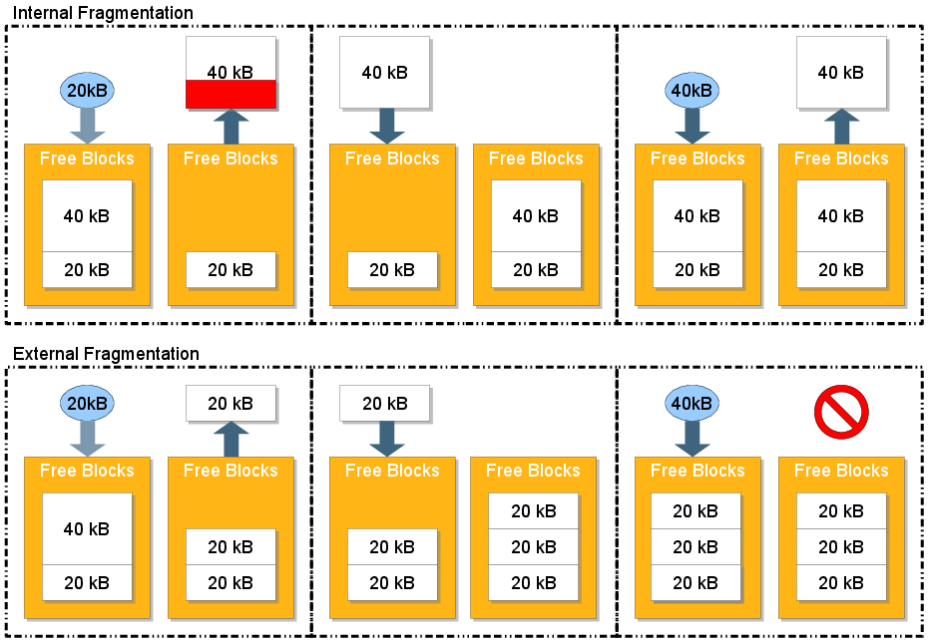
- malloc (που ονομάζεται new στην C++): Μία λειτουργία που δέχεται ως είσοδο ένα απαιτούμενο μέγεθος και επιστρέφει ένα block μνήμης που είναι τόσο μεγάλο τουλάχιστον όσο το απαιτούμενο μέγεθος.

- free (που ονομάζεται delete στην C++): Μία λειτουργία που ελευθερώνει ένα προηγουμένως κατανεμημένο block, επιτρέποντας έτσι την επαναχρησιμοποίηση του για μελλοντικές κατανομές.

Υπάρχουν αρκετοί μη λειτουργικοί περιορισμοί, που πρέπει να έχουμε υπόψη πριν να κρίνουμε ότι ένα DMM είναι βέλτιστο, ή ακόμα και χρησιμοποιήσιμο. Ένας από αυτούς τους παράγοντες είναι ο κατακερματισμός της μνήμης. Έχει ήδη παρουσιαστεί σε προηγούμενη ενότητα, μαζί με την διαφορά ανάμεσα σε δύο είδη κατακερματισμού μνήμης και πιο συγκεκριμένα της εσωτερικής και εξωτερικής μνήμης. Εδώ παρέχεται ένα παράδειγμα, για να διευκρινιστεί καλύτερα αυτή η έννοια.

Στην Εικόνα 6.6, έχουμε δύο παραδείγματα ενός μοτίβου εκχώρησης/αποεκχώρησης, στο πάνω και στο κάτω μέρος. Σε κάθε μοτίβο, κατανέμεται ένα block των 20 KB από την εφαρμογή, που στην συνέχεια αποεκχωρείται, και μετά εκχωρείται από την εφαρμογή ένα block των 40 KB. Και στις δύο περιπτώσεις, το υποσύστημα DMM επιστρέφει το πρώτο block μνήμης, που μπορεί να βρεθεί και ταιριάζει με το απαιτούμενο μέγεθος. Στην πρώτη περίπτωση, το υποσύστημα DMM δεν τεμαχίζει το block των 40 KB που βρίσκει. Αυτό έχει ως αποτέλεσμα, λιγότερη διαθέσιμη ελεύθερη μνήμη από ότι θα είχαμε αν επέστρεφε ένα block των 20 KB. Αυτό ονομάζεται εσωτερικός κατακερματισμός και έχει ως αποτέλεσμα αχρησιμοποίητο χώρο που δεν θα μπορεί η εφαρμογή να χρησιμοποιήσει, όσο το block των 20 KB είναι εκχωρημένο. Το υποσύστημα DMM δεν μπορεί να επιστρέψει το εναπομείναν block των 20 KB, αφού θεωρείται χρησιμοποιημένο ολόκληρο το block των 40 KB, και η εφαρμογή δεν μπορεί να χρησιμοποιήσει το περισσεύουν χώρο, αφού το μόνο που γνωρίζει είναι πως έκανε εκχώρηση σε ένα block που μπορεί να χωρέσει 20 KB. Στην δεύτερη περίπτωση του σχήματος 6.6, το υποσύστημα DMM αποφασίζει να τεμαχίσει το ελεύθερο block των 40 KB για να διαχειριστεί τους εσωτερικούς κατακερματισμούς. Όμως, αυτό έχει ως αποτέλεσμα, η διαδοχική αίτηση για 40 KB, να μην βρει κανένα block διαθέσιμο. Σε μία τέτοια περίπτωση το υποσύστημα DMM πρέπει να ζητήσει περισσότερη μνήμη από το Λειτουργικό Σύστημα (ΛΣ) για να μπορέσει να υπακούσει στο αίτημα της εφαρμογής (με αποτέλεσμα μία μεγαλύτερη δεξαμενή block μνήμης). Την ίδια στιγμή, η μνήμη που απασχολείται από τα ελεύθερα block μνήμης των 20 KB παραμένει αχρησιμοποίητη και έτσι χάνεται. Αυτό ονομάζεται εξωτερικός κατακερματισμός. Σε αυτό το απλό παράδειγμα, το υποσύστημα DMM θα μπορούσε να έχει διαλέξει το block των 20 KB για την αίτηση των 20 KB και θα είχε απαλείψει και τα προβλήματα κατακερματισμού. Στα πραγματικά συστήματα, η εύρεση του πιο ταιριαστού block μπορεί να αποβεί ακριβής λόγω του χρόνου που χρειάζεται για να βρεθεί αυτό το block. Το δεύτερο DMM υποσύστημα θα μπορούσε να έχει συγχωνεύσει τα δύο block 20 KB σε ένα block των 40 KB. Αυτό όμως προκαλεί συμβιβασμούς κόστους στα πραγματικά συστήματα. Επι-

πλέον, η συγχώνευση μπορεί να έχει ως αποτέλεσμα αργότερα τον εσωτερικό ή εξωτερικό κατακερματισμό και έτσι η συγχώνευση σε συγκεκριμένες χρονικές στιγμές μπορεί να μην είναι η βέλτιστη.



