Εθνικό Μετσόβιο Πολυτεχνείο
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων VLSI

Ανάπτυξη Μεθοδολογίας για Δυναμική Ανάθεση Μνήμης σε Πολυπύρηνα Επεξεργαστικά Συστήματα

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

Ιωάννη Κ. Κούτρα

Αθήνα, Οκτώβριος 2016

Εθνικό Μετσόβιο Πολυτεχνείο
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων VLSI

Ανάπτυξη Μεθοδολογίας για Δυναμική Ανάθεση Μνήμης σε Πολυπύρηνα Επεξεργαστικά Συστήματα

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

Ιωάννη Κ. Κούτρα

Συμβουλευτική Επιτροπή : Δημήτριος Σούντρης
Κιαμάλ Πεκμεστζή
Γεώργιος Γκούμας

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 12η Οκτωβρίου 2016.

.....	.....	.....
Δ. Σούντρης	Κ. Πεκμεστζή	Γ. Γκούμας
Αν. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.
.....	.....	.....
Ν. Κοζύρης	Κ. Κοντογιάννης	Α. Χατζηγεωργίου
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Πα.Μακ.
	.....	
	Ά. Μπίλας	
	Καθηγητής Π. Κρήτης	

Αθήνα, Οκτώβριος 2016

.....

Ιωάννης Κ. Κούτρας
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η διατριβή αυτή παρουσιάζει μία μεθοδολογία για τη δημιουργία δυναμικών αναθετών μνήμης (dynamic memory allocators) σε μία μεγάλη ποικιλία υπολογιστικών συστημάτων, από ενσωματωμένα συστήματα έως εξυπηρετητές υψηλού φόρτου. Πιστεύουμε ότι οι σημερινές υλοποιήσεις δυναμικών αναθετών μνήμης χαρακτηρίζονται από γενικεύσεις, με αποτέλεσμα να μη χρησιμοποιούν επαρκώς χαρακτηριστικά και λειτουργίες των μοντέρνων υπολογιστικών συστημάτων πάνω στα οποία εκτελούνται. Προτείνουμε ένα πλαίσιο λογισμικού, το οποίο αποκαλούμε *dmmlib*, μέσα από το οποίο προγραμματιστές εφαρμογών μπορούν να δημιουργήσουν την δική τους βιβλιοθήκη δυναμικής διαχείρισης μνήμης. Επιλέγοντας διαφορετικές οργανώσεις μνήμης, πολιτικές και μηχανισμούς, η παραγόμενη βιβλιοθήκη μπορεί να χρησιμοποιηθεί άμεσα από εφαρμογές χωρίς να είναι αναγκαία η επαναμεταγλώττιση των τελευταίων. Η dmmlib προσφέρει δυνατότητες αλλαγής των παραμέτρων του αναθέτη για την περαιτέρω βελτίωση των επιδόσεων των εφαρμογών, όπως επίσης και τη δυνατότητα εκτενούς παρακολούθησης της λειτουργίας της μνήμης.

Σε αυτήν την διατριβή, δείχνουμε ότι η dmmlib μπορεί να χρησιμοποιηθεί αποδοτικά τόσο σε ενσωματωμένα συστήματα με περιορισμένο οικοσύστημα λογισμικού όσο και σε εξυπηρετητές γενικού σκοπού και υψηλής υπολογιστικής απόδοσης, ενώ μπορεί να αποτελέσει πεδίο μελέτης και ανάπτυξης πολιτικών και μηχανισμών για δυναμική διαχείριση μνήμης. Ξεκινούμε αναλύοντας τις τρέχουσες προσεγγίσεις στη δυναμική διαχείριση μνήμης, συμπεριλαμβανομένων δημοφιλών αναθετών που χρησιμοποιούνται σε ρεαλιστικές εφαρμογές και λειτουργικά συστήματα, και δείχνουμε πλεονεκτήματα και πιθανές ελλείψεις των αναθετών αυτών σε σύγχρονα συστήματα και εφαρμογές. Στη συνέχεια, παρουσιάζουμε την dmmlib, και δείχνουμε τα κίνητρα και τις αντίστοιχες υλοποιήσεις των πολιτικών και των ευρύτερων μηχανισμών. Δείχνουμε επίσης πιθανούς τρόπους εκτέλεσης των εφαρμογών με την dmmlib και πώς αυτές μπορούν να τροποποιήσουν την dmmlib κατά την εκτέλεσή τους.

Παρουσιάζουμε επίσης χρήσεις της dmmlib πάνω σε τρεις διαφορετικές πλατφόρμες. Οι δύο πρώτες αφορούν ενσωματωμένα συστήματα, ενώ η τρίτη συστήματα γενικού σκοπού. Η πρώτη χρησιμοποιεί την P2012, έναν πολυπύρηνο επιταχυντή για ενσωματωμένα συστήματα. Η δυναμική ανάθεση μνήμης στην πλατφόρμα αυτή χρήζει ιδιαίτερης μεταχείρισης, καθώς μία επεξεργαστική μονάδα αναλαμβάνει την διαχείριση των πόρων για μία ομάδα 16 επεξεργαστών. Μεταφέρουμε και αξιολογούμε την dmmlib στην πλατφόρμα αυτή. Τα αποτελέσματά μας δείχνουν ότι αναθέτες που παράγονται από την dmmlib μπορεί να είναι ακόμα και 2,6 φορές πιο γρήγοροι από τον επίσημο αναθέτη της πλατφόρμας, χωρίς να διακινδυνεύει επιπλέον κόστος στη χρήση

μνήμης. Στην δεύτερη πλατφόρμα δείχνουμε πώς η κλιμακωσιμότητα του υλικού επηρεάζει την απόδοση των δυναμικών αναθετών μνήμης. Για συστήματα με πολλαπλές επεξεργαστικές μονάδες, πειραματικά αποτελέσματα δείχνουν ότι δυναμικοί αναθέτες μνήμης χαμηλού επιπέδου αδυνατούν να αντεπεξέλθουν της πολυπλοκότητας και υπολείπονται αναθετών που παράγονται μέσα από την dmmlib. Τέλος, σε συστήματα γενικού σκοπού δείχνουμε πώς μπορούν να εκτιμηθούν τα μεγέθη μελλοντικών αιτημάτων μνήμης και πώς ο τρέχων αναθέτης μπορεί να χρησιμοποιήσει την πληροφορία αυτή, ώστε να τροποποιηθεί κατά την εκτέλεση της εφαρμογής. Η προσέγγισή μας εισάγει ελάχιστο επιπλέον κόστος, ενώ καταφέρνει να διατηρήσει τον κατακερματισμό της μνήμης στο ελάχιστο.

Λέξεις-Κλειδιά: διαχείριση μνήμης, ενσωματωμένα συστήματα, πολυπύρηνες αρχιτεκτονικές, επιτάχυνση υλισμικού, ανάθεση, κλιμακωσιμότητα, προσαρμοστικότητα

Abstract

This dissertation presents a methodology for creating dynamic memory allocators on computing systems, ranging from embedded systems to high-workload servers. Our thesis is that modern implementations of dynamic memory allocators have become very generic and do not use sufficiently features and functions of the modern computing systems where they are being executed. We propose a framework, that we call *dmmlib*, through which application developers can create their own library for dynamic memory management. By selecting different memory organizations, policies and mechanisms, the produced library can be used directly by the applications with recompiling them being optional. The dmmlib framework offers also the possibility to reconfigure the allocator at the runtime in order to improve further the application performance, as well as the ability to profile extensively the memory usage.

In this dissertation, we show that dmmlib can be used efficiently in both embedded systems with limited software ecosystems and general-purposed, high-performance systems, while it could act as a study and development field of policies and mechanisms for dynamic memory management. We begin by evaluating existing approaches in dynamic memory management, including popular allocators that are used in real-world applications and operating systems, and we show the advantages and possible shortcomings in modern applications. Next, we present dmmlib, and we motivate the implementations of policies and the broader mechanisms. We also derive of a possible way of executing and profiling applications against dmmlib and how the firsts can re-configure the latter at their runtime.

We present also two examples of dmmlib usage on top of three different platforms. The first two refer to embedded systems, while the last one to general-purposed systems. The very first uses P2012, a many-core accelerator for embedded systems. Dynamic memory management in this platform requires extra treatment, since a single processing unit is responsible for the resource management of a cluster with 16 processing elements. We port and evaluate dmmlib at this platform. Our results show that the allocators produced by dmmlib can be up to 2.6 times faster than the official allocator of the platform, without compromising additional memory space for the metadata. In the next embedded platform, we show how processor scalability affects the performance of dynamic memory allocators. Experimental results show that low-level memory managers may not cope with scalability if the processors exceed a certain number. Finally, our last target platform is more generic and our work aims on predicting the sizes of future requests and

how the current allocator can use such information in order to re-configure and improve himself at runtime. Our approach introduces minimal overhead, while it manages to keep memory fragmentation low.

Keywords: memory management, embedded systems, multi-core architectures, many-core architectures, hardware acceleration, allocation, scalability, adaptivity

Περιεχόμενα

Κατάλογος σχημάτων

Στους γονείς μου,
πρωτίστως στον πατέρα μου.

Ευχαριστίες

Συνεχίζω να πιστεύω, ίσως πια με μεγαλύτερη θέρμη, ότι το Διδακτορικό είναι ένα ταξίδι που το κάνει κανείς μόνος του, όσο μεγάλο και αν είναι το «team» που τον πλαισιώνει. Ωστόσο, στην διαδρομή αυτή συνάντησα ανθρώπους, οι οποίοι σίγουρα με βοήθησαν με τον δικό τους, άμεσο ή έμμεσο, τρόπο και τους οποίους αξίζει να τους αναφέρω.

Για αρχή θα ήθελα να ευχαριστήσω όλα τα μέλη της τριμελούς επιτροπής μου, προπάντων για την εμπιστοσύνη που έδειξαν στο πρόσωπό μου, αλλά και για την συνεργασία τους. Θα ήθελα να ευχαριστήσω ειδικά τον κύριο Σούντρη για τις πολλαπλές ευκαιρίες διάκρισης, αλλά και εξέλιξης που μου έδωσε. Οι πολυπληθείς και εκτενείς συζητήσεις που είχαμε όλο αυτό το διάστημα για τεχνικά, διαχειριστικά, αλλά και προσωπικά θέματα ήταν τουλάχιστον ενδιαφέρουσες και διαμόρφωσαν καταλυτικά ένα μεγάλο μέρος του τρόπου σκέψης και ιδιοσυγκρασίας μου.

Ιδιαίτερη μνεία αξίζει ο κύριος Οικονομάκος. Οι τεχνικές του γνώσεις και η βοήθειά του τα πρώτα μου διδακτορικά βήματα με έκαναν να αισθάνομαι πιο ασφαλής ερευνητικά. Αν και δεν ολοκλήρωσα την διατριβή μου στο κομμάτι του High Level Synthesis, απέκτησα εμπειρία και βάσεις που αξιοποίησα και συνεχίζω να αξιοποιώ τεχνικά.

Οι άνθρωποι που γνώρισα στα ταξίδια μου και στις επισκέψεις μου στο εξωτερικό επίσης καθόρισαν μεγάλο μέρος της εξέλιξης αυτού του διδακτορικού. Ξεκινάω χρονικά από τον πρώτο σταθμό μου, το Eindhoven: οι κκ. Maurice Kastelijn (Vector Fabrics/IMEC) και Andrei Terechko (Vector Fabrics/Eindhoven University of Technology) μού έμαθαν πώς μπορεί ένας μηχανικός να διεξάγει έρευνα σε εταιρικό επίπεδο χωρίς να χάνει deadlines. Συνεχίζω στην Grenoble, όπου συνεργάτες όπως ο Germain Haugou (STMicroelectronics/ETH Zurich) και ο Thierry Lepley (STMicroelectronics/nVidia) μού έδειξαν πώς ένας μηχανικός μπορεί να εργαστεί σε μια μεγάλη εταιρεία και φυσικά μου επέβαλαν να ξεσκονίσω τα Γαλλικά μου. Τέλος, στο Μιλάνο θα ήθελα να ευχαριστήσω τον William Fornaciari (POLIMI) και την Cristina Silvano (POLIMI) για την φιλοξενία τους, καθώς και όλους όσους γνώρισα από τις ομάδες τους. Οι Giovanni Agosta (POLIMI), Patrick Bellasi (POLIMI/ARM) και Giussepe Massari (POLIMI) είναι λίγοι μόνο με τους οποίους συνεργάστηκα άμεσα, αλλά και πέρασα πολύ ευχάριστα τον ελεύθερο χρόνο μου.

Λίγοι δεν είναι και οι συνάδελφοι στο Εργαστήριο Μικροϋπολογιστών που μου στάθηκαν όλα αυτά τα χρόνια. Ξεκινάμε από τους «seniors», τον Αντώνη Παπανικολάου (ΕΜΠ/Hypertech) και

τον Αλέξανδρο Μπάρτζα (ΕΜΠ/EXUS) που με καθοδήγησαν στα πρώτα χρόνια του Διδακτορικού μου. Σημαντικός συνοδοιπόρος υπήρξε και ο Ηρακλής Αναγνωστόπουλος (ΕΜΠ/SIU), με τον οποίο κάναμε φοβερά σχέδια κατάκτησης του κόσμο με τον DMM. Αυτούς, αλλά και άλλους πολλούς μέσα από το εργαστήριο αισθάνομαι πραγματικά πολύ τυχερός που τους γνώρισα και που έχω την τιμή να τους θεωρώ φίλους.

Όμως υπάρχουν και άλλοι άνθρωποι, ευτυχώς πέρα από το επάγγελμα του ηλεκτρολόγου, τους οποίους και θέλω να ευχαριστήσω κλείνοντας. Πρόκειται για την οικογένειά μου, η οποία με περισσή αγάπη και υπομονή με υποστήριζαν όλα αυτά τα χρόνια. Όλα ξεκίνησαν με αυτόν τον Olivetti υπολογιστή που μας έφεραν στο σπίτι όταν ήμουν μικρός. Τέλος, θα ήθελα να ευχαριστήσω την Αλεξάνδρα, η οποία με βοήθησε στην δύσκολη τελική ευθεία. Το ταξίδι μόλις αρχίζει!

Η έρευνα αυτή χρηματοδοτήθηκε και στηρίχτηκε από αρκετές υποτροφίες του HiPEAC, καθώς και από το ερευνητικό πρόγραμμα FP7-ICT-248716 2PARMA της Ευρωπαϊκής Επιτροπής.

Ιωάννης Κούτρας

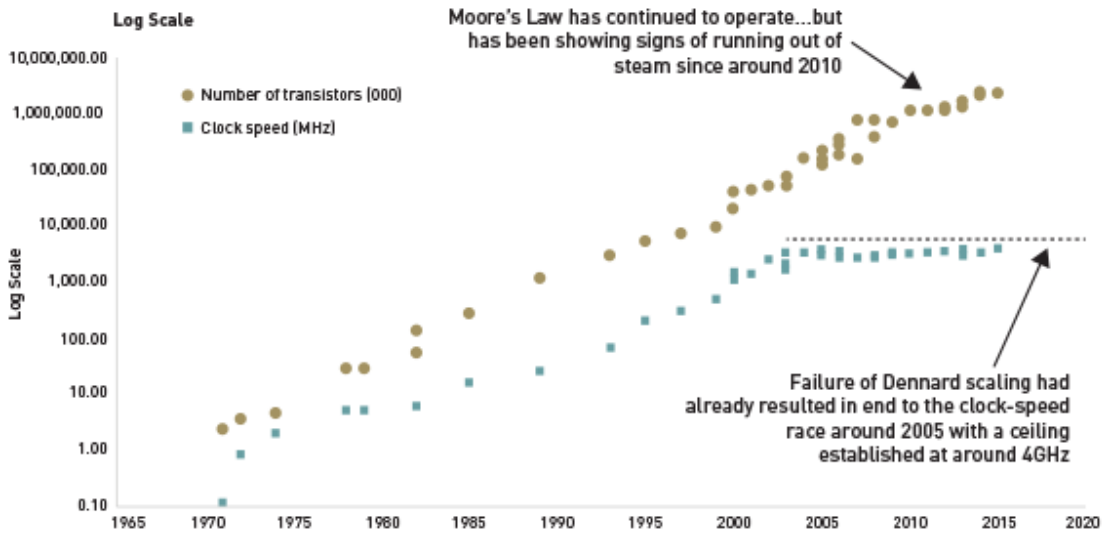
1 Εισαγωγή

Η διατριβή αυτή παρουσιάζει μία μεθοδολογία για να κατασκευαστούν αποδοτικοί δυναμικοί αναθέτες μνήμης σε μία τεράστια ποικιλία υπολογιστικών συστημάτων, από ενσωματωμένα συστήματα έως εξυπηρετητές υψηλού φόρτου. Πιστεύουμε ότι οι σημερινές υλοποιήσεις δυναμικών αναθετών μνήμης χαρακτηρίζονται από γενικεύσεις, με αποτέλεσμα να μη χρησιμοποιούν επαρκώς χαρακτηριστικά και λειτουργίες των μοντέρνων υπολογιστικών συστημάτων πάνω στα οποία εκτελούνται. Προτείνουμε ένα πλαίσιο λογισμικού, το οποίο αποκαλούμε *dmmlib*, μέσα από το οποίο προγραμματιστές εφαρμογών μπορούν να δημιουργήσουν τη δική τους βιβλιοθήκη δυναμικής διαχείρισης μνήμης. Επιλέγοντας διαφορετικές οργανώσεις μνήμης, πολιτικές και μηχανισμούς, η παραγόμενη βιβλιοθήκη μπορεί να χρησιμοποιηθεί άμεσα από εφαρμογές χωρίς να είναι αναγκαία η επαναμεταγλώττισή τους. Η dmmlib προσφέρει δυνατότητες αλλαγής των παραμέτρων του αναθέτη για την περαιτέρω βελτίωση των επιδόσεων των εφαρμογών, όπως επίσης και τη δυνατότητα εκτενούς παρακολούθησης της λειτουργίας της μνήμης. Σε αυτήν την διατριβή, δείχνουμε ότι η dmmlib μπορεί να χρησιμοποιηθεί αποδοτικά τόσο σε ενσωματωμένα συστήματα με περιορισμένο οικοσύστημα λογισμικού όσο και σε εξυπηρετητές γενικού σκοπού και υψηλής υπολογιστικής απόδοσης, ενώ μπορεί να γίνει πεδίο μελέτης και ανάπτυξης πολιτικών και μηχανισμών για δυναμική διαχείριση μνήμης.

1.1 Εισαγωγή: Η μνήμη στα σύγχρονα υπολογιστικά συστήματα

Η εκρηκτική ανάπτυξη των υπολογιστικών συστημάτων τα τελευταία χρόνια έχει δημιουργήσει ένα ευρύ φάσμα νέων ανθρώπινων αναγκών που καλύπτονται από αυτά σε παγκόσμια κλίμακα. Η αγορά δεν κυριαρχείται πλέον από ογκώδεις, ακριβούς υπολογιστές γραφείου, αλλά οι άνθρωποι καλύπτονται από φορητές, "έξυπνες" συσκευές ικανές να κάνουν τα πάντα συμπεριλαμβανομένων τηλεπικοινωνίας μέσω μηνυμάτων, φωνής, ακόμα και βίντεο, πλοήγησης του Διαδικτύου, πραγματοποίησης συναλλαγών, επεξεργασίας κειμένου.

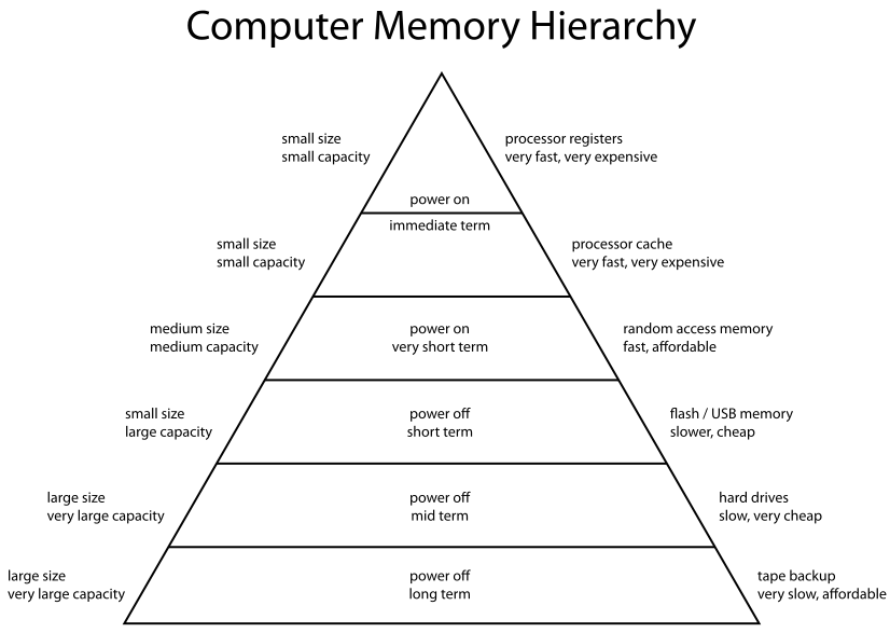
Για περισσότερο από 30 χρόνια (δεκαετίες 1970-2000) η αύξηση του αριθμού των τρανζίστορ (νόμος του Moore) και η αύξηση της συχνότητας λειτουργίας τους οδηγούσαν τις εξελίξεις και προσέφεραν τα θεμέλια για επόμενες τεχνολογικές ανακαλύψεις. Η πορεία αυτή, όμως, έχει εδώ



Σχήμα 1.1: Τάσεις στον αριθμό τρανζίστορ και στις μέγιστες συχνότητες λειτουργίας [1]

και καιρό σταματήσει, όπως μας δείχνει εξάλλου το Σχήμα 1.1. Η συχνότητα λειτουργίας στους επεξεργαστές γενικού σκοπού έχει «κολλήσει» στα 4 Gigahertz από το 2005, με τις εταιρείες να έχουν στρέψει την προσοχή τους σε άλλα θέματα, με σημαντικότερο ίσως την βελτίωση στην διαχείρισης της θερμότητας. Ενώ, λοιπόν, η συχνότητα λειτουργίας των chip παραμένει στα ίδια επίπεδα, παρατηρούμε την τάση αύξησης του αριθμού των τρανζίστορ να μην έχει επηρεαστεί στον ίδιο βαθμό με την συχνότητα, χάρη σε ένα σύνολο από καινούριες τεχνικές που τα σύγχρονα chip υιοθετούν.

Από την αυξημένη δυνατότητα υψηλής ολοκλήρωσης πάντως, ευνοείται δραματικά η τεχνολογία μνήμης υπολογιστών, ιδιαίτερα τα τελευταία χρόνια. Αυτό ενδεχομένως να κλονίσει μακροπρόθεσμα την παραδοσιακή ιεραρχία μνήμης στους υπολογιστές (Σχήμα 1.2). Εξαιτίας οικονομικών και τεχνολογικών λόγων, δεν είναι εύκολο να σχεδιαστεί μία μονή μνήμη που να είναι ταυτόχρονα μεγάλη σε μέγεθος, γρήγορη σε ταχύτητα και φθηνή σε κόστος. Με τη χρήση πολλών διαφορετικών μνημών, άλλοτε λανθανουσών και άλλοτε όχι, καταφέραμε στο παρελθόν, αλλά και συνεχίζουμε να χρησιμοποιούμε υπολογιστικά συστήματα που αντισταθμίζουν τις ιδιότητες αυτές που αναφέραμε για την μνήμη. Ωστόσο, βρισκόμαστε τώρα σε μία ιδιάζουσα τεχνολογική φάση, με δομές από την πυραμίδα ιεραρχίας να αλλάζουν. Οι δυναμικές μνήμες τυχαίας προσπέλασης (Dynamic Random Access Memory - DRAM) έχουν πλέον αποκτήσει μεγέθη συγκρίσιμα με μικρούς σκληρούς δίσκους, ενώ οι δίσκοι στερεάς κατάστασης (Solid State Disks - SSDs) έχουν κατακλύσει την αγορά, αντικαθιστώντας τους συμβατικούς, μηχανικούς σκληρούς δίσκους ακόμα και στο επίπεδο των απλών καταναλωτών - χρηστών. Σε συνδυασμό με τις νεώτε-

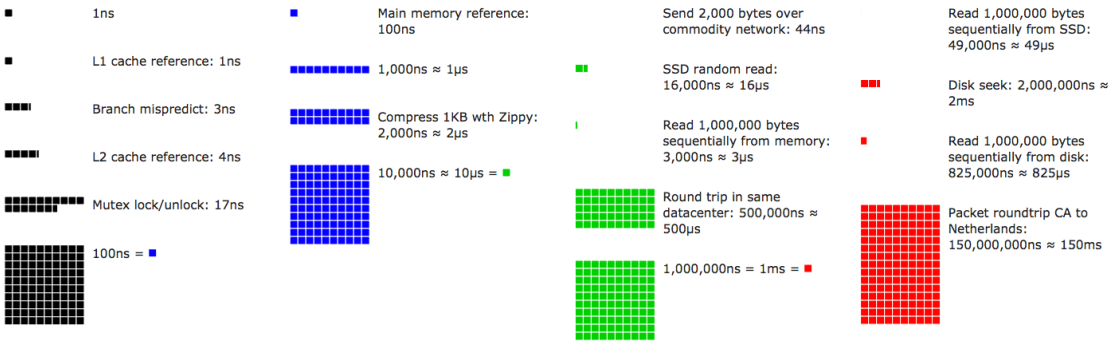


Σχήμα 1.2: Συμβατική ιεραρχία μνήμης υπολογιστών: Πόσο εφαρμόσιμη είναι στις καινούριες τεχνολογίες; Πηγή: [2]

ρες πολυπύρηνες αρχιτεκτονικές επεξεργαστών, οι τεχνολογίες αυτές μάς δείχνουν ότι ο τρόπος με τον οποίο χτίζονται οι υπολογιστές αλλάζει και ότι το μοντέλο προγραμματισμού τους θα πρέπει να προσαρμοστεί.

Σε κάθε περίπτωση, οι εξελίξεις αυτές δε θα πρέπει να οδηγήσουν τον αναγνώστη στην εντύπωση ότι το αποκαλούμενο τείχος μνήμης (memory wall) [4] έχει αντιμετωπιστεί. Οι τεχνολογίες κατασκευής τρανζίστορ μικροεπεξεργαστών και οι αντίστοιχες μνήμης παραμένουν ασύμβατες μεταξύ τους και αυτό δεν προβλέπεται να αλλάζει σύντομα. Στο Σχήμα 1.3 εμφανίζονται οι προβλεπόμενοι αριθμοί καθυστέρησης μεταφοράς το 2020 [3]. Αν δεν αλλάξει δραματικά η τεχνολογία μέσα σε μία πενταετία, τότε θα προκύψουν οι αριθμοί αυτοί, οι οποίοι μάλιστα δε διαφέρουν αρκετά από τα σημερινά νούμερα.

Με το τοπίο, επομένως, να αλλάζει, με πολυπύρηνους επεξεργαστές και εφαρμογές να προσπαθούν να προσπελάσουν μεγάλες ποσότητες μνήμης, η θέμα της διαχείρισης μνήμης αποκτάει νέες συνιστώσες. Αρκετά από τα προβλήματα που είχαν παρουσιαστεί τα πρώτα χρόνια των υπολογιστικών συστημάτων έχουν ήδη μελετηθεί και λυθεί ή ενδεχομένως δεν υφίσταται στα σύγχρονα συστήματα. Στο πεδίο όμως έρευνας έρχονται στο φως νέα θέματα τόσο σε ενσωματωμένα συστήματα, όσο σε μεγαλύτερα συστήματα γενικότερου σκοπού.



Σχήμα 1.3: Προβλεπόμενοι αριθμοί καθυστέρησης μεταφοράς το 2020 [3]

1.2 Περίληψη

Η διατριβή αυτή παρουσιάζει μία μεθοδολογία για τη δημιουργία δυναμικών αναθετών μνήμης (dynamic memory allocators) σε μία μεγάλη ποικιλία υπολογιστικών συστημάτων, από ενσωματωμένα συστήματα έως εξυπηρετητές υψηλού φόρτου. Πιστεύουμε ότι οι σημερινές υλοποιήσεις δυναμικών αναθετών μνήμης χαρακτηρίζονται από γενικεύσεις, με αποτέλεσμα να μη χρησιμοποιούν επαρκώς χαρακτηριστικά και λειτουργίες των μοντέρνων υπολογιστικών συστημάτων πάνω στα οποία εκτελούνται. Προτείνουμε ένα πλαίσιο λογισμικού, το οποίο αποκαλούμε *dmmlib*, μέσα από το οποίο προγραμματιστές εφαρμογών μπορούν να δημιουργήσουν τη δική τους βιβλιοθήκη δυναμικής διαχείρισης μνήμης. Επιλέγοντας διαφορετικές οργανώσεις μνήμης, πολιτικές και μηχανισμούς, η παραγόμενη βιβλιοθήκη μπορεί να χρησιμοποιηθεί άμεσα από εφαρμογές χωρίς να είναι αναγκαία η επαναμεταγλώττιση των τελευταίων. Η dmmlib προσφέρει δυνατότητες αλλαγής των παραμέτρων του αναθέτη για την περαιτέρω βελτίωση των επιδόσεων των εφαρμογών, όπως επίσης και τη δυνατότητα εκτενούς παρακολούθησης της λειτουργίας της μνήμης.

Σε αυτήν την διατριβή, δείχνουμε ότι η dmmlib μπορεί να χρησιμοποιηθεί αποδοτικά τόσο σε ενσωματωμένα συστήματα με περιορισμένο οικοσύστημα λογισμικού όσο και σε εξυπηρετητές γενικού σκοπού και υψηλής υπολογιστικής απόδοσης, ενώ μπορεί να αποτελέσει πεδίο μελέτης και ανάπτυξης πολιτικών και μηχανισμών για δυναμική διαχείριση μνήμης. Ξεκινούμε αναλύοντας τις τρέχουσες προσεγγίσεις στη δυναμική διαχείριση μνήμης, συμπεριλαμβανομένων δημοφιλών αναθετών που χρησιμοποιούνται σε ρεαλιστικές εφαρμογές και λειτουργικά συστήματα, και δείχνουμε πλεονεκτήματα και πιθανές ελλείψεις των αναθετών αυτών σε σύγχρονα συστήματα και εφαρμογές. Στη συνέχεια, παρουσιάζουμε την dmmlib, και δείχνουμε τα κίνητρα και τις αντίστοιχες υλοποιήσεις των πολιτικών και των ευρύτερων μηχανισμών. Δείχνουμε, επίσης,

πιθανούς τρόπους εκτέλεσης των εφαρμογών με την dmmlib και πώς αυτές μπορούν να τροποποιήσουν την dmmlib κατά την εκτέλεσή τους.

Παρουσιάζουμε επίσης δύο χρήσεις της dmmlib πάνω σε δύο διαφορετικές πλατφόρμες. Η πρώτη αφορά την P2012, έναν πολυπύρηνο επιταχυντή για ενσωματωμένα συστήματα. Η δυναμική ανάθεση μνήμης στην πλατφόρμα αυτή χρήζει ιδιαίτερης μεταχείρισης, καθώς μία επεξεργαστική μονάδα αναλαμβάνει τη διαχείριση των πόρων για μία ομάδα 16 επεξεργαστών. Μεταφέρουμε και αξιολογούμε την dmmlib στην πλατφόρμα αυτή. Τα αποτελέσματά μας δείχνουν ότι αναθέτες που παράγονται από την dmmlib μπορεί να είναι ακόμα και 2,6 φορές πιο γρήγοροι από τον επίσημο αναθέτη της πλατφόρμας, χωρίς να διακινδυνεύει επιπλέον κόστος στη χρήση μνήμης. Η δεύτερη πλατφόρμα είναι γενικότερη και αφορά συστήματα γενικού σκοπού. Συγκεκριμένα δείχνουμε πώς μπορούν να εκτιμηθούν τα μεγέθη μελλοντικών αιτημάτων μνήμης και πώς ο τρέχων αναθέτης μπορεί να χρησιμοποιήσει την πληροφορία αυτή, ώστε να τροποποιηθεί κατά την εκτέλεση της εφαρμογής. Η προσέγγισή μας εισάγει ελάχιστο επιπλέον κόστος, ενώ καταφέρνει να διατηρήσει τον κατακερματισμό της μνήμης στο ελάχιστο.

Υποστηρίζουμε ότι με την χρήση της dmmlib, είναι πιο εύκολο να κάνουμε τις εφαρμογές πάνω σε αρκετές πλατφόρμες πιο αποδοτικές σε θέματα δυναμικής διαχείρισης μνήμης. Με μία καλή διαίσθηση των αναγκών των εφαρμογών τους, οι προγραμματιστές μπορούν να επικεντρωθούν στην λογική των εφαρμογών τους, παρά στις λεπτομέρειες της διαχείρισης πόρων, ακόμα και αν αναπτύσσουν τον κώδικα σε γλώσσα χαμηλού επιπέδου. Ο σχεδιασμός της dmmlib επιτρέπει την εύκολη, αρθρωτή (modular) ανάπτυξη πολιτικών και μηχανισμών δυναμικής ανάθεσης μνήμης. Ο στόχος μας είναι να επιτύχουμε ένα πλαίσιο δυναμικής ανάθεσης μνήμης που να λειτουργεί σε πολλές αρχιτεκτονικές υπολογιστικών συστημάτων και να υποστηρίζει πλήθος διαφορετικών επιλογών, ώστε να γίνεται η αξιολόγηση των σχετικών αποφάσεων δυναμικής διαχείρισης μνήμης.

1.3 Δυναμική Διαχείριση Μνήμης

Η ανάθεση μνήμης που εξετάζουμε στην παρούσα εργασία αφορά την διαχείριση μνήμης υπολογιστικών συστημάτων σε επίπεδο εφαρμογών. Η βασική απαίτηση της διαχείρισης μνήμης είναι να αναθέτει δυναμικά τμήματα μνήμης σε εφαρμογές μόλις αυτές τα απαιτήσουν, και αντίστοιχα να αποδεσμεύει τα τμήματα αυτά όταν δεν χρειάζονται πλέον, προκειμένου να χρησιμοποιηθούν ξανά. Αυτό είναι ζωτικής σημασίας για οποιοδήποτε εξελιγμένο υπολογιστικό σύστημα, όπου πολλαπλές διεργασίες μπορούν να ξεκινήσουν οποιαδήποτε στιγμή.

Αρκετές μέθοδοι έχουν επινοηθεί που αυξάνουν την αποτελεσματικότητα της διαχείρισης μνήμης. Τόσο οι σύγχρονοι επεξεργαστές, όσο και τα σύγχρονα Λειτουργικά Συστήματα (ΛΣ) υπο-

στηρίζουν τη χρήση εικονικής μνήμης (virtual memory), ώστε οι διευθύνσεις μνήμης που χρησιμοποιούνται από μια διεργασία να ξεχωρίζονται από τις φυσικές διευθύνσεις. Καθώς αυτό εισάγει ένα επιπλέον επίπεδο αφαιρετικότητας στον προγραμματισμό του συστήματος, η ποιότητα του διαχειριστή εικονικής μνήμης μπορεί να έχει μια εκτεταμένη επίδραση στην συνολική απόδοση του συστήματος.

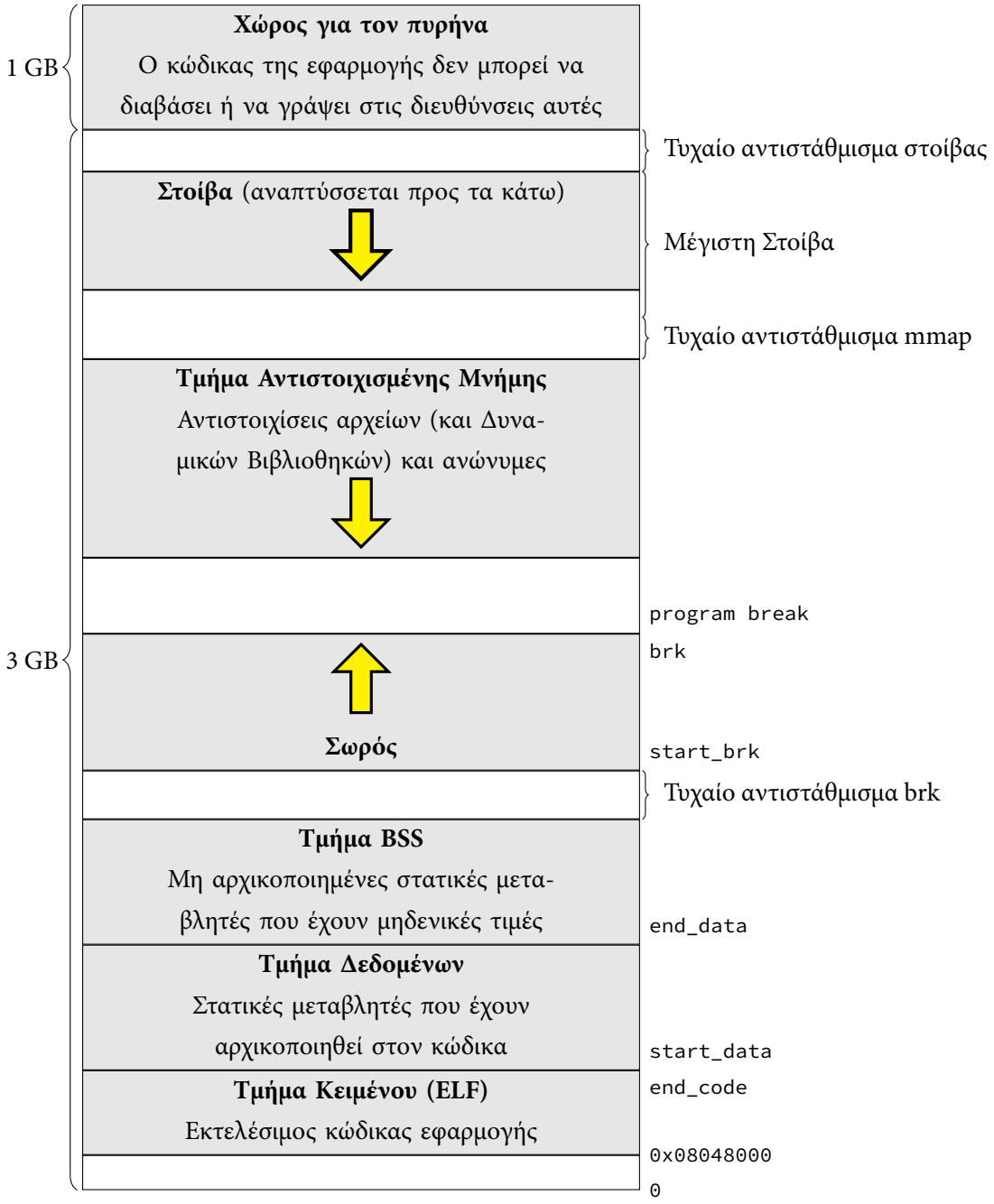
Ο δυναμικός αναθέτης μνήμης διαχειρίζεται αιτήματα μνήμης που συμβαίνουν κατά την εκτέλεση εφαρμογών. Οι δυναμικοί αναθέτες μνήμης είναι υπεύθυνοι για την οργάνωση των δυναμικά ανατεθειμένων δεδομένων στην μνήμη και για την εξυπηρέτηση των αιτημάτων μνήμης των εφαρμογών κατά την εκτέλεσή τους [5], [6]. Τα κύρια αιτήματα που λαμβάνουν χώρα είναι αυτά της ανάθεσης (allocation) και της αποδέσμευσης (de-allocation). Με έννοιες αντικειμενοστραφούς προγραμματισμού, στην περίπτωση που γίνει ένα αίτημα για την ανάθεση χώρου μνήμης ενός νέου αντικειμένου, ο δυναμικός αναθέτης μνήμης επιστρέφει έναν δείκτη στην εφαρμογή, ο οποίος δείχνει στην διεύθυνση μνήμης του ανατεθειμένου αντικειμένου. Στις γλώσσες προγραμματισμού C και C++, η δυναμική ανάθεση μνήμης γίνεται μέσω των κλήσεων στις συναρτήσεις `malloc()` και `new()` (η τελευταία μόνο για την C++). Στην περίπτωση που ένα αίτημα αποδέσμευσης μνήμης για ένα ήδη δυναμικά ανατεθειμένο αντικείμενο, ο αναθέτης φροντίζει να επιστρέψει στην εφαρμογή μία δυαδική τιμή σωστού ή λάθους αναφορική με την επιτυχία της διαδικασίας αποδέσμευσης.

1.3.1 Προγραμματιστική ευκολία και ασφάλεια

Προτού προχωρήσουμε στην δυναμική ανάθεση μνήμης, αξίζει να δούμε την διάταξη εικονικής μνήμης σε σύγχρονα ΛΣ. Στο Σχήμα 1.4 βλέπουμε πώς οργανώνεται η μνήμη σε μία διεργασία ενός Unixοειδούς ΛΣ, όπως το Linux. Στην αριστερή πλευρά βλέπουμε τον περιορισμό που εισάγει ένα ΛΣ 32 bits: οι εφαρμογές πάνω σε αυτό δεν μπορούν να έχουν πρόσβαση σε παραπάνω από 4 GB μνήμης.

Ο πυρήνας του ΛΣ χρησιμοποιεί το πάνω μέρος της μνήμης, προκειμένου να αποθηκευτούν πληροφορίες για την διεργασία αυτήν, τις οποίες μόνο ο πυρήνας μπορεί να τις διαβάσει. Σε περίπτωση που η εφαρμογή προσπαθήσεις να προσπελάσει την περιοχή αυτή, τότε οδηγείται η ίδια σε σφάλμα κατάτμησης (Segmentation Fault) και σταματάει η εκτέλεσή της. Στις πρώτες διευθύνσεις, λίγο πιο πάνω από το `0`, τοποθετείται ο εκτελέσιμος κώδικας (Text Segment), το Τμήμα Δεδομένων (Data Segment) που περιέχει τις στατικές μεταβλητές που έχουν αρχικοποιηθεί από τους προγραμματιστές της εφαρμογής και το Τμήμα BSS (Block Started by Symbol) που αποθηκεύονται οι στατικές μεταβλητές που δεν έχουν αρχικοποιηθεί από τον κώδικα της εφαρμογής.

