Κεφάλαιο 7

Σκιαγράφηση και ανάλυση των δυναμικών εφαρμογών

Όπως έχει εξηγηθεί στο προηγούμενο κεφάλαιο, το πρόβλημα που αφορά τη βελτιστοποίηση του σχεδιασμού των δυναμικών ενσωματωμένων συστημάτων είναι ότι προσπαθούν να εκμεταλλευτούν όσο το δυνατόν περισσότερο τη στατική γνώση σχετικά με τις εφαρμογές (κατά το χρόνο σχεδιασμού), αλλά και να αφήσουν χώρο για τις εκτιμήσεις χρόνου εκτέλεσης που επιτρέπουν να αντιμετωπίσουν τις δυναμικές παραλλαγές, αποκλείοντας τη χειρότερη περίπτωση λύσεων. Αυτό απαιτεί τις εκτενείς πληροφορίες για τα στατικά και δυναμικά χαρακτηριστικά των εφαρμογών. Το κεφάλαιο περιγράφει τις μεθόδους σκιαγράφησης και ανάλυσης, προκειμένου να ληφθούν αυτές οι πληροφορίες και να αποτελέσουν τα μεταδεδομένα του προγράμματος. Αυτά θα καθοδηγήσουν στη βέλτιστη επιλογή του δυναμικού διαχειριστή μνήμης.

7.1 Εισαγωγή

Δεν υπάρχει ένας τυποποιημένος ορισμός ή μια αναπαράσταση των μεταδεδομένων λογισμικού, για να απεικονίσει τα χαρακτηριστικά της δυναμικής συμπεριφοράς πρόσβασης στοιχείων των εφαρμογών ,υποκείμενων στις ποικίλες εισόδους. Το σχήμα 7.1 επεξηγεί πώς η χρήση μιας κοινής αποθήκης πληροφοριών (μεταδεδομένα λογισμικού), μπορεί να μειώσει τη γενική προσπάθεια που απαιτείται για να εφαρμοστούν οι διαφορετικές τεχνικές βελτιστοποίησης σε μια εφαρμογή. Χωρίς μεταδεδομένα, κάθε ομάδα βελτιστοποίησης πρέπει να μελετήσει την εφαρμογή για να εξαγάγει ανεξάρτητα τα χαρακτηριστικά που είναι σχετικά για την εργασία τους. Αν και οι πληροφορίες που απαιτούν οι περιοχές μπορούν να είναι ελαφρώς διαφορετικές για κάθε ομάδα, είναι επίσης αρκετά πιθανό να υπάρχουν και κοινές. Επομένως, ακόμα κι αν η διαδικασία

εξαγωγής μεταδεδομένων μπορεί να είναι πιο σύνθετη από τις πληροφορίες που απαιτούνται για κάθε μια από τις ομάδες στην απομόνωση, η συσσωρευμένη προσπάθεια μπορεί να μειωθεί σημαντικά. Επιπλέον, μόλις εξαχθούν τα μεταδεδομένα, είναι διαθέσιμα χωρίς κανένα κόστος για οποιεσδήποτε πρόσφατες εργασίες βελτιστοποίησης. Σε αυτό το κεφάλαιο προτείνεται μια ομοιόμορφη αναπαράσταση των δυναμικών στοιχείων πρόσβασης και της συμπεριφοράς δέσμευσης των εφαρμογών, που ορίζονται ως τα μεταδεδομένα λογισμικού. Επιπλέον, αυτό το κεφάλαιο περιγράφει τις μεθόδους σκιαγράφησης και ανάλυσης για να λάβει συστηματικά αυτά τα μεταδεδομένα. Τα επιλεγμένα μεταδεδομένα παρέχουν μια ισόπεδη αναπαράσταση συστήματος της συμπεριφοράς των δυναμικών ενσωματωμένων εφαρμογών λογισμικού. Οι συνεισφορές αυτού του κεφαλαίου είναι:

1. Μια ομοιόμορφη αντιπροσώπευση για τα μεταδεδομένα εφαρμογής και μια μέθοδο για να τα εξαγάγουν από τις δυναμικές εφαρμογές.

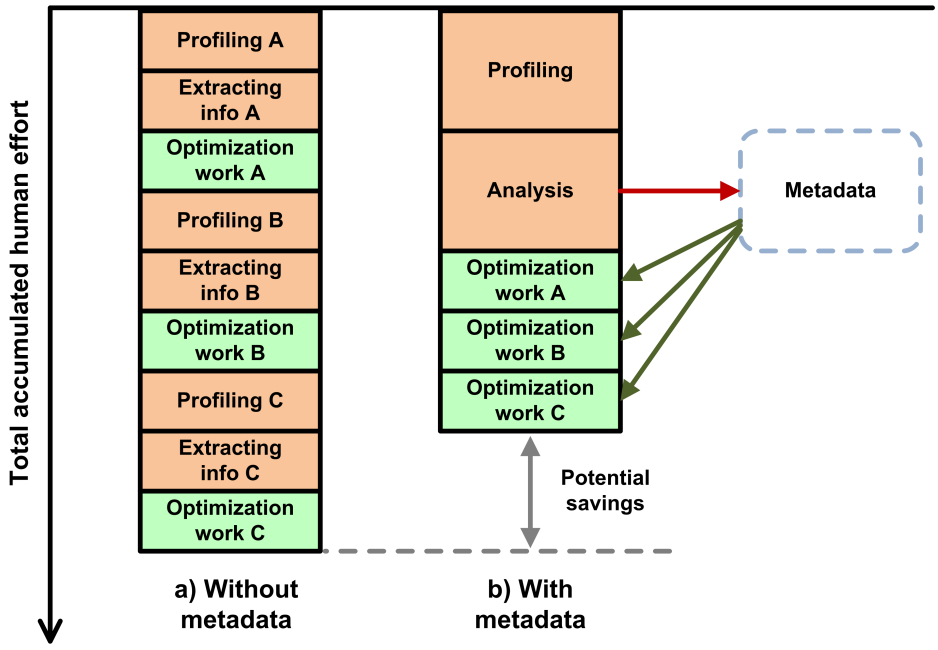
2. Τεχνικές σκιαγράφησης και ανάλυσης που καταδεικνύουν μια συγκεκριμένη εφαρμογή αυτής της μεθόδου.

3. Ένα παράδειγμα για το πώς αυτά τα μεταδεδομένα μπορούν να χρησιμοποιηθούν με τις διαφορετικές μεθόδους επιπέδων συστημάτων και τη σχετικότητά τους στις στοχοθετημένες διαχειριστικές βελτιστοποιήσεις μνήμης. Το υπόλοιπο του κεφαλαίου οργανώνεται ως εξής. Στην αρχή, η αντιπροσώπευση μεταδεδομένων εισάγεται στην Ενότητα 7.2. Έπειτα, συζητείται στο τμήμα 7.3 μια συγκεκριμένη μέθοδος για να λάβουν τις πληροφορίες σχεδιασμού περιγράμματος και οι τεχνικές ανάλυσης για την μετατροπή σε επιθυμητά μεταδεδομένα. Κατόπιν στο τμήμα 7.4, παρουσιάζεται μια περιπτωσιολογία που υιοθετεί αυτά τα μεταδεδομένα για τις διάφορες βελτιστοποιήσεις που αφορούν το δυναμικό αποτύπωμα κατανάλωσης ενέργειας και μνήμης των στοιχείων. Τέλος, στο τμήμα 7.6 περιγράφονται τα συμπεράσματα και η πιθανή μελλοντική εργασία σε αυτήν την περιοχή.

7.2 Η δομή των μεταδεδομένων λογισμικού

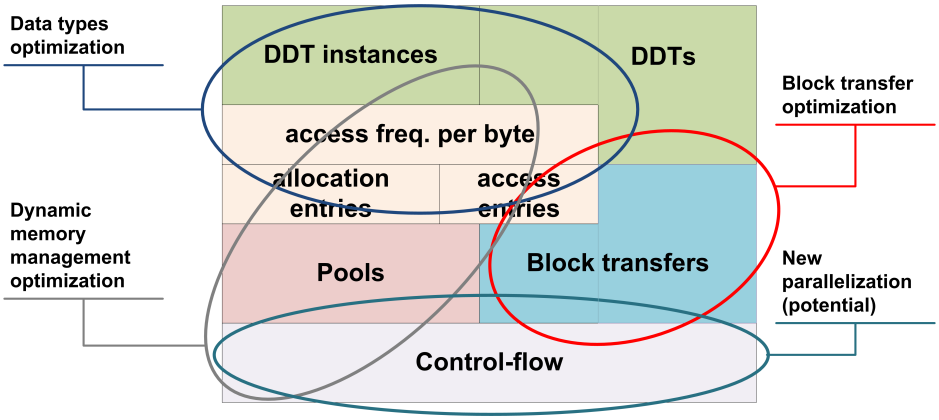
Προτείνουμε να ταξινομήσουμε τις πληροφορίες για μια εφαρμογή λογισμικού σε τρία επίπεδα αυξανόμενης αφαίρεσης: **Μ**ηδενικό επίπεδο μεταδεδομένων: Εκτενής χαρακτηρισμός που λαμβάνεται γενικά μέσω της σκιαγράφησης. **Π**ρώτο επίπεδο μεταδεδομένων: Συνολική αντιπροσώπευση των πληροφοριών στο προηγούμενο επίπεδο που δημιουργείται από τα εργαλεία ανάλυσης. Αυτές

οι πληροφορίες χρησιμοποιούνται και ενημερώνονται από τα εργαλεία βελτιστοποίησης κατά τη διάρκεια του χρόνου σχεδιασμού. Δεύτερο επίπεδο μεταδεδομένων: Πληροφορίες που επεκτείνονται με την τελική εφαρμογή απαριθμώντας τις ανάγκες των πόρων της. Ο διαχειριστής χρόνου εκτέλεσης του ενσωματωμένου συστήματος μπορεί να το χρησιμοποιήσει για να προσαρμόσει την απόδοση του συστήματος στις ανάγκες των εφαρμογών που τρέχουν αυτήν την περίοδο. Αυτές οι πληροφορίες μπορούν να ολοκληρωθούν με τις παραλλαγές των διαθέσιμων πόρων κατά τη διάρκεια του χρόνου (πχ. ικανότητα μπαταριών, νέες ενότητες που είναι συνδεόμενες, κ.λπ.).



Σχήμα 7.1: Ποιοτική σύγκριση του χρόνου που επενδύεται κατά χρησιμοποίηση της προτεινόμενης διανομής των μεταδεδομένων μιας εφαρμογής ή μιας παραδοσιακής ροής σχεδίου χωρίς, μια κοινή βάση πληροφοριών.

Η εστίαση αυτού του κεφαλαίου είναι στα μεταδεδομένα στο πρώτο επίπεδο. Ως εκ τούτου, στο υπόλοιπο αυτού του κεφαλαίου, ο όρος «μεταδεδομένα» χρησιμοποιείται αναφορικά στις πληροφορίες και όχι κατά ομολογία για το ακατέργαστο αρχείο καταγραφής προσβάσεων που παράγεται από τα εργαλεία σκιαγράφησης. Η έννοια των μεταδεδομένων για το λογισμικό που τρέχει στα ενσωματωμένα συστήματα περιλαμβάνει δύο μέρη: οι μετρήσεις και οι καθορισμένες τιμές τους. Οι διαφορετικές μετρήσεις στο σύνολο πληροφοριών μεταδεδομένων μπορούν να ταξινομηθούν σύμφωνα με την κύρια χρήση τους. Αν και τα μεταδεδομένα λογισμικού καθορίζονται συνολικά, τα διαφορετικά εργαλεία βελτιστοποίησης μπορούν να υιοθετήσουν διαφορετικά, επικαλύπτοντας ενδε-



Σχήμα 7.2: Τα διάφορα εργαλεία βελτιστοποίησης μπορούν να χρησιμοποιήσουν διαφορετικά υποσύνολα των μεταδεδομένων εφαρμογής λογισμικού.

χομένως, υποσύνολα. Το Σχήμα 7.2 δίνει μια επισκόπηση αυτών των πιθανών επικαλύψεων. Ένα εργαλείο βελτιστοποίησης μπορεί να πάρει ως εισαγωγή τις τιμές οποιασδήποτε από τις μετρήσεις, να τις χρησιμοποιήσει για να μετασχηματίσει την εφαρμογή και να ενημερώσει τις επηρεασθείσες μετρήσεις με τις τιμές που προσδιορίζονται από τη νέα συμπεριφορά της μετασχηματισμένης εφαρμογής

7.2.1 Καθορισμός και κατηγοριοποίηση των μεταδεδομένων

Η πρώτη ερώτηση που πρέπει να απαντήσουμε καθορίζοντας την έννοια των μεταδεδομένων για τις εφαρμογές λογισμικού που τρέχουν στα ενσωματωμένα συστήματα είναι: Ποιες είναι οι παρούσες μετρήσεις στα μεταδεδομένα; Οποιεσδήποτε πληροφορίες σχετικά με τη συμπεριφορά μιας εφαρμογής που θα μπορούσε ενδεχομένως να χρησιμοποιηθεί από σχεδόν οποιοδήποτε εργαλείο βελτιστοποίησης πρέπει να συμπεριληφθεί. Για τις εφαρμογές λογισμικού αυτό αφορά κυρίως τις απαιτήσεις των πόρων τους (αποτύπωμα μνήμης, εύρος ζώνης μνήμης, προϋπολογισμός κύκλων, αφιερωμένες ανάγκες υλικού, κλπ.), αλλά και οποιεσδήποτε εφαρμόσιμες προθεσμίες, εξαρτήσεις σε άλλες ενότητες λογισμικού, γεγονότα που προκαλούν τη συγκεκριμένη συμπεριφορά, κλπ. Σε μερικές περιπτώσεις, στοιχεία που απαιτούνται από τα εργαλεία βελτιστοποίησης μπορούν να εξαχθούν από τις πληροφορίες των σχεδιασμένων περιγραμμάτων (τα ακατέργαστα στοιχεία) κατά τρόπο απλό, αλλά στις περισσότερες άλλες περιπτώσεις πρέπει να υιοθετηθούν οι πιο επιμελημένες τεχνικές εξαγωγής. Αν και οι μετρήσεις μεταδεδομένων μπορούν να καλύψουν όλες τις σχετικές πτυχές της εφαρμογής, η εστίαση αυτής της εργασίας είναι στην ανάλυση της συμπεριφοράς μνήμης των εφαρμογών.

Η Εικόνα 7.3 παρουσιάζει πλήρη άποψη της δομής των μεταδεδομένων λογισμικού για τη δυναμική συμπεριφορά μνήμης των ενσωματωμένων συστημάτων όπως προτείνεται σε αυτό το κεφάλαιο. Αυτό το σχήμα περιλαμβάνει τις σημασιολογικές διασυνδέσεις μεταξύ των διαφορετικών κατηγοριών, όπως η κληρονομιά, η σύνθεση ή η ένωση. Οι πληροφορίες μεταδεδομένων αγκαλιάζουν τις πτυχές και του αριθμού των προσβάσεων σε μια μεταβλητή ή των σχέσεων μεταξύ των πεδίων εκτέλεσης και των δυναμικών τύπων στοιχείων που προσεγγίζονται. Μερικές τεχνικές βελτιστοποίησης απαιτούν τα μεταδεδομένα καθορίζονται για κάθε δυναμικό τύπο στοιχείων (το DDT) στην εφαρμογή. Συγκεκριμένα, DDTs παρουσιάζεται στον αριθμό και ως αφηρημένες οντότητες και ως συγκεκριμένες εφαρμογές για τις ακολουθίες και τα δέντρα, αλλά θα μπορούσε να προστεθεί στο σχήμα οποιοσδήποτε άλλος δυναμικός τύπος στοιχείων.

Οι διαφορετικές μετρήσεις που περιλαμβάνονται στα μεταδεδομένα μπορούν να ταξινομηθούν σύμφωνα με την κύρια χρήση τους. Η κορυφαία οντότητα στα μεταδεδομένα είναι η ροή ελέγχου. Αυτή η οντότητα φυλάσσει τις πληροφορίες για τα δυναμικά στοιχεία (δυναμική οντότητα στοιχείων) που προσεγγίζονται σε κάθε πλαίσιο υπό μορφή μεμονωμένων μεταφορών block προσβάσεων και στοιχείων μνήμης (οντότητα μεταφοράς block). Αυτές οι προσβάσεις συμβαίνουν στις πραγματικές φυσικές θέσεις μνήμης που αντιπροσωπεύονται από την οντότητα Pool, στην οποία οι περιπτώσεις των διάφορων δυναμικών στοιχείων διατίθενται. Επιπλέον, η οντότητα ροής ελέγχου περιέχει τις αθροισμένες πληροφορίες υπό μορφή ιστογραμμάτων πρόσβασης και δέσμευσης, το ζητούμενο εύρος ζώνης και τη συχνότητα των προσβάσεων ανά διατιθέμενο byte. Η δυναμική οντότητα στοιχείων περιέχει τις συγκεκριμένες πληροφορίες για κάθε δυναμικά διατιθέμενο μεταβλητό ή δυναμικό τύπο στοιχείων, όπως ο αριθμός που καθορίζει τις αναγνώσεις, εγγραφές ή το μέγιστο αποτύπωμα μνήμης. Το είδος αυτών των πληροφοριών είναι το ίδιο για τις μεταβλητές και τους δομημένους τύπους στοιχείων (εντούτοις, οι συγκεκριμένες τιμές θα είναι διαφορετικές για κάθε περίπτωση). Η δομή μεταδεδομένων αποκαλύπτει επίσης τις ενώσεις μεταξύ των δυναμικών οντοτήτων στοιχείων και των συγκεκριμένων περιπτώσεων τους (δυναμική οντότητα περιπτώσεων ημερομηνίας). Ομοίως, η σχέση από τις οντότητες στοιχείων και από τις περιπτώσεις με τις διαδικασίες που εκτελούνται παρουσιάζεται στη δυναμική οντότητα διαδικασιών στοιχείων, εφόσον ενδείκνυται, για παράδειγμα, η μεταβλητή οντότητα δεν έχει καμία διαδικασία, ενώ οι ακολουθίες ή τα δέντρα έχουν διάφορες ξεχωριστές διαδικασίες όπως Πρόσθεση, Αφαίρεση, κ.λπ.

Οι πληροφορίες σχετικά με τις δυναμικές δομές δεδομένων και τις μεταβλητές της εφαρμογής, παρουσιάζονται σε τρία επίπεδα αφαίρεσης: αθροισμένο στα στοιχεία το επίπεδο τύπων, ρητό για κάθε συγκεκριμένη περίπτωση κάθε στοιχείου ή μεταβλητής, και συγκεκριμένος για τις διαδικασίες που εκτελούνται. Κάθε μεταβλητή ή δυναμική οντότητα στοιχείων συνδέεται σε μια Pool

όπου οι συγκεκριμένες περιπτώσεις διατίθενται μέσω του δυναμικού διαχειριστή μνήμης του συστήματος. Επομένως, η οντότητα Pool αθροίζει την δέσμευση, την πρόσβαση, τη συχνότητα ανά byte και τις πληροφορίες εύρους ζώνης για όλες τις περιπτώσεις των μεταβλητών και των δυναμικών τύπων στοιχείων που δημιουργούνται μέσα. Η οντότητα Pool περιέχει, επιπλέον, τις πληροφορίες μεταδεδομένων σχετικά με τις κατανομές και τις προσβάσεις και, το πιο σημαντικό, μια περιγραφή της δομής της Pool, δηλαδή οι αλγόριθμοι και οι δομές δεδομένων που απαιτούνται για να εκτελέσουν τη λογιστική της μνήμης. Τέλος, οι πληροφορίες για τις πιθανές μεταφορές block στοιχείων (όπως το μέγεθος και ο αριθμός χρόνων που εκτελείται κάθε μεμονωμένη μεταφορά block) είναι τοποθετημένες στην οντότητα μεταφοράς block.

Τα μεταδεδομένα ροής ελέγχου

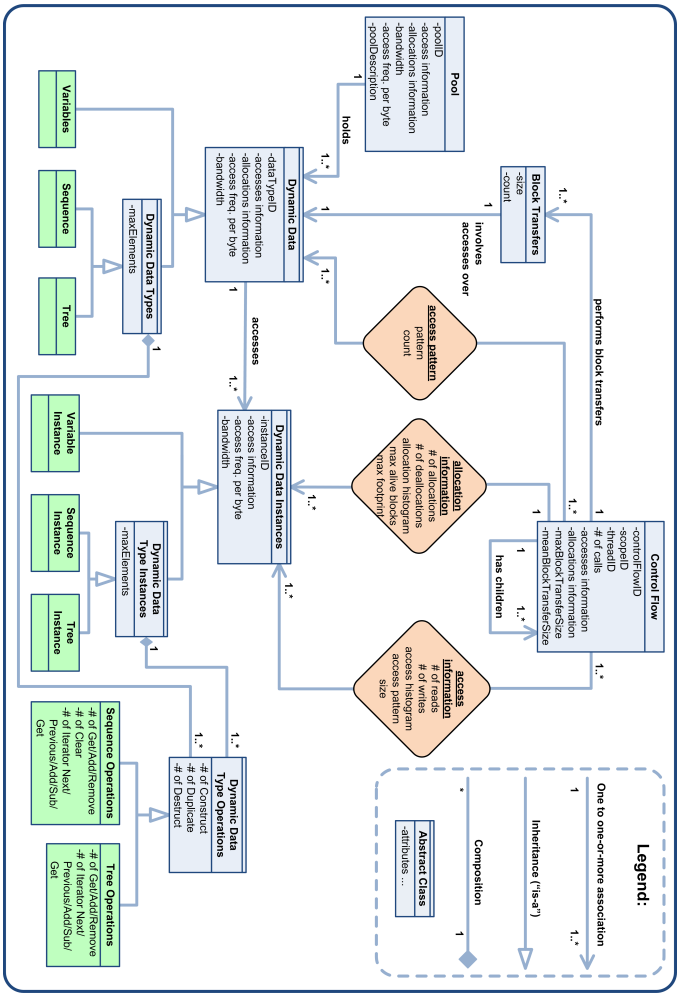
Η οντότητα ροής ελέγχου αντιπροσωπεύει τη ροή ελέγχου της εφαρμογής και μπορούμε να τη δούμε δουν ως σύνολο κλήσεων των διευθύνσεων κώδικα. Ο σκοπός αυτής της οντότητας είναι να ενθυλακώσει μια σημαντική μερίδα της κύριας ροής ελέγχου εφαρμογής, που επιτρέπει την εξαγωγή των πληροφοριών, όπως η δυναμική συμπεριφορά στοιχείων της εφαρμογής στις συγκεκριμένες στιγμές.