Σχήμα 6.6: Παράδειγμα εξωτερικού και εσωτερικού κατακερματισμού.

Ενώ τα περισσότερα λειτουργικά συστήματα πάνε πακέτο με μία στάνταρ βιβλιοθήκη DMM [57]172, αυτές δεν βελτιστοποιούνται ποτέ για λόγους κατανάλωσης ενέργειας. Επιπλέον, δεν λαμβάνουν υπόψη τους, την συμπεριφορά της εφαρμογής σε σχέση με τις εκχωρήσεις/αποεκχωρήσεις μνήμης και των προσπελάσεων μνήμης. Στην παρουσία μίας ιεραρχίας μνήμης, σαν αυτές που βρίσκουμε στα ενσωματωμένα συστήματα, έχει συχνά ως αποτέλεσμα υποβέλτιστες επιλογές για τους εκχωρητές μνήμης [58]. Όπως ο σχεδιασμός των DDTs, ο σχεδιασμός του υποσυστήματος DMM πρέπει να λάβει υπόψη του αρκετούς παράγοντες πριν μπορέσει να θεωρηθεί χρήσιμο για κάποιο συγκεκριμένο σύστημα:

1. Το μοτίβο κατανομής της εφαρμογής στο πέρας του χρόνου άμεσα, αλλά και μέσω της χρήση DDT. Είναι σημαντικό το υποσύστημα DMM να είναι ικανό να εκχωρεί και να αποεκχωρεί αποδοτικά τα block που συχνά εκχωρούνται ή αποεκχωρούνται. Αυτή η αποδοτικότητα τυπικά μετριέται σχετικά και με την υπολογιστικότητα και τις προσπελάσεις μνήμης που εκτελεί κρυφά το DMM γι' αυτές τις εφαρμογές.

2. Το αποτύπωμα μνήμης του συγκεκριμένου εκχωρητή μνήμης δεν καθορίζεται μόνο από τα δυναμικά δεδομένα, τα οποία είναι φυσικά ένας παράγοντας που συμβάλλει στην πολυπλοκότητα του συστήματος, αλλά και από τον κατακερματισμό της μνήμης μέσα στον διαχειριστή δυναμικής μνήμης. Αν αυτό δεν τεθεί υπό έλεγχο, τα δυναμικά δεδομένα εύκολα μπορούν να καταναλώσουν όλους τους πόρους της συγκεκριμένης πλατφόρμας, υποβιβάζοντας έτσι ή και διαλύοντας εντελώς την προσδοκώμενη συμπεριφορά της εφαρμογής Έτσι, ο κατακερματισμός της μνήμης είναι ένας σημαντικός παράγοντας για την συνολική συμπεριφορά του συστήματος, που πρέπει να ληφθεί υπόψιν.

3. Το μοτίβο προσπέλασης της εφαρμογής στα δυναμικά δεδομένα και η δυναμική μνήμη που έχει κατανεμηθεί για χάρη του, καθορίζει την κατανάλωση ενέργειας της εφαρμογής. Με το να τοποθετούμε τα συχνά προσπελασμένα block σε τοπικές και μικρότερες μνήμες είναι δυνατό να έχουμε ένα τεράστιο κέρδος σε κατανάλωση ενέργειας.

Έχοντας αυτό το μοτίβο προσπέλασης κατά νου, το υποσύστημα DMM μπορεί να κάνει σίγουρη την χαμηλότερη κατανάλωση ενέργειας.

6.3 Προτεινόμενη Μέθοδος Βελτιστοποίησης

Όπως εξηγήσαμε στην Ενότητα 6.2 απαιτούνται διάφορες βελτιστοποιήσεις για την μείωση ενεργειακού κόστους που προκαλείται από τις δυναμικές προσπελάσεις μνήμης στην εφαρμογή. Αυτές οι βελτιστοποιήσεις δεν έχουν σκοπό μόνο τις εφαρμογές πολυμέσων που πρέπει να βελτιωθούν, αλλά και τις βιβλιοθήκες, και το υλικό και λογισμικό που χρησιμοποιούν αυτές οι εφαρμογές. Η μέθοδος για την προσέγγιση τέτοιων εφαρμογών είναι συνηθισμένη, όπως και τα βήματα που παράγουν τα συγκεκριμένα αποτελέσματα για κάθε εφαρμογή πολυμέσων. Με την μεταποίηση διάφορων κομματιών στις βιβλιοθήκες και τον χρόνο εκτέλεσης, είναι δυνατό να μειώσουμε σε σημαντικό ποσοστό τα κόστη ενεργειών της δυναμικής μνήμης σε αυτές τις εφαρμογές. Από την στιγμή που οι επιλύσεις είναι συγκεκριμένες για τις εν λόγω εφαρμογές, πρέπει να συλλεχθούν πρώτα οι πληροφορίες που έχουν σχέση με την εφαρμογή.

Θα περιγραφεί τώρα λεπτομερώς η συστηματική μεθοδολογία, η οποία εμφανίστηκε σε προηγούμενη ενότητα. Η μέθοδος που παρουσιάζεται σε αυτό το βιβλίο είναι ανεξάρτητη από την πλατφόρμα (π.χ. εξαρτάται από την περίπτωση οργάνωσης της ιεραρχίας των δεδομένων μνήμης που χρησιμοποιείται και από την πλατφόρμα, αλλά είναι ανεξάρτητη από την αρχιτεκτονική του επεξεργαστή). Εξαρτάται από τα μεγέθη μνήμης που υπάρχουν στο σύστημα, αφού έχει να κάνει με την κατανάλωση ενέργειας και της συμπεριφοράς της μνήμης.