Εκτός από τις στατικές μεταβλητές, μία εφαρμογή έχει και άλλα δεδομένα να αποθηκεύσει. Αρχικά, υπάρχει η στοίβα (Stack), μέσα στην οποία αποθηκεύονται τα πλαίσια της στοίβας (Stack



Σχήμα 1.4: Διάταξη μνήμης για μία διεργασία σε Unixοειδές Λειτουργικό Σύστημα 32 bits

Frames. Τα πλαίσια αυτά περιέχουν τις παραμέτρους εισόδου των συναρτήσεων των οποίων η εκτέλεση δεν έχει ακόμα ολοκληρωθεί. Για λόγους ασφαλείας, δηλαδή για να μην καταφέρει η εφαρμογή να αποκτήσει πρόσβαση σε περιοχές μνήμης που δεν πρέπει, η στοίβα έχει κάποια

τυχαία απόσταση (αντιστάθμισμα - offset). Για επιπλέον ασφάλεια, η στοίβα έχει ένα μέγιστο μέγεθος, το οποίο καθορίζεται από το ΛΣ.

Ο περιορισμός αυτός στο μέγεθος της στοίβας, μαζί φυσικά με το απρόβλεπτο κατά την μεταγλώττιση μέγεθος κάποιων μεταβλητών στο πρόγραμμα, κάνουν απαραίτητη την χρήση δυναμικών δεδομένων.

Αρχικά, λοιπόν, έχουμε την σωρό (Heap), μέσα στην οποία εμφανίστηκαν τα πρώτα δυναμικά δεδομένα. Η σωρός ξεκινάει σχεδόν αμέσως μετά το τμήμα BSS (παραδοσιακά στο σημείο `start_brk`) και έχει δυναμικό μέγεθος. Αυτό σημαίνει ότι το μέγεθός της μπορεί να αλλάξει κατά την διάρκεια εκτέλεσης του προγράμματος. Αν η εφαρμογή χρειαστεί επιπλέον χώρο, τότε ο αναθέτης της θα χρειαστεί να καλέσει μία κλήση συστήματος, ώστε το ΛΣ να εντάξει περισσότερη μνήμη στην εικονική διευθυνσιοδότηση της διεργασίας. Η διαδικασία αυτήν έχει συνδεθεί με την κλήση `brk()`, από πολύ παλιά στα ΛΣ, όταν τα περισσότερα από αυτά προορίζονταν για μονοπύρηνες επεξεργαστικές αρχιτεκτονικές και μονονηματικές εφαρμογές. Πιο συγκεκριμένα, το ΛΣ κρατούσε σε έναν δείκτη ανά διεργασία, τον `brk` (από το program break), το σημείο που τελείωνε η σωρός. Αν τυχόν η εφαρμογή ήθελε περισσότερο χώρο, έπρεπε να καλέσει την `brk()` με όρισμα την νέα διεύθυνση που έδειχνε στο τέλος της σωρού, ή να καλέσει την `sbrk()` με όρισμα τον επιπλέον χώρο που ήθελε να έχει η σωρός. Οι αντίστοιχες κλήσεις με χαμηλότερες διευθύνσεις μνήμης ή αρνητικούς αριθμούς υποδήλωναν ότι η εφαρμογή αποδέσμευε χώρο μνήμης και ΛΣ μπορούσε να το χρησιμοποιήσει όπως ήθελε.

Η προσέγγιση αυτή για την σωρό συνεχίζει να υφίσταται, αν και θεωρείται απαρχαιωμένη (obsolete) στα περισσότερα σύγχρονα ΛΣ. Κάποια μάλιστα από αυτά αρκούνται στην εξομοίωση των κλήσεων αυτών (macOS[1]). Η αντιπρόταση για τις `brk()` και `sbrk()` έρχεται με τις `mmap()` και `munmap()`. Οι συναρτήσεις αυτές χρησιμοποιούνται καταρχάς για να αντιστοιχιστούν στην μνήμη της διεργασίας κάποια αρχεία. Αυτός είναι και ο τρόπος με τον οποίο τα ΛΣ συνδέουν τις δυναμικές βιβλιοθήκες στις εφαρμογές: με την `mmap()` αντιστοιχίζεται ο εκτελέσιμος κώδικας των βιβλιοθηκών αυτών σε διευθύνσεις μνήμης που είναι προσπελάσιμες στην εκάστοτε εφαρμογή. Έκτος όμως από αυτό, η `mmap()` έχει μεγάλη σημασία στην δυναμική ανάθεση μνήμης χάρη στην δυνατότητά της να πραγματοποιεί ανώνυμες αντιστοιχίσεις. Ως ανώνυμες αντιστοιχίσεις, αναφέρουμε τις αντιστοιχίσεις που γίνονται μεταξύ των εικονικών διευθύνσεων των διεργασιών με σελίδες μνήμης του συστήματος που δε χρησιμοποιούνται αλλού. Έτσι, η διεργασία και με την βοήθεια της `mmap()` μπορεί να βρει νέο διαθέσιμο χώρο, του οποίου τις θέσεις μνήμης μπορεί να μηδενίσει, για λόγους ασφαλείας, χωρίς επιπλέον κόστος. Επίσης, τον ίδιο χώρο μπορεί και να αποδεσμεύσει μελλοντικά με την `munmap()`, με αρκετά μεγαλύτερη άνεση από την `brk()`.

[1] `http://opensource.apple.com//source/Libc/Libc-763.12/emulated/brk.c`

1.3.2 Δυναμικοί Αναθέτες Μνήμης

Μέχρι το σημείο αυτό έχουμε περιγράψει πώς μία εφαρμογή που τρέχει σε κάποιο σύστημα, λαμβάνει μνήμη από αυτό. Στο εξής, θα εξετάσουμε πώς μπορεί να διαχειριστεί η εφαρμογή αυτή το χώρο αυτό, μέσω του δυναμικού αναθέτη μνήμης.

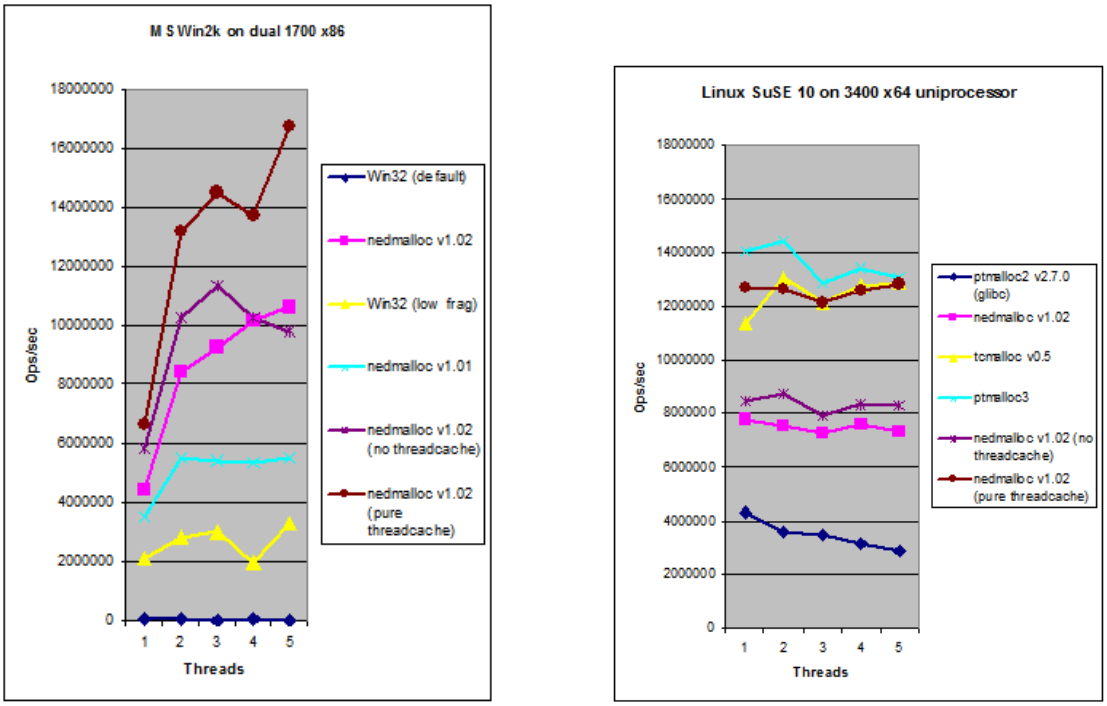
Στην γλώσσα προγραμματισμού C η Προγραμματιστική Διεπαφή Εφαρμογής (ΠΔΕ) των δυναμικών αναθετών μνήμης συμπεριλαμβάνεται ήδη στα πρότυπά της [7]. Στο πρότυπο του 1999 αναφέρονται οι συναρτήσεις `malloc()`, `calloc()`, `realloc()` και `free()`. Η `malloc()` αναθέτει χώρο στην μνήμη για ένα αντικείμενο, το μέγεθος του οποίου δίνεται ως όρισμα. Επιστρέφει την διεύθυνση στην μνήμη ή ένα κενό (null) δείκτη αν δεν μπορέσει να πάρει από το σύστημα την απαιτούμενη μνήμη. Η `calloc()` αναθέτει χώρο στην μνήμη για ένα συγκεκριμένο αριθμό αντικειμένων που έχουν το ίδιο μέγεθος, με τον αριθμό αυτό και το μέγεθος να δίνονται ως ορίσματα στην συνάρτηση. Επιστρέφει αντίστοιχα έναν δείκτη στην διεύθυνση μνήμης του πρώτου αντικειμένου ή δείκτη null αν δεν επιτευχθεί η ανάθεση μνήμης. Η `free()` δέχεται ως όρισμα έναν δείκτη και προσπαθεί να αποδεσμεύσει την μνήμη που αντιστοιχεί σε αυτόν, ώστε να μπορεί να χρησιμοποιηθεί σε επόμενες αναθέσεις. Αν ο δείκτης-όρισμα είναι null, τότε δε λαμβάνει χώρα κάποια δράση από την πλευρά του αναθέτη, ενώ αν ο δείκτης δεν ταιριάζει σε κάποιο δείκτη που είχε δοθεί νωρίτερα από κάποια κλήση των `malloc()`, `calloc()` ή `realloc()`, τότε το πρότυπο δεν ορίζει κάποια συμπεριφορά. Τέλος, η `realloc()` δέχεται δύο ορίσματα, έναν δείκτη και έναν αριθμό. Αποδεσμεύει την μνήμη που αντιστοιχεί στον δείκτη και επιστρέφει έναν νέο δείκτη πάνω σε νέο αντικείμενο, το οποίο έχει το μέγεθος που έχει καθορίσει ο αριθμός-όρισμα. Τα περιεχόμενα του νέου αντικειμένου θα πρέπει να είναι τα ίδια με αυτά του παλιού πριν την αποδέσμευσή του, μέχρι το λιγότερο από το παλιό και το νέο μέγεθος.

Καθώς η λειτουργία του αναθέτη εντάσσεται σε πρότυπα και είναι εφαρμόσιμη σε όλα τα προγράμματα, κοινές βιβλιοθήκες λογισμικού χρησιμοποιούνται στην πλειονότητα των περιπτώσεων. Είναι, επομένως, λογικό οι σχετικές συναρτήσεις να έχουν ήδη συμπεριληφθεί σε πρότυπα όπως αυτό της γλώσσας προγραμματισμού C και διάφορα Λειτουργικά Συστήματα να φέρουν ήδη δυναμικούς αναθέτες μαζί τους.

1.3.3 Επιδόσεις

Εφόσον οι δυναμικοί αναθέτες είναι τόσο ενσωματωμένοι στην τεχνολογική στοίβα που απαρτίζει το λογισμικό των περισσότερων υπολογιστικών συστημάτων, είναι φυσικό να ενδιαφερόμαστε για τις επιδόσεις τους. Είναι φυσιολογικό οι αναθέτες μνήμης να εισάγουν επιπλέον κόστη στην απόδοση ενός συστήματος –- είναι ένα αναμενόμενο τίμημα για την δυναμικότητά του και «αναγκαίο κακό» για την ενίσχυση της ασφάλειάς του.

Μία μελέτη που διεξήχθη το 1994 από την Digital Equipment Corporation παρουσιάζει τα επιπλέον κόστη στην απόδοση που συνεπάγεται για μία ποικιλία από αναθέτες. Ο χαμηλότερος μέσος αριθμός εντολών (instructions) που απαιτούνταν για να ανατεθεί μία περιοχή μνήμης ήταν 52 (όπως είχε μετρηθεί από έναν profiler σε επίπεδο εντολών για διάφορα λογισμικά) [8].



(α) Οι επιδόσεις διαφορετικών δυναμικών αναθετών μνήμης σε περιβάλλον Windows

(β) Οι επιδόσεις διαφορετικών δυναμικών αναθετών μνήμης σε περιβάλλον Linux

Σχήμα 1.5: Οι επιδόσεις διαφορετικών δυναμικών αναθετών μνήμης σε διάφορα λειτουργικά συστήματα το 2013 σύμφωνα με τον αναθέτη nedmalloc [9]

Ωστόσο, το κόστος αυτό δεν είναι πάντα σταθερό. Πριν το 2005, οι δυναμικοί αναθέτες μνήμης που ερχόντουσαν μαζί με τα ΛΣ δεν θεωρούνταν ιδιαίτερα αποτελεσματικοί. Πολλές εφαρμογές ερχόντουσαν με τους δικούς τους αναθέτες, ενώ σιγά-σιγά πρόβαλαν και ανεξάρτητες βιβλιοθήκες με αναθέτες «γενικού σκοπού», οι οποίες μπορούσαν να αντικαταστήσουν τους αντίστοιχους αναθέτες του συστήματος για καλύτερες επιδόσεις [9] χωρίς οι προγραμματιστές να αλλάξουν γραμμή κώδικα από τις εφαρμογές τους. Όπως φαίνεται στα Σχήματα 1.5α και β, το κέρδος που θα μπορούσε να αποκομίσει κανείς με έναν διαφορετικό αναθέτη μπορούσε να κυμανθεί από 2 έως και 3 φορές έναντι του αναθέτη που ερχόταν μαζί με το ΛΣ.

Θα πρέπει, ωστόσο, να σημειωθεί ότι στην σημερινή εποχή και για τα συμβατικά υπολογιστικά

συστήματα, οι επιδόσεις των αναθετών μνήμης έχουν πλέον σταθεροποιηθεί και σπάνια να παρατηρηθεί μεγάλη απόκλιση επιδόσεων μεταξύ των αναθετών του ΛΣ και αναθετών τρίτων. Πέρα από τις επιδόσεις, η έμφαση σήμερα δίνεται και στην διατήρηση της συμβατότητας του συγκεκριμένου μοντέλου στις σύγχρονες αρχιτεκτονικές και εφαρμογές, αλλά και στην κατανάλωση μνήμης.

1.4 Τάσεις σε σύγχρονα υπολογιστικά συστήματα και εφαρμογές

Πολυπύρηνες αρχιτεκτονικές

Όπως αναφέραμε παραπάνω, η τάση αύξησης του αριθμού των τρανζίστορ δεν έχει ελαττωθεί τόσο, όσο ο ρυθμός αύξησης της συχνότητας λειτουργίας. Αυτό έχει ανοίξει τον δρόμο για την κατασκευή πιο πολύπλοκων ψηφιακών κυκλωμάτων και στην παραλληλοποίηση τους δημιουργώντας πολυπύρηνα συστήματα. Οι επιδόσεις ενός μονού πυρήνα από το 2007 διαφέρουν ελάχιστα από τον αντίστοιχο πυρήνα το 2014 [10]. Χάρη όμως στην παραλληλία και την διανυσματοποίηση (vectorization) που υποστηρίζουν οι καινούριοι επεξεργαστές, μπορούμε να επιτύχουμε ακόμα και δύο τάξεις μεγέθους υψηλότερες επιταχύνσεις στην εκτέλεση των εφαρμογών.

Αυτή η επιτάχυνση συνοδεύεται και από μία προς τα πίσω συμβατότητα, ιδιαίτερα στα εμπορικά προϊόντα. Οι καινούριες αρχιτεκτονικές θα πρέπει να υποστηρίζουν εφαρμογές που έχουν γραφτεί σε παλαιότερο κώδικα, ώστε να διασφαλίζεται η εκτέλεση προγραμμάτων σε ένα πλήθος καινούριων και παλιότερων συσκευών χωρίς την επαναμεταγλώττισή τους. Φυσικά, όταν αυτό είναι εφικτό, τα προγράμματα αναβαθμίζουν τον κώδικά τους προκειμένου να υποστηρίξουν τα καινούρια χαρακτηριστικά των νέων αρχιτεκτονικών και ενδεχομένως να βελτιώσουν τις επιδόσεις τους.

Η συγκεκριμένη προσέγγιση δεν παρατηρείται μόνο στα υπολογιστικά συστήματα γενικού σκοπού. Η αγορά των ενσωματωμένων συσκευών έχει ωριμάσει αρκετά, με αποτέλεσμα να υπάρχουν πλέον ισχυρές συσκευές και εκεί, οι οποίες στηρίζονται σε προηγούμενες. Ως εκ τούτου, ακόμα και εδώ, μπορούμε να μιλάμε για πολυπύρηνες πλέον αρχιτεκτονικές και η ανάγκη για λειτουργικό σύστημα για την διαχείριση των πόρων να είναι πλέον ρεαλιστική.

brk και mmap

Αν και η `mmap()` προτιμάται από τα ΛΣ, η `brk()` δεν έχει εκλείψει ακόμα. Πρόκειται για έναν μηχανισμό, ο οποίος έχει δοκιμαστεί εκτενώς στο παρελθόν και είναι πολύ γρήγορος. Από την άλλη πλευρά, η `mmap()` είναι πιο ασφαλής και εγγυάται καλύτερη κλιμακωσιμότητα (scalability), ιδιαίτερα όσο αυξάνει το μέγεθος της μνήμης που πρέπει να διαχειριστεί ο αναθέτης. Όταν

χρησιμοποιείται η `mmap()` είναι ευκολότερο ολόκληρες περιοχές της να αποδεσμευτούν και να επιστραφούν στο σύστημα, σε αντίθεση με την `brk()`, η οποία είναι υποχρεωμένη να διατηρήσει κάποιες περιοχές μνήμης ανεκμετάλλευτες αν τις έχει απελευθερώσει, στην περίπτωση που διατηρεί αντικείμενα σε υψηλότερες διευθύνσεις μνήμης. Επίσης, η `mmap()` δίνει περισσότερες ευκαιρίες στον πυρήνα του ΛΣ να διαχειριστεί την μνήμη των εφαρμογών, καθώς του είναι ευκολότερο να καταλάβει ποιες περιοχές μνήμης χρησιμοποιούνται εκείνη την στιγμή και να μετακινήσει στην μνήμη ανταλλαγής (swap) ό,τι δε χρησιμοποιείται. Τέλος, επεκτάσεις στην ΠΔΕ της `mmap()`, όπως αυτήν της `madvise()`, δίνουν πληροφορίες στον πυρήνα για τις ανάγκες και κατευθύνσεις σε μνήμη της εφαρμογής, ώστε η εφαρμογή αλλά και το σύστημα να συμπεριφερθούν καλύτερα σε ένα πολύπλοκο περιβάλλον, όπου ενδεχομένως πολλαπλές εφαρμογές τρέχουν παράλληλα, αγωνιζόμενες για πρόσβαση στους κοινούς πόρους του συστήματος.

Αυτό, λοιπόν, που συστήνεται για την ώρα στους αναθέτες μνήμης είναι ένας συνδυασμός χρήσης των `brk()` και `mmap()`. Για αναθέσεις μικρού μεγέθους, ο αναθέτης μπορεί να χρησιμοποιήσει την `brk()`, ώστε να έχει ταχύτερη απάντηση από το σύστημα. Αντίστοιχα, όταν η εφαρμογή έχει μεγαλύτερες απαιτήσεις σε μέγεθος, ο αναθέτης χρησιμοποιεί την `mmap()`. Για παράδειγμα, στην ptmalloc, τον επίσημο αναθέτη της glibc, υπήρχε αρχικά ένα κατώφλι στα 128 KiloBytes, κάτω από το οποίο χρησιμοποιούνταν η `brk()` και πάνω από αυτό η `mmap()`. Σε πιο πρόσφατες μάλιστα εκδόσεις του κώδικα (από το 2006 και μετά), το κατώφλι αυτό έγινε δυναμικό: ο συγκεκριμένος αναθέτης προσπαθεί πλέον να προσαρμοστεί στο μοτίβο ανάθεσης μνήμης του εκάστοτε προγράμματος. Αν εντοπίσει ότι ανατίθενται πολλά μεγάλα αντικείμενα και αποδεσμεύονται νωρίς, τότε θα αυξήσει το κατώφλι αυτό με την προοπτική ότι η εφαρμογή θα αναθέτει μεγάλα αντικείμενα και τα αποδεσμεύει πιο συχνά. Αυτή η προσέγγιση ακολουθείται και από άλλους αναθέτες και πραγματικά επιταχύνει αρκετά την διαδικασία της `malloc()`, αλλά με τον κίνδυνο να δημιουργούνται πιθανώς μεγαλύτερες «τρύπες» στην μνήμη.

Εφαρμογές

Οι σημερινές εφαρμογές έχουν γίνει πλέον πολύπλοκες και ταυτόχρονα πιο εύκολες. Γλώσσες προγραμματισμού υψηλού επιπέδου και δαιδαλώδη πλαίσια λογισμικού επιτρέπουν σήμερα σε προγραμματιστές να εκφραστούν με ευκολότερο τρόπο από ό,τι ποτέ. Στην πραγματικότητα όμως, έχουν πλέον προστεθεί επίπεδα αφαιρετικότητας, τα οποία κρύβουν λεπτομέρειες υλοποίησης. Η απόκρυψη αυτή δεν μπορεί ξεκάθαρα να χαρακτηριστεί ευνοϊκή ή ενάντια στην βελτιστοποίηση των συστημάτων (πλατφορμών και εφαρμογών μαζί).

Από την άλλη πλευρά, η μεταφορά προγραμμάτων σε νέες υπολογιστικές εφαρμογές έχει γίνει αρκετά επιθυμητή στην σημερινή αγορά. Η καθετοποίηση που παρατηρείται σε αρκετά «οικοσυστήματα», τόσο σε υπολογιστικά συστήματα γενικού σκοπού (π.χ. Wintel), όσο και ενσωμα-

τωμένα (π.χ. συσκευές για Apple iOS και Google Android), κάνει την μεταφορά των προγραμμάτων δύσκολη. Παρόλα αυτά, για την καλύτερη διείσδυση στην αγορά είναι σχεδόν βέβαιο ότι μία εφαρμογή θα πρέπει να γραφεί για παραπάνω από μία πλατφόρμα.

Σε αυτό το σύνθετο πεδίο των εφαρμογών, η δυναμική ανάθεση μνήμης έχει έναν ιδιότυπο ρόλο. Οι εφαρμογές δε μεριμνούν τόσο για αυτήν, εκτός και αν πρόκειται για εφαρμογές που γράφονται σε επίπεδο συστήματος. Χαρακτηριστικό είναι ότι πολλά προγράμματα γράφονται χωρίς να ελέγχεται ότι υπάρχει δυναμική μνήμη: είναι αρκετά αισιόδοξα ότι το σύστημα θα τους επιστρέφει πάντα μνήμη. Επιπλέον, αρκετές γλώσσες προγραμματισμού παρέχουν διευκολύνσεις στους προγραμματιστές, οι οποίες ξεκινούν από ημιαυτόματους τρόπους (έξυπνοι δείκτες - smart pointers) και καταλήγουν σε εντελώς αυτόματους (garbage collection). Ωστόσο, είναι σχεδόν βέβαιο ότι σε κάποιο σημείο της τεχνολογικής στοίβας εκτελούνται κλήσεις όπως αυτήν της `malloc()` και ότι είναι αρκετά πιθανό να δούμε διαφορές αν χρησιμοποιήσουμε μία διαφορετική της υλοποίηση.

1.4.1 Προβλήματα

Ακόμα και λαμβάνοντας υπόψη τις σύγχρονες τάσεις στα επεξεργαστικά συστήματα και τις εφαρμογές, θα έλεγε κάποιος ότι τα προβλήματα στην δυναμική διαχείριση μνήμης και ειδικότερα στην δυναμική ανάθεση, δεν διαφέρουν αρκετά από αυτά του παρελθόντος. Το σημαντικότερο αντιστάθμισμα, αυτό μεταξύ κατανάλωσης μνήμης και εκτέλεσης κώδικα, συνεχίζει να υφίσταται, μόνο που ενδεχομένως να έχουν αλλάξει οι οπτικές μέσα από τις οποίες το εξετάζουμε.

Συγκεκριμένα, λοιπόν, μας ενδιαφέρει πρωταρχικά η κατανάλωση μνήμης που μπορεί να αποφέρει ένας δυναμικός αναθέτης μνήμης. Το κόστος των δυναμικών δεδομένων στην κατανάλωση χώρου μνήμης και στις επιδόσεις ενδεχομένως να έχει γίνει αρκετά δύσκολο να εκτιμηθεί. Ήδη από το παρελθόν έχει φανεί πόσο δύσκολη μπορεί να είναι η εκτίμηση αυτή [5]. Σε κάθε περίπτωση, χρειαζόμαστε επαρκή εργαλεία μελέτης της επίδοσης των δυναμικών αναθετών.

Ο κατακερματισμός του χώρου παραμένει ένα σοβαρό πρόβλημα. Οι σημερινές μνήμες έχουν πλέον αποκτήσει μεγέθη που θεωρούνταν ασύλληπτα στο παρελθόν, αλλά αυτό δε θα πρέπει να μας κάνει να αγνοήσουμε την σωστή και δίκαιη χρήση της μνήμης. Όταν, εξάλλου, πρόκειται να εκτελέσουμε εφαρμογές, οι οποίες έχουν ανάγκες πολλών Megabytes ή και Gigabytes σε μνήμη, ή αν πρόκειται να εκτελέσουμε παράλληλα πλήθος εφαρμογών, θα πρέπει να προσπαθούμε για την σοφότερη χρήση της μνήμης του συστήματός μας.

Τέλος, αξίζει να αναφερθεί και το ζήτημα των διαφορετικών επεξεργαστικών αρχιτεκτονικών. Οι σημερινές αρχιτεκτονικές, αν και βασίζονται σε παλαιότερες, έχουν αρκετά διαφορετικά στοιχεία. Εξαιτίας των διαφόρων εργαλείων μεταγλώττισης, είναι πολύ εύκολο να μεταφέρει κανείς

παλιό κώδικα στο νέο σύστημα, ακόμα και ενός αναθέτη δυναμικής μνήμης, ο οποίος βρίσκεται στις χαμηλότερες θέσεις της τεχνολογικής στοίβας του λογισμικού. Είναι όμως λάθος να μην μελετάται επαρκώς το μοντέλο μνήμης της νέας επεξεργαστικής αρχιτεκτονικής, καθώς και τα άλλα χαρακτηριστικά της, ώστε η διαχείριση μνήμης να είναι βέλτιστη.

1.5 Στόχοι διατριβής και συνεισφορές

Στα πλαίσια της διατριβής αυτής:

1. παρουσιάζονται οι πλέον εξελιγμένοι αναθέτες μνήμης
2. οργανώνεται συστηματικά ο χώρος επιλογών και υλοποιείται η σχεδίαση της καθεμίας
3. μελετώνται ειδικές περιπτώσεις για την περαιτέρω βελτίωση της δυναμικής διαχείρισης μνήμης σε συγκεκριμένες εφαρμογές και συστήματα

Σε αυτήν την διατριβή παρουσιάζουμε τον σχεδιασμό ενός *πλαισίου ανάπτυξης δυναμικών αναθετών μνήμης* ή αλλιώς *dmmlib*. Παρουσιάζουμε μία μελέτη αξιολόγησης σε δύο διαφορετικές αρχιτεκτονικές πλατφόρμες, δείχνοντας ότι η dmmlib μπορεί να αποφέρει σημαντικά οφέλη στην εκτέλεση εφαρμογών που χρησιμοποιούν δυναμικά δεδομένα σε ενσωματωμένα συστήματα και υπολογιστικά συστήματα γενικού σκοπού.

Η διατριβή αυτή έχει τις εξής συνεισφορές:
Το πλαίσιο ανάπτυξης δυναμικών αναθετών μνήμης dmmlib: Το επίκεντρο αυτής της εργασίας βρίσκεται στον σχεδιασμό και την υλοποίηση της dmmlib. Παρουσιάζουμε τις σχετικές, σύγχρονες προσεγγίσεις στην *δυναμική ανάθεση μνήμης*, προτείνοντας την dmmlib ως ένα ενιαίο πλαίσιο ανάπτυξης και αξιολόγησης αναθετών μνήμης σε διάφορες αρχιτεκτονικές υπολογιστικών συστημάτων. Επίσης, παρουσιάζουμε με λεπτομέρεια το σύνολο των τρεχόντων *πολιτικών* και *μηχανισμών* που απαρτίζουν την dmmlib. Τα μοντέλα αυτά μπορούν να χρησιμοποιηθούν από τους σχεδιαστές συστημάτων για να κατανοήσουν τους παράγοντες που επηρεάζουν την απόδοση των εφαρμογών στα συστήματά τους.

Βελτιστοποίηση της απόδοσης Δυναμική Διαχείριση Μνήμης (ΔΔΜ) σε αρχιτεκτονικές πολλαπλών πυρήνων: Περιγράφουμε μία πολυπύρηνη αρχιτεκτονική για ενσωματωμένα συστήματα, την P2012, η οποία έχει ως στόχο να επιταχύνει συγκεκριμένες εφαρμογές. Παρουσιάζουμε τον τρέχοντα τρόπο με τον οποίο η πλατφόρμα αυτή διαχειρίζεται την μνήμη για τις εργασίες που εκτελούνται πάνω σε αυτήν. Μεταφέρουμε την dmmlib στην συγκεκριμένη αρχιτεκτονική

και παρουσιάζουμε μία υλοποίηση η οποία είναι μέχρι και 2,6 φορές γρηγορότερη της αρχικής χωρίς να μεταβάλλεται η κατανάλωση μνήμης.

Βελτίωση της κλιμακωσιμότητας δυναμικών αναθετών μνήμης: Προτείνουμε την χρήση εξειδικευμένου υλισμικού από την πλευρά του αναθέτη και ενός δυναμικού τρόπου χρήσης των επεξεργαστικών δομών της πλατφόρμας, ώστε να μην παρουσιάζονται προβλήματα απόδοσης όσο αυξάνεται ο αριθμός των επεξεργαστών στο σύστημα. Τα αποτελέσματα μάς δείχνουν ότι για μεγάλο αριθμό επεξεργαστών, η προτεινόμενη λύση μπορεί να ξεπεράσει σε επιδόσεις αναθέτες που έχουν αναπτυχθεί αποκλειστικά σε χαμηλό επίπεδο, εξειδικευμένους στην εκάστοτε πλατφόρμα.

Προσαρμοστική ΔΔΜ Παρουσιάζουμε έναν τρόπο πρόβλεψης του φόρτου εργασίας ενός αναθέτη μνήμης. Χρησιμοποιώντας την πρόβλεψη αυτή, αλλάζουμε κάποιες πολιτικές του αναθέτη κατά την εκτέλεση των εφαρμογών. Τα αποτελέσματα μάς δείχνουν ότι η προσέγγιση αυτή εισάγει ελάχιστο επιπλέον κόστος, ενώ επιτυγχάνει να διατηρήσει τον κατακερματισμό της μνήμης στο ελάχιστο.

1.6 Χάρτης διατριβής

Το υπόλοιπο αυτής της διατριβής οργανώνεται ως εξής:

Στο Κεφάλαιο 2, παρουσιάζουμε τα υπολογιστικά συστήματα στα οποία στοχεύει η εργασία μας, συζητούμε για σύγχρονους δυναμικούς αναθέτες μνήμης, και περιγράφουμε πιθανές ελλείψεις στις τρέχουσες υλοποιήσεις τους. Το Κέφαλαιο 3 περιγράφει με λεπτομέρειες την dmmlib και παρουσιάζει ένα σύνολο πολιτικών και μηχανισμών που συστήνουν το πλαίσιο αυτό.

Στο Κεφάλαιο 4, παρουσιάζουμε εφαρμογές της dmmlib. Πιο συγκεκριμένα, περιγράφουμε την πλατφόρμα για ενσωματωμένα συστήματα P2012 και τον τρόπο λειτουργίας και διαχείρισης των πόρων της. Εξηγούμε τις επιλογές της dmmlib που οδηγούν στην δημιουργία ενός καλύτερου αναθέτη. Επίσης, στο ίδιο κεφάλαιο εξηγούμε προβλήματα που μπορεί να προκύψουν αναφορικά με την κλιμακωσιμότητα του συστήματος ως προς τον αριθμό των επεξεργαστών και τους δυναμικούς αναθέτες μνήμης. Προτείνουμε την δημιουργία ενός αναθέτη που να αξιοποιεί το περιβάλλον του υψηλού αριθμού επεξεργαστών χωρίς όμως να θυσιάζει την απλότητα της διεπαφής του στις εφαρμογές. Στο Κεφάλαιο 5 παρουσιάζουμε έναν τρόπο πρόβλεψης του φόρτου εργασίας ενός αναθέτη, και πώς μέσα από την dmmlib μπορεί να παραχθεί ένας αναθέτης που να προσαρμόζεται στην πρόβλεψη αυτή.

Τέλος, στο Κεφάλαιο 6 παρουσιάζουμε τα συμπεράσματά μας από την εργασία αυτήν και σκια-

γραφούμε περιοχές για μελλοντική εργασία.

2 Πλαίσιο και σχετικές εργασίες

Στο παρόν κεφάλαιο θα αναφερθούμε στα πολυπύρηνα επεξεργαστικά συστήματα πάνω στα οποία εξερευνούμε την δυναμική ανάθεση μνήμης. Κατόπιν θα περιγράψουμε τους υπάρχοντες αναθέτες μνήμης και θα αναλύσουμε τις σύγχρονες τάσεις σε αυτούς, καθώς και προβλήματα στην κάθε προσέγγιση.

2.1 Υπολογιστικά συστήματα πολλαπλών πυρήνων

Η τάση στις σύγχρονες αρχιτεκτονικές μικροεπεξεργαστών είναι ξεκάθαρη: τα πολυπύρηνα συστήματα έχουν εδραιωθεί. Καθώς οι πόροι σε πυρίτιο αυξάνονται συνεχώς, οι σχεδιαστές επεξεργαστών μπορούν να τοποθετήσουν ολοένα και περισσότερους πυρήνες σε μία ψηφίδα, με μαζικές πολυπύρηνες ψηφίδες να βρίσκονται στον ορίζοντα. Ειδικοί της βιομηχανίας έχουν προβλέψει συστήματα πολλαπλών πυρήνων στα μέσα της επόμενης δεκαετίας.

Από τις κάρτες γραφικών σε πιο σύνθετα πολυπύρηνα

Κάποια από τα συστήματα αυτά βρίσκονται ήδη εδώ και μάλιστα προς χρήση των καταναλωτών. Οι σύγχρονες ναυαρχίδες καρτών γραφικών αποτελούνται πλέον από περισσότερα από 7 δισεκατομμύρια τρανζίστορ και 2500 πυρήνες βρίσκονται στην διάθεση των προγραμματιστών [11]. Αυτή η ικανότητα επεξεργασίας των καρτών γραφικών έχει ήδη προτρέψει προγραμματιστές να ξεκινήσουν να εξερευνούν την χρήση των καρτών αυτών σε υπολογισμούς γενικού σκοπού. Αυτό μάς δίνει το λεγόμενο πεδίο των General-Purpose GPUs (GPGPUs) [12]. Στην προσπάθεια να αξιοποιηθεί το περιφερειακό αυτό, μόνο του ή σε συνδυασμό με τα υπόλοιπα επεξεργαστικά στοιχεία ενός υπολογιστικού συστήματος, έχουν προταθεί διάφορα προγραμματιστικά μοντέλα, όπως η CUDA, το OpenCL και τελευταία το Vulkan.

Όσο εξελιγμένες και να είναι οι διεπαφές και τα μοντέλα αυτά, θα πρέπει να τονίσουμε ότι δεν ταιριάζουν τα φορτία όλων των εφαρμογών στις πλατφόρμες αυτές. Μελέτες δείχνουν ότι ένας επεξεργαστής 4 πυρήνων γενικού σκοπού σε συνδυασμό με μία κατάλληλα αναπτυγμένη εφαρμογή μπορεί να συγκριθεί σε απόδοση με μία κάρτα γραφικών ή τουλάχιστον η απόδοση της κάρτας αυτής δεν μπορεί να ξεπεράσει πάνω μία τάξη μεγέθους σε τυπικά φορτία εργασιών [13].

Από την μία, λοιπόν, πλευρά υπάρχουν οι κάρτες γραφικών με τους χιλιάδες, απλούς σε λειτουργία πυρήνες τους, που είναι οργανωμένες με τέτοιο τρόπο που δεν μπορεί να υπάρχει η δυνατότητα ελέγχου ενός πυρήνα μεμονωμένα. Από την άλλη πλευρά, έχουμε τους επεξεργαστές γενικού σκοπού, οι οποίοι είναι καλοί γενικά, αλλά δεν μπορούμε να προσθέσουμε πολλούς πυρήνες σε μία ψηφίδα (στην τρέχουσα τεχνολογία βλέπουμε συστήματα μέχρι και 8 πυρήνες με την μέγιστη απόδοση[1]).

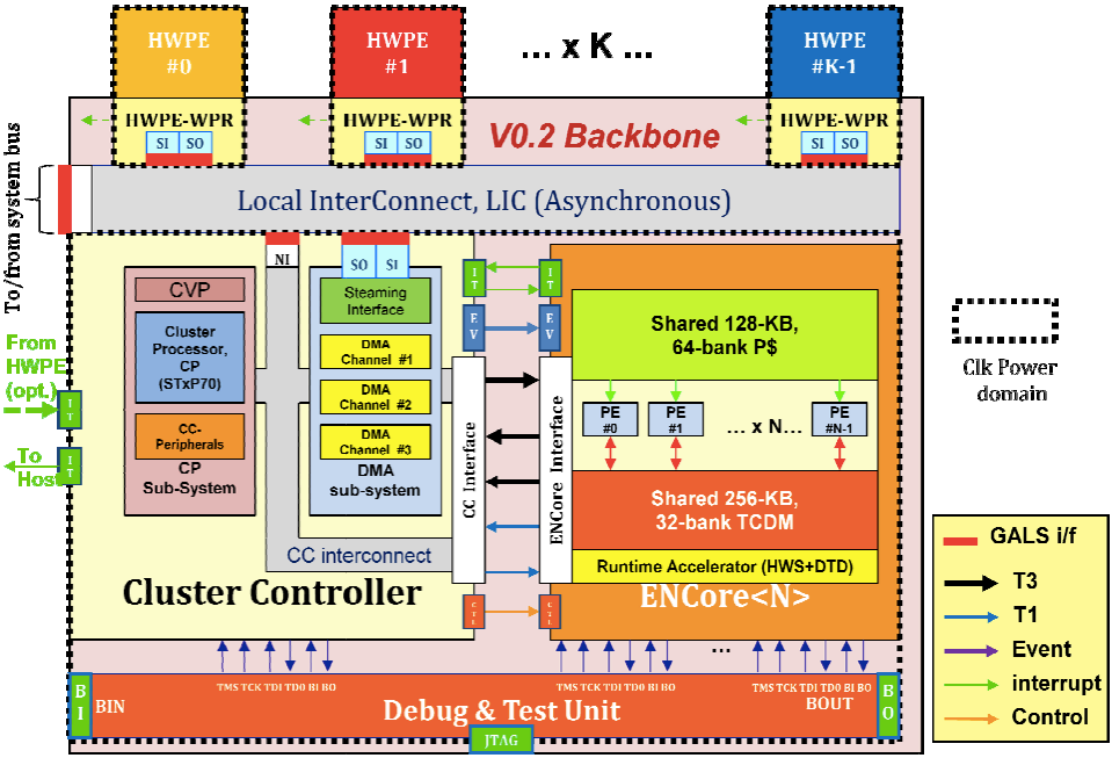
Οι δύο αυτές τάσεις έχουν δημιουργήσει ένα νέο ενδιάμεσο χώρο στα επεξεργαστικά συστήματα, αυτόν με τα πολλαπλούς πυρήνες επεξεργαστικών στοιχείων, οι οποίοι είναι σε θέση να προγραμματίζονται σχετικά ανεξάρτητα μεταξύ τους [14]. Στο χώρο αυτόν ανήκουν υλοποιήσεις που προέρχονται και από τους δύο κόσμους με ορισμένες αλλαγές. Από τον κόσμο των καρτών γραφικών έχουν προστεθεί ιεραρχίες (λανθανουσών) μνημών και τα στοιχειώδη επεξεργαστικά στοιχεία έχουν ομαδοποιηθεί σε μεγαλύτερες, πιο αποδοτικές μονάδες (αρχιτεκτονική Nvidia Fermi). Από την άλλη πλευρά, μία αρκετά απλούστερη αρχιτεκτονική στους πυρήνες δίνει την δυνατότητα αύξησης του αριθμού που μπορούν να ενσωματωθούν σε μία ψηφίδα (αρχιτεκτονική Intel Many Integrated Core - MIC). Να σημειωθεί ότι και οι δύο αρχιτεκτονικές αυτές δεν αποτελούν την κεντρική επεξεργαστική μονάδα ενός υπολογιστικού συστήματος, παρά ένα συνεπεξεργαστή (coprocessor) ο οποίος επικοινωνεί με τον κεντρικό επεξεργαστή, την κεντρική μνήμη και τις περιφερειακές συσκευές με συμβατικούς τρόπους (για παράδειγμα μέσω της διεπαφής PCI Express).