Υπάρχει μια είσοδος που είναι πάντα παρούσα και παίρνει ένα συγκεκριμένο προσδιοριστικό για να αντιπροσωπεύσει την λειτουργία main() της εφαρμογής. Οποιοσδήποτε αριθμός πρόσθετων καταχωρήσεων ροής ελέγχου μπορεί να είναι παρών στις πληροφορίες μεταδεδομένων, αρκεί να προσδιορίζεται μεμονωμένα.

Εκτός από τις πληροφορίες σχετικά με τις διαφορετικές προσβάσεις, τις κατανομές και άλλα μεταδεδομένα που εμφανίστηκαν μέσα σε ένα ιδιαίτερο τμήμα της ροής ελέγχου, κάθε είσοδος περιέχει επίσης τις πληροφορίες για τις οποίες επικαλούνται άλλες καταχωρήσεις ροής ελέγχου, δίνοντας με αυτόν τον τρόπο μια πλήρη άποψη της γραφικής παράστασης ελέγχου-ροής της εφαρμογής. Οι πληροφορίες για τη συμπεριφορά πρόσβασης δέσμευσης ή μνήμης οποιουδήποτε νήματος (μέσω όλων των διαφορετικών πεδίων που ενεργοποιούν) μπορούν να εξαχθούν εύκολα από αυτές τις πληροφορίες, εάν είναι απαραίτητο. Άλλες πληροφορίες, όπως οι προθεσμίες ή ο προϋπολογισμός κύκλων δεν συμπεριλαμβάνονται σε αυτό το κεφάλαιο.

Δυναμικά μεταδεδομένα πρόσβασης και δέσμευσης μνήμης

Οι καταχωρήσεις μεταδεδομένων που εισάγονται σε αυτήν την υποενότητα είναι συγκεκριμένες για τη δυναμική συμπεριφορά πρόσβασης και δέσμευσης μνήμης των εφαρμογών. Αυτές οι πληροφορίες είναι σχετικές με την πλειοψηφία



Σχήμα 7.3: Η προτεινόμενη δομή για τα μεταδεδομένα λογισμικού που έπρεπε να συλλάβει τα χαρακτηριστικά της δυναμικής συμπεριφοράς μνήμης των εφαρμογών που τρέχουν στα ενσωματωμένα συστήματα.

των οντοτήτων μεταδεδομένων που φαίνονται στην Εικόνα 7.3.

Η έννοια της ομάδας της δυναμικής μνήμης [56] είναι κρίσιμη για τις εφαρμογές που χρησιμοποιούν τη δυναμική διαχείριση μνήμης. Δεδομένου ότι η συμπεριφορά των εφαρμογών γίνεται περισσότερη εξαρτώμενη στην εισαγωγή, η ανάλυση του δυναμικού διαχειριστικού υποσυστήματος μνήμης κερδίζει περισσότερη σημασία. Επομένως, οι πληροφορίες σχετικά με τη συμπεριφορά των διαφορετικών ομάδων της εφαρμογής (συγκεκριμένα η οντότητα Pool) είναι ουσιαστικές στα μεταδεδομένα των εφαρμογών λογισμικού προκειμένου να επιτραπούν οι περαιτέρω τεχνικές βελτιστοποίησης (πχ. δυναμικός καθαρισμός μνήμης, δυναμική ανάθεση μνήμης στους πόρους μνήμης, πρόσβαση που σχεδιάζει, κλπ.). Είναι σημαντικό να παρατηρηθεί ότι οι πληροφορίες Pool δεν μπορούν να εξαχθούν από τη φάση ανάλυσης του αρχικού κώδικα εφαρμογής, αλλά θα δημιουργηθούν, θα χρησιμοποιηθούν και θα ενημερωθούν από τα διαφορετικά εργαλεία βελτιστοποίησης, πχ. για τη βελτιστοποίηση DMM, που θα χρησιμοποιήσουν τις πληροφορίες μεταδεδομένων για να επικοινωνήσουν και να περάσουν τους περιορισμούς μεταξύ τους. Τα αναπτυγμένα εργαλεία μπορούν να εξαγάγουν, αντίθετα, τις πληροφορίες όπως ο αριθμός κατανομών και μη-κατανομών , ο μέγιστος αριθμός αντικειμένων που διατίθενται συγχρόνως και το ιστόγραμμα των κατανομών κατά μήκος του χρόνου. Η οντότητα μεταβλητών περιέχει τις πληροφορίες σχετικές με τις προσβάσεις στις δυναμικές μεταβλητές, με μια κοκκοποίηση κάτω σε κάθε δυναμική μεταβλητή που δηλώνεται στην εφαρμογή.

Επιπλέον, είναι δυνατό να εξαχθούν οι πληροφορίες σχετικά με το μέγιστο μήκος, να σημάνουν το μήκος και τον αριθμό όλων των block των στοιχείων που εκτελούνται σε μια οντότητα ροής ελέγχου (είτε πρόκειται για ένα πεδίο ή νήμα), η οποία είναι διαφορετική από τον αριθμό μεμονωμένων προσβάσεων μνήμης. Αυτές οι πληροφορίες, που φυλάσσονται στην οντότητα block για κάθε μεμονωμένη πιθανή μεταφορά block, προσφέρουν επίσης τις λεπτομέρειες στις οποίες το στοιχείο καταγράφει τα μεταφερμένα στοιχεία εκεί που ανήκει, το μέγεθος της μεταφοράς και ο αριθμός χρόνων που εμφανίζονται τέτοιες μεταφορές στοιχείων block.

Δυναμικά μεταδεδομένα τύπων στοιχείων

Η έννοια των πληροφοριών του DDT εμφανίζεται στο σχήμα 7.3. Εντούτοις, δεν είναι πραγματικά ένα μέρος των πληροφοριών μεταδεδομένων οποιασδήποτε εφαρμογής, είναι μια εννοιολογική διεπαφή για να τοποθετήσει τις σχετικές πληροφορίες με οποιοδήποτε δυναμικό τύπο στοιχείων όπως, μια ακολουθία (διάνυσμα), έναν κατάλογο ή ένα δέντρο. Για παράδειγμα, ο αριθμός παρουσιάζει τις πληροφορίες που θα συνδέονταν στους τύπους στοιχείων ακολουθίας και δέντρων.

Οι διαφορετικοί τύποι στοιχείων έχουν διαφορετικές σχετικές πληροφορίες που δεν παρουσιάζουμε εδώ χάριν της περιεκτικότητας. Αντίστοιχα, κάθε συγκεκριμένη περίπτωση ενός δεδομένου τύπου στοιχείων έχει συνδέσει τις πληροφορίες (για παράδειγμα, η ακολουθία και οι οντότητες περιπτώσεων δέντρων). Επιπλέον, η ακολουθία και οι οντότητες διαδικασιών δέντρων απαριθμούν όλες τις διαδικασίες για τις οποίες οι πληροφορίες μπορούν να συλλεχθούν για τέτοιους τύπους στοιχείων. Για οποιουσδήποτε άλλους τύπους στοιχείων που δεν παρουσιάζονται εδώ, κωδικοποιείται ένα νέο σύνολο πληροφοριών για τα στοιχεία σε κάθε περίπτωση. Εντούτοις, ο εννοιολογικός ρόλος που αυτές οι πληροφορίες θα έπαιρναν στη δομή μεταδεδομένων είναι ίδιος με το ρόλο των σχετικών μερών ακολουθίας ή δέντρων.

7.3 Η εξαγωγή μεταδεδομένων

Η αφετηρία για την εξαγωγή μεταδεδομένων είναι ο πηγαίος κώδικας της εφαρμογής. Οι ακατέργαστες πληροφορίες για τη συμπεριφορά των διαφόρων στοιχείων συγκεντρώνονται μέσω μιας εκτενούς σκιαγράφησης στο δυναμικό επίπεδο τύπων στοιχείων, που προκαλεί την εφαρμογή με τα αντιπροσωπευτικά σύνολα δεδομένων εισόδου. Κατόπιν, η φάση ανάλυσης χρησιμοποιεί αυτά τα ακατέργαστα στοιχεία προκειμένου να εξαχθούν οι τιμές για τις μετρήσεις μεταδεδομένων. Μόλις καθοριστούν τα μεταδεδομένα, τα διαφορετικά εργαλεία μπορούν να συνδεθούν με έναν τρόπο σωληνώσεων, όπου χρησιμοποιούν ως εισαγωγή τα τρέχοντα μεταδεδομένα και παράγουν μια ενημερωμένη έκδοση που μπορεί να χρησιμοποιηθεί από την επόμενη. Ο προσδιορισμός των σχετικών συνόλων δεδομένων εισόδου, είναι κρίσιμο να λάβει τις σημαντικές πληροφορίες σχεδιασμού περιγράμματος, επειδή η συμπεριφορά των δυναμικών εφαρμογών επηρεάζεται έντονα από τη φύση της εισόδου. Επομένως, η αξία των μετρήσεων πρέπει να υπολογιστεί για κάθε διαφορετική περίπτωση δεδομένων εισόδου. Σε πολλές περιπτώσεις, θα είναι δυνατό να αφαιρεθεί και να γίνει ένας ενιαίος χαρακτηρισμός επειδή οι τιμές των διαφορετικών μετρικών μεταδεδομένων θα είναι οι ίδιες, αλλά σε μερικά άλλα διαφορετικά σύνολα περιπτώσεων μεταδεδομένων οι τιμές θα απαιτηθούν. Επομένως, ένας μηχανισμός που προσδιορίζει στο χρόνο εκτέλεσης τα χαρακτηριστικά της τρέχουσας περίπτωσης εισαγωγής και συνεπώς το σωστό σύνολο μεταδεδομένων (για να επιλέξει τις σωστές βελτιστοποιήσεις), όπως αυτή που παρουσιάζεται στο [70] μπορεί να χρησιμοποιηθεί εκτός από τις προτεινόμενες τεχνικές.

7.3.1 Ακατέργαστη εξαγωγή στοιχείων μέσω της σκιαγράφησης

Το βήμα για τη σχεδίαση περιγράμματος πρέπει να συλλέξει τις πληροφορίες σχετικές με τη δυναμική συμπεριφορά στοιχείων της εφαρμογής, έτσι ώστε η πιο πρόσφατη ανάλυση μπορεί να εξάγει τα κατάλληλα μεταδεδομένα από αυτές τις πληροφορίες. Δηλαδή, το σχεδιαζόμενο περίγραμμα πρέπει να εξαγάγει τις πληροφορίες για: (α) δέσμευση και ελευθέρωση της δυναμικής μνήμης, (β) προσβάσεις μνήμης (διαβάζει και γράφει), (γ) διαδικασίες στους δυναμικούς τύπους στοιχείων, (δ) πορείες έλεγχου-ροής που οδηγούν στις θέσεις όπου αυτές οι διαδικασίες εκτελούνται και προσδιορίζονται (ε) των νημάτων που εκτελούν αυτές τις διαδικασίες.

Γενικά, η σκιαγράφηση των πληροφοριών μπορεί να ληφθεί μέσω δύο κύριων τρόπων: ερμηνεία δυαδικού κώδικα και ενοργάνωση (τροποποίηση) πηγαίου κώδικα. Οι προσεγγίσεις που λειτουργούν άμεσα στο δυαδικό κώδικα δεν απαιτούν τις τροποποιήσεις στον πηγαίο κώδικα, αλλά συνήθως οι πληροφορίες που εξάγονται δεν μπορούν να αφορούν εύκολα τις μεταβλητές και το DDTs. Σε αντίθεση, οι προσεγγίσεις βασισμένες στην τροποποίηση του πηγαίου κώδικα, είναι συνήθως ικανές να παρέχουν τις πληροφορίες που μπορούν να αφορούν τα συγκεκριμένα στοιχεία και τις θέσεις στον πηγαίο κώδικα. Σε αυτήν την δεύτερη κατηγορία, είναι πάλι δυνατό να γίνει διάκριση μεταξύ των αυτοματοποιημένων προσεγγίσεων (από το μεταγλωττιστής ή άλλο εργαλείο, που μπορεί να λειτουργήσει παρομοίως) και χειρωνακτικά (ο προγραμματιστής τροποποιεί τον πηγαίο κώδικα για να εισαγάγει την πιο συγκεκριμένη ενοργάνωση). Για αυτήν την πρόταση, έχουμε επιλέξει μια ημιαυτόματη προσέγγιση, όπως πρώτα παρουσιάστηκε στο [62]. Με αυτήν την λύση, ο σχεδιαστής πρέπει να σχολιάσει τους τύπους των μεταβλητών που είναι σχετικές, αλλά ο μεταγλωττιστής παίρνει αυτόματα την προσοχή του σχολιασμού όλων των προσβάσεων τους, μέσω του πηγαίου κώδικα. Ένα σημαντικό πλεονέκτημα αυτής της επιλογής είναι ότι ο μεταγλωττιστής εξασφαλίζει ότι καμία πρόσβαση σε οποιεσδήποτε από τις περιπτώσεις των επιλεγμένων τύπων στοιχείων δεν αγνοείται.

Σχεδιάζοντας το περίγραμμα συγκεντρώνονται οι πληροφορίες συμπεριλαμβανομένων των προσβάσεων μνήμης, των κατανομών και μη-κατανομών μνήμης, των προσβάσεων στις κλιμακωτές μεταβλητές και τις συγκεκριμένες περιπτώσεις των δυναμικών τύπων στοιχείων των εφαρμογών (που προσδιορίζονται μεμονωμένα από ένα αριθμητικό προσδιορισμό), των αλλαγών του πεδίου και των νημάτων που εκτελούν κάθε ένα από τα προηγούμενα γεγονότα. Οι αποσπασματικές ακατέργαστες πληροφορίες διατηρούν την ακολουθία, το οποίο σημαίνει ότι μπορούν να δημιουργηθούν τα συσσωρευμένα σύνολα, όπως τα ιστογράμματα των παραλλαγών αποτυπώματος μνήμης κατά μήκος του χρό-

νου.

Περιγραφή και χρήση της βιβλιοθήκης σκιαγράφησης

Σε αυτό το τμήμα, παρουσιάζουμε μια τεχνική για τις εφαρμογές που γράφονται σε C++. Ωστόσο, οι έννοιες μεταδεδομένων είναι ανεξάρτητες από τη γλώσσα προγραμματισμού και ισχύουν ακόμα για άλλες γλώσσες, υπό τον όρο ότι είναι διαθέσιμο το αντίτιμο σχεδιασμού περιγράμματος της μεθόδου. Ο σχολιασμός τύπων εκτελείται μέσω της χρήσης των προτύπων, για να τυλίξει τους τύπους και να τους δώσει έναν νέο τύπο, και στην υπερφόρτωση, για να συλλάβει όλες τις προσβάσεις στις περιπλεγμένες μεταβλητές. Τα πρότυπα είναι χρονικά κατασκευάσματα που περιγράφουν τη γενική συμπεριφορά μιας κατηγορίας (ή της λειτουργίας) βασισμένης στις παραμέτρους τύπων, κατά συνέπεια είναι μερικές φορές γνωστοί ως «παραμετρικοί τύποι». Όταν το πρότυπο κατηγορίας αρχικοποιείται με τον επιθυμητό τύπο, ο μεταγλωττιστής παράγει τις σωστές οδηγίες για να εξετάσει εκείνο τον τύπο.

Η βιβλιοθήκη σκιαγράφησης που χρησιμοποιείται σε αυτό το κεφάλαιο αποτελείται από διάφορα πρότυπα κατηγορίας με σκοπό να είναι ορθογώνια και συναρμολογούμενα. Επιπλέον, η βιβλιοθήκη περιέχει ένα σύνολο βοηθητικών κατηγοριών για τις πληροφορίες που σχηματοποιούν και που καταγράφουν. Για να κρατήσουμε το πεδίο αυτής της εργασίας κάτω από την εστίαση, μόνο τα βασικά στοιχεία απαιτούνται για να καταλάβουμε τη δομή της βιβλιοθήκης. Το πρώτο βοηθητικό στοιχείο στη βιβλιοθήκη είναι η υποδομή. Το ακόλουθο τεμάχιο κώδικα παρουσιάζει τη βασική δομή μιας κατηγορίας αναγραφών που γράφει ένα δυαδικό αρχείο για κάθε γεγονός εφαρμογής:

```
1  class DMMLogger {
2   enum LogType {
3   LOG_VAR_READ = 0,
4   LOG_VAR_WRITE = 1,
5    ...
6   LOG_MALLOC_END = 5,
7    ...
8   };
9
10  public :
11  inline static void log_read ( const void * addr , const unsigned int id ,
12  const size_t sz) {
13  write_header ( LOG_VAR_READ , 4* sizeof ( unsigned int )) << id <<
14  addr << sz << ( unsigned long ) pthread_self ();
15  }
16
17  inline static void log_malloc_end ( const unsigned int id ,
18  const size_t sz , const void * addr ) {
19  write_header ( LOG_MALLOC_END , 4* sizeof ( unsigned int )) << id <<
20  addr << sz << ( unsigned long ) pthread_self ();
21  }
22
23  private :
24  DMMLogger & write_header ( LogType logType , unsigned short logSize ) {
```

```
25  unsigned short logType_s = ( unsigned short ) logType ;
26  if ( logFile_ != NULL ) {
27  fwrite (& logType_s , sizeof ( unsigned short ), 1, logFile_ );
28 fwrite (& logSize , sizeof ( unsigned short ), 1, logFile_ );
29 }
30  return * this ;
31  }
32
33 DMMLogger & operator <<( const unsigned int num ) {
34  if ( logFile_ != NULL )
35  fwrite (& num , sizeof ( unsigned int), 1, logFile_ );
36  return * this ;
37  }
38
39  DMMLogger & operator <<( const void * addr ) {
40  if ( logFile_ != NULL )
41  fwrite (& addr , sizeof ( unsigned int), 1, logFile_ );
42  return * this ;
43  }
44
45  FILE * logFile_ ; // Initialized in the constructor
```

Οι πρόσθετες μέθοδοι, όπως το log_write, log_malloc_begin, log_free_begin, log_free_end, log_scope_begin, log_scope_end, sequence_get, sequence_add, sequence_remove, sequence_clear, map_get, map_add, map_remove ή map_clear, μπορούν να κατασκευαστούν εύκολα με έναν ανάλογο τρόπο. Το σύνολο καταχωρήσεων στο logType πρέπει να διευρυνθεί αντίστοιχα.

Επίσης, απαιτείται μια απλή κατηγορία που εξάγει τα malloc() και τα free(), με τις υπονοούμενες ικανότητες αναγραφών. Μπορεί να εφαρμοστεί εύκολα με τον παρακάτω παρόμοιο κώδικα:

```
1  template <int ID , typename Logger = DMMLogger >
2   class logged_allocator {
3   public :
4   inline static void * malloc ( const size_t sz) {
5   Logger :: log_malloc_begin (ID , sz );
6   void * ptr =:: malloc (sz );
7   Logger :: log_malloc_end (ID , sz , ptr );
8   return ptr ;
9   }
10
11  inline static void free ( void * ptr) {
12  Logger :: log_free_begin (ID , ptr );
13  :: free (ptr );
14  Logger :: log_free_end (ID , ptr );
15  }
16  };
```

Κατά τη διάρκεια της πραγματικής φάσης ενοργάνωσης του πηγαίου κώδικα, ο σχεδιαστής χρησιμοποιεί τα ακόλουθα πρότυπα κατηγορίας:

- «scope» Αυτό είναι το πρότυπο κατηγορίας που χρησιμοποιείται για να χαρακτηρίσει τα διαφορετικά τμήματα του ελέγχου-ροής (control-flow). Το ακόλουθο τεμάχιο κώδικα επεξηγεί την εφαρμογή του:

```
1  template <class Logger = std_scope_logger >
2   class scope {
3   public :
4   scope (std :: string name )
5   : name_ ( name )
6   {
7   Logger :: log_scope_begin ( name_ );
8   }
9
10  ~ scope () {
11  Logger :: log_scope_end ( name_ );
12  }
13  private :
14  std:: string name_ ;
15  };
```

Ο σχεδιαστής μπορεί να δηλώσει μια περίπτωση αυτού του προτύπου κατηγορίας στην αρχή οποιουδήποτε τμήματος του κώδικα και ο μεταγλωττιστής, η συνάρτηση κατασκευής και η αντίστοιχη καταστροφή της δομής θα παραγάγουν τα περίγραμμα σχεδιάζοντας τα σημεία. Μετά, τα εργαλεία ανάλυσης μπορούν να καθορίσουν ποιες περιοχές του κώδικα εκτελέσθηκαν, η ακολουθία ενεργοποιήσεων και, επιπλέον, ποια τμήματα του κώδικα ήταν αρμόδια για τα διαφορετικά περιγράμματα γεγονότων. Επιπλέον, το πρότυπο πεδίο μπορεί να συνδυαστεί με τον προσδιορισμό νημάτων για να λάβει την πλήρη γραφική παράσταση ελέγχου-ροής της εφαρμογής για ένα σύνολο εισαγωγής και να αφορά το υπόλοιπο των γεγονότων (κατανομές, προσβάσεις στους δυναμικούς τύπους στοιχείων) στις συγκεκριμένες μερίδες του κώδικα που εκτελούνται από τα συγκεκριμένα νήματα.