Όμως, οι συμβιβασμοί των διαφορετικών βημάτων βελτιστοποίησης δεν εξαρτώνται από τα επακριβή κόστη ενέργειας της κάθε προσπέλασης μνήμης. Έτσι μόνο τα σχετικά κόστη ενέργειας, και κατά συνέπεια τα σχετικά μεγέθη των διαφορετικών επιπέδων μνήμης (L1, L2, κύρια μνήμη) της ιεραρχίας μνήμης είναι σχετικά, και όχι οι λεπτομέρειες της εν λόγω αρχιτεκτονικής.

Ενώ τα αποτελέσματα και οι ακριβείς τελικές επιλογές σχεδιασμού είναι εξαρτημένες από την πλατφόρμα, η μέθοδος είναι γενική, και εφαρμόζεται σε όλες τις πλατφόρμες, και μόνο οι λειτουργίες κόστους, και η τελική επιλογή σχεδιασμού για ένα συγκεκριμένο σχέδιο θα είναι εξαρτημένες από την πλατφόρμα, και όχι η εφαρμοσιμότητα της ίδιας της μεθόδου.

Εκεί που στις παραδοσιακές προσεγγίσεις βελτιστοποίησης, οι πληροφορίες αφορούσαν στατικά δεδομένα, τα οποία εκδηλώνονται στο πηγαίο κώδικα, σχετικά με τα μεγέθη πινάκων και τα όρια βρόχων, η εισαγωγή δυναμικών δεδομένων αφαιρεί αυτή την πηγή πληροφοριών. Και ενώ μπορούν ακόμα να ταυτοποιηθούν οι βρόχοι παραγωγής, λόγω της προσαρμοστικότητας στο μέγεθος που διαθέτουν τα DDTs (που εξυπηρετούν και ως αποθήκη γι' αυτά τα δυναμικά δεδομένα) σε χρόνο εκτέλεσης, δεν είναι πλέον δυνατό να ταυτοποιήσουμε τις ποσότητες των εν λόγω δεδομένων. Έτσι, οι πληροφορίες που έχουν να κάνουν με τη συμπεριφορά δυναμικών δεδομένων πρέπει να συλλεχθούν κατά το χρόνο εκτέλεσης. Αυτό έχει ως αποτέλεσμα, η μέθοδος να αρχίζει με το βήμα συλλογής μετα-δεδομένων, πριν εφαρμοστούν τα διάφορα βήματα βελτιστοποίησης, που καταλήγει σε μία μέθοδο ροής που διαχειρίζεται την δυναμική μνήμη [59].

Τώρα θα εξετάσουμε αυτή τη μέθοδο ροής, γιατί είναι δομημένη και οργανωμένη όπως είναι, και εξηγώντας τι περιέχουν τα διάφορα βήματα. Πρώτα, η συνολική μέθοδος εξηγείται στην Ενότητα 6.3.1. Μετά, το βήμα που συλλέγει τις πληροφορίες που απαιτούνται για τα διαφορετικά βήματα βελτιστοποίησης εξηγείται στην Ενότητα 6.3.2. Τελικά, τα διαφορετικά βήματα βελτιστοποίησης περιγράφονται στις Ενότητες 6.3.3 και 6.3.5.

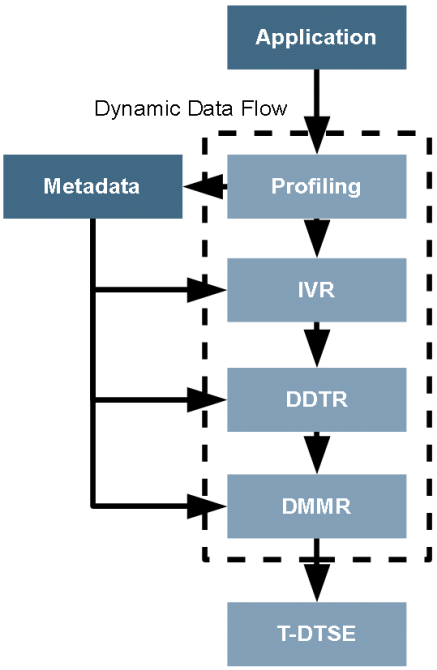
6.3.1 Περίληψη Μεθόδου

Σ' αυτήν την ενότητα δίνουμε μία περίληψη της μεθόδου, εξηγώντας την σειρά των διαφορετικών βημάτων και γιατί αυτή είναι η πιο λογική σειρά. Οι εξηγήσεις των διάφορων βημάτων περιγράφονται περαιτέρω στις ακόλουθες υποενότητες.

Όπως βλέπουμε στο Σχήμα 6.7, το πρώτο βήμα της μεθόδου είναι η συλλογή πληροφοριών σε σχέση με την συμπεριφορά της εφαρμογής. Αυτό που δεν είναι εμφανές στο σχήμα είναι οι βελτιστοποιήσεις υψηλού επιπέδου που είναι δυνατές στον αντικειμενοστραφή προγραμματισμό και στη γλώσσα περιγραφής υψηλού επιπέδου (π.χ. Ενοποιημένη Γλώσσα Μοντελισμού ή Unified Modelling Language, UML [60]). Αυτά πρέπει να γίνουν πρώτα, αφού μπορούν

να τροποποιήσουν εντελώς το σχεδιασμό της εφαρμογής και των υπαρχόντων DDT. Μπορούν να αποσυνδεδεθούν μετά από ακόλουθα βήματα, με την εισαγωγή εκτιμητών υψηλού επιπέδου [52].