Καθώς, λοιπόν, οι αρχιτεκτονικές αυτές αξιοποιούνται ήδη σε εμπορικές πλατφόρμες, η εργασία αυτή ασχολείται με την διαχείριση μνήμης σε τέτοιου είδους συστήματα και με την αντιμετώπιση πιθανών προβλημάτων που προκύπτουν από την ειδική φύση τους.

Πολλαπλοί πυρήνες στα ενσωματωμένα: Η πλατφόρμα P2012

Η τάση που συναντούμε με την αρχιτεκτονική MIC της Intel έχει ήδη μεταβεί στον κόσμο των ενσωματωμένων συστημάτων. Η Platform 2012 (P2012) είναι ένας πολυπύρηνος επιταχυντής, πάνω στον οποίο θα εξετάσουμε εξειδικευμένα στο Κεφάλαιο 4.1 την δυναμική ανάθεση μνήμης. Αναπτύχθηκε από την εταιρεία ST Microelectronics και το ερευνητικό ινστιτούτο CEA. Ο τελικός στόχος της P2012 είναι να καλύψει την περιοχή και το κενό στην ενεργειακή αποδοτικότητα μεταξύ των ενσωματωμένων επεξεργαστών γενικού σκοπού και των επιταχυντών ειδικού σκοπού [15]. Ο στόχος αυτός καθιστά την πλατφόρμα αυτή πλήρως συμβατή με το αντικείμενο των μαζικά πολυπύρηνων επιταχυντών, μέσα στο οποίο εξετάζουμε τη διαχείριση μνήμης στην διατριβή αυτή.

[1] http://ark.intel.com/products/82930/Intel-Core-i7-5960X-Processor-Extreme-Edition-20M-Cache-up-to-3_50-GHz



Σχήμα 2.1: Ένα σύμπλεγμα της P2012 [15]

Τα επεξεργαστικά στοιχεία στην P2012 οργανώνονται σε συμπλέγματα (clusters). Η οργάνωση ενός συμπλέγματος φαίνεται στο Σχήμα 2.1: Τα συμπλέγματα αποτελούνται από επεξεργαστικά στοιχεία γενικού σκοπού και αρχιτεκτονικής STxP70-v4, τα οποία αποκαλούνται επεξεργαστές ENCore. Αν και υπάρχει η επιλογή να συμπεριληφθούν εξειδικευμένοι πυρήνες υλισμικού, όπως και άλλες δυνατότητες που φαίνονται στο Σχήμα 2.1, πρόκειται για επιλογές που είναι εκτός του θέματος διαχείρισης της μνήμης. Τέλος, μία επεξεργαστική μονάδα όμοια με τους επεξεργαστές ENCore από αρχιτεκτονική άποψη, διαχειρίζεται το καθένα σύμπλεγμα.

Ολόκληρος ο επιταχυντής μπορεί να αποτελείται μέχρι στιγμής μέχρι και από τέσσερα συμπλέγματα με 16 επεξεργαστές το καθένα, αλλά εκτιμάται να είναι ακόμα περισσότερο επιδεκτικό διεύρυνσης. Στην περίπτωση που υπάρχουν παραπάνω από ένα συμπλέγματα, τότε απαιτείται ένας ακόμα επεξεργαστής για το συντονισμό τους, τον οποίο η ST αποκαλεί fabric controller.

Σχετικά με την ιεραρχία μνήμης, η P2012 έχει υιοθετήσει μία πολλαπλών επιπέδων: Υπάρχουν τρία επίπεδα μνήμης, ειδικότερα το Επίπεδο 1 (E1), 2 (E2) και 3 (E3). Όλοι οι επεξεργαστές ENCore χρησιμοποιούν τον ίδιο χάρτη μνήμης (memory map) και έχουν την ίδια πρόσβαση σε όλα

τα επίπεδα. Το E1 μοιράζεται σε όλα τα επεξεργαστικά στοιχεία του κάθε συμπλέγματος. Είναι μία πολύ γρήγορη, αλλά μικρή (μόλις 256 kilobytes) μνήμη. Έχει σχεδιαστεί για να είναι προσπελάσιμη με έναν ενιαίο τρόπο, δηλαδή είναι εγγυημένη η προσπέλαση από τους επεξεργαστές ENCore σε ένα σταθερό, μικρό αριθμό κύκλων ανεξάρτητα της κίνησης. Το E2 είναι η μνήμη που μοιράζεται ανάμεσα στα συμπλέγματα με ένα τρόπο NUMA. Δεν είναι ξεκάθαρο πόσους κύκλους θα πάρει σε ένα επεξεργαστικό στοιχείο να το προσπελάσει, καθώς αυτό εξαρτάται από την κίνηση στο Δίκτυο-σε-Ψηφίδα που ενώνει τα συμπλέγματα. Το μέγεθος του E2 είναι ορισμένο προς το παρόν στο 1 megabyte, αλλά μπορούν να υπάρξουν και διαμορφώσεις στις οποίες το E2 παραλείπεται. Τέλος, ένα μέρος της μνήμης του συστήματος που φιλοξενεί τον επιταχυντή είναι προσπελάσιμο από τον τελευταίο ως το E3. Το τυπικό μέγεθος του E3 είναι 256 megabytes. Αν και το μέγεθος αυτό είναι κατά πολύ μεγαλύτερο από το μέγεθος των άλλων δύο επιπέδων, η ποινή στην εκτέλεση είναι υπερβολικά υψηλή κάθε φορά που ένα επεξεργαστικό στοιχείο προσπαθεί να προσπελάσει τη μνήμη αυτή, και έτσι συστήνεται οι προσπελάσεις μνήμης από τον επιταχυντή προς το E3 να διατηρούνται στο ελάχιστο.

2.2 Δυναμική ανάθεση μνήμης

Ο εγγενής και αυξανόμενος δυναμισμός των σύγχρονων εφαρμογών που τρέχουν σε εξίσου σύνθετες, πολυεπεξεργαστικές πλατφόρμες οδηγεί συχνά σε απρόβλεπτες παρεκκλίσεις της κατανάλωσης μνήμης κατά την εκτέλεσή τους. Στην περίπτωση που θέλαμε να εξαλείψουμε τις παρεκκλίσεις αυτές και να αναπτυχθούν οι εφαρμογές με στατικά διαχειρίσιμη μνήμη, χρησιμοποιώντας εκτιμήσεις των χειρότερων περιπτώσεων, αυτό θα επέβαλε σοβαρές επιπτώσεις στην συνολική κατανάλωση μνήμης και ενέργειας. Δεν είναι λοιπόν περίεργο που οι προγραμματιστές ωθούνται να χρησιμοποιήσουν τη μνήμη δυναμικά [5].

Η δυναμική διαχείριση μνήμης (ή αλλιώς αποκαλούμενης και σωρού) είναι ένα ερευνητικό πεδίο με μακρόχρονη δράση. Η εκτενής αναφορά του Wilson [5] υποδεικνύει ότι η ΔΔΜ αφορά ένα πολύ σύνθετο, ιδιαίτερα δυναμικό πρόβλημα που δεν μπορεί να αποδειχτεί ότι λύνεται αποτελεσματικά σε κάθε περίπτωση. Οι σύγχρονοι αναθέτες μνήμης γενικού σκοπού συνδυάζουν ένα σύνολο τεχνικών, ώστε να εκτελούν το έργο του ικανοποιητικά στις περισσότερες περιπτώσεις. Ωστόσο, οι αλγόριθμοι αυτοί είναι κανονικά "στατικοί", δηλαδή μεταχειρίζονται κάθε σενάριο χρήσης μνήμης με τον ίδιο τρόπο.

2.2.1 Για συστήματα γενικού σκοπού

Μία από τις πρώτες απόπειρες παραμετροποίησης του δυναμικού αναθέτη μνήμης σημειώνεται το 1992 με τον αναθέτη CustoMalloc [16] να προτείνει την χρήση αναθέτη παραμετροποιη-

μένου σε συγκεκριμένη εφαρμογή κάθε φορά. Λίγο αργότερα, το 1993, προτείνονται οι πρώτοι αναθέτες μνήμης, οι οποίοι τακτοποιούν τα ελεύθερα μπλοκ της μνήμης σε διαφορετικές κατανομές με βάση το μέγεθός τους [17], σε μια πρώτη προσπάθεια να παραλληλοποιήσουν την εργασία που αναλαμβάνουν οι αναθέτες. Στην έκθεση του Wilson πάνω στους δυναμικούς αναθέτες μνήμης το 1995 [5] γίνεται μια πρώτη απόπειρα χαρτογράφησης του χώρου: Διάφορες πολιτικές και μηχανισμοί αναφέρονται και αξιολογούνται ως προς τον κατακερματισμό της μνήμης που επιφέρουν. Η έκθεση αυτή πρόκειται να ανανεωθεί πολύ αργότερα από μία στρατηγική συστηματικής εξερεύνησης [18], η οποία προορίζεται να πραγματοποιηθεί κατά το χρόνο σχεδιασμού προκειμένου να εξειδικευτούν οι υπηρεσίες δυναμικής ανάθεσης μνήμης στις συγκεκριμένες ανάγκες μίας πολυνηματικής εφαρμογής.

Συμβατικοί αναθέτες γενικού σκοπού

Στο μεταξύ αρκετή δουλειά γίνεται πάνω στους δυναμικούς αναθέτες μνήμης, οι οποίοι συνοδεύουν τα λειτουργικά συστήματα της εποχής: Για το BSD 4.2 έχουμε τον αναθέτη Kingsley [5], για συστήματα GNU/Linux την dlmalloc [19], όπως και τον αντίστοιχο αναθέτη για τα Windows XP [20]. Κάθε αναθέτης από αυτούς παρουσιάζει και μία διαφορετική αντιστάθμιση μεταξύ επιδόσεων και κατανάλωσης μνήμης όπως θα δούμε στην συνέχεια.

Ο αναθέτης Kingsley οργανώνεται σε ξεχωριστές λίστες με βάση δυνάμεις του δύο. Όλα τα αιτήματα για μνήμη στρογγυλοποιούνται στην πλησιέστερη δύναμη του δύο. Τα μπλοκ μνήμης που αποδεσμεύονται δεν μπορούν να συνδυαστούν μεταξύ τους ή να χωριστούν, ώστε να τακτοποιηθούν σε διαφορετικές λίστες από αυτές που θα μπορούσαν να φιλοξενήσουν το αρχικό αίτημα που ικανοποίησε το μπλοκ αυτό. Όπως είναι αναμενόμενο, ο αναθέτης αυτός είναι πάρα πολύ γρήγορος, με τίμημα όμως την ιδιαίτερα αυξημένη κατανάλωση μνήμης και τον κατακερματισμό της.

Η dlmalloc (ή αναθέτης Lea) είναι ένας αναθέτης best-fit (για περισσότερες λεπτομέρειες βλ. Υποενότητα 3.3.3), ο οποίος χειρίζεται τα αντικείμενα με διαφορετικό τρόπο ανάλογα με το μέγεθός τους. Τα μικρά αντικείμενα (μέχρι 64 bytes) τοποθετούνται σε διασυνδεδεμένες λίστες. Σε περίπτωση που έρθει αίτημα για μνήμη μεγαλύτερου μεγέθους (μέχρι 128 Kilobytes), τότε ο αναθέτης επιχειρεί να συγχωνεύσει μπλοκ από τις προηγούμενες λίστες. Σε αιτήματα για ακόμα μεγαλύτερο μέγεθος μνήμης βλέπουμε ότι ο αναθέτης αφήνει για τον διαχειριστή εικονικής μνήμης του συστήματος. Η dlmalloc αποτελεί έναν υποτυπώδη αναθέτη και ο τρόπος αυτός κατηγοριοποίησης θα δούμε ότι ακολουθείται από όλους σχεδόν τους σύγχρονους δυναμικούς αναθέτες μνήμης.

Ο αναθέτης, τέλος, των Windows XP βλέπουμε να χρησιμοποιεί και αυτός την πολιτική best-fit για αιτήματα μνήμης μέχρι και 1 Kilobyte. Τα συγκεκριμένα αιτήματα εξυπηρετούνται από 127

συνδεδεμένες λίστες, η καθεμία από τις οποίες εξυπηρετεί μεγέθη που διαφέρουν κατά 8 bytes από την προηγούμενη και είναι πολλαπλάσια του 8. Για αιτήματα μεγέθους μεγαλύτερου του 1 Kilobyte χρησιμοποιείται μία ακόμα συνδεδεμένη λίστα, ταξινομημένη κατά μέγεθος, η οποία χρησιμοποιεί την πολιτική good fit. Η σημαντικότερη διαφορά του αναθέτη αυτού από τους άλλους δύο αναθέτες είναι η υποστήριξη ατομικών λειτουργιών (atomic operations) αντί κλειδωμάτων συγχρονισμού (synchronization locks), σε μία εποχή όπου οι πολυπύρηνοι επεξεργαστές ξεκινούσαν την διείσδυσή τους στην αγορά των υπολογιστών.

Η εποχή αυτή κλείνει με την πεποίθηση ότι τα προβλήματα κατακερματισμού της μνήμης μπορούν τελικά να λυθούν με την χρήση ενός σωστά υλοποιημένου αναθέτη [21]. Μία ακόμα εργασία [22] παρουσιάζει την διαχείριση σωρού από την πλευρά του λειτουργικού συστήματος και των προγραμματιστών εφαρμογών.

Εξειδικευμένο υλισμικό και δυναμική ανάθεση μνήμης

Από την άλλη πλευρά, υπάρχουν αρκετές ερευνητικές εργασίες σχετιζόμενες με ανάθεση μνήμης και εξειδικευμένο υλισμικό, το οποίο χρησιμοποιείται για να επιταχύνει την πρώτη.

Στην [23] παρουσιάζεται μία υλοποίηση δυναμικού αναθέτη με υλισμικό, όπου το buddy system αντιστοιχίζεται πλήρως σε κυκλώματα προκειμένου να εκτελούνται δυναμικές αναθέσεις μέσα σε ένα κύκλο. Μία ακόμα μονάδα διαχείρισης μνήμης σε υλισμικό, αυτήν την φορά για πολυπύρηνους επεξεργαστές, παρουσιάζεται στην [24] με την ονομασία SoCDMMU. Σε κάθε περίπτωση, πρόκειται για μία κεντρική μονάδα, δηλαδή δεν μπορεί να διανεμηθεί σε επιμέρους μονάδες, γεγονός που την καθιστά πιθανό σημείο συμφόρησης σε πολυεπεξεργαστικά συστήματα. Επιπλέον, η SoCDMMU μπορεί να κατανείμει μόνο ολόκληρες, καθολικές σελίδες μνήμης και η διαχείριση της ανάθεσης μνήμης για τις τοπικές ή ιδιωτικές μνήμες επαφίεται στους επεξεργαστές. Τέλος, μία Μονάδα Διαχείρισης Μνήμης για αρχιτεκτονικές Δικτύου-σε-Ψηφίδα παρουσιάζεται στην [25] προσφέροντας ΔΔΜ γενικού σκοπού για κοινόχρηστη μνήμη με ελάχιστο μέγεθος πάλι ολόκληρες σελίδες μνήμης. Η τελευταία προσέγγιση μοιάζει αρκετά με μία προσπάθεια συγχρονισμού των λανθανουσών μνημών των επιμέρους επεξεργαστικών πυρήνων.

Τέλος, οι συγγραφείς στην [26] μελετούν τη διαχείριση σωρού στον επεξεργαστή Cell, μία παρεμφερής αρχιτεκτονική υλισμικού, στην οποία η προσπέλαση της κοινόχρηστης, καθολικής μνήμης δεν είναι άμεση από τις επιμέρους προγραμματιστικές μονάδες. Αντί αυτού, οι τελευταίες πρέπει να διαχειρίζονται μόνες τους τη δική τους, αποκλειστική μνήμη και επικοινωνούν με το σύστημα και μεταξύ τους με ρητές κλήσεις DMA. Η προσέγγιση αυτή εγείρει ζητήματα στη σωστή παραλληλοποίηση των εφαρμογών, αλλά εν τέλει ακολουθήθηκε αναγκαστικά λόγω του περιορισμού που θέτει η συγκεκριμένη πλατφόρμα. Οι συγγραφείς επανέρχονται στην συνέχεια με μία πιο ολοκληρωμένη υλοποίηση [27], [28], όπου εξηγούν τις αλλαγές που κάνουν στον με-

ταγλωττιστή προκειμένου τα δεδομένα των εφαρμογών και τα μεταδεδομένα του αναθέτη να τοποθετούνται με βέλτιστο τρόπο στην τοπική μνήμη.

Συγχρονισμός και δυναμική ανάθεση μνήμης

Λίγο προτού τα πολυπύρηνα συστήματα γενικού σκοπού διεισδύσουν στην αγορά, ερευνητές ασχολήθηκαν με το ζήτημα της παραλληλίας του δυναμικού αναθέτη μνήμης. Η Vmalloc [29] οργανώνει την μνήμη σε ξεχωριστές περιοχές μνήμης, ώστε διαφορετικές περιοχές να μπορούν να ακολουθούν διαφορετικές πολιτικές και μηχανισμούς μεταξύ τους. Το ίδιο μοτίβο, με τις περιοχές να εμφανίζονται ως υποσωροί (subheaps), βλέπουμε λίγο αργότερα στην LKmalloc [30]. Στην [31] ένας απλός, παράλληλος αναθέτης υλοποιείται, ο οποίος υπόσχεται καλή δυνατότητα κλιμάκωσης λόγω της απλότητάς του και της χρήσης περιοχών μνήμης.

Στις αρχές του 2000, το βάρος των δυναμικών αναθετών μνήμης μετατοπίζεται πλήρως στην υποστήριξη πολυεπεξεργαστικών συστημάτων. Ο Berger συνεχίζει να υποστηρίζει την χρήση των περιοχών για πολυεπεξεργαστικά συστήματα [32], ενώ πλέον ισχυρίζεται ότι εξειδικευμένοι δυναμικοί αναθέτες μνήμης σπάνια μπορούν να ξεπεράσουν καλογραμμένους και ώριμους αναθέτες γενικού σκοπού [33]. Η [34] προτείνει ένα τρόπο να μειωθεί το επιπλέον κόστος της δυναμικής ανάθεσης μνήμης με τη χρήση δομών δεδομένων χωρίς κλειδώματα (lock-free). Τέλος, ένας αναθέτης που ευνοεί την γειτονικότητα στην λανθάνουσα μνήμη σε συγκεκριμένα πολυεπεξεργαστικά συστήματα προτείνεται στην [35].

Κλείνοντας την υποενότητα των συστημάτων γενικού σκοπού, θα θέλαμε να αναφέρουμε δύο σύγχρονους αναθέτες που χρησιμοποιούνται ακόμα και σήμερα και που θα αναλύσουμε στην επόμενη υποενότητα. Ο πρώτος από αυτούς είναι ο Hoard [6], [32], ο οποίος συνδυάζει σωρούς ανά επεξεργαστή και μία καθολική, εξασφαλίζοντας έτσι οριοθέτηση στην κατανάλωση μνήμης, χαμηλό μέσο κατακερματισμό και χαμηλά κόστη συγχρονισμού. Ο δεύτερος ονομάζεται jemalloc [36], [37] και επιδιώκει να αποφύγει τον συγχρονισμό με την χρήση περαιτέρω, βοηθητικών δομών μέσα στην μνήμη και το «διάχυση» των δεδομένων πάνω στην μνήμη.

2.2.2 Για ενσωματωμένα συστήματα

Στα ενσωματωμένα συστήματα η μνήμη αποτελεί έναν πόρο που δε βρίσκεται σε αφθονία. Παράλληλα, αρκετά από αυτά έχουν απαιτήσεις εκτέλεσης σε πραγματικό χρόνο, άρα και ακριβή γνώση του χρόνου εκτέλεσης κατά τη διάρκεια σχεδιασμού, γεγονός που αποτρέπει την εισαγωγή μεθόδων με δυναμικό χρόνο εκτέλεσης και απρόβλεπτες συμπεριφορές. Η δυναμική ανάθεση μνήμης δεν έχει, λοιπόν, μεγάλη διείσδυση στον κόσμο των ενσωματωμένων συστημάτων, τουλάχιστον για συστήματα με μικρή επεξεργαστική ισχύ.

Μεθοδολογίες αντιστοίχισης στην μνήμη

Ίσως από τις πλησιέστερες έρευνες σε ενσωματωμένα συστήματα με την δυναμική ανάθεση μνήμης να αποτελεί η Μεθοδολογία Εξερεύνησης Μεταφοράς και Αποθήκευσης Δεδομένων (Data Transfer and Storage Exploration - DTSE) [38], η οποία αποφασίζει για μια εφαρμογή και τα δεδομένα της σε ποιες ιεραρχίες μνήμης μιας πλατφόρμας θα τοποθετηθούν και αν ο επεξεργαστής πρέπει να μεταφέρει τα δεδομένα σε κάποια χρονική φάση της εκτέλεσης του προγράμματος. Η μεθοδολογία αυτή μπορεί να παρομοιαστεί με έναν αναθέτη μνήμης, ο οποίος εκτελείται κατά τον σχεδιασμό μιας εφαρμογής και οι αποφάσεις του οποίου αξιολογούνται, ώστε στο τέλος να επιλεχθεί η καλύτερη λύση.

Από εκεί και πέρα, η δημιουργία δυναμικών διαχειριστών μνήμης ειδικών για συγκεκριμένη εφαρμογή με ιδιαίτερη έμφαση στην ενεργειακή απόδοση εξετάζεται στη [39] για ενσωματωμένα συστήματα. Σε αυτήν την εργασία μία αυτοματοποιημένη ροή εξερεύνησης δημιουργεί δυναμικούς διαχειριστές μνήμης με κύριο γνώμονα το ενεργειακό κέρδος. Ωστόσο, θα πρέπει να σημειωθεί ότι πρόκειται για αναθέτες μνήμης που προορίζονται για πλατφόρμες με ένα επεξεργαστή, οι οποίες εκτελούν μονονηματικές εφαρμογές. Αντίστοιχη έρευνα παρουσιάζεται και στην [40], όπου εξετάζονται διάφορα δένδρα αποφάσεων και οι επιπτώσεις που έχουν στην απόδοση και την κατανάλωση μνήμης ενός ενσωματωμένου συστήματος.

Η προαναφερθείσα μελέτη επεκτείνεται στην [18]. Οι συγγραφείς εκεί προσθέτουν τους πολυπύρηνους επεξεργαστές στην πλατφόρμα-στόχο της μεθοδολογίας και δίνουν τα καινούρια δένδρα αποφάσεων που προκύπτουν, όπως για παράδειγμα, τους μηχανισμούς συγχρονισμού που θα πρέπει να υποστηρίζει ο αναθέτης σε μια τέτοια πλατφόρμα. Κύριος στόχος είναι οι λύσεις ΔΔΜ να μετατραπούν από γενικού σκοπού σε εξειδικευμένων ανά εφαρμογή χωρίς να φύγουν από το επίπεδο του λογισμικού.

Μνήμες scratchpad και ιεραρχίες μνήμης

Ένα ακόμα πεδίο των ενσωματωμένων συστημάτων παρεμφερές με την δυναμική ανάθεση μνήμης είναι η μνήμη *scratchpad*. Καθώς σε ένα ενσωματωμένο σύστημα το κόστος (ενεργειακό και οικονομικό) και οι διαστάσεις της ψηφίδας συχνά δεν επιτρέπουν την χρήση μεγάλων λανθανουσών μνημών κοντά στον επεξεργαστή, οι σχεδιαστές χρησιμοποιούν εναλλακτικά μία μνήμη ανάμεσα στους επεξεργαστές και την μνήμη RAM του συστήματος που την ονομάζουμε scratchpad. Η μνήμη αυτή αντικαθιστά μερικώς την μνήμη RAM, δηλαδή δεδομένα μεταφέρονται σε αυτήν από την μνήμη RAM για να προσπελάζονται ταχύτερα από τον επεξεργαστή.

Κάποιες από της μεθοδολογίες που αναφέρθηκαν στην προηγούμενη υποενότητα [38], [40] συμπεριλαμβάνουν την χρήση τέτοιων μνημών, ενώ άλλες εργασίες βασίζονται κυρίως σε αυτή.

Πιο συγκεκριμένα, στην [41] ο συγγραφέας παροτρύνει την μεταφορά της σωρού σε μία μνήμη scratchpad. Με τροποποιήσεις στον μεταγλωττιστή (ανάλογες βλέπουμε και στην περίπτωση του Cell παραπάνω [27], [28]), γίνεται η σκιαγράφηση των εφαρμογών και αποφασίζεται κατά την διάρκεια της μεταγλώττισης ποιες μεταβλητές της σωρού πρόκειται να αντιγραφούν στην μνήμη scratchpad.

Αντίστοιχη προσπάθεια, με την συμβολή εξειδικευμένου υλισμικού, καταγράφεται και στην [42]. Οι συγγραφείς προτείνουν την χρήση ενός κυκλώματος, το οποίο θα χρησιμοποιείται για την σκιαγράφηση των περισσότερο χρησιμοποιημένων διευθύνσεων στην μνήμη. Κατόπιν, οι μεταβλητές που αντιστοιχούν στις διευθύνσεις αυτές, θα εντοπίζονται από τον μεταγλωττιστή και έπειτα από περαιτέρω σκιαγράφηση και με βάση την χρήση τους κατά την διάρκεια εκτέλεσης της εφαρμογής, θα αποφασίζεται ποιες μεταβλητές θα παραμένουν στην μνήμη scratchpad και ποιες θα αντικαθιστώνται.

Διαχείριση μνήμης με την βοήθεια υλισμικού

Με εξαίρεση την εργασία [42], οι προαναφερθέντες διαχειριστές δυναμικής μνήμης για ενσωματωμένα συστήματα δεν εκμεταλλεύονται την ύπαρξη κανενός μηχανισμού στο υλισμικό, ακόμα και ευνοϊκών για αυτών όπως η κλιμάκωση τάσης ή / και συχνότητας. Ειδικές, υλισμικές λύσεις ΔΔΜ μπορούν να επιτύχουν υψηλή επίδοση, αλλά οποιαδήποτε μικρή αλλαγή στις λειτουργίες τους οδηγεί στον επανασχεδιασμό ολόκληρης της μονάδας.

Η λύση που προτείνεται στη [43] προσπαθεί να το παρακάμψει αυτό παρέχοντας υπηρεσίες ΔΔΜ εξειδικευμένες ανά εφαρμογή πάνω σε κατανεμημένες πολυεπεξεργαστικές πλατφόρμες κοινόχρηστης μνήμης. Εντούτοις, οι υπηρεσίες αυτές απευθύνονται σε υλισμικό ειδικού σκοπού και έχουν υλοποιηθεί σε κώδικα χαμηλού επιπέδου. Η προσαρμοστικότητα που επιδιώκουμε να εισάγουμε κατά την εκτέλεση των εφαρμογών χρειάζεται αρχικά να εφαρμοστεί σε επίπεδο λογισμικού.

2.2.3 Σύγχρονες τάσεις

Στην παρούσα υποενότητα θα παρουσιάσουμε δύο σύγχρονους δυναμικούς αναθέτες μνήμης, τον Hoard και την jemalloc. Θα δούμε ξεχωριστά χαρακτηριστικά του καθενός και πώς συγκρίνεται γενικά η απόδοσή τους, τόσο σε επιδόσεις (χρόνο εκτέλεσης), όσο και σε κατανάλωση μνήμης.

Οι δύο αυτοί αναθέτες στηρίζονται σε δύο διαφορετικούς στόχους. Ο Hoard προσπαθεί να αποφύγει την «έκρηξη μνήμης» (memory blowup), ενώ η jemalloc θεωρεί ότι ο συγχρονισμός δεδομένων μεταξύ δύο ή παραπάνω πυρήνων κοστίζει πάρα πολύ για να εισάγεται πάντα σε μία

λειτουργία όπως η `malloc()`.

Ο αναθέτης Hoard

Ο αναθέτης Hoard προτάθηκε το 2000 από τον Emery Hoard [6]. Ο αναθέτης αυτός είναι ανοιχτού κώδικα[2], ενώ διατίθεται σε άδεια GPL, αλλά και για μη εμπορική χρήση. Υποστηρίζει τα περισσότερα γνωστά λειτουργικά συστήματα: GNU/Linux, Solaris, macOS και Windows.

Βασικές Ιδέες

Ο Hoard παρουσιάστηκε σε μία μεταβατική εποχή στα υπολογιστικά συστήματα και τα λειτουργικά συστήματα για αυτά, όπου οι πολυπύρηνες αρχιτεκτονικές αποκτούσαν ένα μεγάλο μερίδιο στην αγορά των συστημάτων γενικού σκοπού. Ο Hoard εντόπισε εκείνη την στιγμή τρία βασικά προβλήματα και προσπάθησε να τα αντιμετωπίσει με την υλοποίησή του:

Διαμάχη για Μνήμη Οι πολυνηματικές εφαρμογές συχνά δεν μπορούν να κλιμακωθούν, με την συμφόρηση στην σωρό να αποτελεί συχνό πρόβλημα. Όταν πολλαπλά νήματα της ίδιας εφαρμογής δεσμεύουν ή αποδεσμεύουν ταυτόχρονα μνήμη μέσω του αναθέτη, είναι πιθανό ο αναθέτης να εκτελέσει αυτά τα αιτήματα σειριακά. Ως αποτέλεσμα, τα προγράμματα που χρησιμοποιούν εκτενώς πολλούς πυρήνες και δυναμική μνήμη, στην πραγματικότητα επιβραδύνονται όσο οι πυρήνες που τους είναι διαθέσιμοι αυξάνονται. Ο Hoard προσπαθεί να περιορίσει την διαμάχη (contention) αυτή θέτοντας κατόχους ανά περιοχή μνήμης.

Ψευδής Κοινή Χρήση Οι αναθέτες μνήμης που παρέχονταν από τα λειτουργικά συστήματα εκείνη την εποχή μπορούσαν να προκαλέσουν προβλήματα σε πολυνηματικό κώδικα, τα οποία δεν εντοπίζονταν εύκολα. Πιο συγκεκριμένα, μπορούσαν να οδηγήσουν σε ένα φαινόμενο γνωστό ως «ψευδής κοινή χρήση» (false sharing): νήματα από διαφορετικούς επεξεργαστικούς πυρήνες κατέληγαν να έχουν μνήμη στην ίδια γραμμή κρυφής μνήμης (cache line) ή στην ίδια περιοχή μνήμης. Η προσπέλαση αυτών των ψευδώς κοινόχρηστων γραμμών κρυφής μνήμης είναι εκατοντάδες φορές πιο αργή από την προσπέλαση μη κοινόχρηστων. Ο Hoard φροντίζει να μην μοιράζεται η ίδια περιοχή μνήμης σε διαφορετικά νήματα, καθώς και τα δεδομένα διαφορετικών περιοχών να μην μπορούν να τοποθετηθούν στην ίδια γραμμή κρυφής μνήμης.

Έκρηξη μνήμης Ορισμένοι τύποι αναθετών παρουσιάζουν ένα συγκεκριμένο είδος κατακερματισμού το οποίο στον Hoard αποκαλείται *έκρηξη* (blowup). Συνειρμικά, η έκρηξη μνήμης είναι

[2] `http://www.hoard.org/`

η αύξηση στην κατανάλωση μνήμης που προκαλείται όταν ένας αναθέτης, παρόλο που αποδεσμεύει με επιτυχία μνήμη όταν η εφαρμογή του το ζητάει, δεν μπορεί να την επαναχρησιμοποιήσει για να καλύψει μελλοντικά αιτήματα για μνήμη. Ο Berger ορίζει ως έκρηξη την μέγιστη ποσότητα ανατειθέμενης μνήμης διαιρεμένη με την μέγιστη ποσότητα μνήμης που θα είχε αναθέσει ένας ιδανικός αναθέτης για μονούς επεξεργαστές [6].

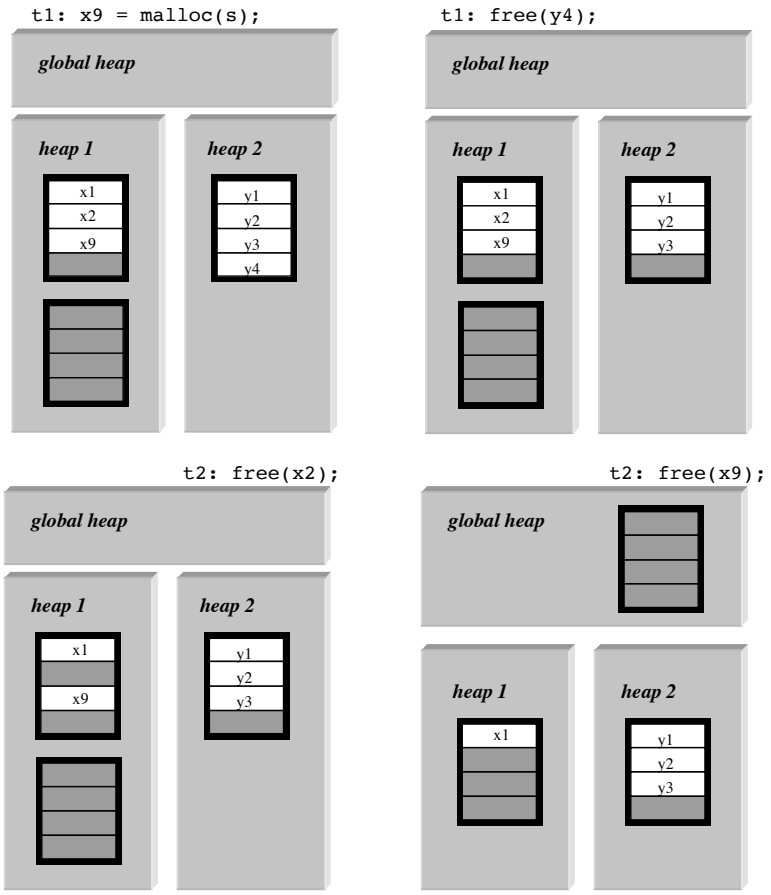
Ο συγγραφέας προσθέτει επίσης ότι οι πολυνηματικές εφαρμογές μπορούν επίσης να οδηγήσουν έναν αναθέτη στην έκρηξη που περιγράψαμε. Το φαινόμενο αυτό στην χειρότερη περίπτωση μπορεί να πολλαπλασιάσει την ποσότητα μνήμης που απαιτείται για την εκτέλεση της εφαρμογής με τον αριθμό των πυρήνων και επεξεργαστών, για παράδειγμα σε μία τετραπύρηνη πλατφόρμα μια εφαρμογή να ζητάει τέσσερις φορές περισσότερη μνήμη από ότι θα ζητούσε σε μία μονοπύρηνη. Ο Hoard προσπαθεί να λύσει το πρόβλημα αυτό με την χρήση μίας καθολικής σωρού, την οποία περιγράφουμε παρακάτω.

Οργάνωση Χώρου

Ο Hoard διατηρεί μία τουλάχιστον σωρό ανά επεξεργαστή, καθώς και μία καθολική σωρό. Κάθε επεξεργαστής έχει πρόσβαση στην δική του σωρό και στην καθολική. Η οργάνωση αυτή είχε προταθεί λίγο νωρίτερα, στον αναθέτη των Vee και Hsu [31]. Όταν η χρήση της σωρού ενός επεξεργαστή πέσει κάτω από ένα συγκεκριμένο ποσοστό, ο Hoard θα μεταφέρει ένα μεγάλο κομμάτι της μνήμης του στην καθολική σωρό, ώστε να είναι αυτό διαθέσιμο και σε άλλους επεξεργαστές. Τα κομμάτια αυτά είναι σταθερού μεγέθους και στον Hoard ονομάζονται superblocks.

Τα superblocks είναι κομμάτια (chunks) μνήμης, τα οποία ο Hoard δεσμεύει από το σύστημα. Κάθε superblock είναι ένας πίνακας κάποιου (μεταβλητού) αριθμού από επιμέρους μπλοκ (αντικείμενα), καθώς και μια επικεφαλίδα για να τα οργανώνει σε μία συνδεδεμένη λίστα. Τα μπλοκ διατάσσονται στην λίστα αυτή με LIFO (για περισσότερες λεπτομέρειες βλ. Υποενότητα 3.3.2) προκειμένου να βελτιωθεί η γειτονικότητα των δεδομένων (data locality). Τέλος, τα μπλοκ ενός superblock ανήκουν στην ίδια τάξη μεγέθους, δηλαδή όπως αναφέρεται παρακάτω, δεν μπορεί ένα μπλοκ να είναι υπερδιπλάσιο από ένα γειτονικό του στο ίδιο superblock.

Όλα τα superblocks έχουν σταθερό μέγεθος, το οποίο είναι πολλαπλάσιο του μεγέθους της σελίδας μνήμης του συστήματος. Τα αιτήματα για δυναμική μνήμη των εφαρμογών που είναι μεγαλύτερα από το μισό ενός superblock, αναθέτονται απευθείας από την `mmap()` και αποδεσμεύονται με την `munmap()`.



Σχήμα 2.2: Παράδειγμα χρήσης της καθολικής σωρού στον Hoard. Πηγή: [6]

Πολιτικές και Μηχανισμοί

Όπως αναφέρθηκε παραπάνω, τα μπλοκ μέσα σε ένα superblock είναι οργανωμένα σε μία λίστα LIFO. Επιπρόσθετα, στον Hoard όλες οι τάξεις μεγέθους των μπλοκ απέχουν μεταξύ τους κατά μία δύναμη ενός αριθμού x (με το x να είναι μεγαλύτερο από ένα). Όλα τα αιτήματα στρογγυλοποιούνται στην πλησιέστερη τάξη. Ο λόγος για τον οποίο ο συγγραφέας διάλεξε τις πολιτικές αυτές ήταν για να φράξει την χειρότερη περίπτωση εσωτερικού κατακερματισμού μέσα σε ένα μπλοκ. Ο κατακερματισμός αυτός, λοιπόν, δεν μπορεί να είναι μεγαλύτερος κατά παράγοντα x.

Ο Hoard παίρνει αντίστοιχα μέτρα για αποφυγή του εξωτερικού κατακερματισμού, που θα δούμε παρακάτω. Κομβικό σημείο αποτελεί το γεγονός ότι τα άδεια superblocks που καταλήγουν στην καθολική σωρό μπορούν να χρησιμοποιηθούν από την αρχή για οποιαδήποτε τάξη μεγέθους.

Φράζοντας την Έκρηξη Κάθε σωρός στον Hoard κατέχει μία σειρά από superblocks. Όταν δεν υπάρχει κάποιο διαθέσιμο superblock στην σωρό ενός επεξεργαστή, ο αναθέτης αποκτά ένα από την καθολική σωρό αν η τελευταία έχει διαθέσιμα. Αν η καθολική σωρός είναι επίσης άδεια, τότε ο Hoard δημιουργεί ένα νέο superblock ζητώντας εικονική μνήμη από το λειτουργικό σύστημα και το προσθέτει απευθείας στην σωρό του επεξεργαστή. Να σημειωθεί στο σημείο αυτό ότι ο Hoard δεν υποστηρίζει την επιστροφή άδειων superblocks στο σύστημα, σε αντίθεση με τα αιτήματα πολύ μεγάλου μεγέθους, τα οποία εξυπηρετούνται άμεσα από την `mmap()` για να αποδεσμευτούν στο τέλος από την `munmap()`. Αντίθετα, τα superblocks παραμένουν στην διαχείριση του Hoard, οποίος τα θέτει διαθέσιμα για επαναχρησιμοποίηση.

Για να μετακινηθεί ένα superblock από την σωρό ενός επεξεργαστή στην καθολική σωρό, ο Hoard χρησιμοποιεί δύο σταθερές, το κατώφλι κενότητας (emptiness threshold - f) και τον ελάχιστο αριθμό superblocks ανά σωρό (K). Για να μετακινηθεί, λοιπόν, ένα superblock στην καθολική σωρό θα πρέπει να ισχύουν τα εξής:

1. Η ωφέλιμη μνήμη της σωρού είναι μικρότερη από τον μη κενό $(1 - f)$ ποσοστό της μνήμης που έχει διαθέσει ο Hoard στην σωρό.

2. Η ωφέλιμη μνήμη της σωρού είναι μικρότερη από την μνήμη που έχει διαθέσει ο Hoard στην σωρό μείον το μέγεθος K superblocks.

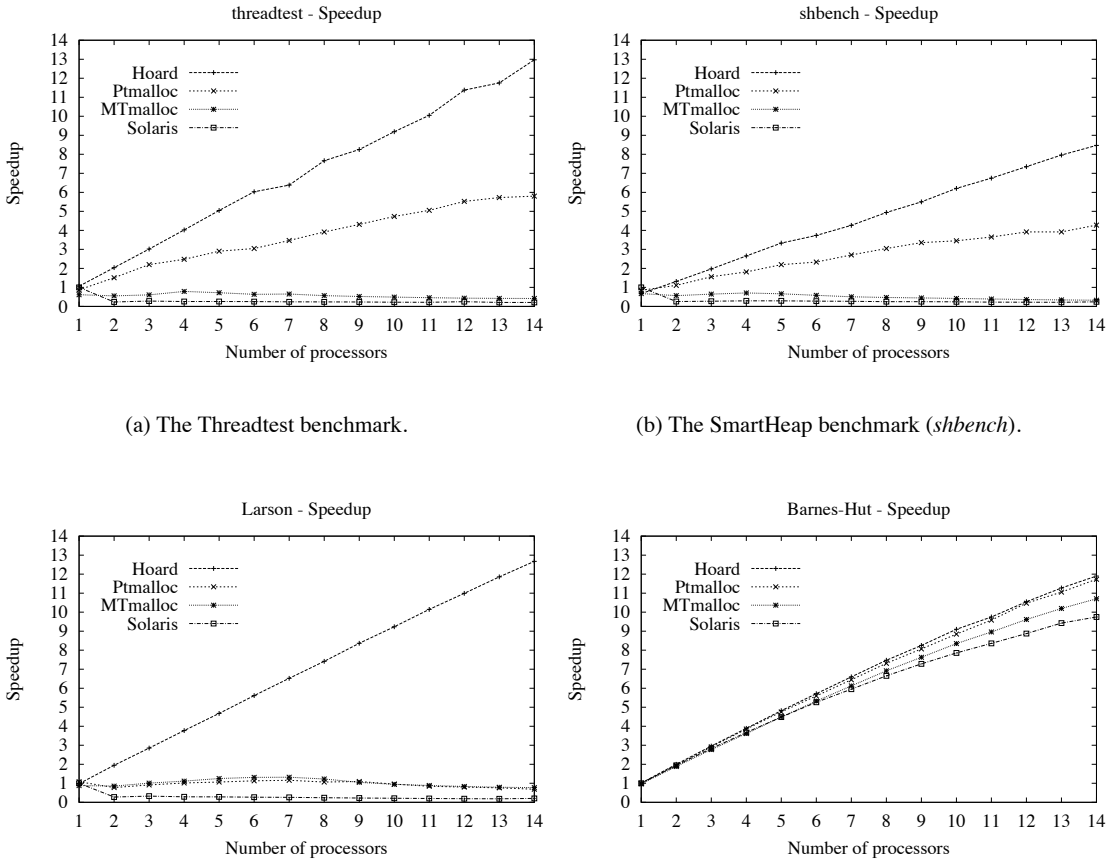
Με τους δύο αυτούς περιορισμούς ο Berger αποδεικνύει στην [6] ότι η έκρηξη μνήμης φράζεται σε ένα σταθερό παράγοντα.