- «Allocated» Οι περιπτώσεις κατηγορίας δημιουργούνται κανονικά και καταστρέφονται μέσω του νέου και διαγράφονται μέσω των χειριστών. Αυτό το πρότυπο κατηγορίας αυτοματοποιεί την υπερφόρτωσή τους για να παραγάγει τη σκιαγράφηση των σημείων. Χαρακτηριστικά, ο σχεδιασμός περιγράμματος των σημείων παράγεται για κάθε μια από τις διαδικασίες και τα αιτήματα διαβιβάζονται στο διαθέτη συστημάτων, μέσω του malloc και των free λειτουργιών:

```
1  template <class Allocator >
2   class allocated {
3   public :
4   void * operator new( const size_t sz) {
5   return Allocator :: malloc (sz );
6   }
7   void operator delete ( void * p) {
8   return Allocator :: free (p);
9   }
10  void * operator new []( const size_t sz) {
11  return Allocator :: malloc (sz );
12  }
```

```
13  void operator delete []( void * p) {
14  return Allocator :: free (p);
15  }
16  };
```

Προκειμένου να καταγραφούν όλα τα δυναμικά γεγονότα μνήμης σχετικά με τις περιπτώσεις μιας κατηγορίας (ή δομής), ο σχεδιαστής θα το δήλωνε ως εξής:

```
1  class NewClass : public allocated < logged_allocator <1> > {
2    ...
3  };
```

Καμία άλλη γραμμή στον πηγαίο κώδικα της κατηγορίας δεν πρέπει να τροποποιηθεί.

- «VAR» Αυτό το πρότυπο κατηγορίας περικλείει τη δήλωση των μεμονωμένων μεταβλητών ή των ιδιοτήτων κατηγορίας και παράγει ένα σχεδιασμό περιγράμματος σημείων για κάθε πρόσβαση μνήμης. Επιπλέον, προέρχεται από το «διατιθέμενο» πρότυπο, με αυτόν τον τρόπο δίνοντας επίσης την αναγραφή δέσμευσης και ελευθέρωσης της συσκευασμένης μεταβλητής. Παρακάτω είναι ένα απόσπασμα της εφαρμογής προτύπων:

```
1  template <typename T, // Type of the wrapped object
2   int ID , // Id used in the profiling tokens
3   class Logger = DMMLogger ,
4   class Allocator = logged_allocator <ID , Logger > >
5   class var : public allocated < typename Allocator > {
6   public :
7   T data_ ;
8
9   // Constructor
10  template <typename T2 >
11  var( const T2 & data )
12  : data_ ( data )
13  {
14  Logger :: log_write (&( data_ ), ID , sizeof (T));
15  }
16
17  // Assignment operator from basic type
18  template <typename T2 >
19  var & operator = ( const T2 & data ) {
20  data_ = data ;
21  Logger :: log_write (&( data_ ), ID , sizeof (T));
22 return * this ;
23  }
24
25  // Assignment operator from wrapped type
26  template <typename T2 , int ID2 >
27  var & operator = ( const var <T2 , ID2 > & other ) {
28  Logger :: log_read (&( other . data_ ), ID2 , sizeof (T2 ));
29  data_ = other . data_ ;
30  Logger :: log_write (&( data_ ), ID , sizeof (T));
31  return * this ;
32  }
```

```
33  };
```

Αυτό το πρότυπο μπορεί να χρησιμοποιηθεί για να τυλίξει τις μεταβλητές οποιουδήποτε βασικού τύπου, συμπεριλαμβανομένων των δεικτών και των ιδιοτήτων κατηγορίας. Στις προσβάσεις σχεδιαγράμματος, στις ιδιότητες μιας κατηγορίας ή μιας δομής, είναι απαραίτητο να τοποθετηθεί εσωτερικά, εκεί που ταξινομούνται και συσκευάζονται οι ιδιότητες των μεταβλητών. Στη συνέχεια, εάν οποιεσδήποτε από τις ιδιότητες είναι επίσης δομές, η τεχνική μπορεί να εφαρμοστεί κατ' επανάληψη, έως ότου επιτυγχάνονται οι βασικοί τύποι:

```
1   var <int , 1> var1 ;
2    var <int , 2> * pointer1 = new var <int , 2>;
3
4    struct A {
5    var <int , 3> foo ;
6    };
7    struct B {
8    A a;
9    var <int *, 4> bar ;
10   };
```

Απαιτείται μια πρόσθετη εκτίμηση κατά το κάλυμμα των δεικτών. Στο ακόλουθο τεμάχιο κώδικα, η πρώτη δήλωση σημαίνει ότι καταγράφονται μόνο οι προσβάσεις στο δείκτη quux, αλλά όχι οι προσβάσεις στους ακέραιους αριθμούς που είναι δείκτες Εντούτοις, όταν και οι δύο προσβάσεις είναι ενδιαφέρουσες, το σχέδιο που δίνεται για τον καθορισμό του baz, είναι ένας συνδυασμός των προηγούμενων σχεδίων, πρέπει να χρησιμοποιηθεί αντ' αυτού:

```
1   var (int , 1) * quux ;
2   var (var(int , 2) *, 3) baz;
```

Χρησιμοποιώντας το πρότυπο κατηγορίας «VAR», ο σχεδιαστής δεν πρέπει να ανησυχήσει για τον προσδιορισμό όλων των θέσεων στον πηγαίο κώδικα, όπου οι μεταβλητές προσεγγίζονται, δεδομένου ότι ο μεταγλωττιστής θα εξασφαλίσει ότι καταγράφονται κατάλληλα.

- «VECTOR: » Αυτό το πρότυπο κατηγορίας αντικαθιστά το διάνυσμα από το STL στο [49] προκειμένου να σχεδιαστεί περίγραμμα συμπεριφοράς χρήσης ακολουθίας στο δυναμικό επίπεδο αφαίρεσης στοιχείο-τύπων. Αυτές οι πληροφορίες μπορούν να χρησιμοποιηθούν, για παράδειγμα, για να αποφασίσουν ποια είναι η ιδανική εφαρμογή στοιχείο-τύπων, όπως φαίνεται στο [61]. Η εφαρμογή του προτύπου ταιριάζει με το STL από λογική άποψη, αλλά προσθέτει την αναγραφή στις σωστές θέσεις για να

δώσει μια υψηλού επιπέδου άποψη του σχεδίου χρήσης της δομής δεδομένων (πχ. συνηθισμένες είσοδοι, γραμμικές οδεύσεις, τυχαίες προσβάσεις, κλπ.). Η δήλωση ενός διανύσματος με αυτό το πρότυπο είναι απλή:

vector <int , 1> aVector ;

Τέλος, ο σχεδιαστής μπορεί να συνδυάσει τα πρότυπα «VECTOR» και «VAR» στο σχεδιάγραμμα μαζί με τις προσβάσεις στα πραγματικά στοιχεία, μέσα στην ακολουθία.

- «Allocated» «VAR» και «VECTOR» παραμετρικά πρότυπα, με ένα μοναδικό προσδιοριστικό που καταγράφεται στα σχεδιασμό περιγράμματος σημεία ως εκ τούτου, πληροφορίες σχεδιασμού περιγράμματος μπορούν εύκολα να συνδεθούν με τον αρχικό κώδικα. Το παραμετρικό πρότυπο «πεδίου» με ένα κατανοητό από τον άνθρωπο προσδιοριστικό για την ευκολότερη αναζήτηση της καλύτερης επιλογής.

7.3.2 Τεχνικές ανάλυσης για το συμπέρασμα μεταδεδομένων

Μόλις εξαχθούν οι πληροφορίες σχεδιασμού περιγράμματος από την εφαρμογή, διάφορα βήματα ανάλυσης μπορούν να εφαρμοστούν για να εξαγάγουν και να υπολογίσουν τις σχετικές μετρήσεις μεταδεδομένων, που θα χρησιμοποιηθούν από τα διάφορα εργαλεία βελτιστοποίησης για να μειώσουν την κατανάλωση ενέργειας, τις προσβάσεις μνήμης και το αποτύπωμα μνήμης. Κάθε ένα από αυτά τα εργαλεία βελτιστοποίησης θα χρησιμοποιήσει τα συγκεκριμένα μέρη των καθορισμένων μεταδεδομένων. Επομένως, οι πληροφορίες μεταδεδομένων μπορούν να αναφέρουν την κατασκευή των μεταβλητών σε διαφορετικές, ενδεχομένως με επικάλυψη, περιοχές ενδιαφέροντος. Σε αυτό το τμήμα, παρουσιάζουμε διάφορες τεχνικές που μπορούν να χρησιμοποιηθούν για να εξαγάγουν τις διαφορετικές μερίδες των πληροφοριών μεταδεδομένων. Η διαδικασία ανάλυσης είναι δομημένη ως σύνολο αντικειμένων που εκτελούν τους συγκεκριμένους στόχους ανάλυσης. Ο κύριος οδηγός διαβάζει κάθε πακέτο από το αρχείο ημερολογίου που παράγεται κατά τη διάρκεια της φάσης του σχεδιασμού περιγράμματος και επικαλείται στη συνέχεια όλα τα αντικείμενα ανάλυσης για να τα επεξεργαστεί. Λόγω του τρόπου ότι οι πληροφορίες μας συγκεντρώνονται κατά το σχεδιασμό του περιγράμματος, δεν είναι σημαντικό να υπάρξει ένα απόλυτα αποτύπωμα χρόνου, διότι ο χρόνος που απαιτεί στο σχεδιάγραμμα εξουσιάζει (έτσι, καταστρέφοντας) το χρόνο εκτέλεσης της εφαρμογής 1. Επομένως, το μέτρο συγχρονισμού καθορίζεται ως προς το ποσό σκιαγράφησης των πακέτων (ενός συγκεκριμένου τύπου, όπως τα πακέτα δέσμευσης ή πρόσβασης) που έχουν περάσει από την αρχή της εκτέλεσης. Εντούτοις, αυτό εί-

ναι ένα καλό μέτρο για τον τύπο ανάλυσης που προτείνεται εδώ, εξαιτίας του γεγονότος ότι όλες οι μετρήσεις μεταδεδομένων, καθώς επίσης και οι μέθοδοι ανάλυσης και βελτιστοποίησης, εξετάζουν τις δυναμικές προσβάσεις μνήμης και όχι με το χρόνο υπολογισμού. Κατά συνέπεια, ο συγχρονισμός είναι βασισμένος στα γεγονότα που αλλάζουν την κατάσταση της δυναμικής μνήμης (πχ. κατανομές) ή που καθορίζει τα κύρια σημεία για το υποσύστημα μνήμης (πχ. αριθμός προσβάσεων).

Ο αλγόριθμος 1 παρουσιάζει την εργασία που εκτελείται από τη συσκευή ανάλυσης των προσβάσεων στις μεταβλητές. Κάθε φορά που διαβάζεται ή γράφεται μια μεταβλητή, το γεγονός βρίσκεται στο αρχείο ημερολογίου. Χρησιμοποιώντας τη διεύθυνση της πρόσβασης μνήμης, είναι δυνατό να βρεθεί η συγκεκριμένη περίπτωση ενός δυναμικού τύπου στοιχείων. Αυτές οι πληροφορίες επιτρέπουν επίσης τον αριθμό προσβάσεων του αντίστοιχων δυναμικών τύπου και της περίπτωσης στοιχείων DDT.

Listing 7.1: "Ανάλυση πρόσβασης"

```
1 // event.type is variable read or write
2 process(event) {
3 Update application reads/writes counter
4 Update reads/writes counters of current control -flow point
5 FindBlockOfAddress (event.address) =>
6 Update block reads/writes counter
7 Update DDT & DDT -instance reads/writes counter
8 }
```

Ένας άλλος σημαντικός στόχος ανάλυσης είναι η συμπεριφορά δέσμευσης δεδομένου ότι τέτοιες πληροφορίες μπορούν να χρησιμοποιηθούν για να σχεδιάσουν τους ιδιαίτερα συντονισμένους οριζόμενους από εφαρμογή δυναμικούς διευθυντές μνήμης. Για να επιτρέψουμε αυτήν την βελτιστοποίηση, προσδιορίζουμε τον αριθμό κατανομών ανά μέγεθος block, τα διαφορετικά μεγέθη block και αριθμό προσβάσεων ανά μέγεθος block. Με αυτές τις πληροφορίες είναι δυνατόν να αυτοματοποιηθεί η εξερεύνηση των οριζόμενων από εφαρμογή δυναμικών διευθυντών μνήμης, όπως καταδεικνύεται στο [71]. Επιπλέον, οι προσβάσεις στις μεταβλητές που αντιμετωπίζονται μεταξύ MallocBegin και MallocEnd, ή μεταξύ FreeBegin και FreeEnd, μπορούν να χρησιμοποιηθούν για να αξιολογήσουν τα γενικά κόστη των διαφορετικών δυναμικών διευθυντών μνήμης. Τέλος, αυτά τα στοιχεία μπορούν επίσης να χρησιμοποιηθούν για να ανιχνεύσουν τις διαρροές μνήμης. Ο αλγόριθμος 2 επιδεικνύει πώς εξάγονται όλες αυτές οι πληροφορίες.

Listing 7.2: "Προσδιορισμός συλλογών πληροφοριών block"

```
1 process(event) {
2 case event of
3 AllocEnd ->
4 Create new live -block with event.size , event.address
5 and the control -flow information from
```

```
6  activeControlFlows .get(threadId ).top()
7  Increase allocation count for event.size
8  DeallocEnd ->
9  FindBlockOfAddress (event.address) =>
10 Increase de -allocation count for event.size
11 Add number of accesses to accesses for block
12 of size event.size
13 Destroy live -block
14 VarRead or
15 VarWrite ->
16 FindBlockOfAddress (event.address) =>
17 Increment read or write counter for this live -block
18 }
19 finalize () {
20 forall block ? live -block {
21 Report memory leak for this block and where it was
22 allocated
23 }
24 }
```

Προσδιορίζοντας τις μεταφορές block μεταξύ των μονάδων που καθορίζουν το περίγραμμα, τα γεγονότα πρόσβασης μνήμης, μας επιτρέπουν την εφαρμογή των βελτιστοποιήσεων μεταφοράς στοιχείων. Παραδείγματος χάριν, οι αφιερωμένοι πόροι υλικού (δηλαδή οι μηχανές DMA) μπορούν να χρησιμοποιηθούν για να εκτελέσουν τις μεταφορές στοιχείων που είναι μακροχρόνιες, χωρίς να επηρεαστούν οι κύκλοι υπολογισμού για τα κύρια στοιχεία επεξεργασίας. Υπάρχουν διάφορες απόψεις για τις αποφάσεις του είδους των βελτιστοποιήσεων που πρέπει να εφαρμοστούν. Για παράδειγμα, στο [72] παρουσιάζεται μια τεχνική για να εκτελεσθούν επιλεκτικά οι μεταφορές στοιχείων που χρησιμοποιούν μια ενότητα DMA στα ενσωματωμένα συστήματα. Επιλέγουμε να καθορίσουμε μια μοναδική μεταφορά στοιχείων, καθώς ένα σύνολο αυστηρά διαδοχικών προσβάσεων στοιχείων, γίνεται στα ίδια δεδομένα από ένα νήμα. Με αυτόν τον καθορισμό, χρησιμοποιούμε τον παρακάτω αλγόριθμο για να προσδιορίσουμε τις μεταφορές στοιχείων της εφαρμογής.

Μια πρόσθετη βελτιστοποίηση που επιτρέπεται από τον προσδιορισμό των σχεδίων στις προσβάσεις στα στοιχεία είναι η τοποθέτηση εκείνων των στοιχείων που προσεγγίζονται συνήθως κατά μνήμες που υποστηρίζουν ταχύτητες ριπής (πχ. DRAM), με αυτόν τον τρόπο μειώνοντας τον αριθμό ενεργοποιήσεων σελίδων και αλλαγής τρόπου στα στοιχεία μνήμης. Εμείς προτείνουμε τον παρακάτω αλγόριθμο για να προσδιορίσουμε τα σχέδια πρόσβασης στις προσβάσεις στους δυναμικούς τύπους στοιχείων.

Αυτός ο αλγόριθμος είναι αρμόδιος της οικοδόμησης ενός καταλόγου σχεδίων πρόσβασης, που εμφανίζονται κατά τη διάρκεια της εκτέλεσης της εφαρμογής. Επιπλέον, κάθε σχέδιο πρόσβασης αποσυντίθεται σε μικρότερα, πιο θεμελιώδη. Κατά αυτόν τον τρόπο ο σχεδιαστής έχει μια πλήρη άποψη με έναν τρόπο πολυ-κοκκοποίησης των σχεδίων πρόσβασης που χαρακτηρίζουν την εφαρμογή. Αυτός ο αλγόριθμος μπορεί να γίνει αποδοτικότερος με τη χρησι-

μοποίηση ενός μέγιστου παραθύρου που εξετάζει προηγούμενες καταστάσεις (look-back).

Μέσω του πειραματισμού έχουμε διαπιστώσει ότι, για τη δυναμική συμπεριφορά στοιχείων, ένα παράθυρο 100 στοιχείων προσδιορίζει το ίδιο σχέδιο με ένα που δεν είναι περιορισμένου παραθύρου, ενώ οι χρόνοι ανάλυσης παραμένουν οριακοί.

Listing 7.3: "Προσδιορισμός μεταφορών στοιχείων"

```
process(event) {
case event of
DeallocEnd ->
FindBlockOfAddress (event.address) =>
Record last active transfer for the block
Destroy block
AllocEnd ->
Create new block with event.address
VarRead or
VarWrite ->
FindBlockOfAddress (event.address) =>
 if consecutive access for event.threadID and
same direction (read/write) {
Update active transfer with event.address
} else {
Record last active transfer (if any)
Create new transfer with event.address
}
}
```

Ενώ ο προηγούμενος περιγεγραμμένος αλγόριθμος είναι πολύ ακριβής και μπορεί να δώσει μια πολύ συγκεκριμένη άποψη των σχεδίων συμπεριφοράς, στην περίπτωση των πολύ δυναμικών συμπεριφορών, δεν θα ανιχνευθούν τα ακριβή ταιριάσματα. Σε αυτήν την περίπτωση, προτείνουμε τον παρακάτω αλγόριθμο, που εξάγει τα μη-ακριβή σχέδια, που δεν παρακολουθούν τον αριθμού επαναλήψεων, και του αριθμού χρόνων στα οποία επαναλαμβάνονται τα γεγονότα. Κατά αυτόν τον τρόπο κάθε σχέδιο δεν φυλάσσει τις λεπτές πληροφορίες κοκκοποίησης σχετικά με τα σχέδια που αποτελούνται. Ένα ενδεικτικό παράδειγμα για αυτό δίνεται εδώ:

Παράδειγμα: Το ταίριασμα σχεδίων. Δεδομένου του ακόλουθου ρεύματος των γεγονότων: AAAABAAABAAABAAAB, όπου το A σημαίνει τις προσβάσεις μέσων variable1 και B σε variable2, τα ακόλουθα σχέδια θα συνάγονταν: Ακριβές ταίριασμα: 4*[3*[A], μη-ακριβές] ταίριασμα B: [[Το A], B] αυτό φαίνεται σαφές ότι τα variable1 και variable2 προσεγγίζονται σε έναν βρόχο. Εάν, εντούτοις, ο αριθμός προσβάσεων σε variable1 ποικίλλει σε αυτόν τον βρόχο λόγω δυναμικών εξαρτώμενων στοιχείων, μπορεί να έχουν ένα ρεύμα όπως: AAAABAAAAABAAABAB. Κατόπιν, το σχέδιο που εξάγεται θα ήταν: Ακριβές ταίριασμα: [4*[A], B, 5*[A], B, 3*[A], B, A, μη-ακριβές] ταίριασμα B: [[Το A], μη-ακριβές] το σχέδιο-ταίριασμα B μπορεί να είναι ευεργετικό για τις εφαρμογές που καταδεικνύουν τη δυναμική συμπεριφορά, ακόμα κι αν προσφέρει τις

λιγότερες πληροφορίες από το ακριβές σχέδιο.