Το βήμα πληροφοριών ιδιαίτερων χαρακτηριστικών, συλλέγεται στην μορφή μετα-δεδομένων λογισμικού, περιγράφοντας την χρήση DDTS και δυναμικών δεδομένων γενικά, και σε σχέση με εκχωρήσεις και αποεκχωρήσεις, όπως και προσπελάσεις, και συνοψίζονται σε σχέση με μία ποικιλία αξόνων. Αυτό το βήμα πληροφοριών μετά κληροδοτείται σε διάφορα βήματα βελτιστοποίησης, κάνοντας τα δυνατά. Το βήμα ανάλυσης και καταγραφής ιδιαίτερων χαρακτηριστικών εξηγείται με περισσότερες πληροφορίες στην Ενότητα 6.3.2.



Σχήμα 6.7: Σύνοψη της προτεινόμενης μεθοδολογίας.

Μετά την συλλογή των πληροφοριών, είναι δυνατό να έχουμε μία ολιστική βελτιστοποίηση σε σχέση με την χρήση DDTs στο επίπεδο αλγόριθμου της εφαρμογής. Τυπικά, τα DDTs χρησιμοποιούνται με δύο τρόπους: ως κεντρική αποθήκη για τα στοιχεία σε έναν αλγόριθμο, εξυπηρετώντας έτσι ως η κατάσταση του αλγόριθμου, ή σαν φορτωτής για την αποσύνδεση φάσεων παραγωγής και κατανάλωσης. Στην δεύτερη περίπτωση, η οποία συχνά παρατηρείται σε εφαρμογές πολυμέσων, η χρήση DDTs έχει ως σκοπό την αποφυγή πολυπλοκότητας και τη βελτίωση της καθαριότητας. Σε τέτοιες περιπτώσεις, είναι δυνατό να βελτιστοποιήσουμε συστηματικά αυτά τα DDTs, με ένα πρωτόγνωρο

βήμα βελτιστοποίησης την Αφαίρεση Ενδιαμέσου Μεταβλητής (Intermediate Variable Removal, IVR). Αυτό το βήμα κάνει δυνατό ένα συμβιβασμό μεταξύ του εύρους ζώνης υπολογισμών και μνήμης. Ανάλογα με την πλατφόρμα και τα σχετικά κόστη προσπελάσεων μνήμης και υπολογισμών, αυτό επιτρέπει βελτιστοποιήσεις που έχουν να κάνουν με κατανάλωση ενέργειας. Ένα μικρό παράδειγμα IVR παρουσιάζεται εδώ, για να δώσει μια ιδέα στον αναγνώστη, ως προς το τι περιέχει. Παραπάνω πληροφορίες για το IVR στην Ενότητα 6.3.3.

Στο πρώτο δείγμα κώδικα του Σχήματος 6.8, δημιουργείται ένα DDT στο οποίο προστίθεται μία ακολουθία στοιχείων (παραγόμενα από μια διαδικασία). Μετά αυτή η ακολουθία στοιχείων καταναλώνεται σε ένα δεύτερο βρόχο. Αν οι δύο βρόχοι είχαν γραφεί ταυτόχρονα, όπως στο δεύτερο δείγμα κώδικα στο Σχήμα 6.8, τα κόστη καταγραφής όλων αυτών των στοιχείων στη μνήμη (όπως και η κράτηση λογαριασμών στα DDT, απαραίτητα για την ανάπτυξη της ακολουθίας), όπως και η ανάγνωση από την μνήμη δεν θα ήταν απαραίτητα. Αντ'αυτού κάθε στοιχείο θα μπορούσε να είχε γραφεί στην τοπική μνήμη (ή ακόμα και σε καταχωρητή) και μετά θα είχε καταναλωθεί, έχοντας ως αποτέλεσμα πολύ μικρότερα κόστη ενέργειας, αφού οι μεγαλύτερες μνήμες απαιτούν περισσότερη ενέργεια. Επιπλέον, η κράτηση λογαριασμών DDT θα μπορούσε να έχει απαλειφθεί οριστικά. Το IVR εστιάζει σε τέτοιες μεταποιήσεις, κάνοντας ταυτόχρονα σίγουρη την τήρηση περιορισμών σε σχέση με την οργάνωση εισόδου/εξόδου, και ασχολείται με πιο πολύπλοκους βρόχους παραγωγής και κατανάλωσης.

Μόνο όταν έχει οριστεί ποια DDTs θα παραμείνουν, είναι αποδοτικό να ξεκινήσει η βελτιστοποίηση της εφαρμογής τους σε ένα βέλτιστο Pareto[1] σχεδιασμό, έχοντας υπόψιν μας την πλατφόρμα στην οποία αποτυπώνεται η εφαρμογή, όπως και η κατά συνέπεια συμπεριφορά της εφαρμογής, σχετικά με τα DDTs. Οι βελτιστοποιήσεις DDT δεν περιγράφονται περαιτέρω σε αυτό το βιβλίο. Αντ'αυτού, αναφέρονται στα [61], [52]. Για ακόμα μία φορά, εκτιμητές υψηλού επιπέδου χρησιμοποιούνται για την αποσύνδεση αυτού του βήματος από τους συγγραφείς των [52]. Παίρνοντας ξανά το παράδειγμα 6.8, αν το ενδιάμεσο DDT δεν είχε αφαιρεθεί, στην ιδανική περίπτωση θα είχε χρησιμοποιηθεί ένα DDT που μοιάζει σε λίστα, επειδή έχει οδηγήσει σε λιγότερες αντιγραφές, ενώ συνέχεια προσθέτει καινούργια στοιχεία στο τέλος του.

Τέλος, όταν οι εφαρμογές των DDTs έχουν καθοριστεί, θα είναι ξεκάθαρο ποια block είναι κατανεμημένα από αυτά τα DDTs, αλλά και την ίδια την εφαρμογή. Αφού οι εφαρμογές DDT καθορίζουν τις τεχνικές εσωτερικής αποθήκης στα block μνήμης, είναι απαραίτητο να έχουμε την τελική εφαρμογή των DDTs, για να δούμε πως είναι η συμπεριφορά κατανομής της εφαρμογής. Για παράδειγμα, μία μικρή εναλλαγή, όπως η αλλαγή της ακολουθίας στυλ vector σε

[1]Στο διάγραμμα pareto απεικονίζονται όλες οι βέλτιστες υλοποιήσεις με τους συμβιβασμούς.

```
vector<int> a;
for (int i = 0; i < SIZE; i++) {
  if (h(i))
    vector.push_back(f(i));
}
for (vector<int>::iterator i = a.begin(); i != a.end(); ++i) {
  g(*i);
}

for (int i = 0; i < SIZE; i++) {
  if (h(i))
    g(f(i));
}
```

Σχήμα 6.8: Παράδειγμα αφαίρεσης της ενδιάμεσης μεταβλητής.