Όσον αφορά τον τρόπο με τον οποίο ο Hoard βρίσκει τέτοια άδεια superblocks (δηλαδή κενά κατά f), αυτός γίνεται σε σταθερό χρόνο, αφού ο αναθέτης φροντίζει να χωρίσει τα superblocks σε διάφορα καλάθια (bins) τα οποία αποκαλεί ομάδες πληρότητας (fullness groups). Κάθε τέτοιο καλάθι περιέχει μία διπλά συνδεδεμένη λίστα superblocks που έχουν παρόμοια κενότητα. Ο αναθέτης μετακινεί τα superblocks από την μία ομάδα στην άλλη όταν χρειάζεται, και πάντα επιλέγει να ικανοποιήσει αιτήματα μνήμης από τα πιο γεμάτα superblocks. Για την βελτίωση της γειτονικότητας (locality), δίνεται προτεραιότητα στην λίστα (τοποθετείται μπροστά) για τα superblocks στα οποία μόλις απελευθερώθηκε ένα μπλοκ. Συνεπώς, για το επόμενο μπλοκ που πρόκειται να αναθέσουμε, είναι πολύ πιθανό να επαναχρησιμοποιήσουμε ένα superblock που βρίσκεται ήδη στην μνήμη. Εφόσον μάλιστα χρησιμοποιείται και διάταξη LIFO είναι επίσης πολύ πιθανό να επαναχρησιμοποιηθεί ένα μπλοκ που βρίσκεται ήδη στην κρυφή μνήμη του επεξεργαστή.

Αποφυγή Ψευδούς Κοινής Χρήσης Όπως εξηγήσαμε στις παραπάνω υποενότητες, ο Hoard χρησιμοποιεί ένα συνδυασμό από superblocks και πολλαπλές σωρούς. Το χαρακτηριστικό αυτό

από μόνο του κάνει τον αναθέτη να αποφεύγει τις περισσότερες περιπτώσεις ψευδούς κοινής χρήσης, τις οποίες ο συγγραφέας του ταξινομεί σε ενεργές και παθητικές. Ενεργώς, λοιπόν, ο αναθέτης αναθέτει μνήμη από διαφορετικά superblocks όταν παράλληλα πολλά νήματα τού ζητούν δυναμική μνήμη. Αντίστοιχα παθητικά, ο αναθέτης φροντίζει να επιστρέψει αποδεσμευμένα μπλοκ στο superblock στο οποίο άνηκαν αρχικά, με αποτέλεσμα μελλοντικές αναθέσεις να προστατεύονται όπως αναφέραμε ενεργητικά, αφού τα υπόλοιπα παράλληλα νήματα δεν μπορούν να ξαναχρησιμοποιήσουν τις γραμμές κρυφής μνήμης του μπλοκ αυτού.

Αξιολόγηση και Προβλήματα



(a) The Threadtest benchmark.

(b) The SmartHeap benchmark (*shbench*).

(c) Speedup using the Larson benchmark.

(d) Barnes-Hut speedup.

Σχήμα 2.3: Σύγκριση Hoard με αναθέτες παλαιότερης γενιάς. Πηγή: [6]

Τα αποτελέσματα που παρουσιάζονται στην [6] είναι αρκετά θεαματικά (Σχήμα 2.3). Σε περιπτώσεις κάποιων πολυνηματικών εφαρμογών ο Hoard επιτυγχάνει γραμμική επιτάχυνση όσο

αυξάνονται οι πυρήνες ενός συστήματος, σε μια εποχή που οι περισσότεροι αναθέτες αποτελούσαν εμπόδιο στην κλιμακωσιμότητα του συστήματος. Βλέπουμε επίσης, ότι ο κατακερματισμός της μνήμης διατηρείται σε λογικά πλαίσια στις περισσότερες εφαρμογές για συστήματα με πολλούς πυρήνες, χάρη κυρίως στους αλγορίθμους που έχει ο αναθέτης αυτός για την αποφυγή της λεγόμενης έκρηξης της μνήμης.

Για σημερινούς αναθέτες μνήμης οι διαφορές αυτές είναι αρκετά μικρότερες, καθώς πλέον οι περισσότεροι σύγχρονοι αναθέτες έχουν θέσει κύρια προτεραιότητα στην υποστήριξη πολυεπεξεργαστικών συστημάτων. Αντίστοιχες βελτιστοποιήσεις έχουν γίνει και στον Hoard, ο οποίος παραμένει υπό ενεργή ανάπτυξη, με τη διαφορά όμως ότι ο ίδιος εξακολουθεί να πιστεύει στην ανάγκη για αποφυγή της έκρηξης μνήμης, σε μια εποχή όπου η μνήμη των συστημάτων έχει γίνει αρκετή και οι σύγχρονοι αναθέτες το εκμεταλλεύονται και θυσιάζουν χώρο στον βωμό της ταχύτητας. Όπως θα δούμε στην Υποενότητα 2.2.3, ο Hoard χάνει αρκετά υπολογιστικό χρόνο στην διαχείριση της καθολικής σωρού και στον συγχρονισμό της, σε αντίθεση με άλλους αναθέτες, οι οποίοι αποφεύγουν με την σειρά τους να θέσουν ένα κοινό σημείο πρόσβασης για όλους τους επεξεργαστές και τα νήματα των εφαρμογών.

Ο αναθέτης jemalloc

Ο επόμενος αναθέτης που εξετάζεται είναι ο *jemalloc*. Δημιουργήθηκε από τον Jason Evans, ο οποίος πήρε τα αρχικά του και πρόσθεσε τη λέξη "malloc" για να φτιάξει το όνομα.Ίσως είναι από τους πιο διαδεδομένους αναθέτες που χρησιμοποιείται πέρα από τους αναθέτες που συμπεριλαμβάνονται σε λειτουργικά συστήματα: το Facebook στηρίζει ολόκληρη την υποδομή του. Από τις εκδόσεις FreeBSD 7.0 και NetBSD 5.0, η jemalloc αποτελεί την επίσημη υλοποίηση της `malloc()` σε αυτά τα λειτουργικά συστήματα. Ο συγγραφέας της εξηγεί αναλυτικά στην εργασία [36] τις αρχικές του ιδέες στην δυναμική ανάθεση μνήμης και στην [37] τις επεκτείνει προσαρμόζοντας τις στις ανάγκες εφαρμογών για εξυπηρετητές. Στις παρακάτω υποενότητες παρουσιάζονται συνοπτικά ως έχουν.

Βασικές ιδέες

Ο ίδιος ο συγγραφέας παραδέχεται ότι προκειμένου να διατηρηθεί υψηλή ταυτόχρονη απόδοση και αποφυγή του κατακερματισμού απαιτείται σημαντική εσωτερική πολυπλοκότητα. Η jemalloc συνδυάζει μερικές πρωτότυπες ιδέες με ένα σύνολο ιδεών που έχουν δοκιμαστεί σε άλλους αναθέτες. Ακολουθεί ένα μείγμα από αυτές τις ιδέες και την φιλοσοφία κατανομής, τα οποία συνδυάζονται για να σχηματίσουν την jemalloc:

- Διαχωρισμός μικρών αντικειμένων βάσει την κλάση μεγέθους και **προτίμηση των χαμη-**

λών διευθύνσεων κατά την επαναχρησιμοποίηση. Η πολιτική αυτή οργάνωσης χώρου προέρχεται από την phkmalloc [44], και είναι καθοριστική για την προβλέψιμη συμπεριφορά χαμηλού κατακερματισμού της jemalloc.

- **Προσεκτική επιλογή των κλάσεων μεγέθους** (επιλογή επηρεασμένη από το Vam [45]). Αν οι κλάσεις αντικειμένων έχουν μακρύ διάστημα μεταξύ τους, τότε τα αντικείμενα τείνουν να έχουν εκτενές αχρησιμοποίητο κενό διάστημα στο τέλος του (εσωτερικός κατακερματισμός). Καθώς αυξάνεται ο αριθμός των κλάσεων μεγέθους, θα υπάρχει αντίστοιχα μία τάση αύξησης της αχρησιμοποίητης μνήμης εξαιτίας των μεγεθών αντικειμένων που δεν χρησιμοποιούνται τόσο πολύ (εξωτερικός κατακερματισμός).

- **Επιβολή αυστηρών ορίων στο επιπλέον κόστος λόγω μεταδεδομένων του αναθέτη**. Αγνοώντας τον κατακερματισμό, η jemalloc περιορίζει τα μεταδεδομένα σε λιγότερο από το 2% της συνολικής χρήσης μνήμης, για όλες τις κλάσεις μεγέθους.

- **Ελαχιστοποίηση των ενεργών σελίδων μνήμης**. Οι πυρήνες των λειτουργικών συστημάτων διαχειρίζονται την εικονική μνήμη σε σελίδες (παλαιότερα καθεμία αποτελούνταν από 4 Kilobytes, πλέον σε συστήματα των 64 bits από 8), οπότε είναι σημαντικό να συγκεντρωθούν όλα τα δεδομένα σε όσο το δυνατόν λιγότερες σελίδες γίνεται. Η phkmalloc επιβεβαίωσε αυτήν την προσέγγιση, σε μια εποχή όπου οι εφαρμογές που χρησιμοποιούσαν πολλές ενεργές σελίδες συχνά τις αντάλλασσαν έξω από την κεντρική μνήμη, στον σκληρό δίσκο (swapping). Στη σημερινή εποχή με τις μεγάλες ποσότητες μνήμης RAM, έχει ακόμα αξία οι εφαρμογές να αποφεύγουν την ανταλλαγή αυτή.

- **Ελαχιστοποίηση συγχρονισμού δεδομένων**. Οι αρένες της jemalloc είναι εμπνευσμένες από την lkmalloc [30], αλλά με την εξέλιξη των πολυεπεξεργαστικών συστημάτων, η tcmalloc κατέστησε απολύτως σαφές ότι είναι ακόμα καλύτερο να αποφεύγεται ο συγχρονισμός δεδομένων το περισσότερο δυνατό, για αυτό και η jemalloc χρησιμοποιεί με λανθάνοντα τρόπο την δυναμική μνήμη σε κάθε νήμα ξεχωριστά (thread-specific caching).

- **Υποστήριξη αναθέσεων γενικού σκοπού**. Όταν η jemalloc ενσωματώθηκε για πρώτη φορά στο ΛΣ FreeBSD, είχε σοβαρά θέματα κατακερματισμού για κάποιες εφαρμογές και η πρόταση που είχε γίνει τότε ήταν να συμπεριληφθούν πολλαπλοί αναθέτες στο ΛΣ, με την έννοια ότι οι προγραμματιστές των εφαρμογών θα έπρεπε να έχουν την δυνατότητα να διαλέξουν τον αναθέτη με βάση τα χαρακτηριστικά της εφαρμογής που ανέπτυσσαν. Η σωστή λύση κατά τον Evans ήταν να απλοποιήσει δραματικά τους αλγορίθμους οργάνωσης χώρου της jemalloc, προκειμένου να βελτιστοποιήσει και την απόδοση και την προβλεψι-

μότητα του αναθέτη. Η επιλογή του αυτή φαίνεται να έχει αποδώσει, καθώς ο αναθέτης συνεχίζει να βελτιστοποιείται ακόμη και σήμερα με κριτήριο την απλότητα αυτή[3].

Οργάνωση Χώρου

Η jemalloc χωρίζει τα αιτήματα που δέχεται σε τρεις κύριες κατηγορίες μεγέθους, οι οποίες για ένα σύστημα 64 bits προεπιλέγονται ως εξής:

- Μικρά: [8], [16, 32, 48, ..., 128], [192, 256, 320, ..., 512], [768, 1024, 1280, ..., 3840]
- Μεγάλα: [4 KiB, 8 KiB, 12 KiB, ..., 4072 KiB]
- Τεράστια: [4 MiB, 8 MiB, 12 MiB, ...]

Η εικονική μνήμη χωρίζεται σε κομμάτια (chunks) μεγέθους 2^k, με τα 4 Megabytes να αποτελούν προεπιλογή. Εφόσον οι περιοχές αυτές είναι σταθερού μεγέθους, ο αναθέτης χρησιμοποιεί έξυπνα τεχνάσματα προκειμένου να προσπελάζει τα μεταδεδομένα όλων των αντικειμένων, των μικρών και μεγάλων σε σταθερό χρόνο με την χρήση δεικτών, ενώ των τεράστιων σε λογαριθμικό χρόνο χρησιμοποιώντας ένα red-black tree.

Τα κομμάτια αυτά, αν δεν πρόκειται για αίτημα τεράστιου μεγέθους, οργανώνονται σε αρένες. Μία αρένα μπορεί να έχει ένα ή παραπάνω κομμάτια. Κάθε κομμάτι με την σειρά του μπορεί να περιέχει μία ή παραπάνω σειρές από σελίδες μνήμης (page runs), μέσα στις οποίες αποθηκεύονται τα αντικείμενα. Η μνήμη που απελευθερώνεται επιστρέφει πάντοτε στην αρένα από την οποία προήλθε ανεξάρτητα από το ποιο νήμα εκτέλεσε την αποδέσμευση.

Μέσα σε κάθε σειρά σελίδων τακτοποιούνται τα αιτήματα με βάση το μέγεθός τους. Τα μικρά αιτήματα ομαδοποιούνται μαζί (βλ. τις αγκύλες στην αρχική λίστα), με επιπλέον μεταδεδομένα στην αρχή κάθε σειράς σελίδων. Αντίθετα, τα μεγάλα αιτήματα είναι ανεξάρτητα το ένα από το άλλο και τα μεταδεδομένα τους βρίσκονται αποκλειστικά στην επικεφαλίδα του κομματιού της αρένας.

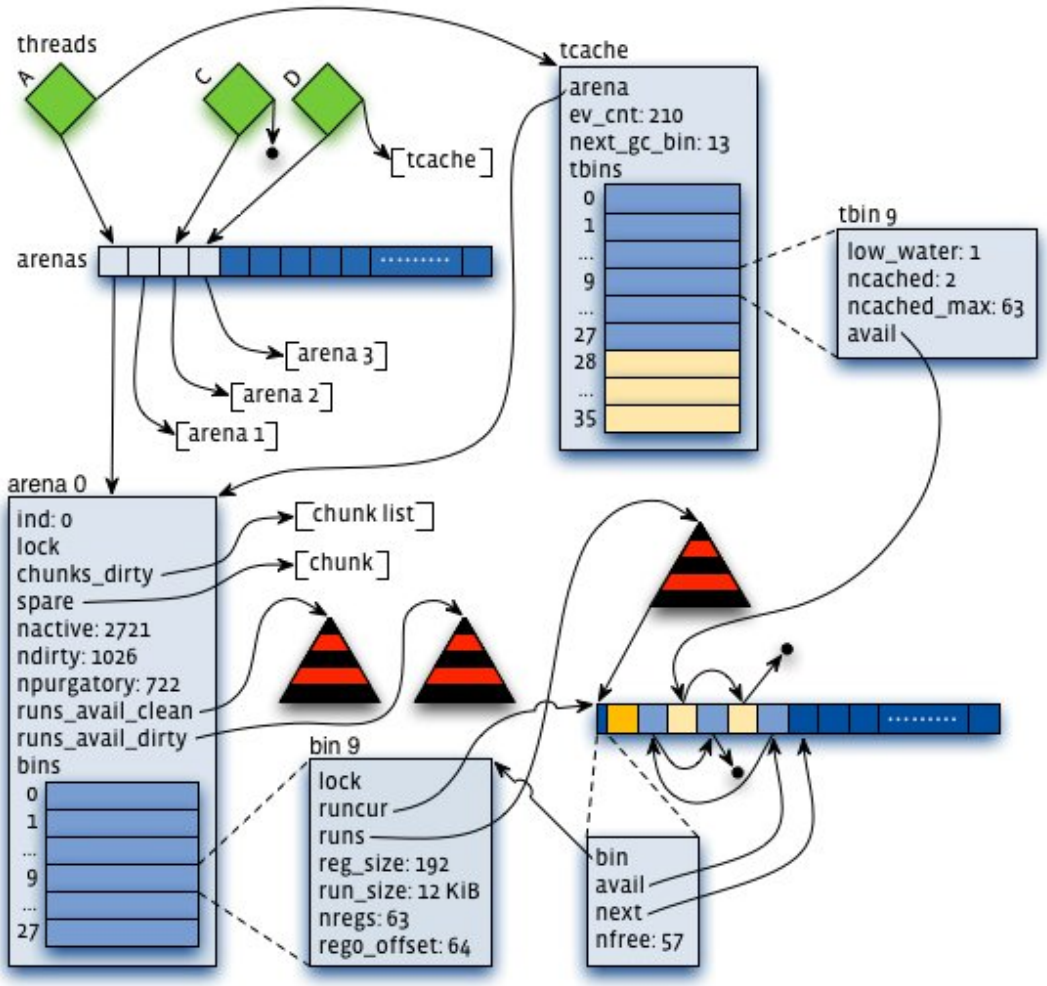
Οι αρένες είναι εντελώς ανεξάρτητες μεταξύ τους. Τα νήματα της εφαρμογής αντιστοιχίζονται σε αυτές εκ περιτροπής (round-robin), μόλις αυτά χρειαστούν να εκτελέσουν κάποιο μικρό ή μεγάλο αίτημα. Τα κομμάτια της αρένας μπορούν να ανήκουν μόνο σε μία αρένα.

Όπως προαναφέρθηκε, κάθε αρένα αποθηκεύει τα μεταδεδομένα της σε μία επικεφαλίδα. Η επικεφαλίδα αυτή περιέχει κυρίως μία δομή δεδομένων για να είναι η αρένα ενήμερη για τις σελίδες μνήμης που κατέχει. Επίσης, κάθε αρένα παρακολουθεί με red-black trees (ένα για κάθε μέγεθος) όσες σειρές σελίδων για μικρά αιτήματα που δεν είναι γεμάτες, και πάντα εξυπηρετεί αιτήματα ανάθεσης χρησιμοποιώντας αυτές ξεκινώντας από την χαμηλότερη διεύθυνση. Οι

[3] `https://github.com/jemalloc/jemalloc`

δομές στις οποίες ανήκουν τα τελευταία trees ονομάζονται καλάθια (bins). Τέλος, κάθε αρένα παρακολουθεί όλες τις διαθέσιμες σειρές σελίδων μέσω δύο ακόμα red-black trees, ένα για τις σειρές που δεν έχουν χρησιμοποιηθεί και ένα για τις χρησιμοποιημένες. Οι σειρές σελίδων αναθέτονται κατά προτίμηση από το tree των χρησιμοποιημένων, χρησιμοποιώντας την πολιτική best-fit (για περισσότερες πληροφορίες βλ. Υποενότητα 3.3.3).

Πολιτικές και Μηχανισμοί



Σχήμα 2.4: Η διάταξη αρένας και thread cache στην jemalloc. Πηγή: [37]

Κρυφή μνήμη ανά νήμα Ο σημαντικότερος μηχανισμός της jemalloc είναι η χρήση κρυφής μνήμης ανά νήμα (thread caching). Στο Σχήμα 2.4 βλέπουμε την διάταξη των σχετικών δομών.

Κάθε νήμα διατηρεί μία κρυφή μνήμη (cache) από μικρά αντικείμενα, όπως και για μεγάλα αντικείμενα μέχρι ένα συγκεκριμένο μέγεθος. Ως αποτέλεσμα, τα περισσότερα αιτήματα για ανάθεση μνήμης πρώτα ελέγχουν στην κρυφή μνήμη για διαθέσιμα αντικείμενα προτού προσπελάσουν μία αρένα. Η ανάθεση μέσω της κρυφής μνήμης αυτής δεν απαιτεί κάποιο συγχρονισμό, ενώ η αντίστοιχη ανάθεση μέσω μιας αρένας απαιτεί τον συγχρονισμό ενός καλαθιού της αρένας ή ακόμα και ολόκληρης της αρένας.

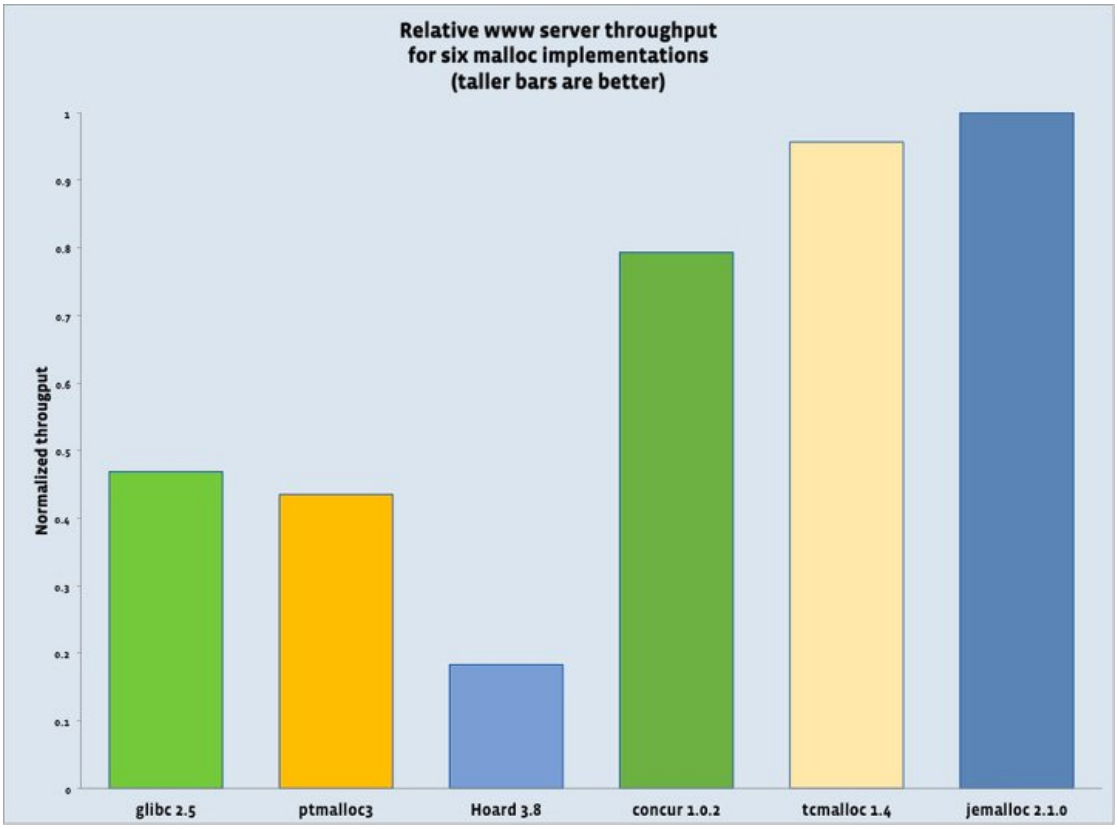
Ο κύριος στόχος των κρυφών μνημών ανά νήμα είναι να μειώσουν τον όγκο των γεγονότων συγχρονισμού. Ως εκ τούτου, ο μέγιστος αριθμός αντικειμένων για κάθε μέγεθος στην κρυφή μνήμη περιορίζεται σε ένα επίπεδο που επιτρέπει μία μείωση στον συγχρονισμό μίας με δύο τάξεων μεγέθους. Υψηλότερα όρια θα βοηθούσαν σε κάποιες εφαρμογές, με κόστος όμως τον επιπλέον κατακερματισμό της μνήμης. Για να αντιμετωπιστεί περαιτέρω ο κατακερματισμός, στην jemalloc εισάγονται έννοιες garbage collection, όπου οι κρυφές μνήμες ανά νήμα αδειάζουν από αντικείμενα αν αυτά δεν χρησιμοποιηθούν μέσα σε συγκεκριμένο αριθμό αιτημάτων που θα κάνει το νήμα.

Έμφαση στην Ταχύτητα Άλλοι μηχανισμοί και τεχνικές που χρησιμοποιήθηκαν στην jemalloc προκειμένου να αυξηθεί η ταχύτητά της και που αναφέρονται στην [37] είναι τα εξής:

- **Χρήση απλών δομών δεδομένων**. Ο συγγραφέας χρησιμοποίησε απλές δομές δεδομένων προκειμένου να αυξήσει την γειτονικότητα των δεδομένων (data locality) και να μειώσει το κρίσιμο μονοπάτι στα συντομότερα αιτήματα.
- **Αυξημένος βαθμός λεπτομέρειας συγχρονισμού**. Αντί καθεμία από τις κεντρικές συναρτήσεις (`malloc()`, `free()` κ.ά.) να έχει κλήσεις συγχρονισμού από την αρχή, ο συγχρονισμός γίνεται σε επίπεδο αρένας ή ακόμα και καλαθιού αρένας.
- **Προτίμηση της χρησιμοποιημένης μνήμης**. Διαχωρίζοντας τις σειρές μνήμης σε χρησιμοποιημένες και μη, καθώς και δίνοντας προτίμηση στις πρώτες, φάνηκε να έχει θετική επίδραση στην απόδοση του αναθέτη.
- **Νέα υλοποίηση red-black trees**. Στην τρέχουσα έκδοση της jemalloc, όπου αναφέρονται red-black trees πρόκειται για υλοποίηση non-recursive left-leaning 2-3 red-black tree αντί για left-leaning 2-3-4 red-black tree που υπήρχε αρχικά.

Αξιολόγηση και Προβλήματα

Στην [37] ο συγγραφέας δίνει και αποτελέσματα του αναθέτη του σε σύγκριση με άλλους αναθέτες (Σχήμα 2.5). Ο jemalloc φέρεται να έχει την καλύτερη απόδοση από όλους, με τους αντί-



Σχήμα 2.5: Σύγκριση jemalloc και άλλων αναθετών με βάση την απόδοση μίας εφαρμογής εξυπηρετητή διαδικτύου. Πηγή: [37]

στοιχους αναθέτες που συνοδεύουν τυπικά τις διανομές GNU/Linux (αυτόν της glibc) να έχει σχεδόν την μισή απόδοση. Αν και η κατανάλωση μνήμης δεν αναφέρεται από τον συγγραφέα, αναμένουμε η χρήση της μνήμης στις εκδόσεις της jemalloc και tcmalloc [46] που παρουσιάζονται να είναι αρκετά πιο υψηλές σε μέσο χρόνο, αλλά και οι μέγιστες τιμές τους.

Η αλήθεια είναι ότι από τότε που έγιναν οι μετρήσεις (2011), οι περισσότεροι από τους αναθέτες αυτούς έχουν αναβαθμιστεί και παρέχουν πλέον εφάμιλλη απόδοση με αυτήν της jemalloc. Ενδεικτικά αναφέρεται η εργασία [47] το 2015, η οποία δείχνει τον αναθέτη της glibc, τον Hoard και την jemalloc να έχουν παρόμοια αποτελέσματα σε χρόνο και απόδοση για μια σειρά από μετροπρογράμματα.

Στο σημείο αυτό θα θέλαμε να κλείσουμε αναφέροντας ότι η jemalloc λειτουργεί όπως ένας σύγχρονος αναθέτης μνήμης σε μοντέρνα συστήματα. Αντιλαμβάνεται ότι λόγω αφθονίας της μνήμης, η χρησιμοποίηση όσο το δυνατόν λιγότερων σελίδων δεν αποτελεί πλέον το σημαντικότερο κριτήριο σε έναν αναθέτη μνήμης για ένα συμβατικό υπολογιστικό σύστημα. Αντίθετα,

δίνεται έμφαση στην αποφυγή συγχρονισμού και στην εκμετάλλευση της γειτονικότητας των δεδομένων. Η αύξηση των επιδόσεων στις εφαρμογές ενδεχομένως να επιτυγχάνεται, αλλά αυτό γίνεται σε βάρος της χρησιμοποιούμενης μνήμης του συστήματος.

Πιστεύουμε ότι ο επιθετικός αυτός τρόπος δυναμικής ανάθεσης μνήμης είναι λειτουργικός στις περισσότερες περιπτώσεις υπολογιστικών συστημάτων σήμερα, χωρίς να τις καλύπτει όλες. Ειδικά στον κόσμο των ενσωματωμένων, ακόμα και αν πλέον μιλάμε για συστήματα με πολυπύρηνους επεξεργαστές και λειτουργικά συστήματα, η κατανάλωση μνήμης παίζει σπουδαίο ρόλο.

3 Το Πλαίσιο Δυναμικής Ανάθεσης Μνήμης dmmlib

Στο προηγούμενο κεφάλαιο αναφέρθηκαν οι σημαντικότερες εξελίξεις στους δυναμικούς αναθέτες μνήμης. Πέρα από την απευθείας σύγκριση του χρόνου εκτέλεσης ή της κατανάλωσης μνήμης, οι αναθέτες μπορεί να έχουν μεμονωμένα χαρακτηριστικά, τα οποία να είναι αξιόλογα, αλλά να χάνονται μέσα στην κάθε υλοποίηση.

Για αυτό το λόγο, στο κεφάλαιο αυτό προτείνουμε μία νέα βιβλιοθήκη δυναμικής ανάθεσης μνήμης, την *dmmlib*, η οποία λειτουργεί ως ένα ευρύτερο πλαίσιο ανάπτυξης δυναμικών αναθετών μνήμης. Η dmmlib προσφέρει ένα σύνολο πολιτικών και μηχανισμών προκειμένου μία εφαρμογή να μπορέσει να διαχειριστεί τη δυναμική της μνήμη. Οι διαφορετικές επιλογές ενεργοποιούνται από τους χρήστες κατά τη μεταγλώττιση και υπάρχει έλεγχος αν οι επιλογές που επιλέχτηκαν είναι συμβατές μεταξύ τους. Η dmmlib είναι επίσης πλήρως συμβατή με την τυπική ΠΔΕ της `malloc()`, αλλά ενσωματώνει και κάποιες συναρτήσεις, οι οποίες μπορούν να βελτιώσουν τη δυναμική ανάθεση μνήμης κατά την εκτέλεση των εφαρμογών. Μπορεί να χρησιμοποιηθεί από ένα ευρύ σύνολο διαφορετικών αρχιτεκτονικών, τόσο γενικού όσο ειδικού σκοπού, πάνω από διάφορα λειτουργικά συστήματα ή απευθείας στο υλικό.

3.1 Επισκόπηση της dmmlib

Η dmmlib έχει υλοποιηθεί στη γλώσσα προγραμματισμού C και κάνει χρήση του εργαλείου CMake[1] για τη ρύθμιση και μεταγλώττισή της. Ο κώδικάς της διατίθεται ελεύθερα στη διεύθυνση `https://git.microlab.ntua.gr/joko/dmmlib/` με άδεια χρήσης Apache 2.0[2]. Η βιβλιοθήκη αυτή προορίζεται τόσο για υπολογιστικά συστήματα με Unixοειδή λειτουργικά συστήματα, όσο για υπολογιστικά συστήματα που δε διαθέτουν κάποιο λειτουργικό σύστημα (baremetal).

Η γλώσσα C επιλέχτηκε ως μία γλώσσα προγραμματισμού συστημάτων, η οποία βρίσκεται πολύ κοντά στην γλώσσα μηχανής. Δύσκολα θα συναντήσει κανείς υπολογιστικά συστήματα που δε διαθέτουν μεταγλωττιστές για τη γλώσσα C. Η τρέχουσα υλοποίηση της dmmlib υποστηρίζει

[1] `https://cmake.org/`

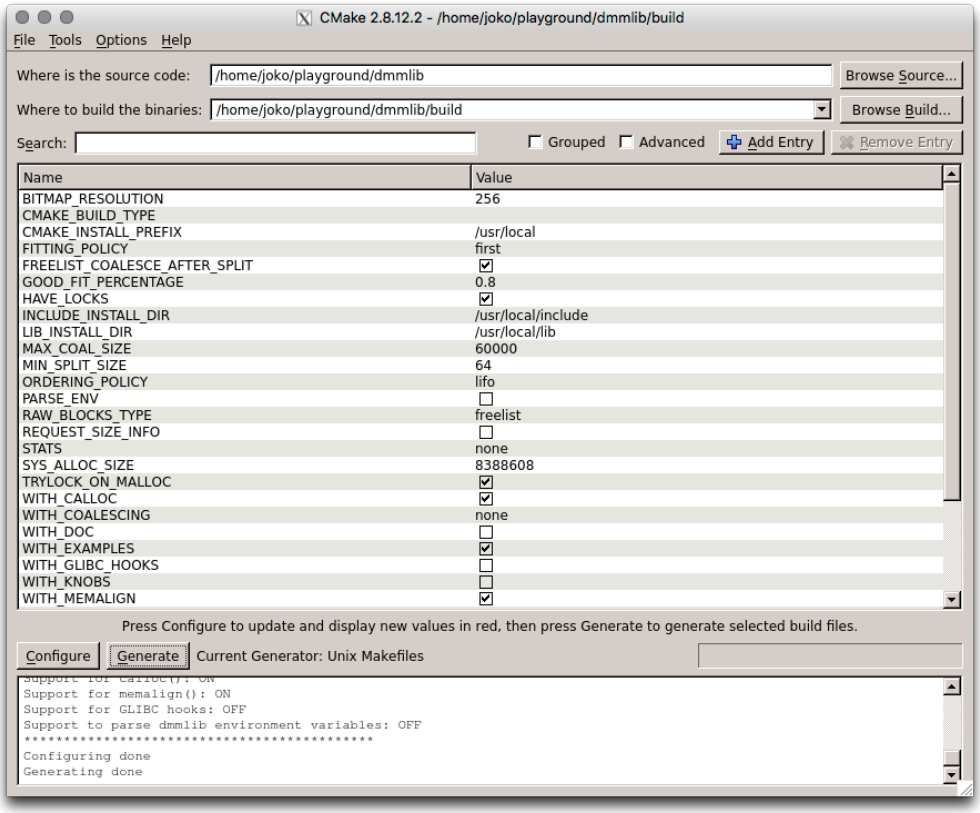
[2] `http://www.apache.org/licenses/LICENSE-2.0`

το πρότυπο C99. Το πιο πρόσφατο πρότυπο είναι το αμέσως επόμενο, το C11, το οποίο όμως δεν είναι εύκολο να το μεταφέρει σε παλαιότερα υπολογιστικά συστήματα, για αυτό και δεν επιλέχτηκε. Οι καινούριες λειτουργίες του προτύπου αυτού αντικαθιστώνται εν μέρει από τα POSIX Threads που συναντώνται σε Unixοειδή λειτουργικά συστήματα.

Όσον αφορά το εργαλείο CMake, αυτό χρησιμοποιείται για να διευκολυνθεί η παραγωγή διαφορετικών αναθετών και να γίνει ένας αυτοματοποιημένος έλεγχος των επιλογών που δίνουν οι χρήστες της βιβλιοθήκης κατά τη δημιουργία ενός νέου αναθέτη. Η ρύθμιση των επιλογών μπορεί να γίνει απευθείας με κείμενο (Παράδειγμα Κώδικα 1) ή με γραφικό τρόπο (Σχήμα 3.1). Με τη βοήθεια του CMake οι χρήστες ουσιαστικά φτιάχνουν με ασφαλή τρόπο ένα σύνολο από macro definitions, τα οποία ο C preprocessor θα χρησιμοποιήσει για να μεταγλωττίσει τα αρχεία κώδικα σε C της dmmlib.

Παράδειγμα Κώδικα 1: Ρύθμιση της dmmlib μέσα από το CMake

```
set(WITH_SYSTEM_CALLS "mmap" CACHE STRING "Choose what system calls can be \
  used, options are: none, sbrk, mmap")
set(SYS_ALLOC_SIZE 8388608 CACHE INTEGER "Choose the default system \
  allocation size")
option(HAVE_LOCKS "Build with POSIX locking mechanisms" ON)
option(TRYLOCK_ON_MALLOC "Use a trylock on malloc()" ON)
set(FITTING_POLICY "first" CACHE STRING "Choose the fitting policy in \
  freelist-organized raw blocks, options are: best, exact, first, good")
set(GOOD_FIT_PERCENTAGE 0.8 CACHE DOUBLE "Choose the good-fit percentage")
set(WITH_COALESCING "none" CACHE STRING "Build with coalescing support")
set(MAX_COAL_SIZE 60000 CACHE INTEGER "Maximum coalescing size")
set(WITH_SPLITTING "none" CACHE STRING "Build with splitting support")
set(MIN_SPLIT_SIZE 64 CACHE INTEGER "Minimum splitting size")
option(WITH_GLIBC_HOOKS "Place dmmlib functions on GLIBC hooks" OFF)
option(WITH_KNOBS "Build with knobs support" OFF)
option(FREELIST_COALESCE_AFTER_SPLIT "Try to coalesce blocks after split" ON)
set(STATS "none" CACHE STRING "Choose if the memory allocator keeps \
  internally statistics per raw block or globally, options are: none, global")
option(REQUEST_SIZE_INFO "Keep request size information in metadata" OFF)
option(WITH_MEM_TRACE "Support for memory traces" OFF)
option(WITH_STATS_TRACE "Support for statistics traces" OFF)
option(WITH_STATIC_LIB "Build a static library" OFF)
option(WITH_SHARED_LIB "Build a shared library" ON)
option(PARSE_ENV "Build with support to parse dmmlib environment \
  variables" OFF)
option(WITH_EXAMPLES "Build with examples" ON)
option(WITH_DOC "Build with documentation" OFF)
```



Σχήμα 3.1: Δημιουργία αναθέτη μέσα από γραφικό περιβάλλον

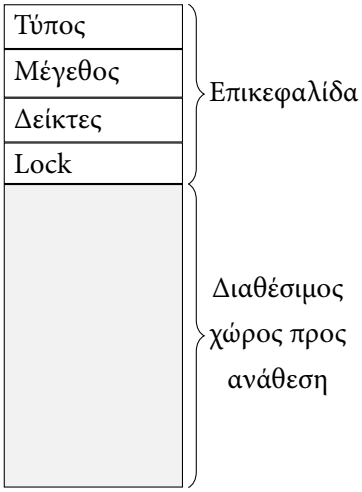
Σε κάθε περίπτωση ο χρήστης θα πρέπει να μεταβεί σε έναν κατάλογο εκτός του πηγαίου καταλόγου (μπορεί ωστόσο να είναι υποκατάλογος του τελευταίου), και να εκτελέσει την αντίστοιχη εντολή του CMake, όπως για παράδειγμα «`cmake -C../linux.preset ..`» στην περίπτωση που χρησιμοποιείται ένας υποκατάλογος.

Οι επιλογές της dmmlib εξετάζονται λεπτομερώς στις επόμενες ενότητες αυτού του κεφαλαίου. Σε περίπτωση που κάποια από τις ρυθμίσεις αντίκειται σε κάποια άλλη, το CMake φροντίζει να σταματήσει τη μεταγλώττιση του αναθέτη και να ειδοποιήσει τον χρήστη για το πρόβλημα. Για τη διευκόλυνση των χρηστών, η βιβλιοθήκη έρχεται με διάφορα σετ από προεπιλεγμένες ρυθμίσεις για λειτουργικά συστήματα Linux και macOS, καθώς και για την πλατφόρμα της STMicroelectronics, P2012 [15] (νυν STHORM). Τα σετ αυτά περιέχουν ρυθμίσεις που αποδίδουν περισσότερο συγκριτικά με άλλες σε γενικά φορτία εργασίας και έχουν συνήθως δοκιμαστεί περισσότερο κατά τη διάρκεια ανάπτυξης της βιβλιοθήκης.

3.2 Οργάνωση χώρου

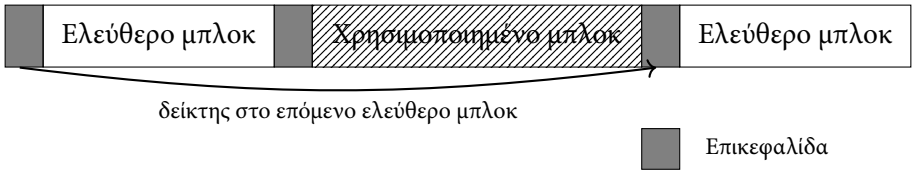
Στην οργάνωση χώρου ανήκουν οι σημαντικότερες αποφάσεις σχεδιασμού ενός αναθέτη. Η οργάνωση χώρου σε συμβατικούς αναθέτες είναι άρρηκτα συνδεδεμένη με τις πολιτικές και τους μηχανισμούς των αναθετών αυτών προκειμένου να υπάρξει βελτιστοποίηση στην εκτέλεση κώδικα από τον αναθέτη. Στην dmmlib θυσιάζουμε λίγη από τη βελτιστοποίηση αυτή προκειμένου να επιτρέψουμε μία πολυμορφία, η οποία θα μας επιτρέψει την ανάπτυξη διαφορετικών αλγορίθμων ανάθεσης μέσα στο κοινό πλαίσιο της βιβλιοθήκης μας.