Αυτοί οι δύο αλγόριθμοι δεν περιορίζονται στις προσβάσεις μπορούν να χρησιμοποιηθούν για το ταίριασμα σχεδίων οποιαδήποτε από το σχεδιασμό περιγράμματος γεγονότα, για παράδειγμα δέσμευση ή πεδίο-γεγονότα. Συμπερασματικά, οι δύο αλγόριθμοι εξυπηρετούν τους διαφορετικούς σκοπούς: Ο ακριβής αλγόριθμος δίνει την κυρίαρχη συμπεριφορά της εφαρμογής, αλλά μπορεί να παραλείψει στα λιγότερο σταθερά σχέδια ο μη-ακριβής αλγόριθμος δίνει ένα σφαιρικό βλέμμα ποιου είδους σχέδια αναμένονταν. Η συγκέντρωση των πληροφοριών για τη στατιστική συμπεριφορά των διαφορετικών ακολουθιών στην εφαρμογή, επιτρέπει να διακριθεί ο μέσος αριθμός στοιχείων στις ακολουθίες κάθε συγκεκριμένου τύπου. Ο αλγόριθμος για να εξαγάγει αυτές τις πληροφορίες είναι απλός κατά συνέπεια, δεν παρουσιάζεται εδώ. Αυτές οι πληροφορίες και το ιστόγραμμα των διαδικασιών επικαλέσθηκαν γιατί κάθε ακολουθία είναι κρίσιμη όχι μόνο για τον προσδιορισμό των κυρίαρχων τύπων στοιχείων, αλλά και για να αποφασίσουν το είδος των δυναμικών τύπων στοιχείων που πρέπει να χρησιμοποιηθούν στο [61].

7.4 Μελέτη περίπτωσης -Ενσωματωμένο παράδειγμα

Εδώ, παρουσιάζουμε μια ολοκληρωμένη προσέγγιση στο απόσπασμα και εκμεταλλευόμαστε τα χαρακτηριστικά των εφαρμογών, που τρέχουν στα ενσωματωμένα συστήματα με τη βοήθεια του καθορισμού μεταδεδομένων ενός των κοινών λογισμικού. Επομένως, είναι σημαντικό να παρουσιαστεί επίσης ένα ενσωματωμένο παράδειγμα όπου οι διαφορετικές μέθοδοι βελτιστοποίησης εφαρμόζονται σε μια ενιαία εφαρμογή. Επιπλέον, αυτό το παράδειγμα επιδεικνύει πως τα μεταδεδομένα λογισμικού επιτρέπουν τις σχετικές βελτιστοποιήσεις με συνέπεια τα σημαντικά κέρδη στην κατανάλωση ενέργειας και στο αποτύπωμα μνήμης. Ένα δίκτυο TCP/IP όπως το σωρό συμπεριλαμβανομένου Deficit Round Robin(DRR), των εξερχόμενων πακέτων χρησιμοποιείται ως δοκιμαστική εφαρμογή για να επεξηγήσουν τη συνεργατική επίδραση που αποκτάται όταν χρησιμοποιούνται τα μεταδεδομένα λογισμικού από διάφορα εργαλεία βελτιστοποίησης στην ίδια εφαρμογή.

Listing 7.4: "Ακριβής αλγόριθμος ταιριάσματος μετα-σχεδίων"

```
1 Type Pattern a = case Kind of
2 Element => a
3 Pattern => record {
4 count :: Integer ,
5 pattern :: List (Pattern a)
6 }
7 current :: List (Pattern a)
```

```
 initialize  () {
current = new List;
}
process(event) {
case event of
type -> patternize(new List(Element type ));
}
patternize(segment) {
n = segment.length;
 if  (n > windowsize || current.length == 0) {
current.append(segment );
}
 if  (current.get  (-1).  pattern == segment) {
// Segment fits a pattern
// Increase pattern count
current.get  (-1).  count = current.get (-1). count + 1;
// Create a new segment to look for
segment = new List(current.get ( -1));
current.dropLast (1);
// Start fresh with higher -order pattern
patternize(segment );
} elseif  (current.range(-n, -1) == segment) {
// Segment fits another segment
// Drop the segment in current
current.dropLast(n);
// Create a new segment to look for
segment = new List(Pattern 2 segment );
// Start fresh with higher -order pattern
patternize(segment );
} else {
// Try to look for a bigger match
segment.prepend(current.get ( -1));
current.dropLast (1);
patternize(segment );
}
}
```

7.4.1 Στόχος και διαδικασία

Ο τελικός στόχος αυτού του παραδείγματος είναι να βελτιστοποιηθεί η εφαρμογή του συστήματος σε τρία διαφορετικά βασικά σημεία: τα δυναμικά στοιχεία δακτυλογραφούν, δυναμικές διαχείριση μνήμης και μεταφορές block των δυναμικών στοιχείων. Τα διενεργηθέντα βήματα είναι:

Listing 7.5: ”Αλγόριθμος εύρεσης μετα-προτύπων χωρίς ακρίβεια ταιριάσματος”

```
Type Pattern a = case Kind of
Element => a
Pattern => List (Pattern a)
current :: List (Pattern a)
 initialize  () {
current = new List;
}
process(event) {
case event of
type -> patternize(new List(Element type ));
```

```
11 }
12 patternize(segment) {
13 n = segment.length;
14 if (n > windowsize || current.length == 0) {
15 current.append(segment );
16 }
17 if (current.get (-1). pattern == segment) {
18 // Segment fits a pattern
19 // Create a new segment to look for
20 segment = new List(current.get ( -1));
21 current.dropLast (1);
22 // Start fresh with higher -order pattern
23 patternize(segment );
24 } elseif (current.range(-n, -1) == segment) {
25 // Drop the segment in current
26 current.dropLast(n);
27 // Create a new segment to look for
28 if (n == 1) {
29 // Keep current segment , namely of one item.
30 } else {
31 // Generate a pattern out of multiple items.
32 segment = new List(Pattern segment );
33 }
34 // Start fresh with higher -order pattern
35 patternize(segment );
36 } else{
37 // Try to look for a bigger match
38 segment.prepend(current.get ( -1));
39 current.dropLast (1);
40 patternize(segment );
41 }
42 }
```

1. Εξαγωγή των μεταδεδομένων του λογισμικού από την αρχική εφαρμογή που χρησιμοποιεί τις τεχνικές σκιαγράφησης και ανάλυσης που παρουσιάζονται σε αυτό το κεφάλαιο.

2. Βελτιστοποίηση των δυναμικών τύπων στοιχείων, για να μειώσει τον αριθμό προσβάσεων μνήμης και το αποτύπωμα μνήμης των δυναμικών δομών δεδομένων, που χρησιμοποιούνται στην εφαρμογή (πχ. συνδεδεμένοι κατάλογοι, διπλά συνδεδεμένοι κατάλογοι).

3. Βελτιστοποίηση της δυναμικής διαχείρισης μνήμης για να υποστηρίξει μια αποδοτικότερη εκτέλεση των λειτουργιών malloc() και των free(). Μετά από αυτό το βήμα, ο αριθμός προσβάσεων που εκτελούνται στη μνήμη για να διαχειριστούν τη δυναμική μνήμη ελαχιστοποιείται. Αντίστοιχα, το συνολικό αποτύπωμα μνήμης που απαιτείται από το δυναμικό διαχειριστή μνήμης για να εξυπηρετήσει όλες τις απαιτήσεις των εφαρμογών ελαχιστοποιείται, δηλαδή με τη μείωση του εσωτερικού και εξωτερικού κατακερματισμού που προκαλείται από τον ίδιο το διαχειριστή στο [56].

4. Βελτιστοποίηση της συμπεριφοράς μεταφοράς block της εφαρμογής για να χρησιμοποιηθούν οι πόροι DMA για τις μεταφορές των block των δυναμικών στοιχείων (σε αντιδιαστολή με τη μεταφορά των στατικών σειρών ή των μεταβλητών). Το αποτέλεσμα αυτού του βήματος λαμβάνει υπόψη την επίδραση των ταυτόχρονων προσβάσεων από τον επεξεργαστή και το DMA, στον τρόπο πρόσβασης συστοιχίας των μνημών DRAM. Η συμπεριφορά μνήμης της εφαρμογής αξιολογείται μετά από κάθε βήμα, προκειμένου να συγκριθεί ο αντίκτυπος κάθε απόφασης. Στο τέλος, η εφαρμογή προσαρμόζεται για να βεβαιώσει τη συσσωρευμένη επίδραση όλων των βελτιστοποιήσεων.

7.4.2 Η περιγραφή της εφαρμογής οδηγών

Ένα TCP/IP υποσύστημα δικτύων χρησιμοποιείται σε όλο αυτό το παράδειγμα ως εφαρμογή οδηγών. Αυτό το σύστημα οργανώνεται σε διάφορα νήματα που επικοινωνούν μέσω των ασύγχρονων σειρών αναμονής FIFO: η παραγωγή ενός νήματος είναι η είσοδος για μια άλλη. Μια πιο λεπτομερής εξήγηση αυτής της εφαρμογής μπορεί να βρεθεί στο [72]. Βασικά, η εφαρμογή διαιρείται στις ακόλουθες ενότητες:

- Έγχυση πακέτων. Μια συλλογή των πραγματικών (ασύρματων) ιχνών 73 δικτύων, χρησιμοποιείται για να παραγάγει τα πακέτα που τροφοδοτούνται το σύστημα.

- Σχηματισμός πακέτων. Η επιγραφή TCP/IP προστίθεται στα στοιχεία προκειμένου να χτιστεί ένα πλήρες πακέτο.

- Κρυπτογράφηση (DES). Αυτή η ενότητα παρακάμπτεται από τα πακέτα των συνόδων που δεν απαιτούν την κρυπτογράφηση.

- Checksum TCP.

- Διαχείριση σχεδίου και ποιότητα–υπηρεσιών. Ο DRR είναι ένας δίκαιος αλγόριθμος σχεδιασμού δικτύων που χρησιμοποιείται συνήθως για το σχεδιασμό σύμφωνα με το διαθέσιμο εύρος ζώνης [73]. Αυτός ο αλγόριθμος έχει εφαρμοστεί στους διάφορους δρομολογητές (πχ. στη σειρά Cisco 12000).

Αυτά τα υποσυστήματα αποτελούν τη βάση ενός απλουστευμένου σωρού δικτύων. Εξαιτίας του γεγονότος ότι χρησιμοποιούμε τα ίχνη δικτύων που συλλέγονται από τα ασύρματα σημεία πρόσβασης μιας πανεπιστημιούπολης στο [74], και όχι από τις πραγματικές συσκευές, η αναπαραγωγή των λεπτομερειών

όπως η αναμετάδοση πακέτων ή ο έλεγχος εύρους ζώνης γίνεται σχεδόν αδύνατη, ως εκ τούτου παραμένουμε στην απλουστευμένη περιγραφή που παρουσιάσθηκε πριν. Δεδομένου ότι η εφαρμογή είναι πολύπλοκη, πολλά πακέτα είναι ζωντανά συγχρόνως κατά τη διάρκεια της εκτέλεσης κατά συνέπεια, οι προσβάσεις μνήμης εκτελούνται ταυτόχρονα από τα πολλαπλάσια νήματα. Πολυνηματώδης, τελικά σημαίνει ότι οι προσβάσεις μνήμης από τα διαφορετικά νήματα παρεμβάλλουν λευκές σελίδες με έναν λεπτόκοκκο τρόπο και η γενική συμπεριφορά δεν μπορεί να περιγραφεί από την ανεξάρτητη συμπεριφορά κάθε τμήματος. Επομένως, ολόκληρο το σύστημα πρέπει να βελτιστοποιηθεί αντί κάθε νήμα ανεξάρτητα. Η αρχιτεκτονική στόχων για αυτό το σύστημα αποτελείται από ένα στοιχείο επεξεργασίας που συνδέεται με μια ενότητα SRAM και με μια εξωτερική ενότητα DRAM (δείτε τον Πίνακα 7.1 για μια περιγραφή των παραμέτρων εργασίας τους), όπως παρουσιάζεται στο [72]. Μια μηχανή άμεσης πρόσβασης μνήμης (DMA), φροντίζει για τις μεταφορές στοιχείων μεταξύ της εξωτερικής μνήμης και της εσωτερικής, με αποδοτικό τρόπο, και από οποιεσδήποτε από τις μνήμες προς τις εξωτερικές συσκευές (δηλαδή απομονωτές υλικού στους προσαρμογείς δικτύων). Ο επεξεργαστής μπορεί να έχει πρόσβαση και στις δύο μνήμες άμεσα εντούτοις, η πρόσβαση στο εξωτερικό DRAM πρέπει να συντονιστεί με το DMA για να αποφύγει τις περιττές ποινικές ρήτρες ενέργειας και λανθάνουσας κατάστασης, λόγω των παρεμβάσεων στις σελίδες.

Πίνακας 7.1: Ενέργεια προς Πρόσβαση. Το SDRAM διαμορφώνεται σύμφωνα με την προδιαγραφή μικρού PC100 υποθέτοντας CL = 2 και ένα ρολόι συστημάτων και επεξεργαστών 100MHz.

Ενέργεια/Πρόσβαση	3.5 nJ
Ενεργοποίηση Ενέργειας/Προφόρτιση	10 nJ
Καθυστέρηση CAS	2 Κύκλοι
Καθυστερημένη Προφόρτιση	2 Κύκλοι
Ενεργό για να διαβάσει ή να γράψει	2 Κύκλοι
Αποκατάσταση γραφής	2 Κύκλοι
Τελευταία Δεδομένα να διαβαστούν/γραφούν εκ νέου	1 Κύκλος
Μέγιστο μήκος ριπής	1.024 λέξεις

7.4.3 Σκιαγράφηση και η ανάλυση

Η εφαρμογή οδηγών ενοργανώνεται και σχεδιάζεται το περίγραμμα, και οι αποσπασματικές πληροφορίες αναλύονται όπως εξηγούνται στο τμήμα 7.4.2. Μετά από αυτό το βήμα, εκτελέστε συνεχώς τις βελτιστοποιήσεις και τα αρχικά μεταδεδομένα λογισμικού είναι έτοιμα. Οι πληροφορίες που εξάγονται περιλαμβάνουν:

- Για κάθε δυναμική δομή δεδομένων, τον αριθμό των λειτουργιών που εκτέλεσε, τον αριθμό προσβάσεων μνήμης και το συνολικό αποτύπωμα μνήμης.

- Για το δυναμικό διαχειριστή μνήμης, τον αριθμό προσβάσεων μνήμης που εκτελούνται για να διαχειριστούν όλα τα ελεύθερα χρησιμοποιημένα block της δυναμικής μνήμης, και το συνολικό ποσό μνήμης που χρησιμοποιείται πραγματικά για να εξυπηρετήσει όλες τις αιτήσεις από την εφαρμογή. Η συνολική χρήση της μνήμης εξαρτάται από τον εσωτερικό και εξωτερικό κατακερματισμό τα οποία προκαλούνται από το διαχειριστή.

- Τέλος, τον αριθμό των προσβάσεων σε κάθε δυναμικό αντικείμενο και το σχέδιο πρόσβασης κάθε νήματος στο σύστημα. Αυτές οι πληροφορίες επιτρέπουν τους πιο σχετικούς δυναμικούς τύπους στοιχείων στην εφαρμογή. Ο πρώτος αντιστοιχεί στα πακέτα που δημιουργούνται για να σταλούν. Ο τεράστιος αριθμός τους απαιτεί την υψηλή αποδοτικότητα από το δυναμικό διαχειριστή μνήμης. Οι επόμενες πιο σχετικές δομές δεδομένων είναι ο κατάλογος κόμβων και η σειρά αναμονής των πακέτων που χτίζονται από την ενότητα DRR. Ο κατάλογος κόμβων αντιπροσωπεύει τους οικοδεσπότες στον οποίο μια σύνδεση είναι ενεργός, δηλαδή, είναι εκεί μια είσοδος για κάθε οικοδεσπότη προορισμού για την οποία υπάρχουν τα πακέτα περιμένοντας να αποσταλλούν Κάθε είσοδος στον κατάλογο περιέχει τη σειρά αναμονής των πακέτων περιμένοντας να σταλεί προς αυτόν τον προορισμό. Και οι δύο δομές δεδομένων είναι δυναμικές επειδή ο αριθμός στοιχείων τους ποικίλλει στο χρόνο εκτέλεσης κατά συνέπεια, είναι καλοί υποψήφιοι για τις τεχνικές βελτιστοποίησης τύπων στοιχείων.

Τέλος, η τελευταία των σχετικών δυναμικών δομών δεδομένων που χρησιμοποιούνται στην εφαρμογή, είναι η ασύγχρονη σειρά αναμονής FIFO που τα νήματα χρησιμοποιούν για να περάσουν τα μηνύματα μεταξύ τους. Αυτή η δομή δεδομένων κάνει τη βαριά χρήση των πρωτόγονων συγχρονισμού, οι οποίοι είναι από το πεδίο αυτής της εργασίας. Εντούτοις, η καλύτερη εφαρμογή αυτής της δομής δεδομένων μπορεί να καθοριστεί με έναν απλό τρόπο επειδή παρουσιάζει ένα πολύ κανονικό σχέδιο πρόσβασης.

Κάθε ένα από τα επόμενα τρία βήματα βελτιστοποίησης στηρίζεται στα μεταδεδομένα λογισμικού που εξάγονται σε αυτό το σημείο και που ενημερώνονται από τα αντίστοιχα εργαλεία.

7.4.4 Δυναμικός τύπος καθαρισμού στοιχείων -DDTR

Η πρώτη βελτιστοποίηση που εφαρμόζεται σε αυτό το παράδειγμα είναι ο καθαρισμός των δυναμικών δομών δεδομένων της εφαρμογής (DDTR). Μετά από τα βήματα σκιαγράφησης και ανάλυσης, οι δυναμικοί τύποι στοιχείων οντοτήτων μεταδεδομένων, οι δυναμικές περιπτώσεις τύπων στοιχείων και οι δυναμικές διαδικασίες τύπων στοιχείων περιέχουν τις πληροφορίες για τον τύπο και τον αριθμό διαδικασιών που εκτελούνται για κάθε τύπο στοιχείων. Επιπλέον, αυτές οι πληροφορίες είναι επίσης διαθέσιμες για τις συγκεκριμένες περιπτώσεις τους, το οποίο το καθιστά πιθανό να μελετήσει και να βελτιστοποιήσει κάθε μια από τις διαδικασίες, που διενεργήθηκαν συχνότερα. Οι μέθοδοι σκιαγράφησης και ανάλυσης επιτρέπουν να διαφοροποιήσουν μεταξύ των προσβάσεων που διανέμονται στις δομές δεδομένων, και των προσβάσεων στις εσωτερικές δομές των τύπων στοιχείων εφαρμογής. Αυτή η διαφοροποίηση το καθιστά πιθανό να προσδιορίσει τους πιο σχετικούς τύπους στοιχείων για τη βελτιστοποίηση, ανεξάρτητα από την αρχική εφαρμογή τους. Επιπλέον, οι τεχνικές βελτιστοποίησης μας εστιάζουν στη βελτιστοποίηση των γενικών εξόδων που επιβάλλονται από κάθε δομή δεδομένων, αλλά δεν μπορούν να μειώσουν τον αριθμό προσβάσεων που οι αλγόριθμοι εφαρμογής εκτελούν. Προκειμένου να μειωθεί ο αριθμός προσβάσεων που διανέμονται στις δυναμικές δομές δεδομένων (σε αντίθεση με τον αριθμό προσβάσεων μνήμης που εκτελούνται πραγματικά στο υποσύστημα μνήμης), οι βελτιστοποιήσεις σε πιο υψηλό επίπεδο αφαίρεσης που είναι από το πεδίο αυτού του κεφαλαίου θα ήταν το ζητούμενο.

Οι αποσπασματικές πληροφορίες αποκαλύπτουν ότι οι δύο δυναμικές περιπτώσεις στοιχείων που συγκεντρώνουν τις περισσότερες από τις προσβάσεις εφαρμογής είναι ο κατάλογος κόμβων στον αλγόριθμο DRR (δυναμική δομή «A») και η σειρά αναμονής των εκκρεμών πακέτων για κάθε έναν από τους κόμβους (δυναμική δομή «B»). Χρησιμοποιώντας τις μεθόδους που εξηγούνται στο προηγούμενο κεφάλαιο, επιλέγουμε την καλύτερη επιλογή για τις αντίστοιχες εφαρμογές τους. Αν και το τρέξιμο μιας εξαντλητικής εξερεύνησης όλων των περιπτώσεων δεν απαιτείται για να πάρει τη βέλτιστη λύση, σε αυτό το πείραμα τρέξαμε ένα πλήρες σκούπισμα όλων των συνδυασμών για να εκτελέσουμε μια πρόσθετη συγκριτική δοκιμή και να επικυρώσουμε την προσέγγιση βελτιστοποίησης. Για κάθε μια από τις δυναμικές δομές δεδομένων, μια από τις ακόλουθες εφαρμογές μπορεί να επιλεχτεί (περισσότερες λεπτομέρειες σε κάθε μια από τις εφαρμογές μπορούν να βρεθούν [75]):

1. Σειρά δεικτών.

2. Σειρά αντικειμένων.

3. Ενιαίος συνδεομένος κατάλογος.
4. Διπλός συνδεομένος κατάλογος.
5. Ενιαίος συνδεομένος κατάλογος με το δείκτη περιπλάνησης.
6. Διπλός συνδεομένος κατάλογος με το δείκτη περιπλάνησης.
7. Ενιαίος συνδεομένος κατάλογος σειρών.
8. Διπλός συνδειμένος κατάλογος σειρών.
9. Ενιαίος συνδεδεμένος κατάλογος σειρών με το δείκτη περιπλάνησης.
10. Διπλός συνδεδεμένος κατάλογος σειρών με το δείκτη περιπλάνησης.