μία ακολουθία σε στυλ διασυνδεδεμένης λίστας, όχι μόνο αλλάζει ριζικά το μοτίβο ανακατανομής, αλλά και τους τύπους των block που κατανέμονται. Στην περίπτωση του vector, θα κατανεμηθεί ένα συνεχόμενο block μνήμης που περιέχει τα στοιχεία, με αποτέλεσμα μία μεγάλη κατανομή ή αρκετές αφού σιγά σιγά ο vector μεγαλώνει και το block μνήμης επεκτείνεται και έτσι ανακατανέμεται. Από την άλλη, αν χρησιμοποιηθεί μία διασυνδεδεμένη λίστα, πολλά μικρά block θα κατανεμηθούν, που το καθένα θα περιέχει ένα και μόνο ένα στοιχείο και τους δείκτες προς το προηγούμενο και το επόμενο block, με μία κατανομή να συμβαίνει κάθε φορά, που προστίθεται ένα στοιχείο. Έτσι, είναι ξεκάθαρο πως η εφαρμογή DDT έχει μεγάλη επιρροή στο μοτίβο κατανομής. Γι' αυτό το βήμα βελτιστοποίησης DDM είναι μετά το βήμα βελτιστοποίησης DDT. Περισσότερες πληροφορίες σε σχέση με την βελτιστοποίηση DDM δίνονται στην Ενότητα 6.3.5.

Σημειώνουμε πως όλα τα παραπάνω βήματα έχουν να κάνουν με δυναμικά δεδομένα και την αποτύπωση τους σε δεξαμενές μνήμης. Μετά από αυτά τα βήματα, είναι δυνατό να συνδυάσουμε αυτές τις αφηρημένες δεξαμενές δεδομένων με γενικά δεδομένα και δεδομένα στοίβας. Αυτό είναι έξω από την θεματολογία αυτού του βιβλίου, αλλά δείχνει πως η προηγούμενη μέθοδος συνδέεται με την υπάρχουσα μελέτη των στατικών δεδομένων. Περισσότερες πληροφορίες στην Ενότητα 6.3.6

6.3.2 Συλλογή Καταγραφών Ιδιαιτέρων Στοιχείων και Μετα-δεδομένων

Για να γίνουν δυνατά τα διάφορα βήματα βελτιστοποίησης, όπως έχει ήδη εξηγηθεί στην ενότητα 6.3, είναι αναγκαίο να κατανοηθεί τι είναι η συμπεριφορά δυναμικής μνήμης της εφαρμογής. Για αυτό τον σκοπό, αφού η εν λόγω εφαρμογή είναι δυναμική, είναι αναγκαίο να καταγραφούν τα ιδιαίτερα χαρακτηριστικά της, για να μπορούμε να έχουμε μία ακριβή εικόνα τον διαφορετικών απαιτήσεων της εφαρμογής σε σχέση με τις προσπελάσεις μνήμης και τις κατανομές μνήμης. Επιπλέον, συλλέγονται πληροφορίες καταγραφής ιδιαιτέρων στοιχείων υψηλού επιπέδου αφαίρεσης, όπως η χρήση διαφόρων μεθόδων που παρέχονται από ένα DDT. Με το να χρησιμοποιήσουμε μία στάνταρ διεπαφή (π.χ. η διεπαφή ακολουθίας από το STL [49]), είναι δυνατή η μελέτη συμπεριφοράς της εφαρμογής σε σχέση με αυτήν την διεπαφή και έπειτα να βελτιστοποιήσουμε την εφαρμογή DDT βάσει πώς χρησιμοποιούνται τα DDTs. Η βιβλιοθήκη καταγραφής ιδιαίτερων στοιχείων που έχει αναπτυχθεί μπορεί να συλλέξει όλες αυτές τις πληροφορίες χωρίς να απαιτεί δραστικές αλλαγές στον πηγαίο κώδικα της εφαρμογής [62].

Αφού συλλεχθούν τα ακατέργαστα δεδομένα καταγραφής ιδιαιτέρων στοιχείων, αναλύονται και συνοψίζονται σε μεταδεδομένα που περιγράφουν τη συμπεριφορά μνήμης της εφαρμογής σε υψηλό επίπεδο. Τυπικές πληροφορίες που θα εντοπισθούν εδώ, είναι πώς χρησιμοποιούνται τα διαφορετικά DDTs της εφαρμογής, ποια είναι η τυπική συμπεριφορά κατανομής της εφαρμογής, που συμβαίνουν οι περισσότερες προσπελάσεις, κλπ. Εκτελώντας αυτή την καταγραφή ιδιαιτέρων στοιχείων και την ανάλυση για μία ομάδα αντιπροσωπευτικών πληροφοριών εισόδου, είναι δυνατό να αποκτήσουμε μια πιο ξεκάθαρη εικόνα της συμπεριφοράς της εφαρμογής. Με αυτές τις πληροφορίες, μπορούμε να ταυτοποιήσουμε ενδιάμεσα DDTs που μπορούν να αφαιρεθούν, να βελτιστοποιήσουμε την εφαρμογή των εναπομεινάντων DDTs και να ορίσουμε έναν βέλτιστο εκχωρητή μνήμης για την εκάστοτε πλατφόρμα εφαρμογής και υλικού.