Θεωρούμε ότι η οργάνωση χώρου οποιουδήποτε αναθέτη που έχει αναπτυχθεί στην dmmlib γίνεται τμηματικά σε περιοχές μνήμης που τις ονομάζουμε *ακατέργαστα μπλοκ (raw blocks)*. Οι περιοχές αυτές μπορούν να έχουν διαφορετική οργάνωση μεταξύ τους. Raw blocks διαφορετικού τύπου μπορούν να συνυπάρξουν: Μέσα στην dmmlib και μέσω των πολιτικών θα ορίζεται πότε και ποια raw blocks δημιουργούνται, καθώς και πότε αυτά επιστρέφονται στο σύστημα (βλ. Υποενότητα 3.3).



Σχήμα 3.2: Επικεφαλίδα σε ακατέργαστο μπλοκ

Η δομή της επικεφαλίδας ενός raw block φαίνεται στο Σχήμα 3.2. Διακρίνουμε ως πρώτο πεδίο τον *Τύπο*, όπου σημειώνεται ένας μαγικός αριθμός. Η τιμή αυτή είναι προφανώς διαφορετική για κάθε τύπο raw block. Ακολουθεί το *Μέγεθος* του μπλοκ, ώστε ο αναθέτης να γνωρίζει τα όρια του κάθε raw block. Έπειτα υπάρχουν οι *Δείκτες*, όπου αποθηκεύονται οι δείκτες άλλων raw blocks, πληροφορία που όπως θα δούμε χρησιμοποιείται στις πολιτικές επιλογής raw block (βλ. Ενότητα 3.3). Τέλος, αποθηκεύουμε στην επικεφαλίδα ένα *Lock*, ώστε να σηματοδοτούμε ότι το raw block χρησιμοποιείται από κάποιο νήμα (για περισσότερες λεπτομέρειες βλ. Υποενότητα 3.4.3). Ο υπόλοιπος χώρος είναι διαθέσιμος προς ανάθεση στις εφαρμογές, με εξαίρεση τον περαιτέρω



Σχήμα 3.3: Παράδειγμα διάταξης μπλοκ μέσα σε μπλοκ ελεύθερων λιστών

χώρο που χρησιμοποιεί ο αναθέτης για μεταδεδομένα και που είναι διαφορετικός ανά τύπο raw block.

Θα πρέπει να τονιστεί ότι ακόμα και ανεξάρτητα τύπου raw block τα περιεχόμενα των επικεφαλίδων μπορούν να διαφέρουν ανά αναθέτη. Για παράδειγμα, μπορεί ένας αναθέτης να έχει παραχθεί για μονονηματικά συστήματα και εφαρμογές, οπότε σε αυτήν την περίπτωση το πεδίο Lock παραλείπεται προκειμένου να εξοικονομηθεί ο χώρος στη μνήμη. Σε ένα τυπικό αναθέτη για πολυπύρηνο σύστημα 64 bits, το μέγεθος της επικεφαλίδας είναι 32 bytes, λαμβάνει δηλαδή χώρο για 4 πεδία των 64 bits.

3.2.1 Μπλοκ τύπου ελεύθερων λιστών

Σε raw blocks ελεύθερων λιστών δημιουργείται μία συνδεδεμένη λίστα (linked list), στην οποία αποθηκεύονται τα επιμέρους μπλοκ που είναι διαθέσιμα προς ανάθεση και που ανήκουν στο ίδιο raw block. Η σύνδεση μπορεί να είναι μονή, ή διπλή ή ακόμα και κυκλική ανάλογα με την πολιτική που έχει ρυθμιστεί να έχει ο αναθέτης. Η λίστα αποθηκεύεται στην αρχή του στα μεταδεδομένα των επιμέρους μπλοκ ως επικεφαλίδα. Τα υπόλοιπα πεδία της επικεφαλίδας κάθε μπλοκ φαίνονται στον Πίνακα 3.1 και προσομοιώνουν οργάνωση που συναντιέται στον αναθέτη του Doug Lea [19]:

Τα πεδία αυτά είναι σημαντικά για τις εσωτερικές πολιτικές των αναθετών που χρησιμοποιούν αυτό το είδος raw block, καθώς και για την εξέλιξη του κάθε raw block τύπου ελεύθερων λιστών. Στο Σχήμα 3.3 φαίνονται σε ένα παράδειγμα τρία μπλοκ όπως ακριβώς διατάσσονται στο χώρο της μνήμης. Πολιτικές όπως αυτή της διαίρεσης και συγχώνευσης (βλ. Υποενότητα 3.3.4) στηρίζονται σε πληροφορίες όπως η διαθεσιμότητα και το μέγεθος προηγούμενου ως προς τον χώρο μνήμης μπλοκ. Αντίστοιχα, οι πολιτικές διάταξης βασίζονται στους δείκτες της επικεφαλίδας για να θέσουν τη σειρά εκτίμησης των μπλοκ όταν γίνονται αιτήματα ανάθεσης.

Τέλος, αξίζει να σημειωθούν δύο τελευταία μεταδεδομένα που αποθηκεύονται στον διαθέσιμο χώρο ενός μπλοκ τύπου ελεύθερων λιστών, αυτό του δείκτη ορίου (border pointer) και του εναπομείναντος μεγέθους (remaining size). Και τα δύο αυτά μεταδεδομένα αποθηκεύονται στην αρχή του διαθέσιμου χώρου του μπλοκ, μαζί με τμήμα της λίστας των ελεύθερων μπλοκ. Ο

Πίνακας 3.1: Πεδία επικεφαλίδας επιμέρους μπλοκ σε μπλοκ τύπου ελεύθερων λιστών

Πεδίο	Τύπος	Περιγραφή
Διαθεσιμότητα	Υποχρεωτικό	Υποδηλώνει αν το μπλοκ είναι κατειλημμένο ή διαθέσιμο προς ανάθεση.
Μέγεθος	Υποχρεωτικό	Το μέγεθος που έχει το μπλοκ συμπεριλαμβανομένης της επικεφαλίδας του.
Αιτούμενο Μέγεθος	Προαιρετικό	Το μέγεθος που αιτήθηκε η εφαρμογή στον αναθέτη.
Διαθεσιμότητα Προηγούμενου Μπλοκ	Υποχρεωτικό	Η διαθεσιμότητα του προηγούμενου ως προς τον χώρο μνήμης μπλοκ.
Μέγεθος Προηγούμενου Μπλοκ	Υποχρεωτικό	Το μέγεθος του προηγούμενου ως προς τον χώρο μνήμης μπλοκ.
Δείκτης 1	Υποχρεωτικό	Ο δείκτης στο επόμενο μπλοκ που βρίσκεται στη λίστα ελεύθερων μπλοκ.
Δείκτης 2	Προαιρετικό	Επιμέρους δείκτης για μπλοκ που βρίσκεται στη λίστα ελεύθερων μπλοκ.

συνδυασμός των δύο αυτών μεταδεδομένων χρησιμοποιείται, όπως θα δούμε παρακάτω, στη δημιουργία νέων μπλοκ αν η λίστα ελεύθερων μπλοκ δεν έχει κάποιο διαθέσιμο, με την προϋπόθεση φυσικά ότι το raw block έχει ακόμα ελεύθερο χώρο εκτός των ήδη δημιουργημένων επιμέρους μπλοκ.

3.2.2 Μεγάλα μπλοκ

Υπάρχουν περιπτώσεις όπου τα προγράμματα ζητούν από τους αναθέτες μνήμης πολύ μεγάλα μεγέθη μνήμης. Σε αυτές τις περιπτώσεις, δεν υπάρχει νόημα ο αναθέτης να δημιουργήσει κάποιο συγκεκριμένο raw block και προτιμάται ο αναθέτης να απευθυνθεί απευθείας στο σύστημα για να πάρει χώρο στη μνήμη, χωρίς να δημιουργήσεις εσωτερικές δομές οργάνωσης στη μνήμη που λαμβάνει για να εξυπηρετήσει το αίτημα.

Τα μπλοκ αυτά που δημιουργούνται στη σωρό ονομάζονται μεγάλα μπλοκ (big blocks). Δεν περιέχουν άλλα μεταδεδομένα παρά μόνο μία επικεφαλίδα που περιέχει τον τύπο τους (μαγικός αριθμός) και το μέγεθός τους. Ο λόγος για τον οποίο γίνεται αυτό είναι όταν η εφαρμογή ζη-

τήσει από τον αναθέτη να αποδεσμεύσει τον χώρο της συγκεκριμένης ανάθεσης, να μπορέσει ο αναθέτης να προσδιορίσει τον ακριβή χώρο που χρειάζεται να επιστρέψει στο σύστημα.

3.3 Πολιτικές

3.3.1 Διαχείριση ακατέργαστων μπλοκ

Ανεξάρτητα από το είδος του raw block που υποστηρίζει ένας αναθέτης της dmmlib, υπάρχουν οι ίδιες συναρτήσεις εισόδου σε όλους. Προκειμένου να διατηρηθεί η συμβατότητα με άλλες υλοποιήσεις, η dmmlib παράγει αναθέτες με την ίδια ΠΔΕ των υπολοίπων. Έτσι συναντούμε συναρτήσεις όπως οι `malloc()`, `free()`, `realloc()` και `calloc()`. Μέσα σε αυτές υπάρχουν οι διαθέσιμες εσωτερικές συναρτήσεις, οι οποίες ενεργοποιούνται ή απενεργοποιούνται με τη βοήθεια του CMake και που εξετάζονται στις επόμενες Υποενότητες.

Στη εξωτερική συνάρτηση της `malloc()` γίνεται αρχικά η επιλογή του raw block που θα φιλοξενήσει το τρέχων αίτημα μνήμης. Το σύνολο των raw blocks βρίσκεται καταχωρημένο σε κάποια δομή δεδομένων μέσα στη γενικότερη δομή μεταδεδομένων των αναθετών, η οποία και αποθηκεύεται ως καθολική μεταβλητή του προγράμματος με το όνομα `tls_allocator`. Η δομή δεδομένων που κρατάει τα raw blocks μπορεί να είναι μία μονά ή διπλά συνδεδεμένη λίστα, ένας πίνακας συγκεκριμένων θέσεων με βάση το μέγεθος των μπλοκ που αποτελούν ένα raw block, ακόμα και υποθετικά ένα red-black tree, όπου αποθηκεύονται τα raw blocks με βάρος τα μεγέθη μπλοκ που δέχονται τα ίδια.

Αν η δομή αυτή δεν έχει κάποιο raw block που να μπορεί να εξυπηρετήσει το αίτημα για μνήμη ή βρισκόμαστε κατά την αρχικοποίηση, όπου δεν έχει δημιουργηθεί κάποιο raw block ακόμα, τότε ο αναθέτης δοκιμάζει να δημιουργήσει εκ νέου ένα raw block. Ο μηχανισμός με τον οποίο ο αναθέτης ζητάει και βρίσκει χώρο γίνεται με κάποια κλήση στο σύστημα και εξηγείται διεξοδικότερα στην Υποενότητα 3.4.1. Με το επιτυχές πέρας της διαδικασίας αυτής, το νέο raw block θα προστεθεί στην δομή που περιέχει τα υπόλοιπα και θα εξεταστεί αν μπορεί να ικανοποιήσει το αίτημα μνήμης που έχει γίνει.

Η εξέταση κάθε raw block διαφέρει ανά τύπο και περιγράφεται για τα raw blocks ελεύθερων λιστών στις δύο παρακάτω Υποενότητες, τις 3.3.2 και 3.3.3. Στην περίπτωση όμως που το μέγεθος του αιτήματος ξεπερνάει ένα συγκεκριμένο αριθμό, τότε χρησιμοποιείται ένα καινούριο μεγάλο μπλοκ.

Η εξωτερική συνάρτηση `free()` αναλαμβάνει αρχικά το έργο εύρεσης του raw block στο οποίο ανήκει ο δείκτης που εισήχθηκε ως όρισμα στη συνάρτηση. Εκμεταλλευόμενοι κάποιους μηχανισμούς ευθυγράμμισης δεδομένων (Υποενότητα 3.4.2) είμαστε σε θέση να εντοπίσουμε πολύ γρήγορα το raw block αυτό: το μέγεθος των raw blocks είναι σταθερός αριθμός, τον οποίο και

γνωρίζουμε ήδη στην μεταγλώττιση. Με λίγες αριθμητικές πράξεις μπορούμε, λοιπόν, εύκολα να προσδιορίσουμε την διεύθυνση μνήμης από την οποία ξεκινάει το raw block. Από εκεί και πέρα, μία εσωτερική συνάρτηση αναλαμβάνει να συνεχίσει το αίτημα της αποδέσμευσης, για παράδειγμα να εντάξει το απελευθερωμένο μπλοκ στην λίστα με τα ελεύθερα. Αν φυσικά ο τύπος του raw block που εντοπίστηκε είναι μεγάλο μπλοκ, τότε ο αναθέτης αναλαμβάνει να επιστρέψει το σύνολο της μνήμης του μεγάλου μπλοκ πίσω στο σύστημα.

Η εξωτερική συνάρτηση `realloc()` έχει να επιτελέσει ένα πολυπλοκότερο έργο από τις δύο προηγούμενες. Θα πρέπει αρχικά να εντοπιστεί το raw block στο οποίο ανήκει η διεύθυνση μνήμης που δηλώθηκε, ακριβώς όπως στην συνάρτηση `free()`. Στη συνέχεια, θα πρέπει να ασκηθεί ο εσωτερικός μηχανισμός που υπάρχει για τον κάθε τύπο raw block. Αν πρόκειται για raw block τύπου ελεύθερων λιστών, τότε ελέγχεται αν μπορεί και εξυπηρετεί το νέο μέγεθος να γίνει η συγχώνευση με το επόμενο στο χώρο μπλοκ. Αν η συγχώνευση αυτή είναι εφικτή, τότε ο αναθέτης την πραγματοποιεί και επιστρέφει στην εφαρμογή την ίδια διεύθυνση μνήμης. Αν η συγχώνευση δεν μπορεί να γίνει, τότε δημιουργείται ένα νέο μπλοκ (καλώντας εσωτερικά την `malloc()`), αντιγράφονται τα δεδομένα του αρχικού μπλοκ στο καινούριο, αποδεσμεύεται το παλιό μπλοκ και ο αναθέτης επιστρέφει την διεύθυνση του καινούριου. Να σημειωθεί, επίσης, ότι δεν είναι απαραίτητο το καινούριο μπλοκ να ανήκει στο ίδιο raw block με το παλιό αν και συνίσταται. Η αντίστοιχη, τελευταία διαδικασία γίνεται και στην περίπτωση των μεγάλων μπλοκ, όπου δεν μπορούμε να ζητήσουμε μνήμη από το σύστημα, η οποία να συνεχίζει απαραίτητα στο χώρο από το τέλος του συγκεκριμένου μεγάλου μπλοκ που ζητείται να αλλάξει μέγεθος. Για αυτόν τον λόγο, γίνεται εκ νέου ανάθεση, αντιγραφή και αποδέσμευση αντίστοιχα για μεγάλα μπλοκ.

Τέλος, η εξωτερική συνάρτηση `calloc()` εκτελεί εσωτερικά την συνάρτηση `malloc()`. Πιο συγκεκριμένα, καλείται η `malloc()` με ζητούμενο μέγεθος το γινόμενου του αριθμού των στοιχείων που ζητούνται με το μέγεθος του κάθε στοιχείου. Κατόπιν καλείται η συνάρτηση `memset()` προκειμένου όλα τα στοιχεία να αρχικοποιηθούν στην τιμή μηδέν.

3.3.2 Διάταξη μπλοκ σε μπλοκ τύπου ελεύθερων λιστών

Τα μπλοκ που βρίσκονται μέσα σε μπλοκ τύπου ελεύθερων λιστών μπορούν το τοποθετούνται στις λίστες των ελεύθερων μπλοκ με διάφορους τρόπους διάταξης. Η dmmlib υποστηρίζει τους εξής:

- **LIFO** Last-In-First-Out. Το μπλοκ που τοποθετήθηκε τελευταίο στην λίστα, εξετάζεται πρώτο στο επόμενο αίτημα ανάθεσης.
- **FIFO** First-In-First-Out. Το μπλοκ που τοποθετήθηκε πρώτο στην λίστα, εξετάζεται πρώτο στο επόμενο αίτημα ανάθεσης.

- **Κατά διεύθυνση** Το μπλοκ που έχει την μικρότερη διεύθυνση στην λίστα, εξετάζεται πρώτο στο επόμενο αίτημα ανάθεσης.
- **Κατά μέγεθος μπλοκ** Το μπλοκ που έχει το μεγαλύτερο μέγεθος στην λίστα, εξετάζεται πρώτο στο επόμενο αίτημα ανάθεσης.

Ήδη από την έρευνα του Wilson [5] το 1995, φάνηκε ότι η επιλογή της διάταξης δεν παίζει πολύ σημαντικό ρόλο στην απόδοση ενός αναθέτη. Ωστόσο, ύστερα από διάφορες μετρήσεις σε διάφορα μετροπρογράμματα, παρατηρήσαμε ότι κάποιες πολιτικές διάταξης δεν βοηθούν στην κατανάλωση μνήμης και προκαλούν προβλήματα κατακερματισμού του χώρου. Οι μετρήσεις αυτές στάθηκαν αφορμή για την περαιτέρω παραμετροποίηση της dmmlib και την δημιουργία των νέων πολιτικών που εμφανίζονται στο Κεφάλαιο 5.

3.3.3 Επιλογή μπλοκ σε μπλοκ τύπου ελεύθερων λιστών

Όπως έχει αντιληφθεί ο αναγνώστης στο σημείο αυτό, η επιλογή του κατάλληλου μπλοκ όταν πραγματοποιηθεί ένα αίτημα ανάθεσης, εξαρτάται από πολλές αποφάσεις, οι οποίες έχουν ήδη αποτυπωθεί στις παραπάνω υποενότητες. Απομένει μόνο το τελευταίο κομμάτι, εκείνο που εκτελείται αφού έχει ήδη επιλεχθεί το κατάλληλο raw block τύπου ελεύθερων λιστών.

Η dmmlib υποστηρίζει μέχρι στιγμής τις ακόλουθες πολιτικές επιλογής μπλοκ (fitting policies):

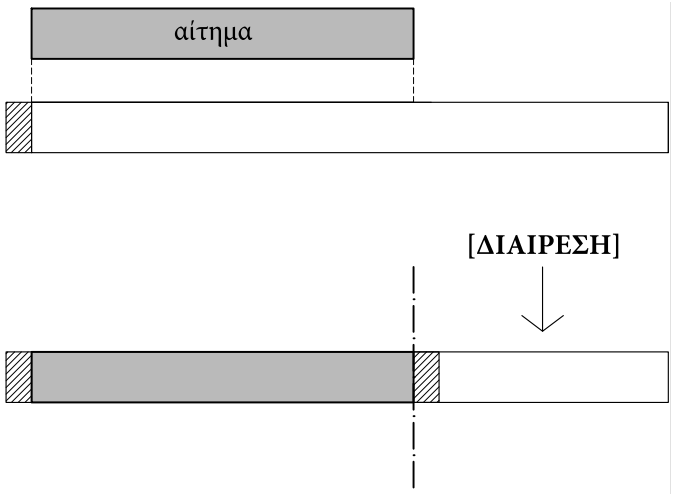
- **First-Fit**. Στην πολιτική αυτή, ο αναθέτης διαλέγει το πρώτο μπλοκ της λίστας που θα ταιριάζει στο αίτημα.
- **Best-Fit**. Στην πολιτική αυτή, ο αναθέτης εξετάζει ολόκληρη την λίστα συγκρατώντας το μπλοκ με το μέγεθος που ταιριάζει καλύτερα στο αίτημα, το οποίο και επιστρέφει στην εφαρμογή στο τέλος.
- **Exact-Fit**. Στην πολιτική αυτή, ο αναθέτης εξετάζει ολόκληρη την λίστα και επιστρέφει μπλοκ μόνο αν αυτό έχει ακριβώς το μέγεθος που έχει ζητήσει η εφαρμογή.
- **Good-Fit**. Παραπλήσια της Best-Fit, η πολιτική αυτή εξετάζει ολόκληρη την λίστα, αλλά μπορεί να σταματήσει αν το μέγεθος του μπλοκ που έχει βρει δεν ξεπερνάει πάνω από ένα συγκεκριμένο, σταθερό ποσοστό το μέγεθος που έχει ζητήσει η εφαρμογή.

3.3.4 Διαίρεση και συγχώνευση σε μπλοκ ελεύθερων λιστών

Μέχρι το σημείο αυτό έχουν περιγραφεί όλες οι γενικές ιδιότητες που θα πρέπει να έχει ένας λειτουργικός δυναμικός αναθέτης μνήμης. Στην dmmlib προσθέσαμε επίσης την δυνατότητα

διαίρεσης και συγχώνευσης των μπλοκ που υπάρχουν σε raw blocks τύπου ελεύθερων λιστών, εμπνευσμένοι από την αντίστοιχη υλοποίηση που υπάρχει στον αναθέτη του Doug Lea [19]. Ο κύριος λόγος για τον οποίο παρέχεται η δυνατότητα αυτή είναι ο κατακερματισμός. Παρατηρήσαμε ότι σε υπολογιστικά συστήματα με μικρή ποσότητα μνήμης, ακόμα και οι πιο συντηρητικές πολιτικές της dmmlib, όπως για παράδειγμα η πολιτική Best-Fit, οδηγούσαν τελικά το σύστημα σε καταστάσεις όπου η μνήμη δεν επαρκούσε, ειδικά όταν οι εφαρμογές που χρησιμοποιούσαν τους παραγόμενους αναθέτες είχαν δυναμικό φορτίο και μεγάλη διάρκεια εκτέλεσης.

Με τη διαίρεση και συγχώνευση δίνεται μία επιπλέον ευκαιρία σε μπλοκ ενός raw block τύπου ελεύθερων λιστών να εξυπηρετήσουν αιτήματα, εφόσον δημιουργούνται από τις ενέργειες αυτές νέα ελεύθερα μπλοκ.



Σχήμα 3.4: Η πολιτική της διαίρεσης σε μπλοκ ελεύθερων λιστών

Η διαδικασία της διαίρεσης λαμβάνει χώρα στην ανάθεση ενός ελεύθερου μπλοκ, το οποίο έχει μεγαλύτερο μέγεθος από αυτό που είχε ζητήσει η εφαρμογή. Ο μηχανισμός της διαίρεσης δημιουργεί ένα καινούριο μπλοκ από το χώρο που περισσεύει και το τοποθετεί στην λίστα με τα υπόλοιπα ελεύθερα μπλοκ. Στην dmmlib ο μηχανισμός της διαίρεσης ενεργοποιείται από την μεταβλητή `WITH_SPLITTING`, η οποία μπορεί να λάβει τις τιμές `never`, `always` και `fixed`. Στην πρώτη τιμή δεν ενεργοποιείται ποτέ ο μηχανισμός διαίρεσης (ούτε μεταγλωττίζεται ή ενσωματώνεται στον αναθέτη), ενώ στην δεύτερη ενεργοποιείται στην επιστροφή κάθε ανάθεσης που εξυπηρετείται από raw block τύπου ελεύθερων λιστών. Τέλος, στην τιμή `fixed` ο χρήστης θα πρέπει να χρησιμοποιήσει και την μεταβλητή `MIN_SPLIT_SIZE` για να δηλώσει ένα κατώφλι ενεργοποίησης: αν το μέγεθος του μπλοκ που προκύπτει είναι μικρότερο του κατωφλίου, τότε η διαίρεση δεν πραγματοποιείται.

Η διαδικασία της συγχώνευσης πραγματοποιείται όταν αποδεσμεύεται ένα μπλοκ που ανήκει σε raw block τύπου ελεύθερων λιστών. Στο γεγονός αυτό, προτού προστεθεί το μπλοκ αυτό στην λίστα των ελεύθερων μπλοκ, εξετάζεται αν τα διπλανά του μπλοκ από άποψη διάταξης μνήμης είναι ελεύθερα και αν είναι, τότε συμπτύσσονται όλα μαζί σε ένα καινούριο μεγαλύτερο ελεύθερο μπλοκ. Η σχετική μεταβλητή στο CMake ονομάζεται `WITH_COALESCING` και παίρνει τις ίδιες τιμές με την `WITH_SPLITTING`. Στην περίπτωση της τιμής `fixed`, χρειάζεται να τεθεί και η μεταβλητή `MAX_COAL_SIZE`, η οποία χρησιμοποιείται ως άνω κατώφλι: αν το μπλοκ που προκύπτει ξεπερνάει το μέγεθος αυτό, τότε η συγχώνευση δε θα γίνει.

Τέλος, η dmmlib παρέχει το μηχανισμό συγχώνευσης μετά από διαίρεση. Αν η μεταβλητή-διακόπτης `FREELIST_COALESCE_AFTER_SPLIT` είναι ενεργοποιημένη, τότε ο αναθέτης θα προσπαθήσει να συγχωνεύσει το ελεύθερο μπλοκ που προκύπτει μετά την διαίρεση με το αμέσως επόμενο στο χώρο μπλοκ, με την προϋπόθεση ότι το τελευταίο είναι ελεύθερο. Στην διαδικασία αυτή μετέχουν και τα δύο κατώφλια των πολιτικών αν έχουν αυτά τεθεί: Η διαίρεση δεν μπορεί να γίνει αν το ελεύθερο μπλοκ που προκύπτει από την διαίρεση είναι μικρότερο από το `MIN_SPLIT_SIZE`, ακόμα και αν μεγαλώσει μετά την συγχώνευση, ούτε αν το ελεύθερο μπλοκ που προκύπτει μετά την συγχώνευση είναι μεγαλύτερο του `MAX_COAL_SIZE`.

3.4 Μηχανισμοί

3.4.1 Κλήσεις στο Σύστημα

Προκειμένου να μπορέσει να λειτουργήσει ένας αναθέτης που παράγεται από την dmmlib, πρέπει αρχικά να γνωρίζει πώς μπορεί να αναλάβει μνήμη από το σύστημα. Διακρίνουμε δύο κύριες περιπτώσεις, την απευθείας ανάθεση μνήμης με δήλωση μέσα στο πρόγραμμα και τη χρήση κάποιας κλήσεις στο λειτουργικό σύστημα (system call).

Η πρώτη περίπτωση είναι η απλούστερη και δηλώνεται στο CMake θέτοντας τη μεταβλητή `WITH_SYSTEM_CALLS` ως `none`. Οι παραγόμενοι αναθέτες δε χρειάζονται κάποιο λειτουργικό σύστημα από πίσω τους για να τους δίνει χώρο στη μνήμη. Αντίθετα, θα πρέπει να δηλωθούν σε αυτούς δύο μεταβλητές, η αρχική διεύθυνση του χώρου μνήμης και το μέγεθός του, ώστε να αρχίσουν να χτίζουν μεταδεδομένα βάσει αυτού. Οι μεταβλητές αυτές μπορούν να δημιουργηθούν και να τεθούν τόσο στατικά (στο χρόνο μεταγλώττισης του αναθέτη), όσο και δυναμικά (μέσα από τον κώδικα της εφαρμογής).

Οι άλλες επιλογές για τη μεταβλητή `WITH_SYSTEM_CALLS` ξεκινούν με την `sbrk`. Πρόκειται για μία κλήση συστήματος που συναντάται σε παλιά Unixοειδή συστήματα και θεωρείται πλέον απαρχαιωμένη σε σύγχρονες εκδόσεις. Η `brk` και η `sbrk` χρησιμοποιούνται για να ελέγξουν το μέγεθος της μνήμης που έχει ανατεθεί από το σύστημα στο τμήμα δεδομένων μίας διεργασίας

και για να το επεκτείνουν αν αυτό χρειαστεί. Η σωρός δηλαδή της εφαρμογής εκτείνεται ουσιαστικά στο τέλος του τμήματος δεδομένων της εφαρμογής, με τις εντολές αυτές να μπορούν να το αυξομειώσουν. Η προσέγγιση ωστόσο αυτή θεωρείται πλέον παρωχημένη, τόσο από άποψη ασφαλείας (η σωρός μπορεί να φτάσει σε διευθύνσεις τη στοίβα αν μια εφαρμογή είναι απαιτητική σε μνήμη), όσο και από άποψη ευελιξίας (η σωρός μπορεί να χρειάζεται να βρίσκεται σε διαφορετικές διευθύνσεις μνήμης ανάλογα με το νήμα που χειρίζεται τα δεδομένα).

Για τους παραπάνω λόγους, προτείνεται η χρήση της `mmap` στην `WITH_SYSTEM_CALLS`, η οποία καλεί την ομώνυμη κλήση που μπορεί να δεσμεύσει και να αποδεσμεύσει πιθανά μη-συνεχόμενες περιοχές εικονικής μνήμης στον χώρο εικονικών διευθύνσεων της διεργασίας. Επιπλέον, η υλοποίηση της `mmap()` σε μοντέρνους πυρήνες λειτουργικών συστημάτων έχει πλέον εμπλουτιστεί με νέους τρόπους σήμανσης στον πυρήνα του πώς χρησιμοποιείται η μνήμη από τις εφαρμογές, ώστε να βελτιωθούν οι επιδόσεις και η κατανάλωση μνήμης. Μελλοντικές εκδόσεις της dmmlib θα μπορούσαν να χρησιμοποιήσουν τα νέα αυτά χαρακτηριστικά. Σε κάθε περίπτωση, η επιλογή της `sbrk` δίνεται ώστε η dmmlib να είναι σε θέση να παράξει αναθέτες για παλαιότερες εκδόσεις λειτουργικών συστημάτων και η επιλογή της `mmap` ώστε να δημιουργούνται αποδοτικοί αναθέτες σε σύγχρονα ΛΣ.

3.4.2 Ευθυγράμμιση Διευθύνσεων και Δεδομένων

Ένα από τα προβλήματα που συναντήσαμε ενώ αναπτύσσαμε την dmmlib ήταν η ευθυγράμμιση των διευθύνσεων και των δεδομένων σε αυτές. Στα αρχικά στάδια σχεδιασμού της dmmlib, οι αναθέτες δεν είχαν κάποιο περιορισμό στο μέγεθος της μνήμης που ζητούσαν από το σύστημα προκειμένου να ικανοποιήσουν τρέχοντα και μελλοντικά αιτήματα μνήμης της εφαρμογής που τους καλούσε. Ο κύριος λόγος για την αρχική αυτή απόφαση ήταν η περαιτέρω εξοικονόμηση χώρου μνήμης, ώστε οι εφαρμογές να μην παίρνουν πολύ επιπλέον χώρο, τον οποίο το σύστημα θα μπορούσε να τον διοχετεύσει σε άλλες εφαρμογές. Καθώς η dmmlib έπαιρνε χώρο στη μνήμη, η διεύθυνση μνήμης που έπαιρνε η βιβλιοθήκη σε επόμενη φορά ήταν ουσιαστικά η αμέσως επόμενη στο χώρο μνήμης. Ως εκ τούτου, πολλές από τις διευθύνσεις μνήμης δεν ήταν καν πολλαπλάσιες του αριθμού 2.

Ενώ λοιπόν στις αρχιτεκτονικές υπολογιστών x86 και x86-64 δε φαινόταν να υπάρχει πρόβλημα με αυτό, στην πλατφόρμα P2012, όπου χρησιμοποιείται αρχιτεκτονική της ARM, συναντήσαμε αρκετές δυσκολίες, όταν οι αναθέτες δε χρησιμοποιούσαν διευθύνσεις ευθυγραμμισμένες στα 32 bits. Τα προβλήματα αυτά απέτρεπαν σε ορισμένες περιπτώσεις την μεταγλώττιση των αναθετών, ενώ άλλες φορές η εκτέλεση των εφαρμογών είχε απρόβλεπτη συμπεριφορά. Για το λόγο αυτό, δημιουργήσαμε την αντίστοιχη επιλογή αρχικά στην dmmlib, όπου όλα τα μεγέθη μνήμης που ζητούνται από το σύστημα, καθώς και το μέγεθος των δομών των μεταδεδομένων

που χρησιμοποιούνται μέσα στη βιβλιοθήκη, να είναι πολλαπλάσια κάποιου αριθμού. Την επιλογή αυτή στο CMake την ονομάζουμε `DMM_DATA_ALIGNMENT` και δέχεται τις τιμές σε bytes (για παράδειγμα στην P2012 ορίζουμε 4 bytes).

Αλλά και στις άλλες πλατφόρμες συναντήσαμε τελικά προβλήματα εξαιτίας της απουσίας ευθυγράμμισης. Αν και η αρχιτεκτονική x86 δεν είχε προβλήματα με μη ευθυγραμμισμένες διευθύνσεις, παρατηρήσαμε ότι η ανάγνωση των δεδομένων από μη-ευθυγραμμισμένες διευθύνσεις ήταν αρκετά πιο αργή. Πιθανολογούμε ότι η λανθάνουσα μνήμη δεν είχε σωστά τοποθετημένα τα μεταδεδομένα που χρησιμοποιούσε ο αναθέτης, αλλά και τα δεδομένα που χρησιμοποιούσε η εφαρμογή, με αποτέλεσμα να χάνονται κύκλοι στη μεταφορά των δεδομένων από τον επεξεργαστή (λανθάνουσα μνήμη) και τη μνήμη RAM του συστήματος. Μόλις θέσαμε τη ρύθμιση `DMM_DATA_ALIGNMENT` να είναι 8 (τα περισσότερα πειράματά μας γίνονται πάνω σε συστήματα των 64 bits), η απόδοση των παραγομένων αναθετών είχε αισθητή βελτίωση.

Επίσης, η dmmlib μπορεί να λάβει υπόψη το μέγεθος της σελίδας μνήμης. Ο πυρήνας του λειτουργικού συστήματος χωρίζει τη μνήμη σε σελίδες συγκεκριμένου συνήθως μεγέθους (εκτός και αν του ζητηθούν πολύ μεγάλα μεγέθη, για τα οποία μπορεί να έχει εξειδικευμένους μηχανισμούς, βλ. Huge TLB). Σε μοντέρνα λειτουργικά συστήματα το μέγεθος αυτό ανέρχεται στα 8192 bytes, καθώς είναι εύκολο για τις σύγχρονες αρχιτεκτονικές επεξεργαστών και τις Μονάδες Διαχείρισης Μνήμης (ΜΔΜ) που διαθέτουν να διαχειρίζονται το εύρος αυτό ως εικονική μνήμη. Συνεπώς, τα περισσότερα σύγχρονα, συμβατικά λειτουργικά συστήματα δίνουν στις εφαρμογές χώρο εικονικής μνήμης χωρισμένο σε πολλαπλάσια των 8192 bytes. Αν, λοιπόν, ο αναθέτης που παράγει η dmmlib ζητήσει λίγο παραπάνω από 8192 bytes, τότε θα πάρει δύο σελίδες χωρίς να το ξέρει. Για αποφευχθεί μία τέτοια σπατάλη χώρου, φροντίζουμε να ορίσουμε κατά τη ρύθμιση του αναθέτη τη μεταβλητή `SYS_ALLOC_SIZE` ως μία τιμή πολλαπλάσια του μεγέθους της σελίδας.

3.4.3 Μηχανισμοί Συγχρονισμού

Οι μηχανισμοί συγχρονισμού επιτρέπουν στους αναθέτες της dmmlib να δημιουργούν σωρούς που με τη σειρά τους να μπορούν να χρησιμοποιηθούν από πολλαπλά νήματα.

Επειδή υπάρχει περίπτωση μία αρχιτεκτονική να είναι μονονηματική, ειδικά σε ενσωματωμένα συστήματα, αλλά και μία εφαρμογή αντίστοιχα να είναι γραμμένη χωρίς πολυνηματική χρήση, δημιουργήσαμε μία μεταβλητή-διακόπτη στο CMake με την ονομασία `HAVE_LOCKS`, ο οποίος προσθέτει στη ροή του παραγόμενου αναθέτη ελέγχους όταν ένα νήμα καλεί τον αναθέτη για ανάθεση ή αποδέσμευση χώρου. Οι έλεγχοι αυτοί πραγματοποιούνται προκειμένου άλλα νήματα να μην κάνουν αντίστοιχες διεργασίες στην ίδια περιοχή μνήμης. Η υλοποίηση που χρησιμοποιούμε στην dmmlib χρησιμοποιεί στην καλύτερη περίπτωση ατομικές εντολές (atomic instructions) και πιο συγκεκριμένα την εντολή Compare-and-swap. Αν τυχόν δεν υποστηρίζεται η εντολή αυτή

εγγενώς από το σύστημα, ο μεταγλωττιστής θα φροντίσει να αντικαταστήσει τον κώδικα με μία αντίστοιχη εξωτερική συνάρτηση, η οποία προφανώς θα έχει μεγαλύτερο κόστος από άποψη χρόνου.

Τέλος, στους μηχανισμούς συγχρονισμού της dmmlib έχουμε εισάγει και την έννοια των trylocks με τη μεταβλητή-διακόπτη του CMake `TRYLOCK_ON_MALLOC`. Σύμφωνα, λοιπόν, με την επιλογή αυτή, ένας αναθέτης θα προσπαθεί να αποκτήσει το lock της περιοχής μνήμης που βρίσκεται υπό εξέταση και αν αυτό είναι στην κατοχή άλλου νήματος, ο αναθέτης θα προχωράει αμέσως στην εξέταση της επόμενης περιοχής. Ο μηχανισμός αυτός δίνεται για την περαιτέρω επιτάχυνση των αναθετών που χρησιμοποιούνται σε πολυνηματικές εφαρμογές.

3.4.4 Στατιστικά

Η dmmlib μπορεί επίσης να παρέχει στατιστικά σχετικά με τη δυναμική μνήμη σε προγραμματιστές εφαρμογών. Επειδή τα στατιστικά αυτά μεγαλώνουν το μέγεθος του κώδικα και επιβαρύνουν την εκτέλεση των εφαρμογών, προτείνεται να χρησιμοποιούνται οι δυνατότητες αυτές μόνο κατά την ανάλυση (profiling) των εφαρμογών από την βιβλιοθήκη.

Η μεταβλητή `STATS` στο CMake καθορίζει την υποστήριξη στατιστικών στους αναθέτες της dmmlib. Μπορεί να δεχτεί τις εξής επιλογές: `none` και `global`. Η πρώτη απενεργοποιεί τα στατιστικά εντελώς από τον παραγόμενο αναθέτη, ενώ η δεύτερη δημιουργεί τις κατάλληλες δομές μεταδεδομένων, ώστε να αποθηκεύσει κατά την εκτέλεση διάφορα στατιστικά. Από αυτά τα σημαντικότερα είναι:

- το μέγεθος συνολικής μνήμης που έχει δεσμεύσει ο αναθέτης από το σύστημα
- το μέγεθος συνολικής μνήμης που η εφαρμογή έχει ζητήσει από τον αναθέτη (διαθέσιμο μόνο με την επιλογή `REQUEST_SIZE_INFO` του CMake και όταν χρησιμοποιούνται μπλοκ ελεύθερων λιστών)
- ο αριθμός των αντικειμένων που ο αναθέτης διατηρεί για την εφαρμογή
- ο αριθμός των κλήσεων `malloc()`
- ο αριθμός των κλήσεων `free()`
- ο αριθμός των κλήσεων `realloc()`

Τα στατιστικά αυτά είναι διαθέσιμα οποιαδήποτε στιγμή στην εφαρμογή που χρησιμοποιεί τον αναθέτη και οι προγραμματιστές μπορούν να τα προσπελάσουν μέσω της καθολικής μεταβλητής `tls_allocator` που εισάγει στο πρόγραμμα ο αναθέτης ως τοποθεσία της κεντρικής ιεραρχίας

των μεταδεδομένων του. Η συνάρτηση `print_stats()` μπορεί να χρησιμοποιηθεί εναλλακτικά για την εμφάνιση των στατιστικών αυτών στην οθόνη ή σε αρχείο συστήματος.

Μία άλλη δυνατότητα παρεμφερής των στατιστικών αποτελεί η δυνατότητα δημιουργίας ιχνών. Ενεργοποιώντας την μεταβλητή-διακόπτη `WITH_MEM_TRACE` στο CMake, μπορεί να δει κάποιος τη σειρά με την οποία ζητούνται από τον αναθέτη χώροι στη μνήμη, το μέγεθος που ζητάει η εφαρμογή και την διεύθυνση που επιστρέφει ο αναθέτης.

Τη λειτουργία των στατιστικών και των ιχνών την ολοκληρώνουν μία ομάδα δεσμών ενεργειών (scripts) τα οποία δημιουργούν γραφήματα από τα στατιστικά και ίχνη. Η υλοποίησή τους είναι σε Python 2 και χρησιμοποιούν τη βιβλιοθήκη κατασκευής διαγραμμάτων matplotlib [3]. Τα γραφήματα που απεικονίζονται στην ενότητα 5.3.1 στηρίζονται σε αυτά.

3.4.5 Επαναρρύθμιση κατά την εκτέλεση

Οι αναθέτες που εξετάσαμε στο κεφάλαιο 2 χαρακτηρίζονται ως «γενικού σκοπού» και οι σημαντικότερες ρυθμίσεις τους γίνονται κατά τη διάρκεια μεταγλώττισής τους. Εξαίρεση αποτελεί ένα σύνολο ρυθμιζόμενων παραμέτρων που η GLIBC malloc() εξάγει μέσω της `mallopt()`. Οι παράμετροι αυτές είναι γενικές και αφορούν κυρίως την διεπαφή του αναθέτη με το σύστημα: πώς και πότε θα καλεί την `mmap()` και αντίστοιχα πώς και πότε θα επιστρέφει μνήμη στο σύστημα.