Τα δέντρα δεν αντιπροσωπεύονται σε αυτήν την ιεραρχία, αλλά μπορούν να χτιστούν χρησιμοποιώντας έναν συνδυασμό αυτών των δομών δεδομένων. Για την αποδοτικότερη εξερεύνηση των δέντρων, θα απαιτούταν μια περαιτέρω μελέτη των πιθανών διεπαφών που ένα δέντρο μπορεί να εκθέσει. Δεδομένου ότι οι περισσότερες εφαρμογές δέντρων είναι μάλλον ειδικές, και από υψηλή άποψη οι διαφορετικοί τύποι δέντρων έχουν τις διαφορετικές ιδιότητες, θα ήταν απαραίτητο να ταξινομεί αρχικά τις διαφορετικές δομές δεδομένων δέντρων, πριν καθοριστούν οι ομάδες διεπαφών. Μερικές δομές δεδομένων δέντρων, όπως τα διατεταγμένα δέντρα, συμπεριφέρονται πλήρως όπως τα σύνολα. Άλλες δομές δεδομένων δέντρων έχουν την πιο θεμελιώδη κωδικοποίηση στη δομή δεδομένων δέντρων, όπως οι ιδιότητες γονέας-παιδιών.

Στο υπόλοιπο αυτού του τμήματος, το AI-BJ αντιπροσωπεύει το συνδυασμό της εφαρμογής για το A και της εφαρμογής j για το B. Παραδείγματος χάριν, A1-B3 αντιπροσωπεύει ότι μια σειρά δεικτών χρησιμοποιείται ως εφαρμογή για τον κατάλογο κόμβων στον αλγόριθμο DRR και ένας ενιαίος-συνδεδεμένος κατάλογος χρησιμοποιείται ως εφαρμογή για τη σειρά αναμονής των εκκρεμών πακέτων για κάθε κόμβο. Υπάρχουν δέκα πιθανές εφαρμογές για κάθε μια από τις δυναμικές δομές. Εξαιτίας του γεγονότος ότι θελήσαμε να κάνουμε μια πλήρη δοκιμή όλων των περιπτώσεων, χρησιμοποιήσαμε δέκα τρία εισηγμένα ίχνη ως είσοδο και εκτελέσαμε δέκα επαναλήψεις που αντιμετωπίζουν τις στατιστικές παραλλαγές, ο συνολικός αριθμός τρεξίματος συνδυασμών που ανήλθε σε 13.000. Εντούτοις, αυτός ο υψηλός αριθμός εκτελέσεων εκτελέσθηκε για να επικυρώσει ακριβώς τη βελτιστοποίηση με μια εξαντλητική δοκιμή, αλλά σε πρακτικές περιπτώσεις δεν υπάρχει καμία ανάγκη να εκτελεσθούν. Για αυτό το εκτενές πείραμα, τα μεταδεδομένα για κάθε μια από τις δυναμικές περιπτώσεις στοιχείων συλλέχθηκαν από τους δυναμικούς τύπους στοιχείων οντοτήτων

μεταδεδομένων, δυναμικές περιπτώσεις τύπων στοιχείων και δυναμικές διαδικασίες τύπων στοιχείων, με τις πρόσθετες πληροφορίες στις πληροφορίες πρόσβασης (σχήμα 7.3). Δύο συμπεράσματα προέρχονται από τα αποτελέσματα των πειραμάτων. Ο πρώτος είναι η επιβεβαίωση της αποδοτικότερης δομής δεδομένων σχετικά με το συνολικό αριθμό προσβάσεων μνήμης. Ο δεύτερος είναι η ανάλυση της δομής δεδομένων που μειώνει πιο πολύ το συνολικό ποσό μνήμης (αποτύπωμα μνήμης) που απαιτείται για να εκτελέσει την εφαρμογή.

Μείωση του αριθμού προσβάσεων μνήμης

Τα πειραματικά αποτελέσματα δείχνουν ότι ο αποδοτικότερος συνδυασμός δυναμικών δομών δεδομένων είναι A3-B3, το οποίο σημαίνει ότι μια ενιαία-συνδεμένη εφαρμογή καταλόγων πρέπει να χρησιμοποιηθεί και για τον κατάλογο ενεργών κόμβων σε DRR και για τη σειρά αναμονής των πακέτων περιμένοντας να σταλεί για κάθε έναν από τους κόμβους. Αυτή η λύση είναι η βέλτιστη για 11, των 13 εισαγωγών που εξετάστηκαν. Ο πίνακας 7.2 παρουσιάζει τον αριθμό προσβάσεων που απαιτούνται από τη βέλτιστη λύση για κάθε εισαγωγή, και τον αριθμό προσβάσεων μνήμης που απαιτούνται από την επιλεγμένη λύση A3-B3. Η τελευταία στήλη παρουσιάζει διαφορά ποσοστού και μεταξύ των δύο. Είναι σχετικό να σημειωθεί ότι η μέση διαφορά από A3-B3 σε μια υποθετική «τέλεια» λύση 2 είναι τόσο μικρή όπως 0.28%.

Πίνακας 7.2: Συγκρίσεις αριθμού προσβάσεων

Είσοδος	Προσβάσεις Μνήμης A3-B3	Βέλτιστη Πρόσβαση Μνήμης	Διαφορά επί %
01	4328501556	4247377390	
02	3589773903	3530754848	1.91
03	8004231	8004231	1.67
04	1867812	1867812	0
05	9622944	9622944	0
06	25225070	25225070	0
07	192869416	192869416	0
08	183636289	183636289	0
09	12699747	12699747	0
10	45429504	45429504	0
11	239756665	239756665	0
12	19082840	19082840	0
13	250188588	250188588	0
Μέση διαφορά			0.28

Μείωση του αποτυπώματος μνήμης

Στην περίπτωση του αποτυπώματος μνήμης, μια βέλτιστη λύση Παρέτο που εξετάζει επίσης την επίδραση στον αριθμό προσβάσεων μνήμης αξιολογήθηκε. Ο Πίνακας 7.3 παρουσιάζει τα αποτελέσματα που επιτυγχάνονται από τα πειράματα. Σε αυτήν την περίπτωση, το A3-B3 είναι βέλτιστο μόνο σε μια από τις 13 εισηγμένες περιπτώσεις. Η γενική διαφορά από αυτήν την λύση στη βέλτιστη λύση αποτυπώματος μνήμης (με τις πολύ κακές ανταλλαγές πρόσβασης μνήμης) είναι μόνο 3.33%. Αυτό σημαίνει ότι, ακόμα κι αν δεν υπάρχει μια «τέλεια» λύση για να μειώσει τη μνήμη αποτυπώματος της εφαρμογής, η επιλογή της καλύτερης λύσης συμβιβασμού για να μειώσει τον αριθμό προσβάσεων δεν επιβάλλει μια σημαντική ποινική ρήτρα στο συνολικό ποσό μνήμης που απαιτείται.

Πίνακας 7.3: Αποτελέσματα Πειραμάτων ως προς το αποτύπωμα μνήμης

Είσοδος	Αποτύπωμα Μνήμης A3-B3	Βέλτιστο Αποτύπωμα Μνήμης	Διαφορά επί %
01	9028421	8992232	0.40
02	7882678	7872560	0.13
03	1978361	1809807	9.31
04	47778	47778	0
05	992569	953663	4.08
06	2178989	2025122	7.60
07	186441	185851	0.32
08	174315	171398	1.70
09	1589535	1532995	3.69
10	3316557	3271309	1.38
11	760281	716898	6.05
12	772158	732497	5.41
13	471254	456433	3.25
Μέση Διαφορά			3.33

7.4.5 Δυναμική διαχείριση καθαρισμού μνήμης-DMMR

Μόλις βελτιστοποιηθούν οι δυναμικές δομές δεδομένων της εφαρμογής και οι αντίστοιχες πληροφορίες μεταδεδομένων ενημερωθούν (δυναμικές σχετικές με τα στοιχεία οντότητες, σχήμα 7.3), εκτελείται η βελτιστοποίηση του δυναμικού διαχειριστή μνήμης, όπως περαιτέρω εκτίθεται λεπτομερώς στο επόμενο κεφάλαιο. Αναφερόμαστε στο δυναμικό διαχειριστή μνήμης ως ενσωματωμένο

σύνολο συγκεκριμένων δυναμικών διαχειριστών μνήμης εφαρμογής (δηλαδή οι διαχειριστές DM επιπέδων εφαρμογής μεταγλωττίστηκαν με κάθε εφαρμογή λογισμικού). Εάν περισσότερες από μια εφαρμογές είναι παρούσες, κατόπιν κάθε ένας χρησιμοποιείται κατά τη μεταγλώττιση της εφαρμογής.

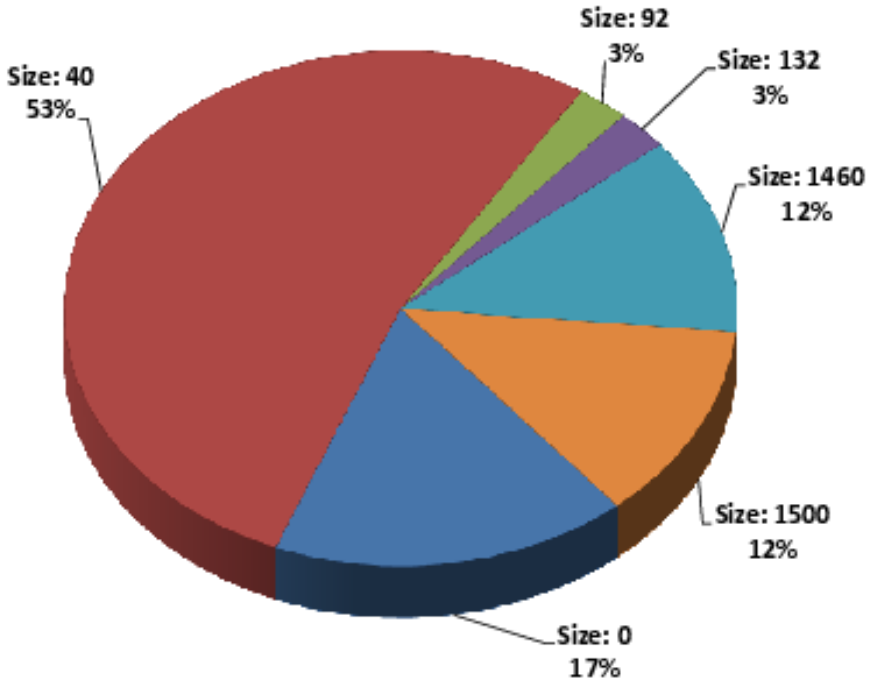
Σε κάθε περίπτωση, όλοι οι προσαρμοσμένοι διαχειριστές DM μοιράζονται τις κοινές υπηρεσίες των επιπέδων του ΛΣ για την παροχή των μεγάλων block μνήμης (πχ. sbrk() και mmap()). Οι πληροφορίες μεταδεδομένων για αυτό το βήμα περιλαμβάνουν τον αριθμό block των ποικίλων μεγεθών που διατίθενται από την εφαρμογή για τα πακέτα δικτύων (οργανισμοί στοιχείων και επιγραφές δικτύων ανεξάρτητα), τον κατάλογο κόμβων προορισμού για τον αλγόριθμο DRR και τις σειρές αναμονής των εκκρεμών πακέτων για κάθε κόμβο.

Η εκτέλεση αυτού του βήματος εφόσον έχουν βελτιστοποιηθεί οι δυναμικές δομές δεδομένων είναι σημαντική, επειδή ο αριθμός κατανομών και μη-κατανομών μνήμης δεν θα αλλάξει άλλο λόγω των τροποποιήσεων εφαρμογής. Κάθε φορά που δημιουργείται μια νέα περίπτωση οποιοιδήποτε από αυτούς τους δυναμικούς τύπους στοιχείων, ο δυναμικός διαχειριστής μνήμης πρέπει να ψάξει για έναν κατάλληλο ελεύθερο block, δηλαδή ένα block ικανοποιητικού μεγέθους που εκτείνεται πέρα από μια σειρά των διευθύνσεων μνήμης, που δεν καταλαμβάνονται αυτήν τη χρονική στιγμή από οποιοδήποτε άλλο αντικείμενο.

Αντιθέτως, όταν καταστρέφεται μια μεταβλητή, το διάστημα μνήμης που χρησιμοποιούσε πρέπει να επανενταχθεί στην ομάδα των διαθέσιμων διευθύνσεων για τις νέες περιπτώσεις. Όλες αυτές οι διαδικασίες παράγουν γενικά κόστη στο συνολικό ποσό εργασίας που το σύστημα πρέπει να κάνει. Επιπλέον, ο διαχειριστής μνήμης απαιτεί κάποια πρόσθετη μνήμη για τη λειτουργία του και ολόκληρη η λογιστική της διαδικασίας αυξάνει το συνολικό αριθμό προσβάσεων μνήμης. Επομένως, η καταλληλότητα ενός δεδομένου δυναμικού διαχειριστή μνήμης εξαρτάται από τα γενικά κόστη που επιβάλλει στο σύστημα και την πρόσθετη μνήμη που απαιτούνται λόγω της ύπαρξης του εσωτερικού και εξωτερικού τεμαχισμού στο[56].

Το μέγεθος των διατιθέμενων block μνήμης και η συχνότητα εμφάνισής τους, μαζί με το σχέδιο των κατανομών και μη-κατανομών, καθορίζουν το χαρακτηριστικό του καταλληλότερου διαθέτη μνήμης. Αυτοί οι αριθμοί καθορίζονται στα μεταδεδομένα λογισμικού της εφαρμογής (δυναμικά στοιχεία και τη Pool σχετικές με οντότητες, Εικόνα 7.3).

Το σχήμα 7.4 παρουσιάζει τη συχνότητα της εμφάνισης κάθε μεγέθους δέσμευσης στην εφαρμογή οδηγών, για τα δημοφιλέστερα μεγέθη. Ο αριθμός το καθιστά σαφές ότι ο διαχειριστής μνήμης για αυτήν την εφαρμογή πρέπει να βελτιστοποιηθεί, για να διαθέσει τις μεγάλες ποσότητες πακέτων από ένα μικρό σύνολο μεγεθών block. Προκειμένου να μειωθεί ο αριθμός προσβάσεων μνήμης, ο διαχειριστής μνήμης πρέπει να εντοπίσει τον πιο κατάλληλο ελεύθερο block με το λιγότερο ποσό προσβάσεων μνήμης. Για αυτόν τον λόγο, ένας διαχειρι-



Σχήμα 7.4: Διανομή συχνότητας των «δημοφιλών» μεγεθών κατανομής για όλες τις περιπτώσεις εισαγωγής.

στής μνήμης που παίρνει ελεύθερο χώρο από μια σφαιρική Pool αρχικά, αλλά έπειτα ελευθερώνει τα block στους καταλόγους συγκεκριμένων μεγεθών επιλέγεται. Επιπλέον, προκειμένου να μειωθεί το ποσό εσωτερικού τεμαχισμού, είναι επίσης δυνατό να δημιουργηθούν οι κατάλογοι ελεύθερων block για έναν μικρό αριθμό πρόσθετων μεγεθών που περιορίζουν το ποσό σπαταλημένης μνήμης για τα ανώμαλα μεγέθη. Η δομή του διαθέτη μνήμης κρατιέται στην είσοδο poolDescription της οντότητας Pool. (σχήμα. 7.3).

Πίνακας 7.4: Διαμόρφωση δυναμικών διαχειριστών μνήμης που αξιολογούνται σε αυτό το πείραμα.

DMM	Περιγραφή
DMM 1	Kingsley-όπως [126] διαχείριση μνήμης με κάδους για τα τεμάχια των 128 διαφορετικών μεγεθών, 8-16384 bytes. Αυτή η δημοφιλής διαχείριση μνήμης χρησιμοποιείται σε όλο το υπόλοιπο αυτού του τμήματος ως σημείο αναφοράς για σύγκριση με προσαρμοσμένους διαχειριστές δυναμικής μνήμης
DMM 2	Ειδικές σωρούς με τις λίστες των ελεύθερων μπλοκ των μεγεθών 40, 1460 και 1500
DMM 3	Προσαρμοσμένη σωρό με τις λίστες των ελεύθερων μπλοκ των μεγεθών ακριβώς 40, ακριβώς 1460, ακριβώς 1500, έως 92, έως 132, έως 256, έως 512 και έως 1024 bytes. Η συγκεκριμένη σειρά περιορισμών της ”έως” και ”ακριβώς” εξασφαλίζουν ότι τα πιο κοινά μεγέθη δέσμευσης απαιτούν τον ελάχιστο αριθμό προσβάσεων για να βρείτε ένα κατάλληλο μπλοκ
DMM 4	Προσαρμοσμένη σωρό με τις λίστες των ελεύθερων μπλοκ των μεγεθών 40, 1460 και 1500 byte, συν υποστήριξη για το διαχωρισμό και συνενώνονται
DMM 5	Ειδικές σωρούς με τις λίστες των ελεύθερων μπλοκ των μεγεθών 40, 1460, 1500, 92 και 132 byte (κατά σειρά προτεραιότητας αναζήτησης), καθώς και υποστήριξη για το διαχωρισμό και συνένωση
DMM 6	Όπως στο DMM 2, καθώς και ειδική μεταχείριση για μπλοκ με μηδενικά bytes (εφαρμογή ειδικής βελτιστοποίησης)
DMM 7	Όπως DMM 3, καθώς και ειδική μεταχείριση για μπλοκ με μηδενικά bytes (εφαρμογή ειδικής βελτιστοποίησης)
DMM 8	Όπως DMM 4, καθώς και ειδική μεταχείριση για μπλοκ με μηδενικά bytes (εφαρμογή ειδικής βελτιστοποίησης)
DMM 9	Όπως DMM 5, καθώς και ειδική μεταχείριση για μπλοκ με μηδενικά bytes (εφαρμογή ειδικής βελτιστοποίησης)

Μια σημαντική εκτίμηση είναι ότι η εξεταζόμενη εφαρμογή πραγματικά

ζητά τα block μηδέν bytes στο μέγεθος. Αυτό δεν είναι ένα λάθος, αλλά μια συνέπεια της ανάγκης να σταλούν αναγνωριστικά πακέτα. Δεδομένου ότι αυτά τα TCP τμήματα 0 bytes κινούνται προς το επίπεδο IP, το σώμα τμήματος μηδέν bytes κρατιέται και μια επιγραφή 40 bytes TCP+IP προστίθενται με τις προηγούμενες εκτιμήσεις, και οι μέθοδοι που παρουσιάζονται στο επόμενο κεφάλαιο, το διάστημα σχεδίου του δυναμικού διαχειριστή μνήμης, μπορεί να στενέψουν σε μερικές επιλογές. Για την εφαρμογή οδηγών αυτού του παραδείγματος, αξιολογήσαμε τους δυναμικούς διευθυντές μνήμης που περιγράφηκαν στον πίνακα 7.4. Μετά από κάθε πείραμα, εξετάσαμε τις πληροφορίες που παράγονται στη Pool, Pool, Allocation Information, Dynamic Data, Dynamic Data Instances και Control Flow στις οντότητες μεταδεδομένων (σχήμα. 7.3) και τις συλλέξαμε για να αξιολογήσουμε την ποιότητα κάθε λύσης.

Προτεινόμενα block: Πρέπει να καθοριστούν τα «πρόσθετα» μεγέθη block μνήμης ίσα με κάθε μέγεθος πακέτων που αντιπροσωπεύει τουλάχιστον 10% των γενικών μεγεθών πακέτων. Το υπόλοιπο των προκαθορισμένων block μνήμης πρέπει να είναι μέγεθος δύναμης του δύο μέχρι το μέγεθος MTU. Στην περίπτωση IEEE 802.11b, αυτό σημαίνει ότι απαιτείται ένα πρόσθετο προκαθορισμένο block μνήμης 40 bytes και ένας πρόσθετο προκαθορισμένο block μνήμης 1.500 bytes. Το υπόλοιπο των προκαθορισμένων block μνήμης πρέπει να έχει 256, 512 και 1.024 bytes, αντίστοιχα.

Για μερικές από τις περιπτώσεις που εξετάστηκαν στα πειράματα αυτού του κεφαλαίου, προσθέσαμε δύο μικρότερα μεγέθη (92 και 132 bytes). Επομένως, τα δημοφιλέστερα αιτήματα μνήμης μπορούν ικανοποιηθούν χωρίς οποιοδήποτε εσωτερικό κατακερματισμό και η τα λιγότερο δημοφιλή αιτήματα μπορεί να ικανοποιηθούν με ένα λογικό εσωτερικό τεμαχισμό. Επιπλέον, η απόφαση σχεδίου του προκαθορισμού τα block των σταθερών μεγεθών, στον χρόνο σχεδιασμού, δίνει το πλεονέκτημα απόδοσης και έτσι δε χρειάζεται να υπολογίσει το μέγεθος block για κάθε αίτημα στο χρόνο εκτέλεσης.

Προτεινόμενη συγχώνευση και διαχωρισμός των block: Κανένας διαχωρισμός ή συγχώνευση των block μνήμης δεν πρέπει να χρησιμοποιηθεί. Δεδομένου ότι το μέγιστο ζητούμενο μέγεθος block είναι ήδη γνωστό (το MTU των πακέτων), δεν υπάρχει καμία ανάγκη να συγχωνευτεί block για να εξετάσει τον εξωτερικό τεμαχισμό. Επιπλέον, ο καθορισμός των μεγεθών block που αποτρέπουν τον μεγαλύτερο μέρος του εσωτερικού τεμαχισμού (δηλαδή ο εσωτερικός τεμαχισμός που παράγεται από τα δημοφιλή αιτήματα), μειώνει την ανάγκη να χωριστούν τα block. Επιπλέον, και οι συγχωνευμένοι μηχανισμοί είναι υπολογιστικά εντατικοί, κατά συνέπεια μπορούν να επιβραδύνουν ουσιαστικά τις διαδικασίες δέσμευσης και απελευθέρωσης και μπορούν επίσης να υποστούν έναν σημαντικό αριθμό προσβάσεων μνήμης για να μετασχηματίσουν τα παλαιά μεγέθη block στα νέα.