6.3.3 Αφαίρεση Ενδιάμεσης Μεταβλητής

Πριν να οριστούν ποιες θα είναι οι εφαρμογές των διάφορων DDT στην εφαρμογή, είναι αναγκαίο να αποφασιστεί ποια DDTs απαιτούνται για την εκτέλεση του αλγορίθμου. Από άποψη σχεδιασμού, είναι καλή ιδέα να αποσυνδέσουμε τις δηλώσεις παραγωγής από τις δηλώσεις κατανάλωσης, λόγω του ξεκάθαρου διαχωρισμού που δημιουργεί με μία στάνταρ διεπαφή, πιο συγκεκριμένα τα ενδιάμεσα DDTs μεταξύ των βημάτων της εφαρμογής. Μερικά DDTs απαιτούνται από μία θεωρητική άποψη και μόνο, επειδή μπορούν να κω-

δικοποιήσουν στην κύρια κατάσταση ενός συγκεκριμένου κομματιού της εφαρμογής, όπου η εφαρμογή ανακυκλώνεται μέσα σε ένα βρόχο και ενημερώνει ανά τακτά διαστήματα την κατάσταση. Από την άλλη, άλλα DDTs υπάρχουν μόνο σε συγκεκριμένες φάσεις ως προσωρινοί διαμεσολαβητές μεταξύ δύο βημάτων του αλγορίθμου.

Από άποψη παραδείγματος αρθρωτού προγραμματισμού και χρήσης των τύπων αφηρημένων δεδομένων [63], [64], που όλα αυτά χρησιμοποιούνται σε όλες τις μοντέρνες γλώσσες προγραμματισμού, η λειτουργικότητα του οποιουδήποτε προγράμματος πρέπει να αποσυνδεθεί και να οριστεί ανεξάρτητα από την συγκεκριμένη εφαρμογή των δομών δεδομένων (δυναμικές ή στατικές) που χρησιμοποιούνται για την αποθήκευση των δεδομένων της εφαρμογής, όσο είναι διαθέσιμες όλες οι μέθοδοι και οι συναρτήσεις πρόσβασης που απαιτούνται από τον εφαρμοσμένο αλγόριθμο [65]. Σύμφωνα με αυτό το παράδειγμα προγραμματισμού λογισμικού, η προτεινόμενη μέθοδος βασίζεται στην υπόθεση ότι τα DDT μπορούν να ταυτοποιηθούν στον αρχικό πηγαίο κώδικα της εξεταζόμενης εφαρμογής, και ότι μπορούν να αντικατασταθούν χωρίς παράπλευρες συνέπειες που απαιτούν τροποποιήσεις της ροής ελέγχου της εφαρμογής. Με άλλα λόγια, ο πηγαίος κώδικας εισόδου της εφαρμογής πρέπει να μην περιέχει εφαρμογές DDT εκεί που εφαρμόζεται η εν λόγω μέθοδος, και πρέπει να έχει οριστεί σε μία ξεχωριστή ενότητα ή στάνταρ βιβλιοθήκη εφαρμογών DDT στην C++, όπως η STL. Ακόμα, όλος ο αλγόριθμος που χρησιμοποιεί τις εφαρμογές DDT πρέπει να αλληλεπιδράσει μ' αυτές μόνο μέσω των στάνταρ και κοινών σετ μεθόδων [66].

Πρέπει να σημειωθεί ότι ο προαναφερθείς περιορισμός δεν περιορίζει σε μετρήσιμο βαθμό την εφαρμοσιμότητα της προτεινόμενης μεθόδου, αφού όλες οι εξεταζόμενες εφαρμογές και οι περισσότερες σύγχρονες εφαρμογές, ακολουθούν αυτή την υπόθεση. Επίσης, όπως υιοθετείται το παράδειγμα αντικειμενοστραφούς προγραμματισμού ως το κοινό στάνταρ για την εκμετάλλευση του παραδείγματος συστήματος πάνω σε chip σε επίπεδο λογισμικού, αυτή η προηγούμενη υπόθεση θα χρειαστεί να επιβληθεί περαιτέρω, λόγω της αναγκαιότητας ορισμού και χρήσης βιβλιοθηκών πολλαπλών χρήσεων, που βασίζονται σε στάνταρ διεπαφές.

Η βελτιστοποίηση IVR χρησιμοποιεί πληροφορίες από το βήμα καταγραφής ιδιαίτερων χαρακτηριστικών, για να προσδιορίσει ποια DDT χρησιμοποιούνται με τρόπο παραγωγού- καταναλωτή. Αυτός είναι ένας από τους λόγους που το βήμα καταγραφής ιδιαιτέρων στοιχείων δεν καταγράφει μόνο τη συμπεριφορά της μνήμης, αλλά και την συμπεριφορά της εφαρμογής, σε σχέση με αφηρημένες λειτουργίες DDT, όπως αυτές ορίζονται από την διεπαφή του DDT. Μόνο όταν είναι γνωστό ποια είναι τα εναπομείναντα DDT, είναι δυνατό να συνεχίσουμε στο επόμενο βήμα, την βελτιστοποίηση των DDT, η οποία δεν παρουσιάζεται με περισσότερες λεπτομέρειες σ' αυτό το βιβλίο. Περισσότερες πληροφορίες σε

σχέση με τις βελτιστοποιήσεις των DDT μπορούν να βρεθούν στο [52].

6.3.4 Βελτίωση Δυναμικού Τύπου Δεδομένων

Τυπικά, υπάρχει ένας συμβιβασμός μεταξύ DDTs που κατά κύριο λόγο προσπελάζονται με τυχαίο τρόπο, και DDT που προσπελάζονται διαδοχικά. Δεν είναι δυνατή η σχεδίαση DDTs, τα οποία θα υποστηρίζουν και τους δύο τύπους εργασιών στο O(1). Για να κατανοήσει ο αναγνώστης το συμβιβασμό, ο Πίνακας 6.2 δίνει μερικές στάνταρ εφαρμογές που παρέχονται από τις περισσότερες γενικές βιβλιοθήκες. Πρέπει να σημειωθεί πως αυτοί οι συμβιβασμοί βρίσκονται στο ελάχιστο της πολυπλοκότητας (η οποία μεταφράζεται και σε προσπελάσεις μνήμης και σε υπολογισμούς).