Στην dmmlib σχεδιάσαμε ένα αντίστοιχο σύστημα, το οποίο μπορεί να επιδρά ακόμα πιο εσωτερικά στους μηχανισμούς του αναθέτη και να τους τροποποιεί κατά τη διάρκεια εκτέλεσης μιας εφαρμογής που χρησιμοποιεί έναν αναθέτη της dmmlib. Ενεργοποιώντας τη μεταβλητή του CMake `WITH_KNOBS`, μία σειρά από παραμέτρους κατωφλίων αποθηκεύεται στη μεταβλητή μεταδεδομένων `tls_allocator` και δίνεται η δυνατότητα στην εφαρμογή να αλλάξει δυναμικά τις πολιτικές του αναθέτη της. Για παράδειγμα, μπορεί κατά την εκτέλεσή της να τροποποιήσει τα μεγέθη διαίρεσης και συγχώνευσης μπλοκ μνήμης, ώστε να επιτρέπει ή να τα περιορίζει περισσότερο και να αλλάξει έτσι εντελώς τη συμπεριφορά του αναθέτη. Περισσότερες λεπτομέρειες του μηχανισμού αυτού παρουσιάζονται στην Ενότητα 5.3, όπου ο μηχανισμός αυτός χρησιμοποιείται για να δημιουργηθεί μία νέα προσαρμοστική πολιτική ανάθεσης.

3.5 Χρήση παραγόμενων αναθετών

Οι αναθέτες που παράγονται από την dmmlib μπορούν να χρησιμοποιηθούν απευθείας από τις εφαρμογές, είτε ως στατικές, είτε ως δυναμικές βιβλιοθήκες. Οι προγραμματιστές δε χρειάζεται να αλλάξουν τον κώδικα των εφαρμογών τους πέρα από την υπόδειξη ότι χρησιμοποιείται η

[3] `http://matplotlib.org/`

dmmlib. Οι κλήσεις `malloc()`, `free()`, `realloc()` κ.ο.κ. παραμένουν ως έχουν. Προαιρετικά, οι προγραμματιστές μπορούν να προσθέσουν κώδικα για να χρησιμοποιήσουν κάποιο επιπρόσθετο χαρακτηριστικό της dmmlib, όπως τα στατιστικά (Υποενότητα 3.4.4) ή την επαναρρύθμιση κατά την εκτέλεση (Υποενότητα 3.4.5).

Στην περίπτωση της στατικής βιβλιοθήκης, οι προγραμματιστές μπορούν να ενεργοποιήσουν τη μεταβλητή `WITH_STATIC_LIB` και να ενσωματώσουν απευθείας τον εκτελέσιμο κώδικα του αναθέτη στην εφαρμογή τους. Για παράδειγμα, αν χρησιμοποιούν τον μεταγλωττιστή gcc, μπορούν να εκτελέσουν την ακόλουθη εντολή:

```
gcc -I{DMMLIB Source Directory}/include \
  -I{DMMLIB Build Directory} \
  {DMMLIB Build Directory}/libdmm_static.a \
  yourapp.c -o yourapp
```

Η χρήση της δυναμικής έκδοσης είναι ευκολότερη και συνίσταται έναντι της στατικής. Η αντίστοιχη μεταβλητή του CMake ονομάζεται `WITH_SHARED_LIB` και δεν αποκλείει την πρώτη, δηλαδή μπορεί να φτιαχτούν δύο εκδόσεις του αναθέτη παράλληλα και για τις δύο χρήσεις. Η αντίστοιχη εντολή του gcc για τη χρήση της δυναμικής έκδοσης είναι:

```
gcc -I{DMMLIB Source Directory}/include \
-I{DMMLIB Build Directory} -ldmm yourapp.c -o yourapp
```

Σε Unixοειδή λειτουργικά συστήματα μπορεί εναλλακτικά να χρησιμοποιηθεί ο μηχανισμός `LD_PRELOAD`. Εφόσον δε χρησιμοποιείται κάποια συνάρτηση της Προγραμματιστικής Διεπαφής Εφαρμογής (ΠΔΕ) της dmmlib που δεν υπάρχει σε συμβατικούς αναθέτες μνήμης, τότε οι εφαρμογές μπορούν να μεταγλωττιστούν χωρίς τη χρήση ή τη δήλωση της dmmlib. Κατά την εκτέλεσή τους όμως, μπορούν οι χρήστες να δηλώσουν στη μεταβλητή περιβάλλοντος (environment variable) `LD_PRELOAD` την πλήρη διαδρομή της δυναμικής έκδοσης του αναθέτη της dmmlib και οι εφαρμογές πρόκειται να χρησιμοποιήσουν τις συναρτήσεις που υπάρχουν στον υλοποιημένο αναθέτη.

3.6 Συμπεράσματα

Στο κεφάλαιο αυτό παρουσιάσαμε την dmmlib, ένα πλαίσιο ανάπτυξης μεθοδολογίας δυναμικών αναθετών μνήμης. Το πλαίσιο αυτό έχει γραφτεί στην γλώσσα προγραμματισμού C, ώστε να μπορεί να χρησιμοποιηθεί σε πολλά υπολογιστικά συστήματα, γενικού σκοπού ή ενσωματωμένα, με ΛΣ ή χωρίς.

Διάφορες πολιτικές και μηχανισμοί έχουν ήδη υλοποιηθεί μέσα στην dmmlib. Αν και η ίδια δεν περιγράφει ακόμα πλήρως τον χώρο των δυναμικών αναθετών μνήμης, είναι ήδη ικανή να συνθέσει πλήρεις αναθέτες, οι οποίοι έχουν καλές επιδόσεις στην εκτέλεση των προγραμμάτων που τους χρησιμοποιούν και κατανάλωση μνήμης. Επιπλέον πολιτικές και μηχανισμοί μπορούν να προστεθούν στο πλαίσιο χωρίς να χρειάζεται η τροποποίηση των ήδη υπαρχόντων.

Κάποιοι από τους μηχανισμούς που περιγράφουμε μπορεί να είναι ήδη περιοριστικοί στην σχεδίαση αναθετών μνήμης για οποιαδήποτε πλατφόρμα. Οι αποφάσεις που έχουμε σχεδιάσει στηρίζονται σε δύο γνώμονες: (α) με βάση κάποιων ιδιοτήτων που πιστεύουμε ότι υπάρχουν στην πλειονότητα των υπολογιστικών συστημάτων που χρησιμοποιούν δυναμική ανάθεση μνήμης, για παράδειγμα η χρήση λανθάνουσας μνήμης, και (β) με βάση την δυνατότητα κάλυψης πολλαπλών αποφάσεων, καθώς κάποιοι μηχανισμοί δεν μπορούν να απομονωθούν από κάποιες πολιτικές και το αντίστροφο. Σε κάθε περίπτωση, πιστεύουμε ότι ένας εξοικειωμένος χρήστης της dmmlib θα μπορεί να τροποποιήσει τον πηγαίο κώδικα αν αυτός δεν καλύπτει τις ανάγκες του, θυσιάζοντας κάποιες από τις επιλογές που παρέχει η βιβλιοθήκη.

Τέλος, με την χρήση της διεπαφής που προσφέρει η dmmlib μπορεί να σκιαγραφηθούν εύκολα οι ανάγκες μιας εφαρμογής σε δυναμική μνήμη. Μία απλή επέκταση της μεθοδολογίας θα επέτρεπε την εκτέλεση μιας εφαρμογής με έναν αναθέτη από την dmmlib για την σκιαγράφησή της, την συλλογή των αποτελεσμάτων και κατόπιν την χρησιμοποίησή τους για την παραγωγή ενός καλύτερου για την εφαρμογή αναθέτη.

4 Εφαρμογές Μεθοδολογίας dmmlib πάνω σε Ενσωματωμένα Συστήματα

Στο παρόν κεφάλαιο αναλύουμε την χρήση της dmmlib πάνω σε ενσωματωμένα συστήματα για δύο συγκεκριμένες εφαρμογές: Η πρώτη αφορά πολυπύρηνους επιταχυντές υλικού, πάνω στους οποίους ένας αναθέτης γενικού σκοπού παρουσιάζει μειωμένη απόδοση εμποδίζοντας ακόμα και τη σωστή λειτουργία των εφαρμογών. Η δεύτερη δείχνει τον τρόπο τον οποίο η κλιμακωσιμότητα των επεξεργαστικών πυρήνων επηρεάζει την δυναμική ανάθεση μνήμης και πώς η dmmlib μπορεί να συνεισφέρει στην καλύτερη αξιοποίηση των πόρων ενός συστήματος μαζικά πολλαπλών πυρήνων.

4.1 Δυναμικές Αναθέσεις Μνήμης σε Πολυπύρηνους Επιταχυντές Υλικού

Η διαχείριση μνήμης είναι μία από τις κεντρικές προκλήσεις και στο σχεδιασμό ενσωματωμένων συστημάτων. Η μνήμη αποτελεί σε αυτά ένα σπάνιο πόρο, και το πρόβλημα αυτό κλιμακώνεται δυσανάλογα καθώς ολοένα και περισσότερα ενσωματωμένα συστήματα χρησιμοποιούν πολυπύρηνες αρχιτεκτονικές. Αν η ασφάλεια και η μέγιστη θεωρητική απόδοση δεν είναι πρόβλημα, ολόκληρη η μνήμη θα πρέπει να μπορεί να προσπελαστεί από όλους τους πυρήνες για να μεγιστοποιηθεί η χρήση της. Στην ενότητα αυτή εξετάζουμε πώς θα μπορούσε να χρησιμοποιηθεί η dmmlib με τέτοιο σκοπό και για τέτοια συστήματα. Η προσέγγισή μας εξετάζεται πάνω στην πλατφόρμα P2012, ένα πολυπύρηνο επιταχυντή υλισμικού από την ST. Από τα αποτελέσματα φαίνεται ότι ένας εξειδικευμένος αναθέτης που δημιουργήθηκε από τη βιβλιοθήκη μας μπορεί να γλιτώσει 62% από τους συνολικούς κύκλους που ξοδεύει η πλατφόρμα για τη δυναμική διαχείριση μνήμης σε σύγκριση με τον τρέχοντα αναθέτη της πλατφόρμας. Επιπλέον, η κατανάλωση μνήμης διατηρείται στα ίδια επίπεδα.

4.1.1 Εισαγωγή

Η τρέχουσα τάση στα Συστήματα-σε-Ψηφίδα (System-on-Chips - SoCs) στηρίζεται αρκετά πάνω σε πολλαπλούς πυρήνες και ως εκ τούτου τοποθετείται πολύ κοντά στο χώρο των συμβατικών Πολυ-Επεξεργαστικών SoC (MPSoC). Όσο η τεχνολογία επιτρέπει την ενσωμάτωση μεγάλων αριθμών τρανζίστορ, η έννοια των μαζικά πολυπύρηνων (many-core) υπολογιστών αναδύεται σταθερά, προτείνοντας δεκάδες επεξεργαστικούς πυρήνες ενσωματωμένους σε μία ψηφίδα, η οποία πλέον αποκαλείται υπολογιστή δομή (computing fabric) [48]. Χωρίς αμφιβολία, η διαχείριση μνήμης σε τέτοιες αρχιτεκτονικές θα μπορούσε να συνεισφέρει στη βελτίωση των επιδόσεων και της δυνατότητας κλιμάκωσης, καθώς οι επεξεργαστές ζορίζονται περισσότερο να προσπελάσουν δεδομένα από κοινόχρηστες περιοχές της μνήμης. Η συμφόρηση αυτή μπορεί να έχει μία ακόμα μεγαλύτερη επίπτωση σε μαζικά πολυπύρηνες αρχιτεκτονικές ενσωματωμένων συστημάτων εξαιτίας του συνήθως μικρού μεγέθους μνήμης που χρησιμοποιείται σε τέτοιο υλισμικό.

Αντίστοιχα, οι εφαρμογές που αναπτύσσονται για τα συστήματα αυτά προσαρμόζονται με αργούς ρυθμούς στο προαναφερθέν μοντέλο, ενώ προσπαθούν παράλληλα να εκμεταλλευτούν κάθε διαθέσιμο πόρο χρησιμοποιώντας παραλληλισμό σε επίπεδο δεδομένων και εργασιών. Αυτό οδηγεί σε εφαρμογές με δυναμική συμπεριφορά και παράλληλη εκτέλεση εργασιών. Ο δυναμισμός αυτός, με τη σειρά του, μπορεί να οδηγήσει σε απρόσμενη κατανάλωση μνήμης και διάφορους κατακερματισμούς, οι οποίοι είναι δύσκολο να ανιχνευθούν στη διάρκεια σχεδίασης των εφαρμογών. Ένας τρόπος αντιμετώπισης θα ήταν η χρήση εκτιμήσεων χειρότερων περιπτώσεων και διαχείριση της μνήμης με ένα στατικό τρόπο, αλλά, όπως έχουμε αναφέρει, κάτι τέτοιο θα προκαλούσε σοβαρά προβλήματα χρήσης επιπλέον μνήμης και μεγαλύτερης κατανάλωσης ενέργειας. Για τους λόγους αυτούς, λοιπόν, οι προγραμματιστές θα πρέπει και πάλι να καταφύγουν στη χρήση δυναμικής ανάθεσης μνήμης [5].

Στην ενότητα αυτήν θα εξετάσουμε τις αναθέσεις μνήμης σε ένα μαζικά πολυπύρηνο επιταχυντή υλισμικού, την P2012, και θα προτείνουμε ένα εξειδικευμένο σύστημα δυναμικής ανάθεσης μνήμης για να βελτιώσουμε την απόδοση στις σχετικές λειτουργίες του επιταχυντή αυτού.

Οι υπόλοιπες υποενότητες οργανώνονται ως ακολούθως: Στην Υποενότητα 4.1.2 εξηγούμε τα διαθέσιμα προγραμματιστικά της μοντέλα και τα patterns εκτέλεσης της P2012. Η Υποενότητα 4.1.3 αφιερώνεται στο σχεδιασμό αναθετών μνήμης για την πλατφόρμα αυτή και δίνει μία σύντομη εποπτεία στον τρέχοντα αναθέτη μνήμης. Εκτελούμε πειράματα πάνω σε αυτήν και τα παρουσιάζουμε στην Υποενότητα 4.1.4. Τέλος, αναφέρουμε τα συμπεράσματα στην Υποενότητα 4.1.5.

4.1.2 Ανάπτυξη λογισμικού στην P2012

Σχήμα 4.1: Λογισμικό και P2012

Η P2012 ενεργεί ως ένας επιταχυντής, η ύπαρξη δηλαδή ενός συστήματος να τη φιλοξενεί είναι απαραίτητη. Το Σχήμα 4.1 δείχνει το σύνολο λογισμικού που χρειάζεται για να αξιοποιήσει μία εφαρμογή από το κεντρικό σύστημα τον επιταχυντή. Η περιγραφή της εφαρμογής και τα προγραμματιστικά μοντέλα που αναφέρονται είναι ενιαία για όλα τα συστήματα, αλλά καθένα από αυτά έχει διαφορετικό περιβάλλον εκτέλεσης. Από την πλευρά του συστήματος που τον φιλοξενεί, υπάρχουν το Λειτουργικό Σύστημα (στην P2012 υποστηρίζονται το GNU/Linux και το Android) και οι οδηγοί συσκευής του επιταχυντή. Από τον τελευταίο, υπάρχει ένα συγκεκριμένο περιβάλλον εκτέλεσης. Το P2012 Software Development Kit (SDK) περιέχει όλα τα απαραίτητα εργαλεία για την ανάπτυξη εφαρμογών για την P2012, όπως και τον πηγαίο κώδικα των υπηρεσιών της πλατφόρμας μέχρι κάποιο βαθμό. Μία κοινότητα χρηστών υποστηρίζει ενεργά το SDK στην [49].

Η εκτέλεση των εφαρμογών οργανώνεται ως εξής: Ένας οδηγός συσκευής στο Linux διαχειρίζεται την επικοινωνία μεταξύ του επιταχυντή και του συστήματος που το φιλοξενεί, φορτώνοντας τον κώδικα εφαρμογής από το σύστημα στον επιταχυντή και ελέγχοντας τους πόρους

της P2012 από την πλευρά του συστήματος. Οι υπηρεσίες εκτέλεσης από την άλλη πλευρά είναι διαθέσιμες σε κάθε ελεγκτή συμπλέγματος. Αυτοί με τη σειρά τους είναι υπεύθυνοι για την τοποθέτηση της εφαρμογής στον κάθε επεξεργαστή ENCore και επιπλέον παρέχουν τον προγραμματισμό (scheduling) και τη διαχείριση όλων των πόρων του συμπλέγματος όσον αφορά την ανάθεση και την αποδέσμευση επεξεργαστών και μνήμης, αλλά και τη διαχείριση της ενέργειας. Ο κώδικας του περιβάλλοντος εκτέλεσης εκτελείται τοπικά στους ελεγκτές συμπλεγμάτων της P2012 και η Προγραμματιστική Διεπαφή Εφαρμογής (ΠΔΕ) του χρησιμοποιείται προκειμένου να αναπτυχθούν τα προγραμματιστικά μοντέλα της πλατφόρμας.

Οι εφαρμογές μπορούν να αναπτυχθούν αυτήν τη στιγμή σε δύο προγραμματιστικά μοντέλα, τα οποία είναι διαθέσιμα και υποστηρίζονται από την P2012: Το πλαίσιο Open Compute Language (OpenCL) και το Native Programming Model (NPM). Το πρώτο είναι γρήγορα αποδεχόμενο ως το βιομηχανικό πρότυπο, καθώς οι περισσότερες από τις μεγαλύτερες εταιρίες υλισμικού είναι ενεργά μέλη του Khronos Compute Working Group που είναι υπεύθυνο για την ανάπτυξη των επίσημων προδιαγραφών OpenCL. Το NPM αποτελεί μια πιο συγκεκριμένη προσέγγιση να προγραμματιστεί η πλατφόρμα. Χειρίζεται την σύνδρομη προσπέλαση με ένα μοντέλο Actor. Άλλα προγραμματιστικά μοντέλα μπορούν να αναπτυχθούν πάνω από το NPM, ακόμα και το περιβάλλον εκτέλεσης του OpenCL για την P2012 έχει χτιστεί πάνω από αυτό [15].

Και τα δύο προγραμματιστικά μοντέλα χρησιμοποιούν το περιβάλλον εκτέλεσης της P2012 για να προσπελάσουν πόρους και τον χρονοπρογραμματιστή εργασιών χαμηλού επιπέδου. Στην πραγματικότητα χρησιμοποιούν και οι δύο το μοντέλο εκτέλεσης που απεικονίζεται στο Σχήμα 4.2. Πρώτα, το σύστημα που φιλοξενεί τον επιταχυντή προετοιμάζει τον τελευταίο στέλνοντας τον απαραίτητο κώδικα που πρέπει να εκτελεστεί. Η ανάθεση πόρων, η αντιστοίχηση εργασιών σε συγκεκριμένους επεξεργαστές ENCore και οι αναθέσεις μνήμης για τις στοίβες των επεξεργαστών και τις ενδιάμεσες αποθηκεύσεις είναι το επόμενο βήμα στο σχεδιάγραμμα της εκτέλεσης. Στην πραγματικότητα η ανάθεση πόρων πρέπει να έχει εντελώς ολοκληρωθεί προτού ξεκινήσει η φάση εκτέλεσης εργασιών. Κατά τη διάρκεια εκτέλεσης εφαρμογών δεν υπάρχει καινούρια ανάθεση μνήμης: ακόμα και οι νέες υποστάσεις εργασιών χρησιμοποιούν το χώρο της μνήμης που ήταν καταμερισμένος προηγουμένως στην αρχική ίδια εργασία. Μετά την ολοκλήρωση κάθε εργασίας στους επεξεργαστές ENCore, η αποδέσμευση πόρων λαμβάνει χώρα και στο τέλος η P2012 τίθεται σε αδράνεια.

Σχήμα 4.2: Εκτελώντας μία εφαρμογή στην P2012

4.1.3 Δυναμική Ανάθεση Μνήμης στην P2012

Αρχικός Αναθέτης Μνήμης

Ο αναθέτης μνήμης που χρησιμοποιεί η P2012 είναι μία παραλλαγή του δυναμικού αναθέτη μνήμης Doug Lea, δηλαδή της dlmalloc [19]. Ο αναθέτης Lea θεωρείται από τους καλύτερους γενικά αλγορίθμους και ανταγωνίζεται ακόμα και εξειδικευμένους [33].

Για κάθε κλήση ανάθεσης ένα μπλοκ μνήμης δίνεται στην εφαρμογή, το οποίο περιέχει το απαιτούμενο μέγεθος οριοθετημένο από τις πληροφορίες μεγέθους του μπλοκ. Αυτό βοηθάει στο διαχωρισμό κάθε μπλοκ και βελτιώνει διαδικασίες πάνω στο επίπεδο των δεδομένων, όπως η συγχώνευση μπλοκ.

Τα ελεύθερα μπλοκ μνήμης διατηρούνται σε κυκλικές, διπλά συνδεδεμένες λίστες: κάθε ελεύθερο μπλοκ περιέχει δείκτες στο προηγούμενο και το επόμενο μπλοκ στη λίστα για τις περιοχές που προορίζονται για τα δεδομένα της εφαρμογής. Υπάρχουν 128 τέτοιες λίστες που περιέχουν μπλοκ ανάλογα με το μέγεθός τους. Για παράδειγμα, υπάρχουν λίστες για μπλοκ των 16, 32 bytes και ούτω καθεξής. Ο μέγιστος αριθμός μπλοκ που μπορεί να συμπεριληφθεί σε αυτές τις λίστες μειώνεται λογαριθμικά καθώς το μέγεθος των μπλοκ μεγαλώνει.

Τα μπλοκ μπορούν να ταξινομηθούν και να αναζητηθούν είτε από το λιγότερο πρόσφατα χρησιμοποιημένο, είτε με πρώτη εισαγωγή, πρώτη εξαγωγή. Επίσης, υποστηρίζεται η δυνατότητα συγχώνευσης και διαίρεσης μπλοκ.

Δημιουργία εξειδικευμένου αναθέτη για την P2012

Η dmmlib είναι μια φορητή βιβλιοθήκη δυναμικής ανάθεσης γραμμένη σε C, η οποία επιτρέπει τη δημιουργία εξειδικευμένων δυναμικών αναθετών μνήμης διαλέγοντας τα επιθυμητά χαρακτηριστικά και πολιτικές.

Η μεθοδολογία παρέχει εξειδικευμένες υλοποιήσεις για δυναμική ανάθεση μνήμης, επανάθεση και αποδέσμευση, οι οποίες αντικαθιστούν τυπικές κλήσεις συστήματος όπως η `malloc()`, η `realloc()` και η `free()`. Οι παραγόμενες συναρτήσεις μπορούν να είναι εντελώς αυτόνομες αν ο προγραμματιστής γνωρίζει εξαρχής τη συνολική μνήμη που πρόκειται να διαχειριστεί, ή μπορούν να χρησιμοποιούν κλήσεις του λειτουργικού συστήματος, όπως η `sbrk()` ή η `mmap()`, για να προσπελάσουν το χώρο μνήμης. Η πολυνηματική ΔΔΜ μπορεί να υποστηριχτεί χρησιμοποιώντας POSIX mutexes ή θεμελιακά στοιχεία συγχρονισμού συγκεκριμένα για την εκάστοτε πλατφόρμα. Η μνήμη μπορεί να οργανωθεί σε πολλαπλές σωρούς, καθεμία από τις οποίες μπορεί να περιέχει λίστες ελεύθερων μπλοκ σταθερού μεγέθους. Υπάρχουν αρκετές υλοποιήσεις οργανώσεων μπλοκ (μονά, διπλά συνδεδεμένες λίστες λίστες κ.λπ.) και υποστηρίζεται μία ποικιλία από αλγορίθμους αναζήτησης μπλοκ (πρώτης, καλύτερης, επόμενης, καλής, ακριβής εφαρμογής κ.λπ.) Επιπλέον, υπάρχει δυνατότητα για συγχώνευση και διαίρεση μπλοκ προκειμένου να αποφευχθεί ο υπερβολικός κατακερματισμός της μνήμης. Σχετικά κατώφλια μπορούν να δοθούν κατά το χρόνο σχεδιασμού ή ακόμα και εκτέλεσης για τους προηγούμενους μηχανισμούς. Τέλος, η βιβλιοθήκη υποστηρίζει ένα πλούσιο σετ από στατιστικά, όπως ο αριθμός των προσπελάσεων στη μνήμη και τα απαιτούμενα μεγέθη ανάθεσης, τα οποία είναι διαθέσιμα κατά την εκτέλεση με κάποιο επιπλέον κόστος στις λειτουργίες του αναθέτη και τις δομές των μεταδεδομένων.

Μία γενικότερη τάση της βιβλιοθήκης είναι η δυνατότητα αλλαγής κάποιων χαρακτηριστικών κατά το χρόνο εκτέλεσης των εφαρμογών. Ο κάθε προγραμματιστής θα πρέπει να επιλέξει τη δυνατότητα αυτή προτού δημιουργηθεί ο εκτελέσιμος κώδικας της βιβλιοθήκης. Εκτός των χαρακτηριστικών, μπορούν να τεθούν και κατώφλια είτε στατικά στο χρόνο σχεδιασμού, είτε δυναμικά με τη χρήση παραμέτρων κατά την εκτέλεση. Η προοπτική αυτή της ρύθμισης παραμέτρων κατά την εκτέλεση ώστε να δημιουργηθούν περισσότερο προσαρμόσιμοι αναθέτες ξεκίνησε να ερευνάται διεξοδικά στην [50]. Σε κάθε περίπτωση, θα πρέπει να σημειωθεί ότι υπάρχει μία ξεκάθαρη αντιστάθμιση μεταξύ μέγεθος κώδικα και προσαρμοστικότητας, καθώς ο προγραμματιστής διαλέγει να υλοποιήσει χαρακτηριστικά και πολιτικές στατικά ή όχι.

Η μεθοδολογία dmmlib ενσωματώθηκε στο P2012 SDK. Πιο συγκεκριμένα, επιλέχτηκε το

περιβάλλον εκτέλεσης της P2012, επειδή αυτό θεωρείται το χαμηλότερο επίπεδο για διαχείριση πόρων μέσα σε ένα σύμπλεγμα της P2012. Ως αποτέλεσμα της βαθιάς ενσωμάτωσης μέσα στο περιβάλλον εκτέλεσης, κάθε εξειδικευμένος αναθέτης μνήμης που παράγεται από την dmmlib, μπορεί να χρησιμοποιηθεί άμεσα και από τα δύο διαθέσιμα προγραμματιστικά μοντέλα. Ένα ακόμα κέρδος αυτής την ενσωμάτωσης είναι ότι η χρήση του βελτιστοποιημένου αναθέτη είναι διαφανής προς τους προγραμματιστές, δηλαδή δεν απαιτούνται αλλαγές στον κώδικα των εφαρμογών.

Επιλογές στην dmmlib για ένα αναθέτη της P2012

Η dlmalloc είναι ένας πολύ αποτελεσματικός αναθέτης μνήμης γενικού σκοπού, αλλά δεν ταιριάζει στη τρέχουσα στρατηγική εκτέλεσης εφαρμογών της P2012. Προκειμένου να εξερευνηθεί περισσότερο η ενσωμάτωση της dmmlib στην P2012, ένας εξειδικευμένος αναθέτης μνήμης δημιουργήθηκε με χαρακτηριστικά και επιλογές που θεωρήσαμε βέλτιστες για την πλατφόρμα αυτή.

Η κύρια συλλογιστική για τις περισσότερες επιλογές ήταν να επιτευχθεί απλότητα, καθώς η πλατφόρμα του στόχου μας είναι τμήμα ενός ενσωματωμένου συστήματος και δε διαθέτει πολλούς υπολογιστικούς πόρους. Πιο συγκεκριμένα, λοιπόν, επιλέξαμε την οργάνωση των μπλοκ να είναι παρόμοια με της dlmalloc, ώστε να αποφύγουμε το επιπλέον κόστος στα μεταδεδομένα. Όμως, καθώς πρόκειται για ένα περιβάλλον με λίγη μνήμη, χρησιμοποιήσαμε κυκλικές, μονά συνδεδεμένες λίστες αντί για κυκλικές, διπλά συνδεδεμένες, ώστε να επιτύχουμε μείωση στην κατανάλωση μνήμης σε σύγκριση με τον αναθέτη. Επιπλέον, αντί να χρησιμοποιήσουμε πολλαπλές λίστες από ελεύθερα μπλοκ, επιλέγουμε να χρησιμοποιήσουμε μία μόνο τέτοια λίστα. Η απόφαση αυτή απλοποιεί τη διαδικασία εύρεσης κατάλληλου, ελεύθερου μπλοκ μνήμης, το οποίο να είναι αρκετά μεγάλο ώστε να φιλοξενήσει το αίτημα της ανάθεσης. Τέλος, απενεργοποιήσαμε τη συγχώνευση και διαίρεση, ώστε να απλοποιηθούν ακόμα περισσότερο οι κλήσεις `malloc()` και `free()` και να έχουμε περισσότερα κέρδη στις επιδόσεις.

Ο αναθέτης μνήμης στον οποίο καταλήξαμε είναι ένας ακραία απλός, ο οποίος ωστόσο ταιριάζει απόλυτα στην τρέχουσα ροή εκτέλεσης της P2012. Σε διαφορετικές περιστάσεις θα προξενούσε προβλήματα κατακερματισμού σε εφαρμογές που εκτελούν ποικίλες εργασίες ακόμα και μη παράλληλα. Αξίζει να σημειωθεί πάντως ότι οι τρέχουσες υλοποιήσεις των προγραμματιστικών μοντέλων δε δημιουργούν τέτοιες εργασίες ή χειρίζονται τους πόρους με τέτοιο τρόπο.

4.1.4 Πειραματικά Αποτελέσματα

Υποδομή Πειραμάτων

Προκειμένου να συγκριθεί ο προτεινόμενος εξειδικευμένος αναθέτης μνήμης με τον τρέχοντα, χρειάζονται προσομοιώσεις ακρίβειας τουλάχιστον επιπέδου κύκλου για διάφορες εφαρμογές που εκτελούνται πάνω στην P2012. Επιλέξαμε, λοιπόν, την προσομοίωση σε επίπεδο μεταφοράς καταχωριστών, καθώς η κίνηση στο Δίκτυο-Σε-Ψηφίδα και οι χρόνοι προσπέλασης της μνήμης είναι οι πιο ακριβείς σε αυτό το είδος προσομοίωσης.

Ωστόσο, η προσομοίωση αυτή είναι επίσης και μία από τις πιο χρονοβόρες. Όλες οι δοκιμασμένες εφαρμογές χρησιμοποιούν ένα σύμπλεγμα και έτσι η προσομοίωση μόνο ενός είναι αρκετή για τις ανάγκες των μετροπρογραμμάτων. Ακόμα και έτσι, η προσομοίωση απαιτεί αρκετό χρόνο ακόμα και για την εκτέλεση απλών εφαρμογών. Για αυτόν το λόγο, απλοποιήσαμε τις εφαρμογές: Καθώς όλα τα αιτήματα ανάθεσης και αποδέσμευσης συμβαίνουν στον ελεγκτή του συμπλέγματος, δε χρειάζεται να ενεργοποιήσουμε τους επεξεργαστές ENCore. Εξάγοντας τα ίχνη (traces) των εφαρμογών, δημιουργούμε απλές δοκιμαστικές εφαρμογές, όπου μόνο αυτές οι εντολές εκτελούνται.

Τα ίχνη προέρχονται από εφαρμογές διαθέσιμες μέσα από το επίσημο SDK της P2012 [49], και είναι οι ακόλουθες:

- **Integral** Υπολογίζει το ολοκλήρωμα ενός πίνακα που εισάγεται κατά την εκκίνηση.
- **Gaussian** Εφαρμόζει ένα εφέ θολώματος με χρήση Γκαουσιανών συναρτήσεων πάνω σε μια εικόνα που εισάγεται κατά την εκκίνηση.
- **FAST** Features from Accelerated Segment Test, μία υλοποίηση αλγορίθμου ανίχνευσης γωνιών που χρησιμοποιείται για μηχανική όραση.
- **Matrix** Εκτελεί πολλαπλασιασμό ενός πίνακα που εισάγεται κατά την εκκίνηση με ένα σταθερό πίνακα.

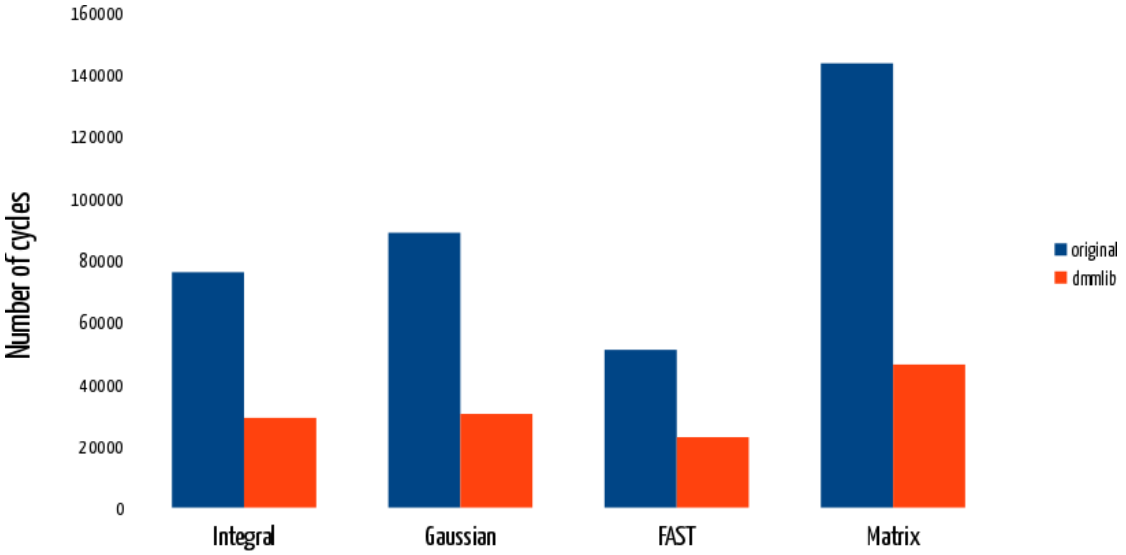
Αξιολόγηση των αναθετών

Χάρη στην απλότητά του, ο παραγόμενος αναθέτης έχει μικρότερο μέγεθος από τον αναθέτη του Doug Lea: 60 Kilobytes έναντι 88 Kilobytes, μία δηλαδή βελτίωση 32% στο μέγεθος κώδικα.

Τα μεταδεδομένα, όπως για παράδειγμα η εσωτερική οργάνωση της σωρού, οι πληροφορίες χρήσης και τα στατιστικά) είναι ίδια και στους δύο αναθέτες, εφόσον η δομή του μπλοκ εμφανίζεται στη μνήμη δύο φορές, μία πριν και μία μετά από το χώρο που προορίζεται για τα δεδομένα της εφαρμογής. Αυτό είναι ισοδύναμο με 64 bits ανά μπλοκ, καθώς η πληροφορία μεγέθους στη

dmmlib καταλαμβάνει το μέγεθος μιας λέξης και οι επεξεργαστές ENCore της P2012 χρησιμοποιούν λέξεις των 32 bits.

Η επιτάχυνση από τη χρήση του εξειδικευμένου αναθέτη είναι εξίσου ουσιαστική. Στο Σχήμα 4.3 φαίνεται μία βελτίωση 60% στους κύκλους κατά μέσο όρο για την εκτέλεση όλων των αναθέσεων που απαιτεί καθεμία εφαρμογή. Στο Σχήμα 4.4 συμπεριλαμβάνονται και οι αποδεσμεύσεις της μνήμης στο συνολικό χρόνο, αλλά η επίπτωσή τους στον τελικό χρόνο είναι ελάχιστη.

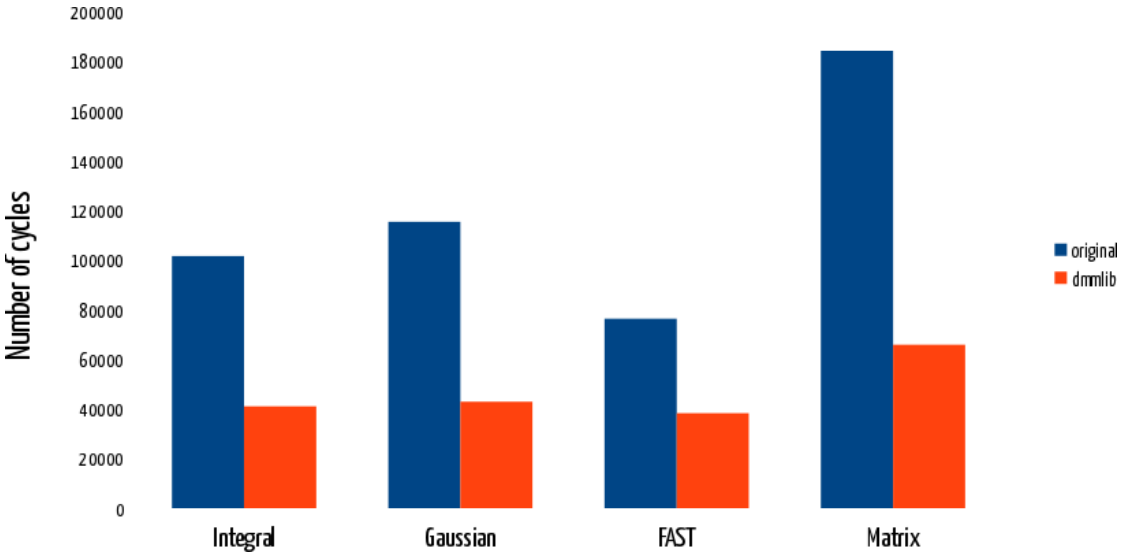


Σχήμα 4.3: Σύγκριση των επιδόσεων των δύο αναθετών μνήμης σε διάφορες εφαρμογές

Οι Πίνακες 4.1 και 4.2 παρουσιάζουν μία πιο καθαρή εικόνα στο μέσο και μέγιστο χρόνο κάθε αναθέτης χρειάζεται για ένα μόνο γεγονός, καθώς και την αντίστοιχη διαφορά.

Πίνακας 4.1: Μέσες και μέγιστες τιμές χρόνων ανάθεσης

Εφαρμογή	Καταμ.	Μέσοι Κύκλοι	Μέση Διαφορά	Μεγ. Κύκλοι	Μέγ. Διαφορά
Integral	αρχικός	1224	-62.54%	2278	-66.11%
	νέος	329		772	
Gaussian	αρχικός	886	-62.86%	2293	-66.33%
	νέος	466		772	
FAST	αρχικός	737	-55.5%	1205	-49.88%
	νέος	328		604	
Matrix	αρχικός	1462	-67.85%	2334	-67.65%
	νέος	470		755	



Σχήμα 4.4: Σύγκριση των επιδόσεων των δύο αναθετών μνήμης συμπεριλαμβανομένης της αποδέσμευσης σε διάφορες εφαρμογές

Η τάση που μπορούμε να εξάγουμε από τους χρόνους αυτούς είναι ότι οι εφαρμογές που χρησιμοποιούν πολλά μικρά μπλοκ (όπως για παράδειγμα η εφαρμογή Matrix) επωφελούνται περισσότερο από τον εξειδικευμένο αναθέτη μνήμης της dmmlib.

Εκτίμηση ανάθεσης μνήμης σε περιπτώσεις πιο δυναμικής εκτέλεσης εργασιών

Όσο η P2012 ωριμάζει, είναι σχεδόν βέβαιο ότι θα μπορούμε να έχουμε πιο σύνθετες, δυναμικές ροές εκτέλεσης εργασιών. Οι πόροι θα πρέπει τότε να αποδεσμεύονται και επανααναθέτονται με ένα πιο λεπτομερειακό τρόπο, για παράδειγμα ανάμεσα στις εκτελέσεις των εργασιών και όχι μόνο μετά το τέλος της εκτέλεσης της τελευταίας εργασίας της εφαρμογής. Για αυτό το λόγο, λοιπόν, χρειάζεται να δοκιμάσουμε τους αναθέτες μνήμης σε περιπτώσεις χρήσης όπου τα δεδομένα αναθέτονται και αποδεσμεύονται με ένα πιο δυναμικό τρόπο.

Δυστυχώς δεν υπάρχουν ακόμα διαθέσιμες εφαρμογές οι οποίες ταιριάζουν με αυτήν τη συμπεριφορά. Αυτό που προτείνουμε προκειμένου να εκτιμήσουμε τις επιδόσεις των αναθετών μνήμης σε τέτοιες περιπτώσεις, είναι να χρησιμοποιήσουμε τα ίχνη των προηγούμενων εφαρμογών και να ανακατέψουμε με τυχαίο τρόπο τις σειρές των αποδεσμεύσεων των δεδομένων. Οι εξαρτήσεις των αναθέσεων θα είναι ακόμα σεβαστές (για παράδειγμα, δε θα μπορούμε να αποδεσμεύσουμε μνήμη που δεν έχει ανατεθεί προηγουμένως) και τα αιτήματα μνήμης θα παραμείνουν ρεαλιστικά αρκετά.