Προτεινόμενες δεξαμενές: Πρέπει να δημιουργηθούν διάφορες δεξαμενές

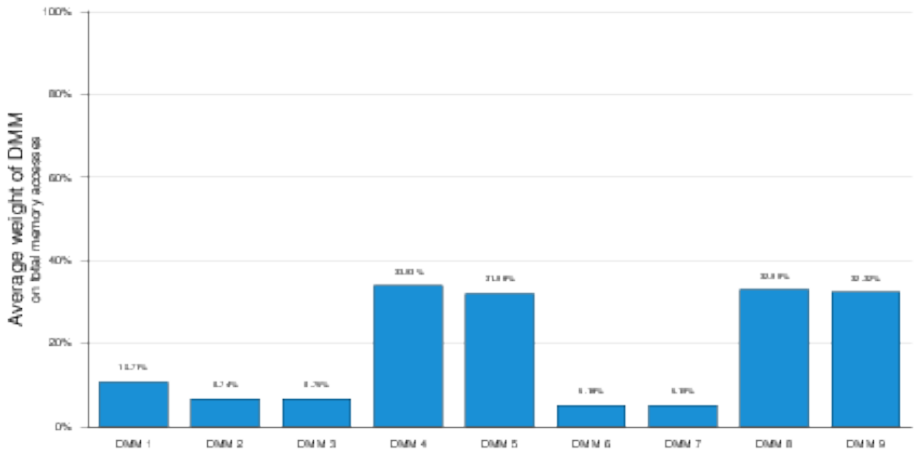
ίσες με τον αριθμό των προκαθορισμένων μεγεθών block. Στην παρούσα περίπτωση, δύο πρόσθετες δεξαμενές που κρατούν τα 40 bytes και τα 1500 bytes, και μια δεξαμενή για κάθε πρόσθετο μέγεθος block που δημιουργείται.

Αυτή η αδύνατη οργάνωση Pool προτιμάται αντί πιο σύνθετων επειδή επιτρέπει μια γρηγορότερη πρόσβαση στη συγκεκριμένη Pool μνήμης, που θα συντηρήσει κάθε αίτημα: στη χειρότερη περίπτωση, ο αριθμός προσβάσεων μνήμης που θα απαιτηθεί για να βρει τη Pool που κρατά το block του κατάλληλου μεγέθους είναι της τάξεως του αριθμού Pool. Τέλος, μόλις ληφθεί η απόφαση της μη υποστήριξης της συγχώνευσης ούτε του διαχωρισμού των block, το DMM δεν πρέπει να υποστηρίξει (τις δαπανηρές στην απόδοση) μετακινήσεις των block μεταξύ των Pool.

Προτεινόμενοι κατάλληλοι αλγόριθμοι: Προκειμένου να επιλεχτεί μια Pool, προτείνονται οι κατάλληλοι αλγόριθμοι. Η ακριβής τακτοποίηση (best fit) χρησιμοποιείται μόνο για να κάνει διακρίσεις μεταξύ των πρόσθετων Pool, ενώ η πρώτη τακτοποίηση (first fit) χρησιμοποιείται μεταξύ των υπολοίπων. Μόλις επιλεχτεί μια Pool, η πρώτη μόνο τακτοποίηση χρησιμοποιείται για να επιλέξει τον κατάλληλο block. Αυτή η διαμόρφωση απαιτεί τις λιγότερες προσβάσεις προκειμένου να βρεθούν τα block μνήμης για να ικανοποιήσει ένα αίτημα μνήμης. Εάν γίνει λανθασμένος συνδυασμός μεγεθών Pool και κατάλληλων αλγορίθμων, ο αριθμός προσβάσεων μνήμης που εκτελούνται από το διαχειριστή DM μπορεί να αυξήσει σημαντικά και να οδηγήσει στους πιο μακροχρόνιους χρόνους εκτέλεσης και τις μεγαλύτερες καταναλώσεις ενέργειας. Σαν τελική εκτίμηση για να παρακινήσει τη σχετικότητα του δυναμικού διαχειριστή μνήμης για τη σφαιρική απόδοση της πλήρους εφαρμογής που χρησιμοποιείται σε αυτό το παράδειγμα, η Εικόνα 7.5 παρουσιάζει το βάρος των προσβάσεων μνήμης λόγω της δυναμικής διαχείρισης μνήμης πέρα από το συνολικό αριθμό προσβάσεων μνήμης, για κάθε έναν από τους διαχειριστές μνήμης που χρησιμοποιούνται στα πειράματά μας. Οι δυναμικοί διαχειριστές μνήμης 2, 3, 6 και 7 απαιτούν έναν σχετικά χαμηλό αριθμό προσβάσεων μνήμης για να εκτελέσουν την εργασία τους. Αντίθετα, τα DMMs 4, 5, 8 και 9 εισάγουν περισσότερα από ένα τρίτο των προσβάσεων μνήμης που απαιτούνται από την εφαρμογή για να ρυθμιστεί ακριβώς η δυναμική μνήμη. Το DMM 1 εισάγει ένα μέτριο 10% των πρόσθετων προσβάσεων.

Το Σχήμα 7.6 παρουσιάζει συνολικό ποσό προσβάσεων μνήμης που απαιτούνται για να επεξεργαστούν κάθε περίπτωση εισαγωγής με τα DMMs 1, 6 και 7. Κατόπιν, το Σχήμα 7.7 αποκαλύπτει ότι για τις εντατικές περιπτώσεις στοιχείων, όπου τα συνήθως μεγάλα πακέτα στέλνονται και η εργασία που απαιτείται για να επεξεργαστεί τα στοιχεία εκτοπίζει την εργασία που απαιτείται για να διαθέσει τα block, ο αριθμός προσβάσεων λόγω της δυναμικής διαχείρισης μνήμης αντιπροσωπεύει λιγότερο από 1% του συνόλου (είσοδοι 1 και 2 και στους δύο αριθμούς). Εντούτοις, όταν μπορεί η εισαγωγή των δυνάμεων

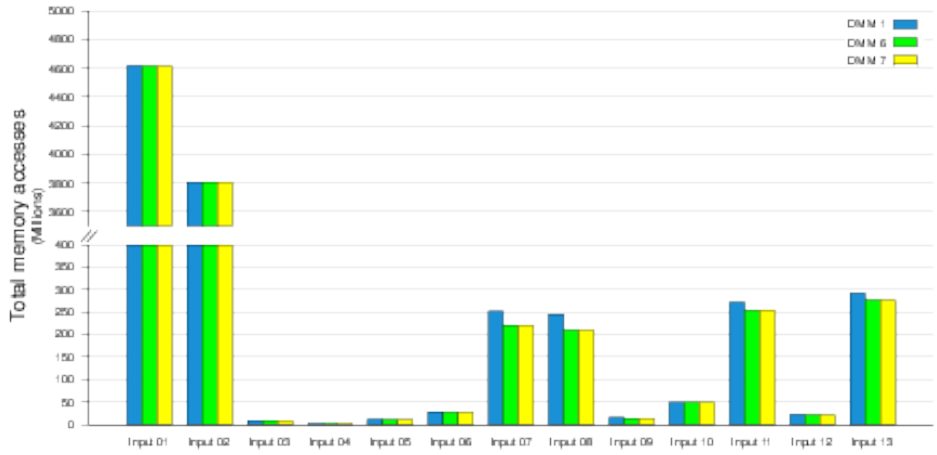
εφαρμογής για να στείλει πολλά μικρά πακέτα (δηλαδή τα μικρά τμήματα TCP στέλνονται λόγω της ανάγκης), ο αριθμός προσβάσεων λόγω της διαχείρισης μνήμης να χρησιμοποιήσει μέχρι ένα 24% του συνόλου για την εισαγωγή. Ενδιαφέρον είναι επίσης, ότι τα γενικά κόστη των DMMs 6 και 7 είναι περίπου το μισό από αυτό, που υπογραμμίζει τη σημασία της απόδοσης του δυναμικού διαχειριστή μνήμης. Μόλις ερευνήσουμε τις βασικές παραμέτρους του διαχειριστή μνήμης και μειώσουμε το διάστημα σχεδίου για το διαχειριστή μνήμης σε ένα εύχρηστο μέγεθος, τρέξαμε διάφορες προσομοιώσεις για να αναλύσουμε την απόδοση κάθε επιλογής όσον αφορά τον αριθμό προσβάσεων μνήμης και αποτυπώματος μνήμης. Ένα μειωμένο διάστημα σχεδίου επιτρέπει μια λεπτομερή ανάλυση και τον καταλληλότερο προσαρμοσμένο δυναμικό διαχειριστή μνήμης.



Σχήμα 7.5: Ποσοστό των προσβάσεων διαχειριστών μνήμης πέρα από τις συνολικές προσβάσεις εφαρμογής.

Μείωση του αριθμού προσβάσεων μνήμης

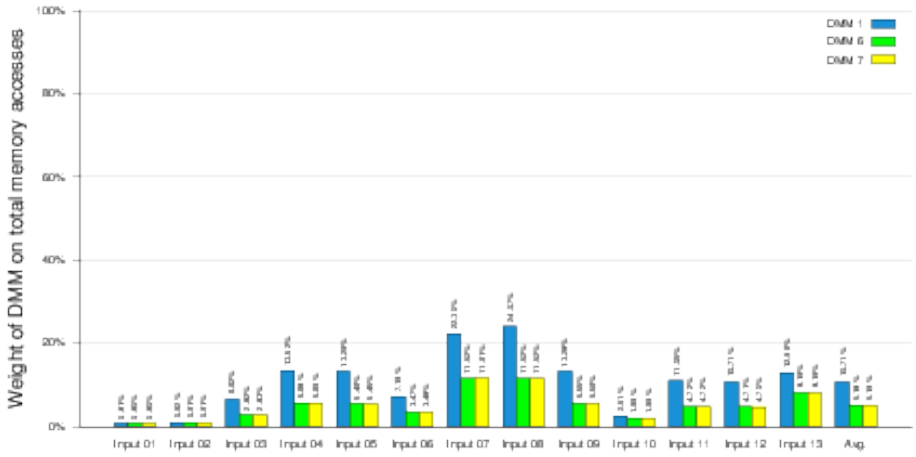
Έχοντας ως στόχο την ελαχιστοποίηση των προσβάσεων μνήμης, ο δυναμικός διαχειριστής μνήμης πρέπει να είναι σε θέση να προσκομίσει τα block για τα δημοφιλέστερα μεγέθη γρήγορα. Ομοίως, η διαδικασία για τα ελεύθερα block που γίνονται αχρησιμοποίητα, πρέπει να είναι γρήγορη. Αυτό προτείνει την ειδίκευση των καταλόγων ελεύθερων block για τα δημοφιλή μεγέθη. Εντούτοις, η χρησιμοποίηση ενός υψηλού αριθμού καταλόγων αυξάνει το χρόνο αναζήτησης, και επομένως ο αριθμός καταλόγων πρέπει να μειώσει ειδικά αυτά που λαμβάνουν τις περισσότερες προσβάσεις. Ο πίνακας 7.5 παρουσιάζει τη διαφορά στον αριθμό προσβάσεων μνήμης σχετικών άμεσα με τη διαχείριση της δυναμικής μνήμης στη βέλτιστη λύση για κάθε περίπτωση εισαγωγής.



Σχήμα 7.6: Αποτελέσματα της δυναμικής διαχείρισης μνήμης στο συνολικό αριθμό προσβάσεων μνήμης: συνολικός αριθμός προσβάσεων για κάθε εισαγωγή.

Το DMMs 6 και 7 είναι οι διαχειριστές μνήμης που υφίστανται τα χαμηλότερα γενικά κόστη, με το DMM6 να είναι καλύτερο για τις περισσότερες από τις περιπτώσεις εισαγωγής (11 από τα 13). Έναντι του DMM 1, ο δυναμικός διαχειριστής μνήμης που χρησιμοποιείται ως αναφορά για αυτήν την περιπτωσιολογική μελέτη, ενώ τα DMMs 6 και 7 λαμβάνουν μια βελτίωση μέχρι 62.35% και 62.33%, αντίστοιχα (για 5 εισόδους). Κατά μέσον όρο, και οι δύο διαχειριστές επιτρέπουν μια μείωση 45.67% και 45.62% των προσβάσεων μνήμης, λόγω της διαχείρισης της δυναμικής μνήμης, αντίστοιχα (σχήμα. 7.8).

Η μείωση του αριθμού προσβάσεων μνήμης λόγω DMM είναι σημαντική, αλλά ο τελικός στόχος είναι να μειωθεί ο συνολικός αριθμός προσβάσεων μνήμης της εφαρμογής. Ο πίνακας 7.6 παρουσιάζει τη διαφορά στο συνολικό αριθμό προσβάσεων μνήμης πότε η χρησιμοποίηση καθενός των διαχειριστών σύγκρινε με το βέλτιστο για κάθε περίπτωση εισαγωγής. Τα αποτελέσματα είναι συνεπή και, πάλι, τα DMMs 6 και 7 επιτρέπουν τη μεγαλύτερη μείωση των προσβάσεων μνήμης, με μια αμελητέα μέση απόκλιση στο βέλτιστο τρόπο για κάθε διαφορετική περίπτωση εισόδου. Με παρόμοιο τρόπο με την προηγούμενη ανάλυση, η Εικόνα 7.9 επεξηγεί τη βελτίωση που επιτυγχάνεται από αυτούς τους δύο δυναμικούς διαχειριστές μνήμης (μέχρι 14.18% για την είσοδο 08 και 5.98% κατά μέσον όρο). Για να γίνει κατανοητή η διαφορά μεταξύ των δύο λύσεων που είναι οι πιο κοντινές στο βέλτιστο για τις περισσότερες περιπτώσεις, το σχήμα 7.10 παρουσιάζει τη διαφορά μεταξύ τους για κάθε περίπτωση εισόδου. Η βελτίωση του DMM 6 ως προς το DMM 7 είναι πολύ μικρή. Αυτή η διαφορά μελετάται περαιτέρω παρακάτω.



Σχήμα 7.7: Αποτελέσματα της δυναμικής διαχείρισης μνήμης στο συνολικό αριθμό προσβάσεων μνήμης: ποσοστό πέρα από τις συνολικές προσβάσεις μνήμης που αντιστοιχούν στον διαχειριστή δυναμικής μνήμης.

Μείωση του αποτυπώματος μνήμης

Η χρησιμοποίηση της λιγότερης μνήμης είναι ευεργετική για μια εφαρμογή που τρέχει σε ένα ενσωματωμένο σύστημα από πολλές απόψεις. Παραδείγματος χάριν, το σύστημα μπορεί να χρησιμοποιήσει μια μικρότερη μνήμη, η οποία θα μεταφράσει σε έναν γρηγορότερο χρόνο πρόσβασης και μια πιο μικρή κατανάλωση ενέργειας. Ο διαχειριστής συστημάτων μπορεί ακόμη και να είναι σε θέση να ελαττώσει την κατανάλωση ενέργειας στις αχρησιμοποίητες ενότητες μνήμης για να μειώσει περαιτέρω τη συνολική κατανάλωση ενέργειας. Κατά συνέπεια, η μείωση του αποτυπώματος μνήμης της εφαρμογής είναι ένας σχετικός στόχος βελτιστοποίησης. Προκειμένου να μειωθεί το αποτύπωμα μνήμης της εφαρμογής, ο δυναμικός διαχειριστής μνήμης πρέπει να εξασφαλίσει ότι το ποσό μνήμης που σπαταλιέται πρέπει, λόγω του εσωτερικού ή/και εξωτερικού τεμαχισμού, να περιορίζεται στο ελάχιστο. Επομένως, μια σημαντική βελτιστοποίηση είναι να σιγουρευτεί ότι τα block με τα σωστά μεγέθη χρησιμοποιούνται για κάθε δέσμευση. Το DMM 6 χτίζει τους καταλόγους ελεύθερων block για τα δημοφιλέστερα μεγέθη (40, 1460 και 1500 bytes), για να μειώσει τον αριθμό προσβάσεων που απαιτούνται για να βρουν το σωστό block. Το DMM 7 προσθέτει μερικά λιγότερο συχνά ζητούμενα μεγέθη (92 και 132 bytes), καθώς επίσης και ενδιάμεσα (256, 512 και 1024 bytes), για να αποφύγει μια υψηλή ποινική ρήτρα για τις κατανομές των ενδιάμεσων μεγεθών. Εκτελέσαμε διάφορα πειράματα για να επικυρώσουμε τις προηγούμενες υποθέσεις βασισμένες στα αποσπασματικά μεταδεδομένα λογισμικού. Ο Πίνακας 7.7 και το σχήμα 7.11 παρουσιάζει το ποσό μνήμης που απαιτείται από την εφαρμογή για κάθε σύνολο δεδομένων

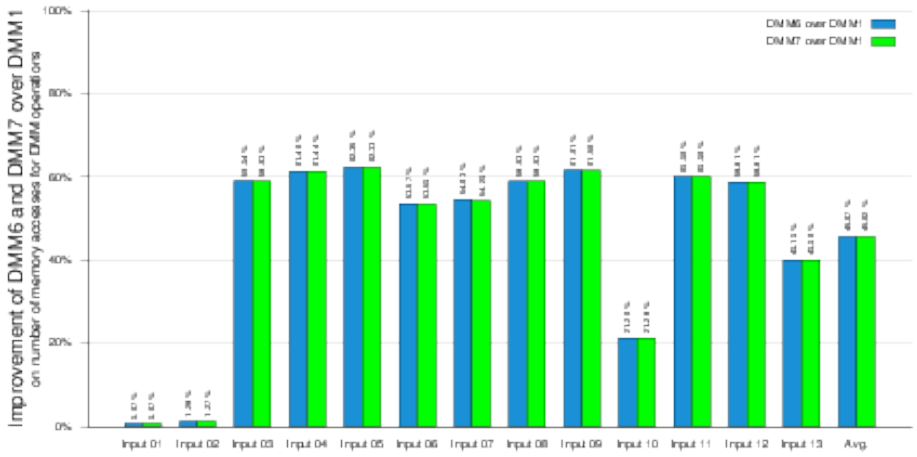
εισόδου, χρησιμοποιώντας κάθε έναν από τους διαφορετικούς διαχειριστές. Για την εφαρμογή του DDTs, χρησιμοποιήσαμε τον προηγουμένως επιλεγμένο συνδυασμό: A3-B3. Ο πίνακας 7.8 παρουσιάζει την απόκλιση των διαφορετικών εφαρμογών όσον αφορά το βέλτιστο, για κάθε περίπτωση εισαγωγής. Εξετάζοντας το αποτύπωμα μνήμης, το DMM 7 είναι ο καλύτερος διαχειριστής μνήμης για τις περισσότερες από τις περιπτώσεις εισαγωγής. Εντούτοις, η διαφορά σε μια υποθετική τέλεια λύση για όλες τις περιπτώσεις είναι τώρα μεγαλύτερη, με μια μέση απόκλιση 10.02%, αλλά ένα μέγιστο λάθος 81.9% (για 10 εισόδους). Αντίθετα, η μέση διαφορά του DMM 6 ως προς το βέλτιστο είναι τώρα αρκετά υψηλότερη. Η υψηλότερη παραλλαγή των αποτελεσμάτων που επιτυγχάνονται για το αποτύπωμα μνήμης, έναντι των αποτελεσμάτων που επιτυγχάνονται για τον αριθμό προσβάσεων μνήμης, οφείλεται στο γεγονός ότι το αποτύπωμα μνήμης είναι πιο ευαίσθητο στις μικρές παραλλαγές στον αριθμό block κάθε μεγέθους που διατίθενται, ακόμα κι αν ο αριθμός προσβάσεων για να βρει τα block παραμένει ο ίδιος. Το σχήμα 7.12 παρουσιάζει τη διαφορά στο αποτύπωμα μνήμης κάθε δυναμικού διαχειριστή μνήμης στο βέλτιστο, που υπολογίζεται κατά μέσο όρο για όλες τις περιπτώσεις εισαγωγής. Το σχήμα 7.13 παρουσιάζει το συνολικό αποτύπωμα μνήμης (στις bytes) που απαιτείται από τους διαφορετικούς δυναμικούς διευθυντές μνήμης για κάθε περίπτωση εισόδου.