Πίνακας 6.2: Τυπικοί συμβιβασμοί πολυπλοκότητας μεταξύ διαδοχικών εφαρμογών που συναντώνται συχνά σε γενικές βιβλιοθήκες

Εφαρμογή	Τυχαία προσπέλαση	Append	Prepend	Εισαγωγή	Αφαίρεση
Διάνυσμα	O(1)	O(1)	O(N)	O(N)	O(N)
Λίστα	O(N)	O(1)	O(1)	O(1)	O(1)
Fingertree	O(log N)	O(1)	O(1)	O(log N)	O(log N)

Όμως, άμα θεωρήσουμε την κατανάλωση ενέργειας, πρέπει να ληφθούν υπόψιν η αποτύπωση μνήμης, όπως και άλλοι παράγοντες. Αυτοί οι παράγοντες παρουσιάζονται λεπτομερώς παρακάτω.

Η εφαρμογή των εργασιών των DDT εξαρτάται από το επιλεγμένο δοχείο και κάθε ένα από αυτά μπορεί να προτιμά συγκεκριμένο τρόπο προσπέλασης δεδομένων και των μοτίβων αποθήκευσης. Κάθε εφαρμογή μπορεί να φιλοξενήσει ένα συγκεκριμένο νούμερο διαφορετικών δοχείων σύμφωνα με το συγκεκριμένο μοτίβο προσπέλασης δεδομένων της και αποθήκευσης στον αλγόριθμο. Διαλέγοντας μια ακατάλληλη εφαρμογή δοχείου για ένα αφηρημένο τύπο δεδομένων θα έχει σημαντικές αρνητικές επιπτώσεις στο υποσύστημα δυναμικής μνήμης του ενσωματωμένου συστήματος [61] [52]. Από την μία, η ανεπαρκής προσπέλαση δεδομένων και οι ανεπαρκείς διεργασίες μνήμης μπορούν να δημιουργήσουν προβλήματα στις επιδόσεις, λόγω της επιπρόσθετης υπολογιστικής επιβάρυνσης των εσωτερικών DDT μηχανισμών. Από την άλλη, κάθε προσπέλαση των DDT στην φυσική μνήμη (όπου αποθηκεύονται τα δεδομένα) καταναλώνει ενέργεια. Οπότε, οι μη αναγκαίες προσπελάσεις μπορούν να αποτελέσουν ένα σημαντικό ποσοστό απορρόφησης ρεύματος του συστήματος [67], [68]. Στην πραγματικότητα, η κατανάλωση ενέργειας είναι ένας από τους πιο καθοριστικούς παράγοντες στο πόσο λειτουργικές μπορεί να είναι τέτοιες συσκευές, γιατί φορητοί υπολογιστές όπως τα tablets και τα notebooks στηρίζονται σε περιορισμένη μπαταρία για την λειτουργία τους. Έτσι, ο σχεδιασμός του

DDT για ένα συγκεκριμένο φορητό ενσωματωμένο σύστημα και μία εφαρμογή πολυμέσων πρέπει να θεωρήσει πολλούς συνδυασμένους παράγοντες:

1. Ο αλγόριθμος εφαρμοσμένος σε σχέση με υψηλού επιπέδου τελεστές τους οποίους το DDT παρέχει. Αφού διαφορετικές εφαρμογές έχουν διαφορετικά κόστη για το εύρος των διεργασιών, αυτός είναι ένας από τους κύριους παράγοντες. Ενώ συγκεκριμένες εφαρμογές μπορεί να έχουν 0(1) προσπέλαση, τυπικά θα απαιτούν 0(N) για εισαγωγή, ενώ τον 'N' έχει το μέγεθος του DDT. Υπάρχει πληθώρα επιλογών, αλλά καμία διεργασία DDT έχει O(1) για όλες τις διεργασίες και έτσι υπάρχει ένας συμβιβασμός μεταξύ των διαφορετικών τελεστών. Αυτό το κόστος εκφράζεται αναφορικά με τους υπολογισμούς, αλλά και με τον αριθμό προσπελάσεων, και έχει σημαντικές επιπτώσεις στην γενική κατανάλωση ενέργειας

2. Η απαίτηση μνήμης των DDT εφαρμογών ποικίλει βάσει της επιλεγμένης εφαρμογής. Συγκεκριμένες εφαρμογές απαιτούν παραπάνω δείκτες για την δυνατότητα διαχείρισης της διαθέσιμης μνήμης. Άλλες δεν απαιτούν τέτοια επιβάρυνση, αλλά δεν επιτρέπουν τον διαχωρισμό μνήμης που χρησιμοποιείται από τα στοιχεία σε μικρότερα block και έτσι φθηνότερης μνήμης. Αυτοί οι συμβιβασμοί έχουν μεγάλη επιρροή και στο αποτύπωμα μνήμης και στην αποτύπωση στην ιεραρχία της μνήμης, και έμμεσα στην κατανάλωση ενέργειας.

3. Η συμπεριφορά προσπέλασης και μετατόπισης των στοιχείων μέσα σε ένα DDT, και πιο συγκεκριμένα η κατανάλωση δεδομένων, επίσης έχει μεγάλη επιρροή στην επιλογή σχεδιασμού ενός DDT. Όπως είδαμε, τα DDT μπορούν να καταναλωθούν με διάφορους τρόπους. Αυτό μπορεί να έχει σοβαρές συνέπειες και σε σχέση με την πολυπλοκότητα στους υπολογισμούς και στην προσπέλαση μνήμης που απαιτείται για τέτοιες μετατοπίσεις, όπως και στη συμπεριφορά αποθήκευσης στην κρυφή μνήμη του εν λόγω DDT.

6.3.5 Βελτιστοποιήσεις Διαχειριστή Δυναμικής Μνήμης

Το τελικό βήμα βελτιστοποίησης για τα δυναμικά δεδομένα, είναι η βελτιστοποίηση του εκχωρητή δυναμικής μνήμης. Αυτό το βήμα έρχεται μετά από την βελτιστοποίηση των εφαρμογών του DDT, αφού αυτές θα επηρεάσουν το μοτίβο κατανομής και προσπέλασης της εφαρμογής. Στο βήμα βελτίωσης διαχειριστή μνήμης (DDMR), χρησιμοποιούνται από τα μεταδεδομένα καταχωρημένων χαρακτηριστικών, πληροφορίες για την εφαρμογή σε σχέση με τα μοτίβα κατανομής αλλά και προσπέλασης στα κατανεμημένα block. Έτσι, καθορίζεται ποιοι τύποι block μνήμης χρησιμοποιούνται λιγότερο και ποιοι περισσότερο.