Για το κομμάτι αυτό χρησιμοποιήσαμε τις εφαρμογές FAST και Matrix, καθώς οι εργασίες

Πίνακας 4.2: Μέσες και μέγιστες τιμές χρόνων αποδέσμευσης

Εφαρμογή	Καταμ.	Μέσοι Κύκλοι	Μέση Διαφορά	Μέγ. Κύκλοι	Μέγ. Διαφορά
Integral	αρχικός	412	-52.91%	603	-38.97%
	νέος	194		368	
Gaussian	αρχικός	409	-52.57%	603	-66.33%
	νέος	194		368	
FAST	αρχικός	367	-38.69%	550	-12.55%
	νέος	225		481	
Matrix	αρχικός	416	-51.92%	636	-41.04%
	νέος	200		375	

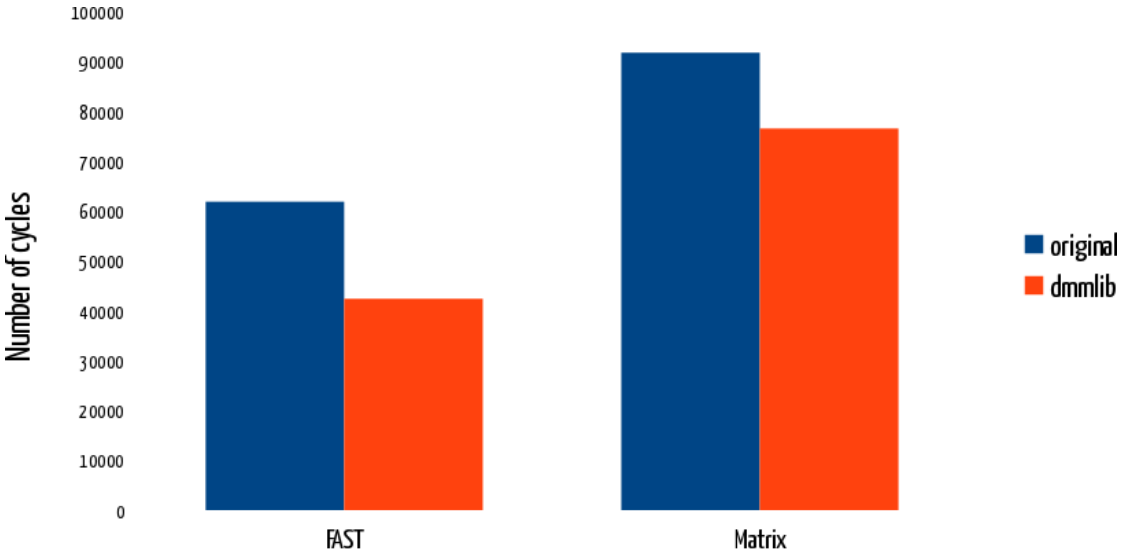
αυτών των εφαρμογών είναι πιο κοντά σε πυρήνες εφαρμογών στον πραγματικό κόσμο. Στο Σχήμα 4.5 υπάρχει μία επισκόπηση των συνολικών κύκλων που δαπανήθηκαν για λειτουργίες διαχείρισης μνήμης. Ο νέος αναθέτης μνήμης ακόμη υπερτερεί του αρχικού, αλλά η διαφορά είναι πλέον μικρότερη. Οι Πίνακες 4.3 και 4.4 δείχνουν τις μέσες και χειρότερες περιπτώσεις χρόνου που απαιτείται για την ανάθεση και την αποδέσμευση αντίστοιχα των δεδομένων. Θα πρέπει να σημειωθεί ότι ο αριθμός των αναθέσεων για την εφαρμογή FAST είναι μικρότερος από τον αντίστοιχο της εφαρμογής Matrix, καθώς η εφαρμογή FAST πραγματοποιεί πιο λεπτομερείς αναθέσεις μνήμης μέσα στον πυρήνα της. Ως αποτέλεσμα, ο εξειδικευμένος αναθέτης μνήμης φαίνεται να έχει καλύτερες επιδόσεις στην περίπτωση της εφαρμογής FAST έναντι της Matrix.

Πίνακας 4.3: Μέσες και μέγιστες τιμές χρόνων ανάθεσης για πιο δυναμικές περιπτώσεις

Εφαρμογή	Καταμ.	Μέσοι Κύκλοι	Μέση Διαφορά	Μέγ. Κύκλοι	Μέγ. Διαφορά
FAST	αρχικός	549	-29.14%	1153	-32.35%
	νέος	389		780	
Matrix	αρχικός	583	-6%	1184	-9.62%
	νέος	548		1310	

4.1.5 Συμπεράσματα

Η δημιουργία εξειδικευμένων αναθετών μνήμης μάς επέτρεψε να επιταχύνουμε τη διαδικασία εξερεύνησης αποτελεσματικών αναθετών μνήμης για πολυπύρηνες αρχιτεκτονικές, όπως αυτή της P2012. Ο εξειδικευμένος αναθέτης μνήμης που προτείνουμε για την P2012 πετυχαίνει μία επιτάχυνση της τάξεως 2,6x χωρίς να διακινδυνεύει επιπλέον κόστος στα μεταδεδομένα, σε σύ-



Σχήμα 4.5: Σύγκριση των επιδόσεων των δύο αναθετών μνήμης σε δυναμικές περιπτώσεις

Πίνακας 4.4: Μέσες και μέγιστες τιμές χρόνων αποδέσμευσης για πιο δυναμικές περιπτώσεις

Εφαρμογή	Καταμ.	Μέσοι Κύκλοι	Μέση Διαφορά	Μέγ. Κύκλοι	Μέγ. Διαφορά
FAST	αρχικός	347	-38.69%	565	-12.55%
	νέος	225		481	
Matrix	αρχικός	353	-34.28%	617	-21.07%
	νέος	232		487	

γκριση πάντα με τον επίσημο αναθέτη μνήμης του περιβάλλοντος εκτέλεσης της P2012.

Καθώς η P2012 και το οικοσύστημα των πολυπύρηνων επιταχυντών αναπτύσσονται, αναμένουμε πολλαπλά σενάρια χρήσης και αυξημένη αλληλεπίδραση μεταξύ των συστημάτων αυτών και του περιβάλλοντος που τα περικλείει. Η διαχείριση μνήμης αναμένεται να παίζει τότε μεγαλύτερο ρόλο και ενδεχομένως νέα χαρακτηριστικά και πολιτικές θα πρέπει να προταθούν, όπως δυνατότητα αλλαγών πολιτικών κατά την εκτέλεση των εφαρμογών, δυνατότητα συγχώνευσης και διαίρεσης μπλοκ, υποστήριξη ανάθεσης και αποδέσμευσης παράλληλα από περισσότερους επεξεργαστές, ακόμα και η δυνατότητα πιο λεπτομερούς διαχείρισης μνήμης για συγκεκριμένες περιοχές μνήμης. Προερχόμενη από την πλατφόρμα x86, όπου τέτοια σενάρια ήδη υφίστανται, η dmmlib αναμένεται να καλύψει τις ανάγκες αυτές.

4.2 Δυναμική Ανάθεση Μνήμη και Κλιμακωσιμότητα

Όπως έχει προαναφερθεί, η διαχείριση μνήμης σε πολυπύρηνες αρχιτεκτονικές αποτελεί μία βασική πρόκληση για την συνολική επίδοση ενός υπολογιστικού συστήματος. Σε έναν από τους δημοφιλέστερους σχεδιασμούς μνήμης, αυτόν της Ανομοιόμορφης Προσπέλασης Μνήμης (ΑΠΜ), οι πόροι της μνήμης κατανέμονται σε κόμβους για ταχύτερες τοπικές προσπελάσεις. Σε δυναμικά φορτία εργασίας που εξαρτώνται ιδιαίτερα σε αιτήματα μνήμης, μία ανεπαρκής διαχείριση μνήμης μπορεί να οδηγήσει σε σοβαρά προβλήματα συμφόρησης και υποβιβασμό της επίδοσης. Στο κεφάλαιο αυτό εστιάζουμε στην βελτιστοποίηση της δυναμικής ανάθεσης μνήμης στις εν λόγω πλατφόρμες με ανάλογα φορτία εργασίας και παρουσιάζουμε ένα δυναμικό διαχειριστή μνήμης που δημιουργήθηκε μέσα από την dmmlib, ο οποίος είναι στοχεύει σε κατανεμημένα, ενσωματωμένα συστήματα που χρησιμοποιούν προγραμματιζόμενους επιταχυντές υλικού. Ο διαχειριστής αυτός αξιοποιεί την ύπαρξη τέτοιων επιταχυντών, ενώ παρέχει μία ΠΔΕ σε γλώσσα C για τους προγραμματιστές. Τα πειραματικά αποτελέσματα που θα δούμε στην πορεία, δείχνουν κέρδη στις επιδόσεις 10% κατά μέσο όρο σε σχέση με αναθέτες που είναι γραμμένοι καθαρά σε C, καθώς και την απαραίτητη κλιμακωσιμότητα ενώ το μέγεθος της πλατφόρμας αυξάνει.

4.2.1 Εισαγωγή

Ο νόμος του Moore περικλείει την κλιμακωσιμότητα που παρατηρείται στις ψηφίδες. Τα δίκτυα διασύνδεσης έχουν μετατραπεί σε ιδιαίτερα πολύπλοκα, προερχόμενα από τα συμβατικά δίκτυα bus και καταλήγοντας σε εξεζητημένα Δίκτυα-σε-Ψηφίδα (Network-on-Chips - NoCs). Αν και η υλοποίηση του μοντέλου Κατανεμημένης Διαμοιραζόμενης Μνήμης (Distributed Shared Memory - DSM) πάνω σε τέτοια δίκτυα συνοδεύεται με προβλήματα συμφόρησης στην μνήμη, οι σχεδιαστές των συστημάτων αυτών το επιλέγουν, επειδή το μοντέλο προγραμματισμού που προκύπτει είναι αρκετά εύκολο.

Σε τέτοια, λοιπόν, συστήματα οι προγραμματιστές συνηθίζουν να ακολουθούν μία συγκεντρωτική προσέγγιση και να μην εκμεταλλεύονται στο έπακρο τα ιδιαίτερα χαρακτηριστικά της αρχιτεκτονικής. Στην προηγούμενη ενότητα, 4.1, είδαμε ένα τέτοιο παράδειγμα, με την πλατφόρμα P2012 και τον fabric controller. Η συγκεντρωτική αυτή προσέγγιση δημιουργεί ένα κεντρικό πιθανό σημείο βλάβης, που μπορεί να θέσει ολόκληρο το σύστημα σε αχρηστία αν παρουσιαστεί αστοχία στον κεντρικό πυρήνα. Επιπλέον, ο κεντρικός πυρήνας παρεμποδίζει την κλιμακωσιμότητα, επειδή αποτελεί σημείο συμφόρησης για την επεξεργασία και την επικοινωνία.

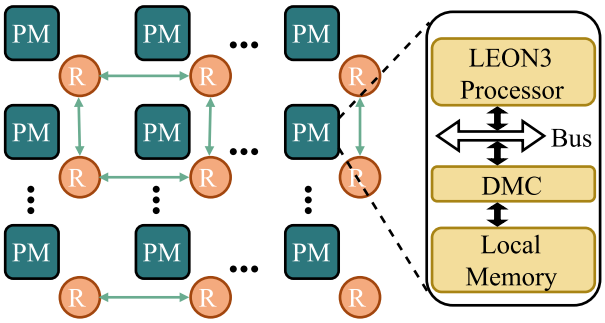
Στην ενότητα αυτή, προτείνουμε έναν πρωτότυπο δυναμικό αναθέτη μνήμης για ενσωματωμένα συστήματα που ακολουθούν το μοντέλο DSM. Ο αναθέτης αυτός επιταχύνεται από εξειδι-

κευμένο υλισμικό και καταφέρνει να διατηρήσει την κλιμακωσιμότητά του. Οι κύριες συνεισφορές του προτεινόμενου αναθέτη είναι οι εξής: (α) Παρουσιάζει ένα ενοποιημένο πεδίο στοίβας που είναι προσπελάσιμο από οποιονδήποτε κόμβο· (β) Αξιοποιεί την παρουσία προγραμματιζόμενων ελεγκτών μνήμης χρησιμοποιώντας εξειδικευμένες εντολές χαμηλού επιπέδου, προκειμένου να επιταχύνει τις συναρτήσεις δυναμικής διαχείρισης μνήμης· (γ) Απαιτεί ελάχιστη αρχική ρύθμιση και (δ) Προσφέρει μία ΠΔΕ για να διαχειρίζονται οι εφαρμογές την στοίβα, παρόμοια με αυτή της `malloc()` και `free()`.

4.2.2 Αξιοποιώντας εξειδικευμένους ελεγκτές μνήμης

Ας υποθέσουμε ότι έχουμε ένα σύστημα DSM όπως αυτό του Σχήματος 4.6: Πολλαπλοί κόμβοι Επεξεργαστή-Μνήμης (Processor-Memory - PM) είναι διασυνδεδεμένοι σε ένα πολυγωνικό (mesh) δίκτυο μεταγωγής πακέτων αποτελούμενο από δρομολογητές (Routers - R). Κάθε κόμβος περιέχει επιπλέον ένα Dual Microcoded Controller (DMC), έναν προγραμματιζόμενο επιταχυντή υλισμικού, ο οποίος αναλαμβάνει τα αιτήματα μνήμης [51]. Η πλατφόρμα αυτή έχει αποδειχτεί ήδη ικανή να επιταχύνει τις λειτουργίες μνήμης σε περιβάλλοντα DSM [43], [51], [52] με τον προγραμματισμό του DMC και παρέχοντας υπηρεσίες όπως: εικονική-σε-φυσική (virtual-to-physical - V2P) μετάφραση διευθύνσεων, συγχρονισμό, συνοχή κρυφής μνήμης (cache coherency), συνοχή μνήμης (memory consistency) και προσπέλαση κοινόχρηστης μνήμης [51].

Η υπηρεσία V2P είναι αυτή που παρέχει ουσιαστικά το μοντέλο DSM: Οι τοπικές μνήμες οργανώνονται σε ιδιωτικά και κοινόχρηστα τμήματα και, χρησιμοποιώντας την υπηρεσία V2P μέσω του τοπικού τους DMC, οι επεξεργαστές είναι σε θέση να προσπελάσουν οποιαδήποτε κοινόχρηστη μνήμη χρησιμοποιώντας ένα σύστημα υψηλότερης διευθυνσιοδότησης σε σχέση με το τοπικό (π.χ. `0x4020000` έναντι `0x00000`).



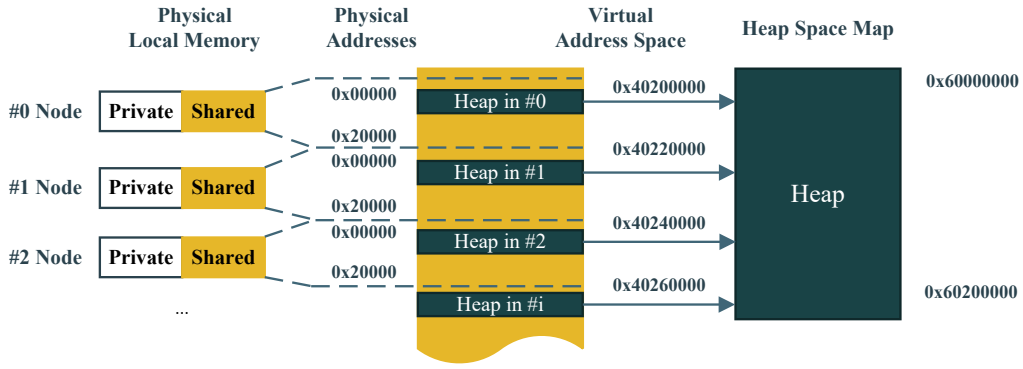
Σχήμα 4.6: Η πλατφόρμα DSM με προγραμματιζόμενους ελεγκτές μνήμης όπως παρουσιάζεται στην [51].

Χάριν στην υπηρεσία V2P, η μεταφορά ενός δυναμικού αναθέτη μνήμης από μία γενικότερη

αρχιτεκτονική είναι μία απλή και ξεκάθαρη διαδικασία. Μπορέσαμε να μεταφέρουμε στην πλατφόρμα του Σχήματος 4.6 τον αποδοτικότερο ενεργειακά αναθέτη της [39] με ελάχιστες αλλαγές στον κώδικά του. Ο αναθέτης αυτός εκλαμβάνει την σωρό ως μία μεμονωμένη, με αποτέλεσμα η ανάπτυξη εφαρμογών που προσπελάζουν πολλές και διαφορετικές μνήμες από διάφορους κόμβους να είναι απλούστερη. Δυστυχώς όμως, θα αποδείξουμε στην συνέχεια ότι οι επιδόσεις του αναθέτη αυτού είναι αρκετά χαμηλές, καθώς δεν υπάρχει κάποια επίγνωση της τοπικότητας της μνήμης: ο αναθέτης επιλέγει περιοχές μνήμης ανεξάρτητα από το ποιος επεξεργαστής έχει κάνει το συγκεκριμένο αίτημα μνήμης.

Το πρόβλημα της τοπικότητας φαίνεται να λύνεται από τον αναθέτη που παρουσιάστηκε στην εργασία [43], όπου ο χώρος της στοίβας είναι οργανωμένος σε επιμέρους στοίβες, μία ανά κόμβο, έτσι ώστε με συγκεκριμένες εντολές στον DMC (μικροκώδικας - microcode από εδώ και στο εξής) να επιταχύνονται οι διαδικασίες ανάθεσης μνήμης. Η προσέγγιση αυτή αποφέρει καλύτερες επιδόσεις από τον προηγούμενο αναθέτη, αλλά θα πρέπει να σημειωθεί επίσης ότι έρχεται και με πλεονάζοντα κόστη ανάπτυξης: Οι προγραμματιστές θα πρέπει να γνωρίζουν εκ των προτέρων ποιοι επεξεργαστές χρειάζεται να προσπελάσουν ποιες στοίβες και να ορίσουν αντίστοιχο ένα πίνακα προτεραιοτήτων ανά κόμβο.

Ουσιαστικά, αναγνωρίζουμε στην κατάσταση αυτή ένα συμβιβασμό μεταξύ ευκολίας στον προγραμματισμό (μέσω της μονής στοίβας) και αύξησης επιδόσεων (μέσω της επιτάχυνσης υλισμικού). Προκειμένου να διατηρήσουμε λοιπόν μία μονή, κατανεμημένη στοίβα, ενώ ακόμα προσφέρουμε ΔΔΜ, προτείνουμε να υλοποιήσουμε δύο νέα χαρακτηριστικά για την πλατφόρμα που εξετάζουμε: την *Αντιστοίχηση Χώρου Στοίβας (Heap Space Map)* και την *Οκνηρή Επιλογή Στοίβας (Lazy Heap Selection)*.

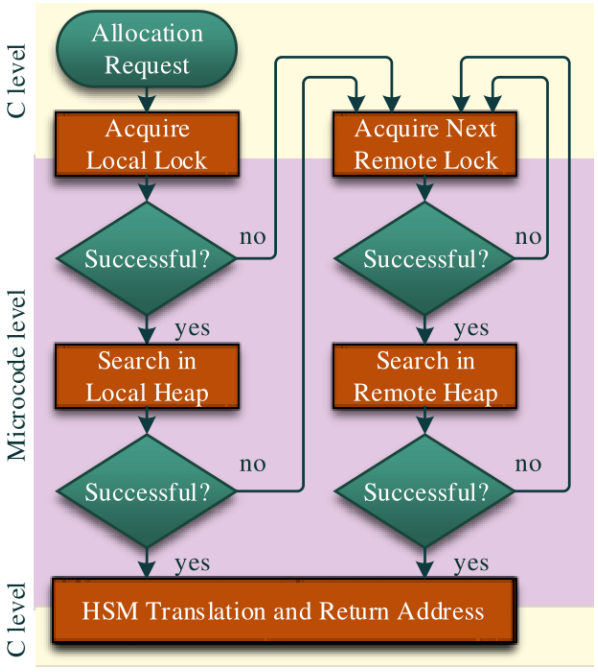


Σχήμα 4.7: Η τεχνική Heap Space Map πάνω στην υπηρεσία μετάφρασης V2P.

Η τεχνική Heap Space Map (HSM) αφορά την δημιουργία ενός δεύτερου επιπέδου εικονικών διευθύνσεων όπως φαίνεται στο Σχήμα 4.7. Το νέο σχέδιο διευθυνσιοδότησης θα πρέπει να ξεκι-

νάει υψηλότερα από αυτό της υπηρεσίας V2P (για παράδειγμα από την περιοχή `0x60000000`) και να ενώνει τις μικρότερες σωρούς που συναντώνται σε κάθε κόμβο, έτσι ώστε μία συνεχόμενη περιοχή μνήμης να είναι διαθέσιμη ως σωρός στην εφαρμογή. Η μετάφραση από την σωρό σε κόμβο λαμβάνει χώρα σε δύο επίπεδα: (α) αρχικά από την υπηρεσία HSM (που βρίσκεται μαζί με την dmmlib, σε επίπεδο C) και (β) δεύτερον από τον αντίστοιχο ελεγκτή μνήμης (μικροκώδικας υπηρεσίας V2P).

Επιπρόσθετα της HSM, προτείνουμε και μία μεθοδολογία Οκνηρής Επιλογής Στοίβας για να αποφύγουμε την χρήση στατικών πινάκων προτεραιοτήτων. Στην εργασία [43] κάθε κόμβος αποθηκεύει στην ιδιωτική περιοχή της τοπικής του μνήμης τις πιθανές σωρούς στις οποίες πρέπει να αιτηθεί προκειμένου να εξυπηρετήσει ο ίδιος ένα αίτημα μνήμης. Αν μία στοίβα είναι απασχολημένη όταν ο αναθέτης θελήσει να ελέγξει μόνο την κατάστασή της, τότε ο τελευταίος θα πρέπει να περιμένει μέχρι η πρώτη να ολοκληρώσει την προηγούμενη εργασία της. Αυτό που προτείνουμε είναι αντίθετα, να γίνεται αξιολόγηση χρήσης όλων των απομακρυσμένων σωρών ανεξαιρέτως, αφού πρώτα επιβεβαιώσουμε ότι η τοπική σωρός δεν έχει ελεύθερο χώρο για νέο αίτημα. Επιπλέον, αν ο αναθέτης δοκιμάσει να αποκτήσει το κλειδί μίας απομακρυσμένης στοίβας και αποτύχει, δε θα πρέπει να περιμένει, αλλά να προχωρήσει στην επόμενη απομακρυσμένη σωρό.



Σχήμα 4.8: Η πλήρης ροή ανάθεσης του προτεινόμενου αναθέτη.

Βασιζόμενοι στον χώρο διευθύνσεων εικονικής μνήμης που αναφέραμε παραπάνω και στον αλγόριθμο επιλογής σωρού, προτείνουμε μία ροή ανάθεσης όπως φαίνεται στο Σχήμα 4.8. Η αποδέσμευση συμβαίνει αντίστοιχα: έπειτα από μία μετάφραση που πραγματοποιεί η υπηρεσία HSM, ένα μήνυμα στέλνεται στο κατάλληλο DMC για να αποδεσμεύσει το μπλοκ μνήμης. Ο προτεινόμενος αναθέτης υλοποιεί σε επίπεδο μικροκώδικα όλες τις μικρές, εξειδικευμένες ενέργειες που απαιτούνται για την δυναμική ανάθεση και αποδέσμευση μνήμης. Οι ΠΔΕ και ο μηχανισμός με τον οποίο εισάγονται τα κλειδώματα συγχρονισμού στην υπηρεσία HSM είναι δανεισμένος από την dmmlib.

Οι κύριες διαφοροποιήσεις της δουλειάς αυτής είναι συνεπώς οι ακόλουθες: Ο προτεινόμενος αναθέτης (α) θεωρεί μία καθολική, συνεχόμενη περιοχή μνήμης για αιτήματα όπως ο αναθέτης σε C της εργασίας [39] και όχι όπως ο αναθέτης σε μικροκώδικα [43], (β) αξιοποιεί την χρήση του μικροκώδικα για τους DMC όπως ο [43], ενώ ο αναθέτης σε C καθόλου, (γ) δεν χρησιμοποιεί πίνακες προτεραιοτήτων όπως κάνει ο [43], (δ) αξιολογεί με οκνηρό τρόπο την διαθεσιμότητα μνήμης σε απομακρυσμένες σωρούς, ενώ ο αναθέτης σε μικροκώδικα πρέπει να αξιολογήσει με εξαντλητικό τρόπο μία πιθανή ανάθεση μνήμης σε μία απομακρυσμένη σωρό προτού κλειδώσει και αξιολογήσει την επόμενη και (ε) δεν χρειάζεται κάποια αλλαγή προκειμένου να χρησιμοποιηθεί σε μία διαφορετική εφαρμογή.

4.2.3 Αξιολόγηση

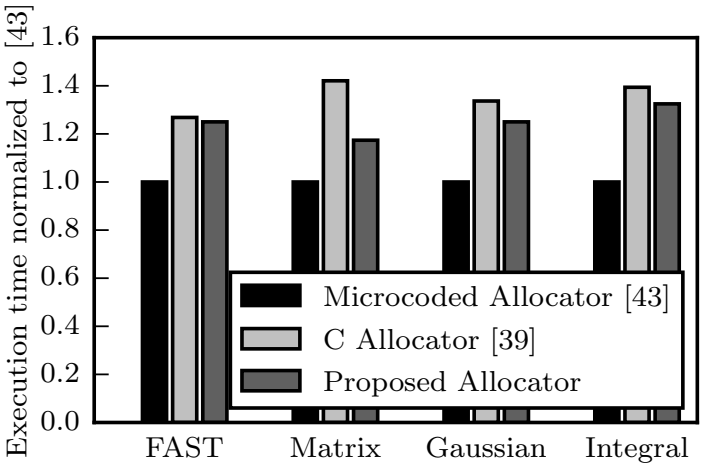
Αξιολογήσαμε την προσέγγισή μας συγκρίνοντάς την με άλλους αναθέτες στην πλατφόρμα που παρουσιάστηκε στην εργασία [51]. Καμία αλλαγή στο υλισμικό δεν έγινε: Κάθε κόμβος αποτελείται από έναν επεξεργαστή LEON3, έναν DMC και μνήμη, η οποία μπορεί να διαμοιραστεί στους κόσμους. Όλοι οι ελεγκτές DMC είναι διασυνδεδεμένοι με το Nostrum [53], ένα πολυγωνικό δίκτυο μεταγωγής πακέτων.

Για την εκτίμηση της απόδοσης του προτεινόμενου αναθέτη χρησιμοποιήσαμε ίχνη εφαρμογών προερχόμενα από τέσσερα μετροπρογράμματα που παρουσιάστηκαν στην προηγούμενη ενότητα της P2012, αλλά και στην εργασία [54]. Τα μετροπρογράμματα αυτά αποδείχτηκαν ακόμα και στην τελευταία ενδεικτικά της απόδοσης ενός αναθέτη μνήμης για ενσωματωμένα συστήματα. Πιο συγκεκριμένα αυτά είναι: (α) το FAST (Features from an Accelerated Test), ένας πυρήνας αναγνώρισης ακμών για όραση υπολογιστή, (β) ένας Γκαουσιανός πυρήνας για εφέ θολούρας, (γ) το Integral, ένας πυρήνας υπολογισμού του ολοκληρώματος στοιχείων ενός πίνακα και (δ) έναν πυρήνα πολλαπλασιασμού πινάκων. Όλα τα μετροπρογράμματα ακολουθούν τον ίδιο σχεδιασμό αφέντη—σκλάβου στον κώδικά τους: Ένας κόμβος ενεργεί ως ο ελεγκτής της πλατφόρμας και είναι υπεύθυνος για την διαχείριση των εργασιών στην πλατφόρμα, ενώ οι υπόλοιποι είναι διαθέσιμοι για την εκτέλεση των εργασιών. Η δυναμική διαχείριση μνήμης είναι

σημαντική σε τέτοιες πλατφόρμες: Αν μόνο ο κόμβος—αφέντης αναλαμβάνει την διαχείριση της μνήμης, τότε προσομοιώσεις δείχνουν ότι οι κύκλοι που ξοδεύει προκειμένου να κάνει ΔΔΜ για τα μετροπρογράμματα που προαναφέραμε, αποτελούν κατά μέσο όρο το 18.8% των συνολικών κύκλων της εφαρμογής (ελάχιστο 10.83% για την εφαρμογή FAST και μέγιστο 23.12% για την εφαρμογή του Γκαουσιανού φίλτρου).

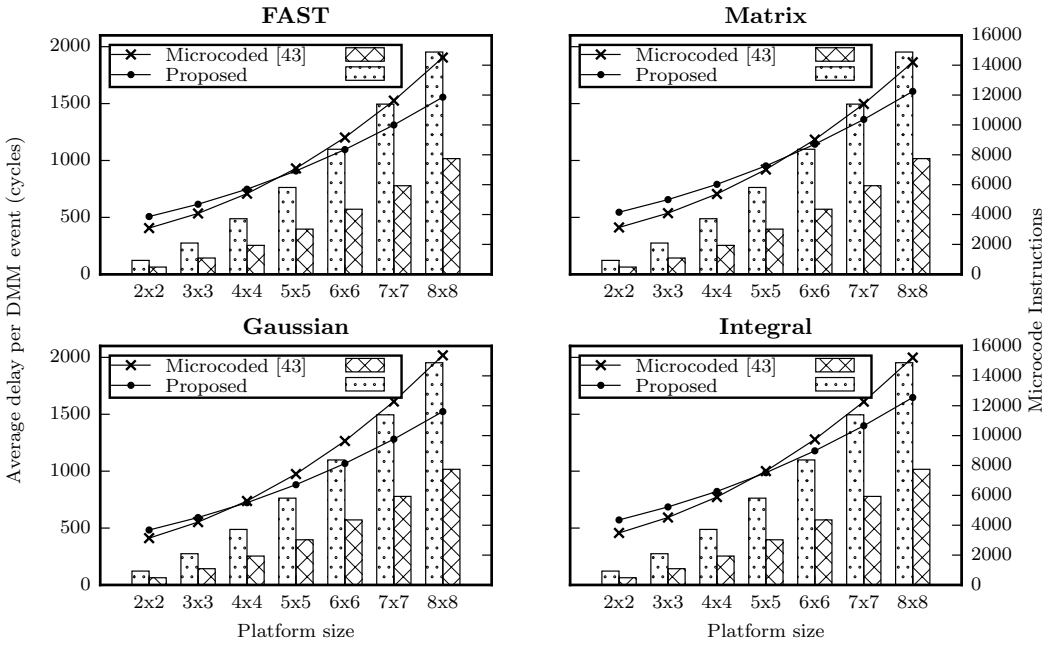
Συγκρίναμε, λοιπόν, αρχικά την απόδοση του προτεινόμενου αναθέτη με αυτή μιας λύσης καθαρά βασιζόμενης στο υλισμικό και με αυτή ενός αναθέτη καθαρά γραμμένου σε λογισμικό, πάνω σε ένα 2×2 σύστημα: (α) τον διαχειριστή δυναμικής μνήμης που είναι γραμμένος σε μικροκώδικα και έχει επίγνωση της κατανομής της μνήμης [43] και (β) έναν αναθέτη μνήμης που χρησιμοποιεί αυστηρά ιδιωτική σωρό, που επιλέχθηκε από την [39] και που δημιουργήθηκε μέσα από την dmmlib (σε γλώσσα C). Οι πολιτικές και οι μηχανισμοί που επιλέχθηκαν για τον τελευταίο αναθέτη έγιναν, επειδή ήταν κατά μέσο όρο 29% ταχύτερος σε σχέση με άλλες λύσεις γενικού σκοπού, ενώ παρουσίαζε μειωμένο κατακερματισμό. Τα δύο αυτά χαρακτηριστικά του είναι ιδιαίτερα σημαντικά για ενσωματωμένα συστήματα, όπως αυτά της μελέτης μας. Η απόδοση των συγκρινόμενων αναθετών, κανονικοποιημένη στην απόδοση του αναθέτη σε μικροκώδικα απεικονίζεται στο Σχήμα 4.9. Ο αναθέτης σε μικροκώδικα είναι ο ταχύτερος των τριών, καθώς όλες οι διαδικασίες και οι αποφάσεις γίνονται σε χαμηλό επίπεδο. Ωστόσο, η υλοποίηση αυτή στερείται μίας ΠΔΕ υψηλού επιπέδου, γεγονός που καθιστά την ενσωμάτωσή του σε εφαρμογές ιδιαίτερα δύσκολη. Ο προτεινόμενος αναθέτης είναι κατά μέσο όρο 25% πιο αργός από αυτόν σε μικροκώδικα και 10% ταχύτερος από αυτόν που υλοποιείται αποκλειστικά σε C. *Ο αναθέτης που παρουσιάζουμε σχεδιάστηκε για να παρέχει υπηρεσίες ΔΔΜ χωρίς να χρησιμοποιεί μία μονή, ιδιωτική σωρό. Ο στόχος είναι να επιτρέψει σε όλους τους κόμβους να είναι ενήμεροι για την κατάσταση της σωρού συνολικά με μικρή ποινή στην απόδοση, η οποία στην περίπτωση πολλαπλών ιδιωτικών σωρών δεν υφίσταται. Με αυτόν τον τρόπο, δείχνουμε ότι ο προτεινόμενος αναθέτης είναι ταχύτερος από έναν αναθέτη, ο οποίος σχεδιάστηκε με γνώμονα τις επιδόσεις, αλλά χωρίς να έχει επίγνωση της πλατφόρμας.*

Προχωρήσαμε έπειτα στην επιβεβαίωση της κατανεμημένης συμπεριφοράς και της κλιμακωσιμότητας του αναθέτη που παρουσιάζουμε συγκρίνοντας (α) τον μέσο όρο των κύκλων ανά γεγονός ανάθεσης και αποδέσμευσης μνήμης και (β) τον αριθμό των εντολών μικροκώδικα που απαιτούνται για την ενδοεπικοινωνία των κόμβων, με τα αντίστοιχα μεγέθη του αναθέτη σε μικροκώδικα της [43] για διάφορα μεγέθη πλατφόρμας. Ο αριθμός των κόμβων στις μετρήσεις μας κυμαίνεται από 4 (2×2) έως 64 (8×8) και είναι όλοι διασυνδεδεμένοι σε ένα πολυγωνικό δίκτυο δύο διαστάσεων. Εξ όσων γνωρίζουμε και πιστεύουμε, ο τύπος δικτύου αυτός αποτελεί μία από τις δημοφιλέστερες τοπολογίες, ενώ δύσκολα αναμένεται να ξεπεραστεί ο αριθμός των 64 κόμβων σε μία ενσωματωμένη πλατφόρμα.



Σχήμα 4.9: Σύγκριση του συνολικού χρόνου εκτέλεσης του προτεινόμενου αναθέτη με τους [43] και [39] υπό μία αυστηρά ιδιωτική σωρό για μία πλατφόρμα 2×2. Ο προτεινόμενος αναθέτης συμβιβάζεται μεταξύ απόδοσης και ευκολίας στην χρήση.

Όπως φαίνεται στο Σχήμα 4.10, ο αναθέτης σε μικροκώδικα [43] χρειάζεται κατά μέσο όρο λιγότερους κύκλους προκειμένου να εξυπηρετήσει ένα γεγονός σχετικό με μνήμη όταν το μέγεθος της πλατφόρμας είναι μικρότερο από 5×5. Ωστόσο, *όταν το μέγεθος της πλατφόρμας αυξηθεί πέραν των 25 κόμβων, ο προτεινόμενος αναθέτης χρειάζεται λιγότερους κύκλους*. Αυτό συμβαίνει επειδή ο αναθέτης σε μικροκώδικα [43] βασίζεται σε πίνακες προτεραιοτήτων. Χρησιμοποιώντας την τεχνική των πινάκων προτεραιοτήτων, κάθε κόμβος αποθηκεύει στην τοπική του μνήμη, ως μία μονά συνδεδεμένη λίστα [43], τους πιθανούς κόμβους που πρέπει να δοκιμαστούν στην περίπτωση ενός αιτήματος ΔΔΜ. Για ένα NoC 2×2 ο πίνακας αυτός περιέχει 4 εγγραφές, ενώ για ένα NoC 8×8 οι εγγραφές φτάνουν μέχρι και τις 64 ανά κόμβο. Παράλληλα, πάρα πολλοί κόμβοι έχουν τους ίδιους κόμβους ως στόχους κατά την διαδικασία ενός αιτήματος ανάθεσης ή αποδέσμευσης μνήμης. Όπως δείχνεται σε άλλα πειραματικά αποτελέσματα [43], σχεδόν το 80% του χρόνου για ΔΔΜ αναλώνεται στην απόκτηση κλειδώματος. Ως αποτέλεσμα, καθώς αυξάνεται το μέγεθος της πλατφόρμας, η διαχείριση και χρήση αυτών των πινάκων απαιτεί περισσότερους κύκλους. Αντίθετα, ο προτεινόμενος αναθέτης χρησιμοποιεί έναν ελαφρύτερο και πιο γενικό τρόπο επικοινωνίας μεταξύ των κόμβων, υποστηρίζοντας αυθαίρετα μεγέθη πλατφόρμας και κλιμακώσιμος ικανοποιητικά καθώς το μέγεθος πλατφόρμας αυξάνεται. Το Σχήμα 4.10 δείχνει ότι ο αναθέτης σε μικροκώδικα χρειάζεται κατά μέσο όρο 29% περισσότερους κύκλους για να εξυπηρετήσει ένα γενονός κάθε φορά που η πλατφόρμα μεγαλώνει, ενώ ο προτεινόμενος έχει μικρότερη αύξηση (περίπου 20% περισσότερους κύκλους). Αν και ο αναθέτης σε μικροκώ-



Σχήμα 4.10: Μέσος όρος κύκλων ανά γεγονός ΔΔΜ και αριθμός εντολών μικροκώδικα που απαιτούνται για την ενδοεπικοινωνία για διάφορα μεγέθη πλατφόρμας και εφαρμογών.

δικα εκτελεί $1.9\times$ κατά μέσο όρο λιγότερες εντολές μικροκώδικα, όπως φαίνεται με τις μπάρες και τους δεξιά άξονες του Σχήματος 4.10, η προτεινόμενη λύση αποδεικνύεται ταχύτερη, καθώς αιτείται μνήμης από καταλληλότερους κόμβους.

4.3 Συμπεράσματα

Στην ενότητα αυτήν παρουσιάσαμε έναν αποδοτικό, κλιμακώσιμο δυναμικό αναθέτη μνήμης με υποστήριξη επιτάχυνσης μέσω υλισμικού για ενσωματωμένα συστήματα πολλαπλών πυρήνων, που χρησιμοποιούν NoC. Ένα μεγάλο μέρος του κώδικα της dmmlib αντικαταστήθηκε από εντολές μικροκώδικα. Ο αναθέτης παρέχει μία συνεχή περιοχή διευθύνσεων για τα δυναμικά δεδομένα των εφαρμογών και η αναζήτηση ελεύθερου χώρου δεν είναι διακοπτόμενη. Ο αναθέτης αυτός είναι γενικός για την πλατφόρμα και δεν χρειάζεται επαναμεταγλώττιση για διαφορετικές εφαρμογές. Απαιτεί, ωστόσο, να αρχικοποιείται με το μέγεθος της μνήμης ανά πλατφόρμα και ανά κόμβο. Πειραματικά αποτελέσματα απέδειξαν (α) ότι ο αναθέτης είναι κλιμακώσιμος αρκετά ώστε να ξεπεράσει σε απόδοση αναθέτες υλοποιημένους σε χαμηλό κώδικα για μεγάλα μεγέθη πλατφόρμας, (β) ότι παρέχει δυνατότητα κατανομής της μνήμης προσφέροντας διαφο-

ρετικά μέρη της κοινόχρηστης μνήμης ως μία συνεχόμενη σωρό και (γ) ότι εξυπηρετεί αιτήματα κατά μέσο όρο 10% ταχύτερα συγκρινόμενος με αναθέτες υλοποιημένους σε υψηλό επίπεδο, χωρίς μάλιστα να μειωθεί η ευχρηστία.

5 Προσαρμόσιμοι Δυναμικοί Αναθέτες Μνήμης

Στην ενότητα αυτή θα παρουσιαστεί ένας προσαρμόσιμος αναθέτης δυναμικής μνήμης υψηλής απόδοσης. Σε αυτήν την προσέγγιση, ο αναθέτης μνήμης ρυθμίζεται αυτόματα κατά την εκτέλεση των εφαρμογών βάσει προβλέψεων της ποσότητας μνήμης που απαιτούν οι εφαρμογές. Τα πειραματικά αποτελέσματα λήφθηκαν χρησιμοποιώντας εφαρμογές από τη σουΐτα μετροπρογραμμάτων PARSEC και την dmmlib. Τα αποτελέσματα δείχνουν ότι οι προσαρμόσιμοι αναθέτες μνήμης μπορούν να βελτιώσουν προβλήματα κατακερματισμού (fragmentation) και να οδηγήσουν σε αποδοτικότερη χρήση της μνήμης.

Στη σημερινή εποχή τα πολυεπεξεργαστικά συστήματα έχουν πλέον επικρατήσει και η τάση είναι προς τη μεγιστοποίηση της χρήσης όλων των διαθέσιμων πόρων μέσα σε ένα σύστημα. Ενώ τα συστήματα μεγαλώνουν και γίνονται αρκετά πολύπλοκα, οι προγραμματιστές πρέπει να επικεντρώνονται στο να αναπτύσσουν αποδοτικά τις αντίστοιχες εφαρμογές. Προχωρημένα λειτουργικά συστήματα και γλώσσες προγραμματισμού υψηλού επιπέδου βοηθούν σε αυτήν τη διαδικασία: Τα λειτουργικά συστήματα αναλαμβάνουν τη διαχείριση των πόρων και οι γλώσσες υψηλού επιπέδου μαζί με τις διεπαφές προγραμματισμού εφαρμογών επιτρέπουν στους προγραμματιστές να εκφραστούν με ένα πιο αφηρημένο τρόπο. Χωρίς αμφιβολία, και τα δύο αυτά στοιχεία έχουν οδηγήσει στη γρηγορότερη συγγραφή αποδοτικότερων εφαρμογών.