Πίνακας 7.5: Αποτελέσματα ως προς τους 9 διαχειριστές

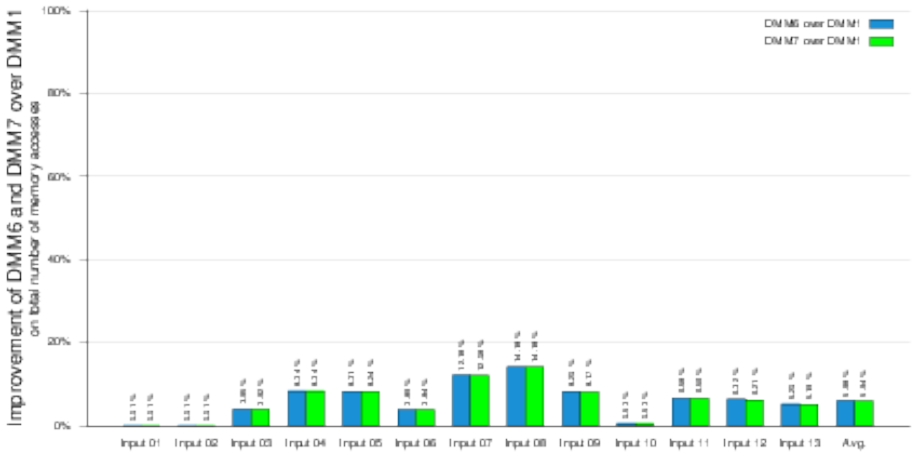
Input	DMM1(%)	DMM2(%)	DMM3(%)	DMM4(%)	DMM5(%)	DMM6(%)	DMM7(%)	DMM8(%)	DMM9(%)
01	0.67	0.01	0.01	5024.59	4881.74	0	0	4986.61	4454.37
02	1.30	0.18	0.19	2240.55	2220.80	0	0.01	2268.15	2239.02
03	144.17	26.92	25.39	720.54	735.17	0	0.29	547.52	572.22
04	159.45	32.33	33.54	744.68	730.43	0	0.04	523.58	514.36
05	165.61	43.62	43.37	844.33	862.57	0	0.05	650.53	698.31
06	115.39	27.89	28.19	865.83	868.86	0	0.16	598.72	629.96
07	120.39	35.03	35.96	551.71	552.20	0	0.94	590.30	614.06
08	143.51	42.13	42.13	572.62	572.98	0	0	566.18	567.98
09	160.48	42.85	42.97	820.09	820.18	0	0.12	651.85	575.89
10	27.02	0.07	0.06	1620.66	1481.76	0.03	0	1420.49	1427.60
11	150.39	43.99	43.97	793.95	938.43	0.02	0	1080.67	931.31
12	142.78	37.94	38.03	834.40	835.10	0	0	806.23	962.48
13	66.95	19.41	19.48	1916.14	571.26	0	0.08	2863.90	1968.18
Μέση Τιμή	107.55	27.10	27.17	1350.01	1236.27	0	0.13	1350.39	1242.75

Πίνακας 7.6: Διαφορά στο συνολικό αριθμό προσβάσεων μνήμης από διαφορετικούς δυναμικούς διαχειριστές

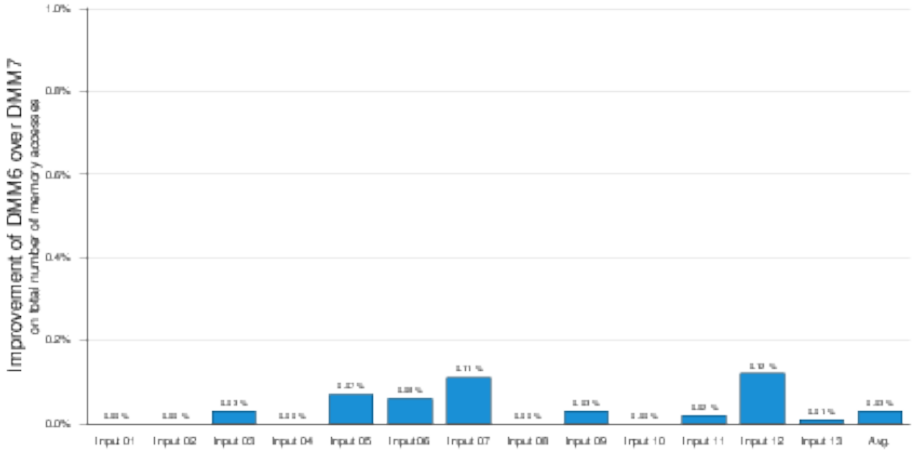
Input	DMM1(%)	DMM2(%)	DMM3(%)	DMM4(%)	DMM5(%)	DMM6(%)	DMM7(%)	DMM8(%)	DMM9(%)
01	0.01	0	0	40,41	39,26	0	0	40,10	35,82
02	0,01	0	0	20,38	20,20	0	0	20,63	20,36
03	4,11	0,76	0,73	20,38	20,74	0	0,03	15,46	16,18
04	9,10	1,84	1,95	42,34	41,53	0	0	29,76	29,24
05	9,06	2,43	2,41	46,09	47,11	0	0,07	35,56	38,13
06	4,05	1,01	1,02	30,07	30,19	0	0,06	20,81	21,88
07	13,87	4,03	4,14	63,55	63,61	0	0,11	68,00	70,74
08	16,53	4,85	4,85	65,94	65,99	0	0	65,20	65,41
09	8,93	2,40	2,42	45,54	43,53	0	0.,03	36,19	31,98
10	0,54	0	0	32,22	29,46	0	0	28,25	28,39
11	7,04	2,04	2,08	37,47	44,36	0	0,02	50,99	44,03
12	6,75	1,91	1,88	39,39	39,42	0	0,12	38,01	45,30
13	5,48	1,59	1,60	156,88	46,77	0	0.01	234,48	161,15
Average	6,57	1,76	1,78	49,28	41,09	0	0.03	52,57	46,82



Σχήμα 7.8: Η βελτίωση του αριθμού προσβάσεων μνήμης λόγω DMM κατά την χρησιμοποίηση του DMM 6 ή 7, αντί του DMM 1.



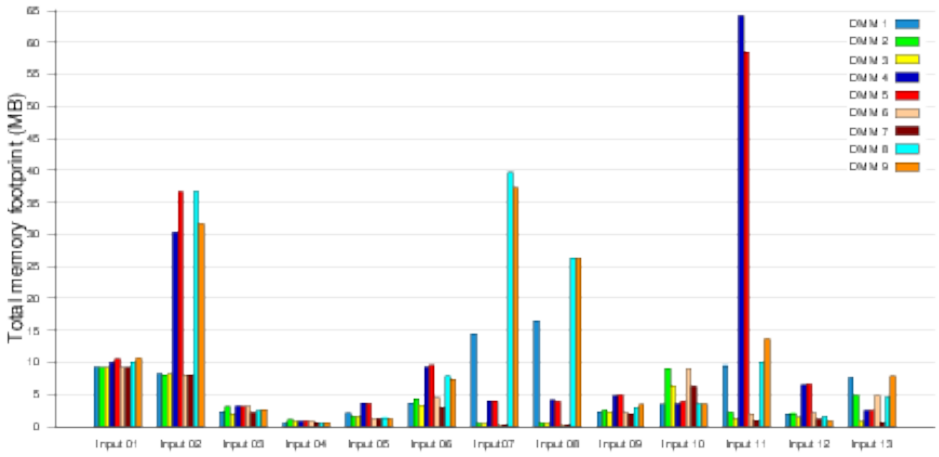
Σχήμα 7.9: Βελτίωση στο συνολικό αριθμό προσβάσεων μνήμης λόγω των βελτιστοποιήσεων στο διαχειριστή δυναμικής μνήμης: βελτίωση DMM 6 και 7 πέρα από το διαχειριστή δυναμικής μνήμης αναφοράς (DMM 1).



Σχήμα 7.10: Βελτίωση στο συνολικό αριθμό προσβάσεων μνήμης λόγω των βελτιστοποιήσεων στο διαχειριστή δυναμικής μνήμης: οριακή βελτίωση DMM 6 άνω των DMM 7 για το συνολικό αριθμό προσβάσεων μνήμης

Πίνακας 7.7: Συγκρίσεις των διαχειριστών ως προς διαφορετικές εισόδους

Input set	DMM1	DMM2	DMM3	DMM4	DMM5	DMM6	DMM7	DMM8	DMM9
Input 01	9292957	9158053	9179392	9990414	10564220	9158542	9176738	10071532	10659022
Input 02	8261672	8114640	8146305	30344312	36688944	8069856	8100836	36775346	31719557
Input 03	2324829	3101201	1955788	3138464	3084152	3191153	2141262	2508024	2668710
Input 04	583720	1068413	793154	805870	848280	858652	638234	564347	618625
Input 05	2041645	1541750	1541532	3707705	3590099	1240996	1229171	1346610	1232973
Input 06	3657349	4387276	3302356	9287451	9497449	4450461	2957479	7873662	7400638
Input 07	14509008	491348	488828	4030953	3985615	263972	261308	39644311	37313804
Input 08	16552416	535652	510296	4077566	3964176	279673	249007	26375050	26260057
Input 09	2280466	2576408	2257745	4787656	4938782	2135969	1938658	3025090	3396929
Input 10	3441933	9070597	6375234	3487045	3944477	9064424	6260861	3511306	3510688
Input 11	9476152	2228468	1261357	64222751	58511747	1946504	978527	10085511	13690317
Input 12	1959322	2038829	1417485	6459893	6600368	2093518	1122728	1581100	897078
Input 13	7667048	4888330	756467	2465793	2481203	4834146	600508	46420012	7808988

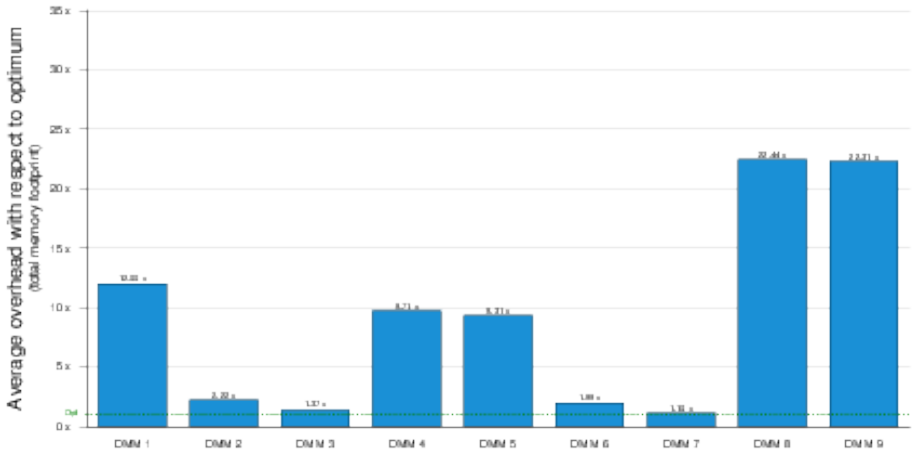


Σχήμα 7.11: Συνολικό αποτύπωμα μνήμης που απαιτείται από κάθε έναν από τους διαχειριστές δυναμικής μνήμης για κάθε περίπτωση εισαγωγής

Η εφαρμογή απαιτεί ένα ορισμένο ποσό μνήμης, αλλά ο δυναμικός διαχειριστής μνήμης χρειάζεται κάποια πρόσθετη μνήμη για τις εσωτερικές δομές του. Επιπλέον, τα αποτελέσματα του εσωτερικού και εξωτερικού τεμαχισμού παράγουν μια σημαντική αύξηση στο πραγματικό ποσό μνήμης, που απαιτείται για να εξυπηρετήσει τις ανάγκες της εφαρμογής. Για τον καλύτερο κατάλληλο διαχειριστή μνήμης, το DMM 7, τα γενικά κόστη είναι μόνο 40.26% (έναντι 1562.44% του διαχειριστή αναφοράς, DMM 1 που είναι διαχειριστής μνήμης γενικού σκοπού). Εντούτοις, εάν ο διαχειριστής μνήμης χρησιμοποιούμενος δεν είναι μεγάλης ακρίβειας που συντονίζεται στη συμπεριφορά δέσμευσης της εφαρμογής, τα γενικά κόστη μπορούν να αυξηθούν μέχρι 2996.77% (κατά μέσον όρο, κατά τη χρησιμοποίηση του DMM 8).

Πίνακας 7.8: Η διαφορά ως προς το βέλτιστο αποτύπωμα μνήμης για την κάθε είσοδο

Input	DMM1(%)	DMM2(%)	DMM3(%)	DMM4(%)	DMM5(%)	DMM6(%)	DMM7(%)	DMM8(%)	DMM9(%)
01	1.5	0	0.2	9.1	15.3	0.0	0.2	10.0	16.4
02	2.4	0.6	0.9	276.0	354.6	0.0	0.4	355.7	293.1
03	18.9	58.6	0.0	60.5	57.7	63.2	9.5	28.2	36.4
04	3.4	89.3	40.5	42.8	50.3	52.1	13.1	0.0	9.6
05	66.1	25.4	25.4	201.6	192.1	1.0	0.0	9.5	0.3
06	23.7	48.3	11.7	214.0	221.1	50.5	0.0	166.2	150.2
07	5452.4	88.0	87.1	1442.6	1425.3	1.0	0.0	15071.5	14179.6
08	6547.4	151.1	104.9	1537.5	1484.8	12.3	0.0	10492.1	10445.9
09	17.6	32.9	16.5	147.0	154.8	10.2	0.0	56.0	75.2
10	0	163.5	85.2	1.3	14.6	163.3	81.9	2.0	2.0
11	868.4	127.7	28.9	6463.2	5879.6	98.9	0.0	930.7	1299.1
12	118.4	127.3	58.0	620.1	635.8	133.4	25.1	76.2	0.0
13	1176.8	714.0	26.0	310.6	313.2	705.0	0.0	673.0	1200.4
Μέση τιμή	1099.8	122.4	37.3	871.3	830.7	99.3	10.0	2144.0	2131.4

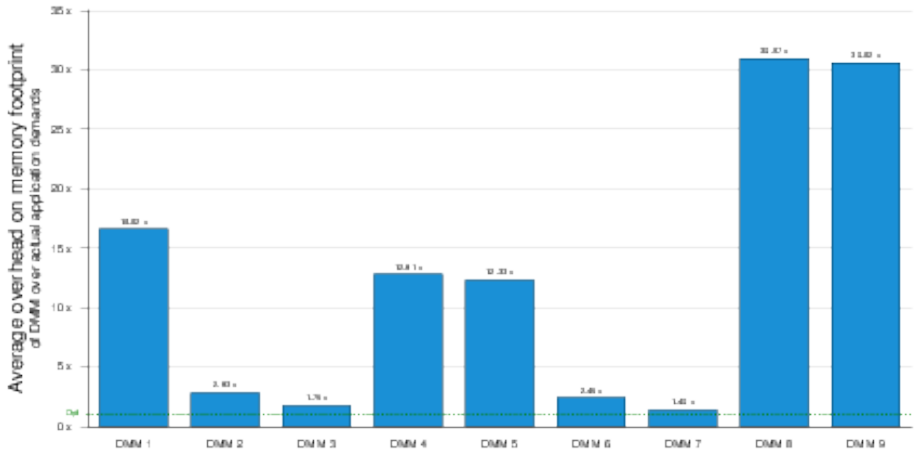


Σχήμα 7.12: Ανάλυση του αποτυπώματος μνήμης με κάθε δυναμικό διαχειριστή μνήμης: μέση διαφορά στο αποτύπωμα μνήμης κάθε διαχειριστή μνήμης στο βέλτιστο. Αν και το DMM 7 δεν είναι η καλύτερη λύση για όλες τις περιπτώσεις εισαγωγής, είναι η καλύτερη γενική λύση.

Με τα προηγούμενα αποτελέσματα, φαίνεται λογικό να υιοθετήσει κάποιος το DMM7 ως τελικό διαχειριστή μνήμης για την εφαρμογή. Εντούτοις, στο τμήμα 7.4.6, αναλύεται αυτός ανεξάρτητα από την εφαρμογή των βελτιστοποιήσεων στη μεταφορά των block των δυναμικών στοιχείων και εξετάζεται η επίδραση και των δύο δυναμικών διαχειριστών μνήμης.

7.4.6 Δυναμική βελτιστοποίηση μεταφοράς block μνήμης

Η παραδοσιακή εφαρμογή της μεταφοράς DMA είναι να μετακινηθούν τα στοιχεία στη μνήμη (ή μεταξύ της μνήμης και των εξωτερικών συσκευών), απελευθερώνοντας τους κύκλους των επεξεργαστών. Εντούτοις, στα ενσωματωμένα συστήματα είναι επίσης κοινή πρακτική να χρησιμοποιηθεί η μεταφορά DMA για να μειώσει τη μέση λανθάνουσα κατάσταση στα στοιχεία πρόσβασης από την κύρια μνήμη (πχ. που αντιγράφει τα στοιχεία στις πιο στενές μνήμες όπως «scratchpad» προτού να τους χρειαστεί πραγματικά η ΚΜΕ στο [76]). Το DRAM είναι συνήθως η τεχνολογία μνήμης που επιλέγεται για να εφαρμόσει την κύρια μνήμη των ενσωματωμένων συστημάτων. Αυτές οι συσκευές οργανώνονται εσωτερικά στις συστοιχίες και τις σελίδες, με τον περιορισμό ότι μόνο μια σελίδα μπορεί να είναι ενεργός σε κάθε συστοιχία οποιαδήποτε στιγμή. Η αλλαγή της ενεργού σελίδας σε μια συστοιχία DRAM έχει κύκλους τους μη-αμελητέους δαπανών ως προς την κατανάλωση ενέργειας. Αυτός ο τύπος οργάνωσης ευνοεί τα διαδοχικά σχέδια πρόσβασης. Εντούτοις, όταν χρησιμοποιεί το DMA την κύρια μνήμη παράλληλα με τον επεξεργαστή, έχει πρόσβαση και



Σχήμα 7.13: Ανάλυση του αποτυπώματος μνήμης με κάθε δυναμικό διαχειριστή μνήμης: Υπερυψωμένο Μέσο αποτυπώματος μνήμης κάθε δυναμικού διαχειριστή μνήμης.

από τους δύο παρεμβάλλει λευκές σελίδες με έναν ακαθόριστο τρόπο. Επομένως, δύο σχετικοί στόχοι βελτιστοποίησης για τις εφαρμογές που τρέχουν στα ενσωματωμένα συστήματα είναι ο αποδοτικός σχεδιασμός των μεταφορών στοιχείων για τα block των δυναμικών στοιχείων που χρησιμοποιούν την ενότητα DMA και η σωστή παρεμβολή λευκών σελίδων των προσβάσεων από τον επεξεργαστή και το DMA για να αποφύγουν τις περιττές ενεργοποιήσεις σειρών στις σελίδες των ενοτήτων DRAM. Σε αυτό το τμήμα, αξιολογούμε την εφαρμογή των τεχνικών βελτιστοποίησης βασισμένων στις πληροφορίες που παρέχονται από τα μεταδεδομένα λογισμικού για τη μεταφορά των δυναμικών block στοιχείων. Τα μεταδεδομένα λογισμικού περιέχουν τις πληροφορίες για τον αριθμό μεταφορών block που περιλαμβάνουν τις περιπτώσεις δυναμικών τύπων στοιχείων, μήκους, κατεύθυνσης (κύρια μνήμη) και του νήματος που τις άρχισαν (κυρίως, η οντότητα μεταφορών block από το σχήμα. 7.3). Αυτές οι πληροφορίες διευκολύνουν τη βελτίωση της χρησιμοποίησης της ενότητας DMA για την εφαρμογή οδηγών αυτού του παραδείγματος. Εάν η περίπτωση εισαγωγής παράγει τη μακροχρόνια σειρά διαδοχικών προσβάσεων (δηλαδή τα μακριά πακέτα διαδικασιών συστημάτων κυρίως), είναι καλοί υποψήφιοι που εκτελούνται από το DMA. Αντίθετα, εάν το σύστημα πρέπει να επεξεργαστεί πολλά μικρά πακέτα, τα γενικά κόστη του προγραμματισμού του DMA μπορούν να είναι υψηλότερα από τον αριθμό κύκλων που θα απαιτούταν η μεταφορά αν ήταν εκτελεσμένος άμεσα από τον επεξεργαστή. Επιπλέον, ο σχεδιασμός των προσβάσεων στους δυναμικούς τύπους στοιχείων πρέπει να εξετάσει επίσης δύο περισσότερες περιστάσεις. Κατ' αρχάς, όταν εκτελείται μια μεταφορά block από

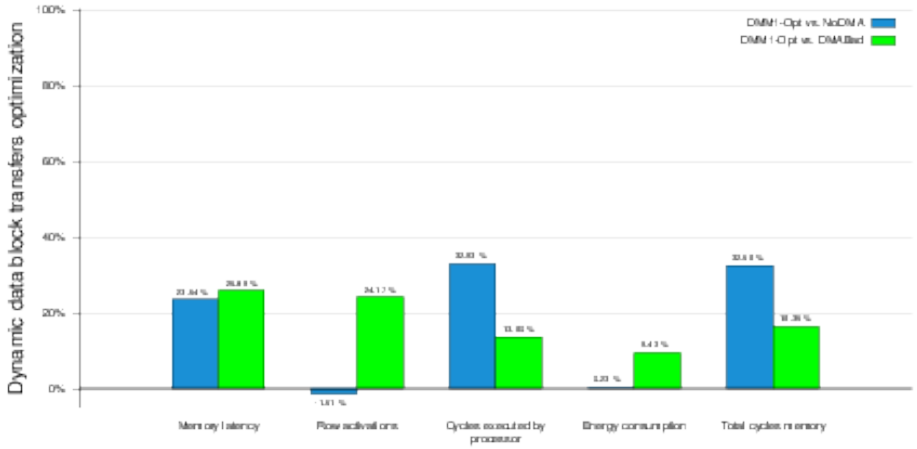
το DMA παράλληλα με τον επεξεργαστή, οι εξωτερικές ενότητες DRAM λαμβάνουν δύο ταυτόχρονα ρεύματα των προσβάσεων που μπορούν να αναγκάσουν τις πρόσθετες ενεργοποιήσεις σειρών. Δεύτερον, η ενότητα DMA μπορεί να ωφεληθεί από τη χαμηλότερη λανθάνουσα κατάσταση των τρόπων έκρηξης DRAM ως εκ τούτου, μπορεί να είναι ενδιαφέρον να βεβαιωθεί ότι ο επεξεργαστής δεν προαγοράζει την ενότητα DMA από το δίαυλο κατά τη διάρκεια των μεταφορών. Εντούτοις, μπορεί να είναι απαραίτητο να εγγυηθεί ότι ο επεξεργαστής μπορεί να έχει πρόσβαση στη μνήμη σε έναν οριακό αριθμό κύκλων, για παράδειγμα, για να προσκομίσει τον κώδικα διακόπτει τη ρουτίνα από την κύρια μνήμη.