Με αυτές τις πληροφορίες, είναι δυνατό να καθοριστεί πως πρέπει να δομηθεί ο εκχωρητή μνήμης σε σχέση με δύο συγκεκριμένους παράγοντες.

Πρώτον, βοηθάει με την σχεδίαση της δομής του εκχωρητή μνήμης. Αν και πολλά διαφορετικά μεγέθη block χρησιμοποιηθούν από την εφαρμογή, τότε ο διαχωρισμός και ο συγκερασμός είναι σίγουρα ωφέλιμος, αφού αλλιώς ένα μεγάλο κομμάτι του αποτυπώματος μνήμης θα χαθεί λόγω κατακερματισμού. Επιπλέον, βοηθάει στο να οριστεί ποια block μνήμης είναι συχνά κατανεμημένα, έτσι επιτρέποντας την λήψη απόφασης ως προ το ποια block μνήμης θα πρέπει να έχουν τις δικές τους προσαρμοσμένες ελεύθερες λίστες για γρήγορη κατανομή αυτών των block.

Δεύτερον, βοηθάει με την τοποθέτηση block μνήμης επάνω στις δεξαμενές μνήμης. Οι δεξαμενές μνήμης τυπικά είναι μεγάλα block μνήμης τα οποία τοποθετούνται σε διαφορετικά στοιχεία της ιεραρχίας της μνήμης. Η πιο απλή προσέγγιση είναι να υπάρχει μία δεξαμενή μνήμης ανά ιεραρχικό στοιχείο block που συχνά προσπελάζονται και πρέπει μετά να τοποθετηθούν σε μικρότερες δεξαμενές μνήμης, που βρίσκονται πιο κοντά στον επεξεργαστή (CPU), ενώ block που δεν προσπελάζονται τόσο συχνά πρέπει να τοποθετούνται σε δεξαμενές μνήμης μακριά από τον επεξεργαστή. Ένα καλό μέτρο είναι ο αριθμός προσβάσεων ανά byte, από την στιγμή που δεν βγάζει νόημα το να τοποθετήσουμε ένα συχνά προσπελάσιμο block σε μία μικρότερη δεξαμενή μνήμης αν αυτό το block είναι μεγαλύτερο από δύο δεξαμενές που μαζί έχουν περισσότερες προσπελάσεις.

Για να επιτραπεί αυτό το βήμα βελτιστοποίησης, απαιτείται κάποια εξερεύνηση ώστε να βρούμε τις ακριβείς παραμέτρους που μας δίνουν τον βέλτιστο κατανομέα μνήμης. Έτσι, είναι απαραίτητη η ύπαρξη βιβλιοθήκης συνδυασμών που επιτρέπουν την κάλυψη του χώρου σχεδιασμού των εκχωρητών μνήμης. Αυτό το σύνολο συνδυασμών θα πρέπει να είναι εύκολα συναρμολογούμενο, ώστε οι εκχωρητές μνήμης οποιαδήποτε πολυπλοκότητας να μπορούν χτιστούν αντ'αυτών.

6.3.6 Μεταφορά Δεδομένων και Εξερεύνηση Μνήμης σε Εργασιακό Επίπεδο

Μετά την ολοκλήρωση των προηγούμενων βημάτων που είναι ανεξάρτητα της αρχιτεκτονικής, υπάρχει μια βέλτιστη αντιστοίχηση τύπων δυναμικών δεδομένων της εφαρμογής σε δεξαμενές μνήμης. Χρησιμοποιώντας δεξαμενές μνήμης που προκύπτουν από αυτά τα δεδομένα σωρού, μαζί με την κατανάλωση μνήμης και προσπελάσεων λόγω δεδομένων στοίβας και σωρού, είναι δυνατό να αποτυπωθεί σε μία πραγματική ιεραρχία μνήμης, λαμβάνοντας υπόψιν τις αλληλεπιδράσεις διάφορων ομάδων διεργασιών. Αυτή η τελική φάση, ονο-

μαζόμενη Μεταφορά Δεδομένων και Εξερεύνηση Μνήμης σε Εργασιακό Επίπεδο(Data Transfer and Storage Exploration, T-DTSE), αναλύεται στο [69]. Όμως το κομμάτι που καλύπτεται σε αυτό το βιβλίο, έχει σχέση με την φυσική κατανομή μνήμης και ανάθεση μνήμης και είναι συμπληρωματικό του T-DTSE, αφού δίνει την δυνατότητα εφαρμογής των εντολών προσπέλασης. Ταυτόχρονα, συνδυάζεται με διάφορες επιτρεπόμενες τροποποιήσεις πηγαίου κώδικα, εξαρτημένες από την αρχιτεκτονική.

6.4 Συμπεράσματα

Σε αυτό το κεφάλαιο είδαμε πως οι εφαρμογές πολυμέσων τελευταίας τεχνολογίας, με στόχο τα νέα φορητά ενσωματωμένα συστήματα (π.χ. 3D παιχνίδια, video-players) μοιράζονται διάφορα χαρακτηριστικά σε σχέση με την εκτέλεση πολλαπλών διεργασιών και την πολύπλοκη διαχείριση μνήμης, που τις καθιστά τις τέλειες υποψήφιες για βελτιστοποιήσεις δυναμικών δεδομένων. Αυτές οι πολύπλοκες και δυναμικές εφαρμογές προέρχονται από τα αρχικά επιτραπέζια συστήματα, τα οποία έχουν πολλή περισσότερη μνήμη και δυνατότητες. Επομένως, έχουμε αναλύσει σε αυτό το κεφάλαιο τα χαρακτηριστικά των δυναμικών δεδομένων σε αυτή την ομάδα καινούργιων εφαρμογών πολυμέσων, και έχουμε προτείνει ένα καινούργιο σχεδιασμό ροής, για να βελτιστοποιήσουμε και να εγκαθιδρύσουμε τη διαχείριση δυναμικής μνήμης στα φορητά ενσωματωμένα συστήματα.