Λαμβάνοντας αυτές τις εκτιμήσεις υπόψη, εξετάζουμε για αυτό το πείραμα τις 3 διαφορετικές πολιτικές σχεδιασμού. Ο πρώτος εκτελεί όλες τις προσβάσεις στη δυναμική μνήμη με τον επεξεργαστή. Ο δεύτερος χρησιμοποιεί την ενότητα DMA για τα block περισσότερων από 32 bytes (οκτώ λέξεις), αλλά ο μέγιστος αριθμός κύκλων ότι η μηχανή DMA μπορεί να κρατήσει το δίαυλο κατά τη διάρκεια των συναλλαγών έκρηξης περιορίζεται σε οκτώ λέξεις (επομένως, μόλις χορηγηθεί στο DMA η πρόσβαση στο δίαυλο, μπορεί να μεταφέρει χωρίς διακοπές τουλάχιστον τόσα bytes όσο η πιο σύντομη μεταφορά). Τέλος, η τρίτη διαμόρφωση υιοθετεί την ενότητα DMA για τις μεταφορές τουλάχιστον 32 bytes, αλλά εξασφαλίζει ότι το DMA μπορεί να έχει πρόσβαση μέχρι μια ολόκληρη σελίδα DRAM σε μια ενιαία συναλλαγή έκρηξης η αποδοτικότητα επιπλέον, αυτή η τελευταία πολιτική χρησιμοποιεί τις τεχνικές που παρουσιάζονται στο [72] και στο [59] για να αποφασίσουν για να χρησιμοποιήσει το DMA ή όχι εάν το σύστημα είναι μπορεί να αναγνωρίσει την παρούσα περίπτωση εισαγωγής. Αναφερόμαστε σε αυτές τις πολιτικές ως «κανένα DMA (No DMA)» «το DMA προκαλεί πρόβλημα (DMA Bad)» και «το DMA προαιρετικό (DMA opt)», αντίστοιχα. Τα αποτελέσματα αυτής της μελέτης αποκαλύπτουν ότι η χρησιμοποίηση της ενότητας DMA μπορεί να απομακρύνει ένα μη αμελητέο ποσό κύκλων των επεξεργαστών (μέσος όρος 43% κατά τη χρησιμοποίηση της βέλτιστης διαμόρφωσης DMA με DMM7), αλλά μόνο εάν το DMA χρησιμοποιείται κατάλληλα. Διαφορετικά, μπορεί να ασκήσει σημαντική αρνητική επίδραση στην κατανάλωση ενέργειας του υποσυστήματος μνήμης, της μέσης λανθάνουσας κατάστασης μνήμης και σπαταλημένους κύκλους αναμονής επεξεργαστών, για να έχει πρόσβαση στη μνήμη.

Ανάλυση των πραγματοποιημένων βελτιώσεων

Το σχήμα 7.14 παρουσιάζει την επίδραση ενός καλού σχεδιασμού: Με τη χρησιμοποίηση της σωστής διαμόρφωσης, μια βελτίωση μέχρι 33% επιτυγχάνεται στον αριθμό των κύκλων που ξοδεύει ένας επεξεργαστής στην πρόσβαση της μνήμης σε σύγκριση με απουσία χρησιμοποίησης DMA. Υπάρχει επίσης

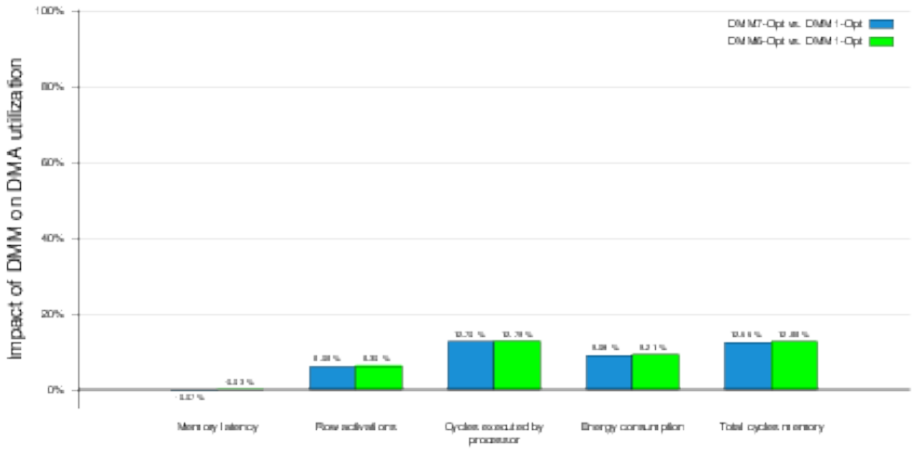
μια μικρή βελτίωση στην κατανάλωση ενέργειας. Επιπλέον, έναντι μιας κακής διαμόρφωσης DMA που δεν περιορίζει αρκετά τις παρεμβάσεις μεταξύ των δύο στοιχείων, ένα 24% της μέσης βελτίωσης είναι εφικτό για τον αριθμό ενεργοποιήσεων ακολουθίας προσβάσεων DRAM, 14% για τον αριθμό των κύκλων των επεξεργαστών που ξοδεύονται, πρόσβαση της μνήμης και 9% για την κατανάλωση ενέργειας.



Σχήμα 7.14: Βελτίωση που πραγματοποιήθηκε χρησιμοποιώντας μια μεταφορά DMA για τα δυναμικά στοιχεία. Τα περισσότερα εναπομείναντα block παρουσιάζουν τις βελτιώσεις που επιτυγχάνονται με τη σωστή διαμόρφωση DMA έναντι της μη χρησιμοποίησης των σωστών βελτιώσεων block του DMA σε σύγκριση με τη λανθασμένη χρησιμοποίησης που δεν περιορίζει την παρέμβαση του DMA με τον επεξεργαστή.

Το σχήμα 7.15 επεξηγεί την επίδραση που έχουν στην απόδοση του συστήματος οι βελτιστοποιήσεις που εκτελούνται προηγουμένως στον δυναμικό διαχειριστή μνήμης, κατά τη χρησιμοποίηση μιας ενότητας DMA. Εδώ, χρησιμοποιούμε την καλύτερη διαμόρφωση DMA με τους δυναμικούς διαχειριστές μνήμης που επιλέγονται στο προηγούμενο βήμα, τα DMM 6 και 7. Κατόπιν, συγκρίνουμε τα τελικά αποτελέσματα απόδοσής τους με τα αποκτηθέντα, χρησιμοποιώντας την ίδια διαμόρφωση DMA με το διαχειριστή αναφοράς, DMM 1. Τα αποτελέσματα αυτής της σύγκρισης είναι αποκαλυπτικά: Ο συνδυασμός του DMA με οποιαδήποτε από το βελτιστοποιημένο DMMs επιτυγχάνει τις βελτιώσεις, σε σύγκριση με το συνδυασμό του DMA και της αναφοράς DMM, περίπου 6% στο μέσο αριθμό ενεργοποιήσεων σειρών DRAM, 13% στον αριθμό κύκλων που ξοδεύονται από τον επεξεργαστή που έχει πρόσβαση στη μνήμη και 9% στην κατανάλωση ενέργειας του υποσυστήματος μνήμης.

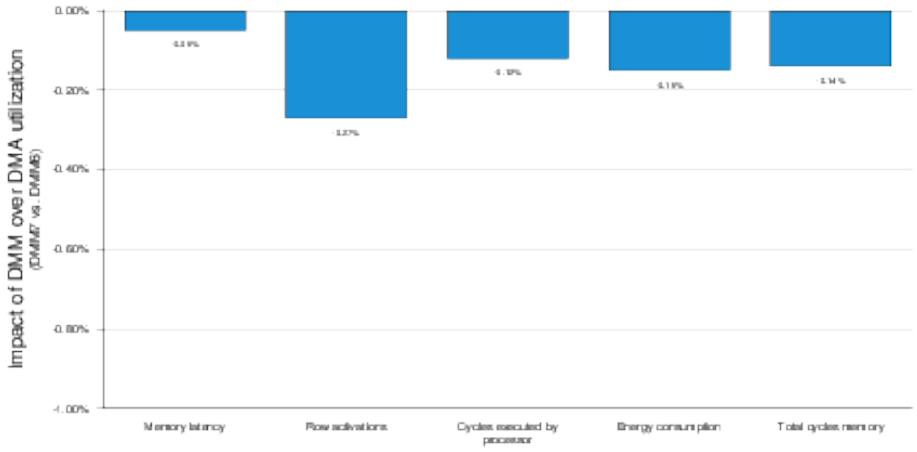
Η έκβαση του βήματος βελτιστοποίησης δυναμικής διαχείρισης μνήμης



Σχήμα 7.15: Η επιλογή ενός από τους βελτιστοποιημένους δυναμικούς διαχειριστές μνήμης που βελτιώνουν τη γενική απόδοση κατά την προσθήκη μιας ενότητας DMA.

ήταν ότι το DMM 7 είναι η καλύτερη μέση λύση όταν εξετάζεται το αποτύπωμα μνήμης, και το DMM6 κατά την εξέταση του αριθμού προσβάσεων μνήμης. Εντούτοις, η διαφορά μεταξύ των δύο κατά την εξέταση του αριθμού προσβάσεων μνήμης ήταν αμελητέα. Έτσι, κρατήσαμε και τους δύο δυναμικούς διαχειριστές μνήμης, ακριβώς χάριν της ανάλυσης του αντίκτυπού τους στην απόδοση της ενότητας DMA. Το σχήμα 7.16 δείχνει ότι και οι δύο διαχειριστές έχουν μια παρόμοια επίδραση στη συμπεριφορά ολόκληρου του υποσυστήματος μνήμης, με το μεγαλύτερο αντίκτυπο που είναι χαμηλότερο από 0.30% (για τον αριθμό ενεργοποιήσεων σειρών DRAM). Επομένως, μπορούμε τώρα να καταλήξουμε στο συμπέρασμα ακίνδυνα, ότι ο DMM7 είναι ο διαχειριστής μνήμης που πρέπει πάντα να χρησιμοποιείται σε αυτό το σύστημα. Η διαθεσιμότητα μιας κοινής αντιπροσώπευσης μεταδεδομένων λογισμικού απλοποιεί αυτόν τον τύπο ανάλυσης.

Στο σχήμα 7.17 παρουσιάζει γενικές βελτιώσεις που επιτυγχάνονται χρησιμοποιώντας το DMM7 με τη βέλτιστη διαμόρφωση DMA, έτσι ώστε τα αποτελέσματα των τεχνικών βελτιστοποίησης DMMR και DMA συνδυάζονται και αναλύονται από κοινού. Μέχρι 43% των κύκλων επεξεργαστών είναι τώρα ελεύθερα να χρησιμοποιηθούν για οποιοδήποτε σκοπό εκτός από την πρόσβαση των δυναμικών στοιχείων από την κύρια μνήμη, και είναι δυνατή μια μέση μείωση 9% στην κατανάλωση ενέργειας του υποσυστήματος μνήμης . Επιπλέον, έναντι μιας λανθασμένης διαμόρφωσης της ενότητας DMA, μια μείωση 29% στον αριθμό ενεργοποιήσεων σειρών DRAM και 18% στην ενέργεια που ξοδεύεται στις ενότητες μνήμης είναι δυνατή.

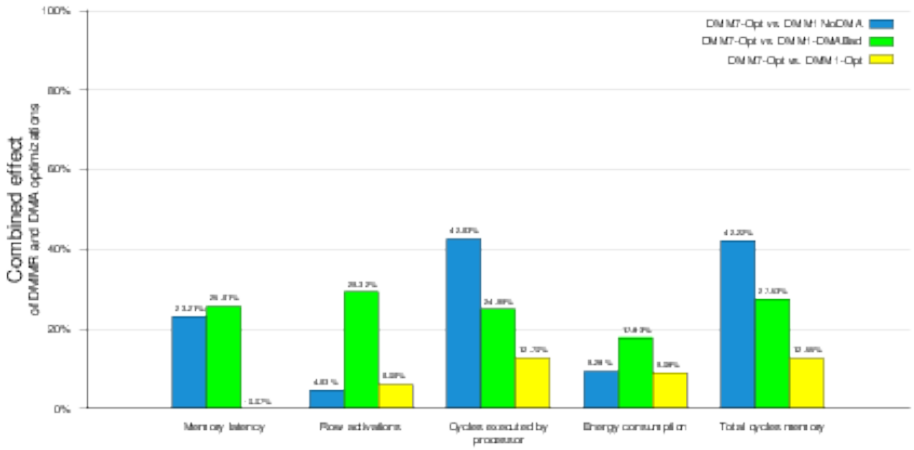


Σχήμα 7.16: Αντίκτυπος των διαφορετικών βελτιστοποιημένων διαχειριστών μνήμης από το προηγούμενο βήμα στη χρησιμοποίηση του DMA: Τα αποτελέσματα των DMM 6 και 7 είναι σχεδόν ίδια.

7.5 Σύγκριση με σχετικές εργασίες

Ο κύριος διαφοροποιητής σε αυτό το κεφάλαιο όσον αφορά τη σχετική εργασία είναι ότι η προσέγγισή μας στα μεταδεδομένα λογισμικού δεν στοχεύει να καθορίσει την εφαρμοσμένη μηχανική των εφαρμογών λογισμικού, ούτε για να αναλύσει ή να χαρακτηρίσει τη δομή τους. Αντ' αυτού, προσπαθούμε να αντιπροσωπεύσουμε τα χαρακτηριστικά της συμπεριφοράς τους, όταν υπόκεινται στις συγκεκριμένες εισόδους, με έναν επαναχρησιμοποιήσιμο τρόπο. Επιπλέον, περιορίζουμε συγκεκριμένα αυτήν την εργασία στο πεδίο των εφαρμογών που εξουσιάζονται από τους δυναμικά διατιθέμενους τύπους στοιχείων που τρέχουν στα ενσωματωμένα συστήματα. Τα παραχθέντα μεταδεδομένα λογισμικού μπορούν να χρησιμοποιηθούν με τα μεταδεδομένα υλικού της πλατφόρμας για να προσαρμόσουν τη διαχείριση των πόρων και να εφαρμόσουν τις διαφορετικές τεχνικές βελτιστοποίησης. Όσον αφορά το θέμα της σκιαγράφησης, αυτό το κεφάλαιο παρουσιάζει μια μέθοδο για να σχολιάσει την εφαρμογή DDTs με τα πρότυπα έτσι ώστε όλες οι προσβάσεις στις μεταβλητές τους καταγράφονται.

Αυτή η μέθοδος απαιτεί μόνο μια μεταγλώττιση της εφαρμογής· μπορεί έπειτα να οργανωθεί με τις διαφορετικές εισόδους για να πάρει το διαφορετικό σχεδιασμό περιγράμματος στις εκτελέσεις. Το κύριο πλεονέκτημα της εξηγημένης μεθόδου υποσημειώσεων είναι ότι ο μεταγλωττιστής διαδίδει την ενοργάνωση προσβάσεων αυτόματα, που εγγυάται ότι όλες αυτές καταγράφονται. Αυτή η μέθοδος υποστηρίζει πολυνημάτωση εντούτοις, όπως συμβαίνει με την αναγραφή οποιουδήποτε πολύπλοκου κώδικα, είναι δυνατό ότι λόγω του θεω-



Σχήμα 7.17: Τελική συνδυασμένη επίδραση των βελτιστοποιήσεων DMM και DMA.

ρήματος του «Heisenburg» η παρατήρηση αλλάζει ελαφρά το αποτύπωμα εκτέλεσης της εφαρμογής. Πιστεύουμε ότι αυτό δεν είναι ένα πρόβλημα για το καλά συμπεριφερόμενο λογισμικό πολυμέσων που εφαρμόζει τα κατάλληλα κλειδώματα συγχρονισμού και ασφάλειας των νημάτων. Συγκρίνοντας, με ένα εργαλείο όπως το Valgrind, που τρέχει τον κώδικα μπορεί να διατηρήσει τον κώδικα πανομοιότυπο χωρίς σκιαγράφηση. Εντούτοις, θα ήταν επίσης πιο δύσκολο να συνδεθούν οι πληροφορίες περιγράμματος με τις συγκεκριμένες γραμμές στον πηγαίο κώδικα.

7.6 Συμπεράσματα

Σε αυτό το κεφάλαιο έχουμε εισαγάγει την έννοια των μεταδεδομένων λογισμικού για να χαρακτηρίσουμε τη δυναμική συμπεριφορά δέσμευσης και πρόσβασης στοιχείων των εφαρμογών λογισμικού, στα ενσωματωμένα συστήματα. Έχουμε καθορίσει και την έννοια των μεταδεδομένων λογισμικού και τις σχετικές πληροφορίες για να εκτελέσουμε τις βελτιστοποιήσεις στο δυναμικό υποσύστημα μνήμης. Αυτές οι πληροφορίες μπορούν να μοιραστούν και να χρησιμοποιηθούν από τα διαφορετικά εργαλεία βελτιστοποίησης, ακόμη και από τους ανεξάρτητους παρόχους, επιτρέποντας τη βελτιστοποίηση της σύνθετης ενσωματωμένης κατανάλωσης ενέργειας συστημάτων, ως προς τις προσβάσεις μνήμης και το αποτύπωμα μνήμης. Στην πραγματικότητα, οποιοδήποτε εργαλείο μπορεί να ενημερώσει τις τιμές μεταδεδομένων για τις μετρήσεις που επηρεάζονται από τις βελτιστοποιήσεις του. Οι ενημερωμένες πληροφορίες μπορούν έπειτα να χρησιμοποιηθούν από τα επόμενα εργαλεία βελτιστοποίησης.

Η εφαρμογή αυτής της μεθοδολογίας θα μείωνε το χρόνο για το χαρακτηρισμό των εφαρμογών, επειδή οι πληροφορίες που παράγονται κατά τη διάρκεια μιας ενιαίας φάσης σκιαγράφησης και ανάλυσης θα ήταν διαθέσιμες σε όλες τις επόμενες τεχνικές βελτιστοποίησης.

Η μέθοδος για να εξαγάγει τα μεταδεδομένα των ενσωματωμένων εφαρμογών λογισμικού που έχουμε παρουσιάσει, διαιρείται σε δύο βήματα: σκιαγράφηση των εφαρμογών, η οποία παράγει τις ακατέργαστες πληροφορίες, και την ανάλυση αυτών των πληροφοριών, για να λάβει τις τελικές τιμές μεταδεδομένων.

Για το σχεδιασμό περιγράμματος βήμα, έχουμε προτείνει τη χρήση μιας εξειδικευμένης βιβλιοθήκης προτύπων. Για το βήμα ανάλυσης, έχουμε παρουσιάσει ένα σύνολο αλγορίθμων που μπορούν να χρησιμοποιηθούν για να συμπεράνει τα διαφορετικά χαρακτηριστικά των εφαρμογών.

Τέλος, έχουμε παρουσιάσει μια περιπτωσιολογική μελέτη στην οποία τα μεταδεδομένα λογισμικού επιτρέπουν να προσδιορίσουν τα πιο σχετικά χαρακτηριστικά μιας εφαρμογής και επιτρέπουν στη συνεπή χρησιμοποίηση των πολλαπλάσιων βελτιστοποιήσεων για να επιτύχουν τις βελτιώσεις στην κατανάλωση ενέργειας, τις προσβάσεις μνήμης και το αποτύπωμα μνήμης.

Όσον αφορά τις μελλοντικές επεκτάσεις, η έννοια των μεταδεδομένων μπορεί να χρησιμοποιηθεί στο χρόνο εκτέλεσης, με τη διευκόλυνση της δυναμικής επιλογής του DDTs και του DMMs που χρησιμοποιούνται για να εκτελέσει τις διαφορετικές φάσεις (με τα διαφορετικά σχέδια πρόσβασης) μιας εφαρμογής σε ένα ορισμένο ενσωματωμένο σύστημα. Αυτό θα μπορούσε να επιτευχθεί μέσω της χρήσης των σεναρίων συστημάτων και μιας βιβλιοθήκης που βρίσκεται στους συνδυασμούς απλών στοιχείων για να εφαρμόσει τα σχέδια που παρήχθησαν από τις μεθοδολογίες DDTR και DMMR. Η έννοια των σεναρίων συστημάτων είναι βασισμένη στον προσδιορισμό των σχετικών (διαφορετικών) εισαγόμενων όρων στον σχέδιο-χρόνο και τον υπολογισμό των κατάλληλων λύσεων DDTR και DMMR για κάθε μια από αυτές τις καταστάσεις. Αργότερα, μια ενότητα ελέγχου προσδιορίζει στο χρόνο εκτέλεσης το τρέχον σενάριο, έτσι ώστε το λειτουργικό σύστημα να μπορεί να εφαρμόσει τις κατάλληλες λύσεις DDTR και DMMR, που συνδυάζουν τα συστατικά από τη βιβλιοθήκη.