Παρόλα αυτά, οι εφαρμογές έχουν γίνει επίσης πιο πολύπλοκες από ό,τι στο παρελθόν, καθώς χρειάζονται πολλαπλά νήματα εκτέλεσης και αυξημένες ανάγκες αποθήκευσης δεδομένων. Επιπλέον, η έντονη δυναμικότητά τους οδηγεί σε απρόβλεπτα αποτυπώματα μνήμης και παραλλαγές κατακερματισμού που είναι άγνωστες κατά το χρόνο σχεδιασμού. Η ανάπτυξη δυναμικών πολυνηματικών εφαρμογών χρησιμοποιώντας εκτιμήσεις της χειρότερης περίπτωσης ώστε να διαχειριστεί η μνήμη με ένα στατικό τρόπο, θα επέβαλε σοβαρές επιπτώσεις στο αποτύπωμα μνήμης, αλλά και στην κατανάλωση ενέργειας του συστήματος. Προκειμένου να αποφευχθούν τέτοιοι τύποι υπερεκτιμήσεων, οι προγραμματιστές ενθαρρύνονται να αξιοποιήσουν δυναμικά τη μνήμη [5]. Οι αυστηρές απαιτήσεις κατά το χρόνο εκτέλεσης μπορεί να τους ωθήσουν να γράψουν κώδικα σε χαμηλότερο επίπεδο. Στο παρελθόν οι στατικοί καταμερισμοί μνήμης ήταν ο

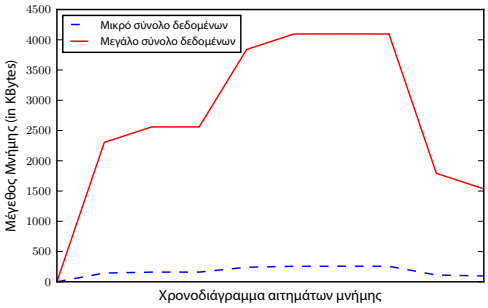
κλασσικός τρόπος για να μεγιστοποιήσει κανείς την απόδοση. Η πρακτική αυτή σήμερα θεωρείται αμφίβολη για δύο κυρίως λόγους: α) είναι πολύ σύνηθες το σύστημα να μην έχει τους ίδιους πόρους διαθέσιμους κάθε στιγμή και β) οι σύγχρονες εφαρμογές έχουν πολύ δυναμικό φόρτο εργασίας. Για το πρώτο μέρος, το λειτουργικό σύστημα ζητείται να είναι το πιο υπεύθυνο, ωστόσο οι εφαρμογές οφείλουν να προλαμβάνουν τη διάθεση ή έλλειψη πόρων ενώ οι ίδιες εκτελούνται. Αντίστοιχα, οι εφαρμογές πρέπει να προσαρμόζουν τα αιτήματά τους για πόρους έτσι, ώστε και άλλες εφαρμογές να μπορούν να χρησιμοποιούν τους ελεύθερους πόρους όταν οι πρώτες δεν τους χρειάζονται.

Το συγκεκριμένο κεφάλαιο επικεντρώνεται στη δυναμική διαχείριση μνήμης περισσότερο αποτελεσματικά εκ μέρους της εφαρμογής. Στο γενικότερο πλαίσιό της εκτελείται δυναμική διαχείριση μνήμης, όπου τα δεδομένα σωρού (heap) εδρεύουν. Οι αναλύσεις στη [50] υποδεικνύουν ότι υπάρχει ανάγκη να γίνουν οι τρέχουσες υλοποιήσεις πολυνηματικών ΔΔΜ περισσότερο προσαρμόσιμες, ώστε να μπορούν να παραμετροποιηθούν κατά την εκτέλεση της εκάστοτε εφαρμογής στο σύστημα. Όπως σημειώνεται στη [55], η προσαρμογή κατά την εκτέλεση, που ελέγχεται από λογισμικό, θα είναι η μελλοντική προσέγγιση στο σχεδιασμό συστημάτων εξαιτίας των υπερβολικών διακυμάνσεων που παρατηρούνται πλέον. Για αυτόν το λόγο, αντί να χρησιμοποιούνται αλγόριθμοι γενικού σκοπού (μία επισκόπηση αυτών των αλγορίθμων μπορεί να βρεθεί στη [5] και με συγκεκριμένα κατώφλια, θα πειράζουμε τον αναθέτη μνήμης κατά τη διάρκεια εκτέλεσης των εφαρμογών. Στόχος είναι οι εφαρμογές να έχουν ελάχιστη ποινή στις επιδόσεις τους (εξαιτίας της ύπαρξης επιπλέον μηχανισμών προσαρμογής), ενώ το σύστημα θα αναθέτει πλέον μικρότερα ποσά μνήμης στην εφαρμογή για να λειτουργήσει.

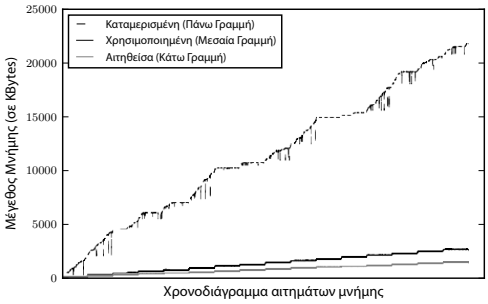
Το κεφάλαιο οργανώνεται ως εξής: Η Ενότητα 5.1 εξηγεί τη σημασία της διαχείρισης μνήμης στη σύγχρονη σημερινή εποχή των μεγάλων δεδομένων. Η Ενότητα 5.2 συζητάει τις παραμέτρους ενός αναθέτη μνήμης, οι οποίες επηρεάζουν τις επιδόσεις του και την αποδοτικότητά του. Η Ενότητα 5.3 ορίζει την προσέγγισή μας στη ρύθμιση αυτών των παραμέτρων. Η Υποενότητα 5.3.1 αναφέρει την υποδομή των πειραμάτων και παρουσιάζει τα αποτελέσματα, μέσα από τα οποία εξηγούμε τις αντισταθμίσεις για το μηχανισμό που προτείνεται. Τέλος, η Ενότητα 5.4 κλείνει την Ενότητα αυτή.

5.1 Κίνητρα για Προσαρμοστικότητα

Ως απόρροια της εποχής των μεγάλων δεδομένων (big data) στην οποία ζούμε πλέον, οι εφαρμογές έχουν σταματήσει να είναι συντηρητικές ως προς την κατανάλωση μνήμης. Το Σχήμα 5.1β δείχνει τις απαιτήσεις σε μνήμη του *blackscholes*, μίας εφαρμογής οικονομικών αναλύσεων [56]. Η χαμηλότερη, διακεκομμένη γραμμή δείχνει πόση μνήμη η εφαρμογή απαιτεί όταν ένα μικρό



(α) Τρέχοντας την ίδια εφαρμογή για διαφορετικές εισόδους δεδομένων



(β) Πρόβλημα στη δυναμική διαχείριση μνήμης: Αδυναμία επαναχρησιμοποίησης

σύνολο δεδομένων χρησιμοποιείται ως είσοδος. Η συμπαγής γραμμή δείχνει τον αριθμό αιτημάτων για μνήμη όταν η εφαρμογή χρησιμοποιεί ένα πιο ρεαλιστικό σύνολο δεδομένων. Ο αριθμός αιτημάτων για μνήμη είναι ο ίδιος και για τις δύο εισόδους, αλλά το απαιτούμενο μέγεθος μνήμης των δύο παραδειγμάτων είναι διαφορετικό. Θα πρέπει να σημειωθεί ακόμα ότι οι χρόνοι εκτέλεσης είναι επίσης διαφορετικοί, καθώς το σύστημα πρέπει να επεξεργαστεί περισσότερα δεδομένα στη δεύτερη περίπτωση, αλλά αυτό δεν εξετάζεται στην παρούσα φάση. Από το Σχήμα 5.1β μπορούμε να δούμε ότι το μέγεθος των δεδομένων στη σωρό, που αναθέτονται και αποδεσμεύονται, εξαρτάται από την είσοδο της εφαρμογής και γενικότερα τη συμπεριφορά του χρήστη.

Καθώς οι αριθμοί γίνονται μεγαλύτεροι, αποκτάει περισσότερο νόημα να ανατεθεί η διαχείριση της μνήμης σε μία μεμονωμένη, συγκεκριμένη διαδικασία. Οι δυναμικοί αναθέτες μνήμης είναι υπεύθυνοι για την οργάνωση των δυναμικά καταμερισμένων δεδομένων στη μνήμη και γενικότερα για την αποδοχή των αιτημάτων της εφαρμογής για μνήμη (για ανάθεση ή αποδέσμευση) κατά την εκτέλεση. Στην περίπτωση μιας εφαρμογής γραμμένης σε C ή C++, τα αιτήματα για μνήμη γίνονται χρησιμοποιώντας τις συναρτήσεις βιβλιοθήκης `malloc()` και `new` και ο αναθέτης πρέπει να επιστρέψει ένα δείκτη στην εφαρμογή, ο οποίος δείχνει στην περιοχή της μνήμης του καταμερισμένου χώρου. Χρησιμοποιώντας τις συναρτήσεις `free()` και `delete` αντίστοιχα, η καταμερισμένη μνήμη μπορεί να αποδεσμευτεί (για παράδειγμα, η περιοχή να σημειωθεί ως άδεια και έτσι να γίνει διαθέσιμη σε ένα μεταγενέστερο ανάθεση.

Το μέγεθος της δουλειάς που πραγματοποιεί ένας αναθέτης είναι συνήθως μεταβλητό για κάθε κλήση και δεν φέρει απαραίτητα το καλύτερο αποτέλεσμα. Η κατάσταση του χώρου μνήμης και τα μεγέθη στα αιτήματα για μνήμη αλλάζουν κατά την εκτέλεση ακόμα και της ίδιας εφαρμογής, ενώ ο αναθέτης προσπαθεί να επαναχρησιμοποιήσει το διαθέσιμο χώρο μνήμης. Ωστόσο, μερικές φορές είναι αδύνατο να επαναχρησιμοποιηθεί αποτελεσματικά η μνήμη, εξαιτίας των προηγούμενων επιλογών που έκανε ο αναθέτης. Το Σχήμα 5.1β απεικονίζει τη χειρότερη περίπτωση για ένα αναθέτη. Η διακεκομμένη γραμμή (υψηλότερα και με κόκκινο χρώμα) δείχνει το συνολικό

μέγεθος της μνήμης που ο αναθέτης διαχειρίζεται κατά τη διάρκεια εκτέλεσης της ίδιας εφαρμογής που αναφέραμε. Η μεσαία γραμμή (μπλε) παρουσιάζει το συνολικό μέγεθος των μπλοκ μνήμης που χρησιμοποιούνται αυτήν τη στιγμή και η χαμηλότερη (πράσινη) την πραγματική μνήμη που ζητήθηκε από την εφαρμογή. Η υψηλότερη και η μεσαία γραμμή δεν είναι ίσες, επειδή υπάρχουν μπλοκ μνήμης τα οποία αποδεσμεύονται μεν, αλλά δεν μπορούν να απελευθερωθούν πίσω στο σύστημα (υπάρχουν ανάμεσα σε μπλοκ που χρησιμοποιούνται αυτήν τη στιγμή), με αποτέλεσμα να παραμείνουν στη σωρό. Η διαφορά μεταξύ των γραμμών του μεγέθους μνήμης που χρησιμοποιείται και αυτού της μνήμης που ζητήθηκε δείχνει επιπλέον κόστη που εισάγει ένας αναθέτης: α) για να κρατήσει τα μεταδεδομένα, β) οι σύγχρονες εφαρμογές έχουν πολύ δυναμικό φόρτο εργασίας. Στο Σχήμα 5.1β φαίνεται ένας αναθέτης που δεν μπορεί να επαναχρησιμοποιήσει το χώρο στη μνήμη. Κατά τη διάρκεια εκτέλεσης μιας εφαρμογής, συμβαίνουν αιτήματα για μνήμη, τα οποία είναι ελαφρώς μεγαλύτερα από τα διαθέσιμα, αποδεσμευμένα μπλοκ μνήμης. Ως αποτέλεσμα, ο αναθέτης μνήμης δεν μπορεί να χρησιμοποιήσει τα προηγούμενα μπλοκ και πρέπει να ζητήσει από το σύστημα περισσότερη μνήμη. Την ίδια στιγμή, κάποια από τα μεγαλύτερα αποδεσμευμένα μπλοκ χρησιμοποιούνται για αιτήματα μικρού μεγέθους. Τελικά, η χρήση μνήμης του αναθέτη θα προσεγγίσει την εκτίμηση της χειρότερης χρήσης μνήμης, όπου η εφαρμογή διευθετεί μνήμη στατικά και ξεχωριστά κάθε μεταβλητή και δεν την απελευθερώνει παρά μόνο μετά το πέρας της εκτέλεσής της.

Φυσικά υπάρχουν τρόποι για να διαχειριστεί κανείς τέτοιες καταστάσεις με αντίτιμο επιπλέον υπολογιστικά κόστη και αυξημένη πολυπλοκότητα στον αναθέτη. Οι πιο προφανείς λύσεις είναι να βελτιωθεί η πολιτική αναζήτησης κατάλληλων μπλοκ μνήμης, ώστε να επιτευχθεί μία καλύτερη προσαρμογή του μεγέθους στα αιτήματα της μνήμης στα διαθέσιμα ελεύθερα μπλοκ μέσα στη σωρό, και / ή να υποστηρίζονται οι μηχανισμοί διαίρεσης / συγχώνευσης μπλοκ έτσι ώστε τα επόμενα αιτήματα μνήμης να βρίσκουν κατάλληλα μπλοκ ευκολότερα. Κάθε βελτίωση του αναθέτη στον τομέα χώρου μνήμης συνήθως αποφέρει μία ποινή στην ταχύτητά του. Η προσαρμοστικότητά που εισάγουμε μέσα στις πολιτικές και τους μηχανισμούς του αναθέτη προσπαθεί να βρει μία μέση λύση έτσι, ώστε όλες αυτές οι χρονοβόρες πολιτικές να λαμβάνουν χώρα μόνο όταν αυτό χρειάζεται.

Ο στόχος της δουλειάς αυτής είναι να προσφερθεί προσαρμόσιμη, δυναμική διαχείριση μνήμης για πολυνηματικές εφαρμογές που εκτελούνται σε πολυεπεξεργαστικές πλατφόρμες. Η διαχείριση μνήμης μπορεί να γίνει σε επίπεδο υλισμικού ή λογισμικού. Και οι δύο προσεγγίσεις έχουν τα πλεονεκτήματα και τα μειονεκτήματά τους, όπως αναφέρθηκε στην Ενότητα 2.2.

Στην παρούσα εργασία, επεκτείνουμε τις παραπάνω λύσεις δημιουργώντας ένα μηχανισμό πρόβλεψης του απαιτούμενου μεγέθους μπλοκ μνήμης για τα μελλοντικά αιτήματα ανάθεσης. Ο προτεινόμενος μηχανισμός υλοποιείται εφαρμόζοντας μία προσέγγιση από τη θεωρία ελέγ-

χου, χρησιμοποιώντας έναν ελεγκτή δύο σημείων για να κάνει την πρόβλεψη του μεγέθους στα αιτήματα μνήμης. Βασιζόμενοι σε δυνατότητα ζωντανής ρύθμισης της πολιτικής του αναθέτη όπως παρουσιάζεται στην [50], χρησιμοποιούμε τις προβλέψεις αυτές έτσι, ώστε ο αναθέτης να διευκολύνει περαιτέρω τα αιτήματα των εφαρμογών.

Η ιδέα, άλλωστε, της χρήσης ελεγκτών δύο (αναλογικού - ολοκληρώματος) και τριών (αναλογικού - ολοκληρώματος - παραγώγου) σημείων για τη διαχείριση κοινόχρηστων πόρων εξετάζεται στο υποσύστημα μνήμης στην [57] και στο Δίκτυο-σε-Ψηφίδα στην [58]. Παρόμοια προσέγγιση συναντάει κανείς στην [59], όπου η χωρητικότητα της λανθάνουσας μνήμης ανά εκτελούντων εφαρμογών ελέγχεται από έναν ελεγκτή δύο σημείων. Στην εργασία αυτή μία Μηχανή Πεπερασμένων Καταστάσεων (Finite State Machine) χρησιμοποιείται παράλληλα με τον αναθέτη προκειμένου να αλλάξει τις πολιτικές και τις παραμέτρους του κατά την εκτέλεση. Ενώ η προσέγγιση αυτή είναι άκρως αποδεκτή για μία υλοποίηση σε υλισμικό, στην περίπτωση που εξετάζουμε, αυτήν των ΔΔΜ εκφρασμένων σε λογισμικό, η Μηχανή αυτή καταναλώνει πάρα πολλούς κύκλους και για αυτό προτιμήθηκε ένα απλούστερο μοντέλο.

5.2 Σχεδιαστικές Αποφάσεις

Η έννοια της δυναμικής διαχείρισης μνήμης σημαίνει ότι οι εφαρμογές αναμένεται να ελευθερώσουν χώρο στη μνήμη κατά τη διάρκεια της ζωής τους. Ο αναθέτης μνήμης πρέπει να έχει πλήρη επίγνωση αυτών των γεγονότων αποδέσμευσης, έτσι ώστε να μπορεί να χρησιμοποιήσει εκ νέου τον χώρο που απελευθερώθηκε αντί να ζητήσει επιπλέον χώρο από το σύστημα. Στην ενότητα αυτή παρουσιάζουμε τις αποφάσεις που είναι σημαντικές για τη διαχείριση της μνήμης. Οι αποφάσεις αυτές εισάγουν παραμέτρους που μπορεί να τελειοποιηθεί ώστε να βελτιώσει την απόδοση της εκχώρησης μνήμης για μια συγκεκριμένη εφαρμογή.

5.2.1 Ορολογία

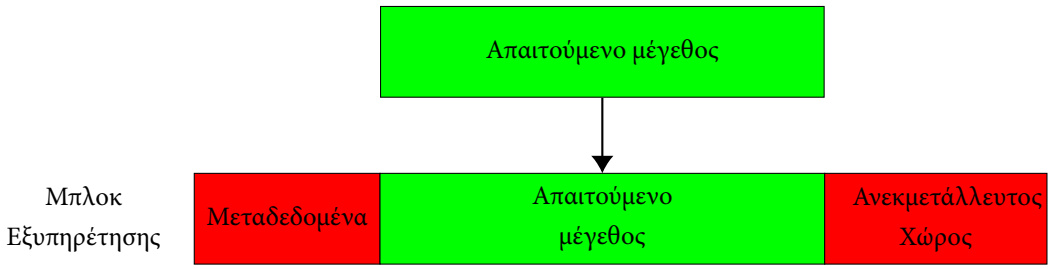
Προτού προχωρήσουμε σε λεπτομέρειες είναι αναγκαίο να αναφέρουμε ορισμένους όρους που πρόκειται να χρησιμοποιηθούν εκτενώς. Καθώς δεν υπάρχει κάποια προτυποποιημένη ορολογία στη βιβλιογραφία, η διαδικασία αυτή είναι χρήσιμη ακόμα για τους ειδικούς στη διαχείριση μνήμης για την καλύτερη κατανόηση της εργασίας αυτής.

Σωρός Η σωρός αναφέρεται στη δεξαμενή μνήμης που είναι διαθέσιμη για την ανάθεση και την αποδέσμευση δυναμικών δεδομένων (μπλοκ δεδομένων αυθαιρέτου μεγέθους, τα οποία ανατίθενται σε αυθαίρετη χρονική σειρά και τα οποία ζουν για ένα αυθαίρετο χρονικό διάστημα). Ο χώρος μνήμης της σωρού εκχωρείται από το λειτουργικό σύστημα και πλέον, στα περισσότερα

λειτουργικά συστήματα αποτελείται συνήθως από εικονικές διευθύνσεις μνήμης για να αποτραπούν καταχρήσεις. Οι δυναμικοί αναθέτες μνήμης λειτουργούν μόνο στο χώρο μνήμης της σωρού της κάθε εφαρμογής που διαχειρίζονται. Στην περίπτωση που υπάρχει ζήτηση για περισσότερη μνήμη, θα πρέπει να χρησιμοποιηθεί μία κλήση συστήματος, όπως για παράδειγμα η `shrk()` ή η `mmap()` προκειμένου να επεκταθεί η σωρός. Όπως καταλαβαίνει κανείς, η σημασία της σωρός για τους αναθέτες είναι τεράστια. Σύμφωνα με τον Berger [6], οι αναθέτες μπορούν να χωριστούν σε πέντε κατηγορίες βάσει του πώς χρησιμοποιούν τη σωρό:

1. Αναθέτες σειριακοί μονής σωρού
2. Αναθέτες συντρέχοντες μονής σωρού
3. Αναθέτες αμιγώς ιδιωτικής σωρού
4. Αναθέτες ιδιωτικής σωρού με ιδιοκτησία
5. Αναθέτες ιδιωτικής σωρού με κατώφλια

Κατακερματισμός Σωρού Ο όρος *κατακερματισμός* αφορά δύο τύπους: τον εσωτερικό και τον εξωτερικό. Ο εσωτερικός κατακερματισμός σχετίζεται με το κάθε καταμερισμένο μπλοκ και προκαλείται από τα μεταδεδομένα που ο αναθέτης πρέπει να κρατάει για το συγκεκριμένο μπλοκ μαζί με τον ελεύθερο χώρο που το μπλοκ έχει, εκτός από το ζητούμενο μέγεθος (Σχήμα 5.2).

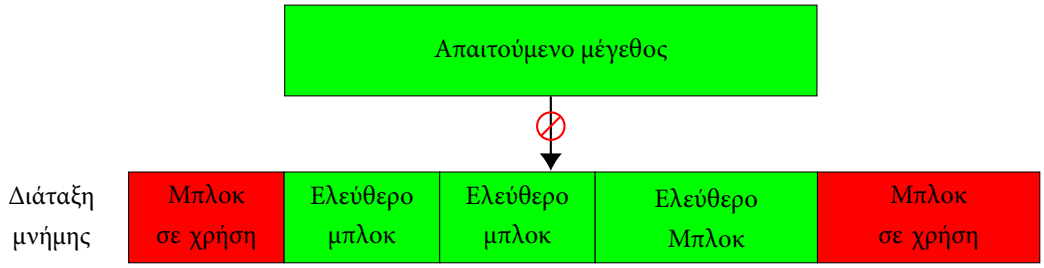


Σχήμα 5.2: Εσωτερικός κατακερματισμός

Για την ακρίβεια, ο τρέχων εσωτερικός κατακερματισμός μιας σωρού ορίζεται ως εξής:

$$\frac{\sum\{\text{καταμερισμένα μπλοκ}\} - \sum\{\text{απαιτούμενη μνήμη}\}}{\sum\{\text{καταμερισμένα μπλοκ}\}} \tag{5.1}$$

Από την άλλη πλευρά, ο εξωτερικός κατακερματισμός αναφέρεται στις περιπτώσεις που ένα αίτημα μνήμης δεν μπορεί να εξυπηρετηθεί από ελεύθερα μπλοκ μνήμης της σωρού, παρόλο που



Σχήμα 5.3: Περίπτωση εξωτερικού κατακερματισμού

υπάρχει χώρος στη μνήμη αν κάποια από τα διαθέσιμα μπλοκ συγχωνευόντουσαν. Μία τέτοια περίπτωση φαίνεται στο Σχήμα 5.3:

Αντίστοιχα λοιπόν, ο εξωτερικός κατακερματισμός ορίζεται ως εξής:

$$\frac{\sum\{\text{διαχειριζόμενα μπλοκ}\} - \sum\{\text{καταμερισμένα μπλοκ}\}}{\sum\{\text{διαχειριζόμενα μπλοκ}\}} \tag{5.2}$$

5.2.2 Σχεδιαστικές Αποφάσεις

Ο αναθέτης μπορεί να ρυθμιστεί να παίρνει αποφάσεις σχετικά με τις σχεδιαστικές του παραμέτρους και επιλογές. Ένας χώρος σχεδιασμού για δυναμικούς αναθέτες μνήμης μονο- και πολυνηματικών εφαρμογών που τρέχουν σε πολυεπεξεργαστικές πλατφόρμες παρουσιάζεται στη [18]. Βάσει αυτού του χώρου έχει αναπτυχθεί μία νέα βιβλιοθήκη δυναμικών αναθέσεων μνήμης σε γλώσσα ANSI C [60], αποκαλούμενη `dmmlib`, η οποία καλύπτει όλες τις διαφορετικές επιλογές, τόσο σε μεταξύ- (inter), όσο και ενδο- (intra) νηματικές επιλογές σχεδίασης. Επιπρόσθετα, η `dmmlib` προσφέρει προσαρμοστικές τεχνικές ανάθεσης, όπως αυτές που περιγράφονται στην αμέσως επόμενη ενότητα, την 5.3. Περιληπτικά, οι υποστηριζόμενες επιλογές σχεδίασης είναι:

- Οι μεταξύ-νημάτων αποφάσεις σχεδιασμού, τις οποίες ο σχεδιαστής μπορεί να διαλέξει προκειμένου να ρυθμίσει τον αναθέτη και είναι: η αρχιτεκτονική της σωρού (ή των στοιβών), η συνεκτικότητα δεδομένων, η απόφαση χρήσης συγκεκριμένης στοίβας, αποφάσεις αποδέσμευσης και αποφυγής κατακερματισμού.
- Οι ενδονηματικές αποφάσεις σχεδιασμού, με τις οποίες ο σχεδιαστής μπορεί να διαλέξει για να ρυθμίσει τις σωρούς που έχει καθορίσει και είναι: δομή μπλοκ, οργάνωση δεξαμενής (pool) - διάταξη, ανάθεσης και αποδέσμευση μπλοκ (πολιτικές αναζήτησης και εφαρμογής), αποφάσεις διαίρεσης και ένωσης.

Η νέα βιβλιοθήκη προσφέρει επιπλέον χαρακτηριστικά που υποστηρίζουν την προσαρμοστικότητα κατά την εκτέλεση αλλάζοντας κάποιες συγκεκριμένες παραμέτρους ενώ η εφαρμογή

εκτελείται και χρησιμοποιεί τη σωρό μέσω του αναθέτη που έχει δημιουργηθεί από την `dmmlib`.

Αποφάσεις διάταξης μπλοκ Ο χώρος μνήμη που ένας αναθέτης μνήμης χειρίζεται οργανώνεται σε μπλοκ μνήμης. Ενώ ο αναθέτης προσπαθεί να ικανοποιήσει τα αιτήματα για μνήμη, φτάνει στο σημείο που εξετάζει τα διαθέσιμα, ελεύθερα μπλοκ. Οι πιθανές πολιτικές διάταξης είναι οι εξής:

- *First-In First-Out (FIFO).* Τα μπλοκ μνήμης που αποδεσμεύτηκαν πρώτα είναι αυτά που εξετάζονται πρώτα αν ταιριάζουν στο μέγεθος ανάθεσης του επόμενου αιτήματος.
- *Last-In First-Out (LIFO).* Τα μπλοκ μνήμης που αποδεσμεύτηκαν πιο πρόσφατα, δηλαδή τα τελευταία, είναι τα πρώτα υποψήφια σε νέα αιτήματα ανάθεσης.
- *Διεύθυνση.* Τα μπλοκ μνήμης που αρχίζουν σε χαμηλότερη διεύθυνση μνήμης έχουν προτεραιότητα στα αιτήματα ανάθεσης.
- *Μέγεθος.* Τα μπλοκ διατάσσονται σε λίστα με βάση το μέγεθός τους, από το μικρότερο στο μεγαλύτερο.

Η απόδοση της κάθε πολιτικής εξαρτάται αρκετά από την εφαρμογή. Υποστηρίζεται συχνά ότι ακόμα και απλές διατάξεις, όπως αυτή της LIFO, αποδίδουν εξίσου καλά όσο και πολυπλοκότερες [5]. Σε κάποιες περιπτώσεις όμως, μπορεί να υπάρχουν ρυθμίσεις του συστήματος που να δίνουν προβάδισμα σε συγκεκριμένες πολιτικές διατάξεις, π.χ. να ευνοούν μία πολιτική χρήσης του λιγότερο χρησιμοποιημένου ή το αντίστροφο. Ωστόσο, στο πεδίο εφαρμογής ενός αναθέτη μνήμης για κανονικές εφαρμογές, τα πλεονεκτήματα αυτά θεωρούνται ασήμαντα.

Η πολιτική διάταξης θεωρείται κανονικά μία στατική πολιτική, δηλαδή είναι μία απόφαση η οποία λαμβάνεται ενώ σχεδιάζεται ο αναθέτης. Μπορεί να υπάρχουν, ωστόσο, περιπτώσεις εφαρμογών όπου διαφορετικές πολιτικές διάταξης είναι καλύτερες από άλλες σε συγκεκριμένα, χρονικά διαστήματα. Υπό αυτήν την έννοια μία επαναδιάταξη ίσως να βελτιώσει την επίδοση του αναθέτη. Για την εργασία αυτή θεωρούμε την πολιτική διάταξης ως μία στατική απόφαση.

Πολιτική Αναζήτησης και Εφαρμογής Η πολιτική αναζήτησης επηρεάζει στο μεγαλύτερο βαθμό από τους υπόλοιπους παράγοντες την ακρίβεια και τις επιδόσεις του αναθέτη. Μία ενδελεχής αναζήτηση πάνω στα διαθέσιμα μπλοκ μνήμης μπορεί να βρει τον ιδανικό υποψήφιο σε βάρος του χρόνου. Από την αντίθετη μεριά, μία πολύ γρήγορη, άπληστη προσέγγιση μπορεί να προσφέρει στην εφαρμογή ένα μπλοκ μνήμης με τον γρηγορότερο τρόπο, αλλά θυσιάζει πολύτιμο χώρο στη μνήμη.

Η καλύτερη από άποψη κατανάλωσης μνήμης πολιτική είναι γνωστή ως *best-fit*. Η πολιτική αυτή δίνει εγγυημένα το καλύτερο διαθέσιμο μπλοκ, αλλά μπορεί να χρειαστεί να διασχίσει ολόκληρη τη λίστα των διαθέσιμων μπλοκ μνήμης, εκτός και αν βρει νωρίτερα ένα μπλοκ ιδανικού μεγέθους ή στη λίστα τα μπλοκ διατάσσονται ανά μέγεθος. Το κύριο πλεονέκτημα της είναι η σωστότερη χρήση της μνήμης και ως συνέπεια ο χαμηλότερος κατακερματισμός της τελευταίας. Ωστόσο, στα μειονεκτήματά της συγκαταλέγονται οι χαμηλές αποδόσεις, καθώς στις περισσότερες περιπτώσεις η αναζήτηση αυτή είναι εξαντλητική.

Την πολιτική αυτή μπορούμε να την επηρεάσουμε με την εισαγωγή μιας παραμέτρου, την οποία θα ονομάσουμε `search_percentage` (ποσοστό αναζήτησης). Η παράμετρος αυτή θα μπορεί να αλλάξει τη συμπεριφορά της πολιτικής και να κάνει την αναζήτηση να σταματάει εφόσον ένα συγκεκριμένο ποσοστό μπλοκ της λίστας έχουν εξερευνηθεί και υπάρχει κάποιο που να εφαρμόζει ήδη. Με αυτόν τον τρόπο μπορούμε να διαλέξουμε ανά πάσα στιγμή στην εφαρμογή τι χρειάζεται, επιδόσεις ή σωστότερη χρήση της μνήμης.

Η πιο άπληστη πολιτική αναζήτησης λέγεται *first-fit*: Διανέμει στην εφαρμογή το πρώτο διαθέσιμο μπλοκ που έχει τουλάχιστον το αιτούμενο μέγεθος. Η συνάρτηση της first-fit διασχίζει τις λίστες που αναφέρουν τα ελεύθερα μπλοκ, οι οποίες μπορεί να είναι λίστες μπλοκ μεταβλητού μεγέθους ή και σταθερού [5], και αν ένα μπλοκ ίσου ή μεγαλύτερου ελεύθερου χώρου βρεθεί, τότε βγαίνει από τη λίστα και ο δείκτης του μπλοκ αυτού επιστρέφεται στην εφαρμογή. Αν δεν υπάρχει τέτοιο μπλοκ, τότε η καθολική σωρός (αν υπάρχει τέτοια) προσπελάζεται, ενώ ως τελευταία προσφυγή γίνεται μία κλήση στο σύστημα για επιπλέον μνήμη.

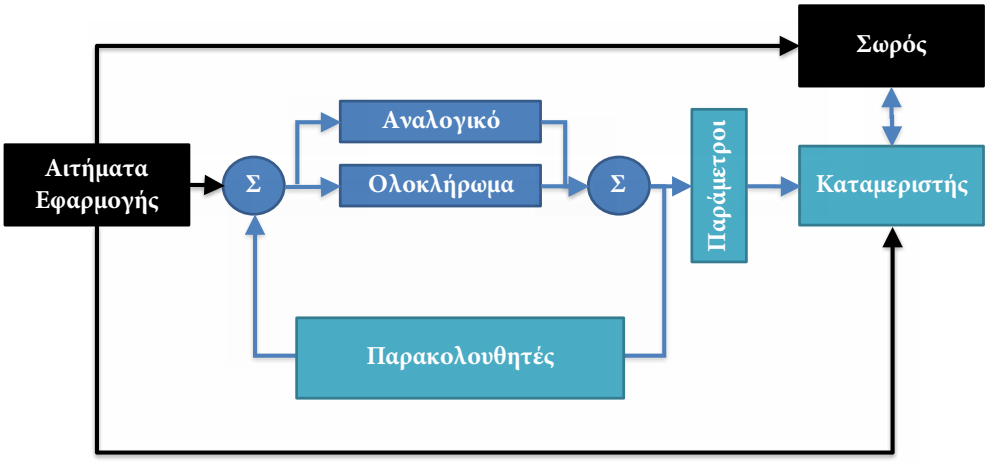
Και οι δύο οι προσεγγίσεις έχουν πλεονεκτήματα και μειονεκτήματα. Η first-fit συμπεριφέρεται πολύ γρήγορα σε σύγκριση με τον ανταγωνισμό, αλλά μπορεί επίσης να προκαλέσει τα χειρότερα προβλήματα ανάθεσης. Από την άλλη πλευρά, η προσπάθεια εύρεσης του πιο εφαρμοστού ελεύθερου μπλοκ μπορεί να απαιτεί πολλούς υπολογιστικούς πόρους χωρίς μάλιστα να εγγυηθεί ότι υπάρχει ένα διαθέσιμο εφαρμοστό μπλοκ.

Μία ενδιαφέρουσα εναλλακτική πολιτική είναι η *good-fit*. Η good-fit μπορεί να συμπεριφερθεί ακριβώς όπως η best-fit, αλλά μπορεί επίσης να εξυπηρετήσει μπλοκ μεγαλύτερου μεγέθους χωρίς να διασχίσει ολόκληρη τη λίστα βάσει ενός συγκεκριμένου λόγου (ratio), το οποίο ονομάζουμε *goodness factor*. Για παράδειγμα, στην περίπτωση που γίνει ένα αίτημα μνήμης των 80 bytes και το goodness factor είναι 80%, ένα μπλοκ των 100 bytes θα μπορούσε να θεωρηθεί ιδανικό για το αίτημα αυτό.

5.3 Προσαρμοστικότητα

Στην υποενότητα αυτή εξηγούμε πώς μπορούμε να τροποποιήσουμε ένα αναθέτη μνήμης κατά το χρόνο εκτέλεσής του, πώς να εκτιμώνται τα μεγέθη των μπλοκ και τέλος πώς μπορούμε να χρησιμοποιήσουμε την πληροφορία αυτή για να ρυθμίσουμε τον αναθέτη.

Η κεντρική ιδέα είναι ότι επηρεάζουμε την ταχύτητα και την κατανάλωση μνήμης πειραματιζόμενοι με τον goodness factor ενός αναθέτη good-fit. Ανάλογα με τις τρέχουσες ανάγκες για μνήμη της εφαρμογής και την τρέχουσα κατάσταση του χώρου μνήμης, η «αυστηρότητα» της πολιτικής αναζήτησης τροποποιείται. Μία υψηλή τιμή στον goodness factor χαλαρώνει τις απαιτήσεις σε κόστος ενός μεγαλύτερου κατακερματισμού.



Σχήμα 5.4: Βασική αρχιτεκτονική πρόβλεψης

Προκειμένου να ρυθμιστεί ο goodness factor, προσπαθούμε να προβλέψουμε το μέγεθος των μελλοντικών αιτημάτων μνήμης. Υπάρχουν πολλές προσεγγίσεις εκτέλεσης αυτής της πρόβλεψης, κυρίως με ελεγκτές PID και φίλτρα Kalman [61]. Στην παρούσα εργασία χρησιμοποιούμε έναν ελεγκτή δύο σημείων PI, επειδή είναι ένα σχετικά εύκολο σχέδιο προς υλοποίηση και τα πιο σύνθετα συστήματα, λόγου χάρη ένα φίλτρο Kalman, δεν εγγυώνται μία καλύτερη πρόβλεψη των μελλοντικών μεγεθών στα αιτήματα μνήμης. Οι ελεγκτές με αναλογική και ολοκληρωτική δράση (Proportional Integral - PI) κλειστού βρόχου (closed loop) χρησιμοποιούνται σε εφαρμογές κυμαινόμενες από έλεγχο θερμοκρασίας έως και εκλεπτυσμένα ρομπότ διαστήματος. Ο ελεγκτής παίρνει κάποια συστεμική συμπεριφορά ως προδιαγραφή του, παρακολουθεί το σύστημα και εφαρμόζει τις απαραίτητες διορθώσεις σε μία εκτέλεση κλειστού βρόχου, ώστε να

επιτύχει την απαιτούμενη συμπεριφορά.

Ένας τέτοιος ελεγκτής θα μπορούσε να είχε και διαφορική (differential) δράση, αλλά για την περίπτωση του αναθέτη μνήμης προτιμήσαμε να την παραλείψουμε. Η διαφορική δράση θα επέφερε λίγη αύξηση στην πολυπλοκότητα του αναθέτη, αλλά θα δημιουργούσε θέματα εξαιτίας της υψηλής δυναμικότητας που ενδεχομένως να χρειάζεται να έχουν οι προβλέψεις του ελεγκτή. Αν η διαφορά μεταξύ του μεγέθους στο προηγούμενο αίτημα και στο τρέχον είναι μεγάλη, τότε ο διαφορικός όρος θα οδηγούσε σε αστάθεια τη διαδικασία και λανθασμένες τιμές θα δίνονταν στις παραμέτρους του αναθέτη. Για αυτόν το λόγο, διαλέξαμε να μη συμπεριλάβουμε το διαφορικό όρο και να διατηρούσε την αναπαράσταση της εξόδου ελέγχου ως:

$$\text{PI έξοδος} = K_p * \text{σφάλμα} + K_i * \sum \text{σφάλμα} \tag{5.3}$$

Ως ορίζουμε την παρέκκλιση του απαιτούμενου μεγέθους από το προβλεπόμενο. Βασιζόμενη στο τρέχον σφάλμα (αναλογικό μέρος) και στο συνολικό σφάλμα των προηγούμενων μετρήσεων (ολοκληρωτικό μέρος), η έξοδος ελέγχου διορθώνεται για να συγκλίνει η έξοδος του συστήματος με την έξοδο PI. Οι τιμές K_p και K_i είναι σταθερές που καθορίζουν τα βάρη των μερών που προαναφέραμε. Η διαμέτρηση (calibration) του ελεγκτή PI γίνεται είτε εμπειρικά, είτε θεωρητικά. Καθώς πολλοί ελεγκτές PI ρυθμίζονται χειροκίνητα ακόμα και στο πεδίο της θεωρίας ελέγχου [62], πράξαμε αντίστοιχα: δοκιμάσαμε αρκετά πειράματα φόρτου εργασίας της μνήμης δυναμικά και καταλήξαμε στις τιμές $(K_p, K_i) = (0.1, 0.3)$, πράγμα που δείχνει την προτίμηση στη συνολική ιστορία προβλέψεων σε σχέση με το πιο πρόσφατο αίτημα.

Το Σχήμα 5.4 απεικονίζει τον ελεγκτή PI συνδεδεμένο με τον αναθέτη. Τα αιτήματα εφαρμογής δίνονται στον αναθέτη μέσω του ελεγκτή. Ο ελεγκτής υπολογίζει το αναλογικό και ολοκληρωτικό μέρος και βγάζει ως έξοδο το άθροισμά του. Το αποτέλεσμα χρησιμοποιείται στη συνέχεια ως ανάδραση στην είσοδο του ελεγκτή και ως είσοδο στις παραμέτρους του αναθέτη. Το μέγεθος του τρέχοντος αιτήματος μνήμης τροφοδοτείται επίσης στις παραμέτρους, όπου θα δημιουργηθεί τελικά το προβλεπόμενο μέγεθος ως το άθροισμα του τρέχοντος μεγέθους και της εξόδου του ελεγκτή. Οι παράμετροι ελέγχουν τον goodness factor, όπως ορίζεται στον Αλγόριθμο 1, έτσι ώστε το τρέχον αίτημα να εξυπηρετηθεί από ένα μπλοκ μνήμης που έχει μέγεθος μέχρι το προβλεπόμενο. Τελικά, ο αναθέτης βρίσκει το μπλοκ αυτό, επιστρέφει στην εφαρμογή τη διεύθυνσή του και η ίδια το χρησιμοποιεί απευθείας από τη σωρό.

Όπως αναφέρεται στην Ενότητα 5.2.2 ο goodness factor επηρεάζει την αυστηρότητα των αλγορίθμων αναζήτησης και εφαρμογής, κάνοντας έτσι ευκολότερη ή δυσκολότερη την εύρεση ενός μπλοκ κατάλληλου να εξυπηρετήσει το αίτημα ανάθεσης. Αυτό έχει άμεσες συνέπειες στις επιδόσεις του αναθέτη και την κατανάλωση της μνήμης όπως βλέπουμε στη συνέχεια.

Αλγόριθμος 1 Ρύθμιση του goodness_factor

αν requested size $>$ predicted size **τότε**
    goodness factor $\leftarrow 1.0$
διαφορετικά
    goodness factor $\leftarrow$ απαιτούμενο μέγεθος/προβλεπόμενο μέγεθος
τέλος αν

5.3.1 Πειράματα

Προκειμένου να αξιολογηθεί η αποτελεσματικότητα των αναθέσεων που παρουσιάζουμε, χρησιμοποιήσαμε εφαρμογές από τη σουΐτα μετροπρογραμμάτων Princeton Application Repository for Shared-Memory Computers (PARSEC) [56], όπως επίσης και το μετροπρόγραμμα Larson [30]. Όλα τα πειράματα έγιναν πάνω σε ένα περιβάλλον Linux με πολυπύρηνους επεξεργαστές. Συγκεκριμένα, ο επεξεργαστής που χρησιμοποιήθηκε ήταν ένας AMD Phenom II X4 965 και το σύστημα ήταν εξοπλισμένο με 8 GB μνήμης RAM. Το περιβάλλον Linux ήταν πλήρως 64-bit και κάθε μετροπρόγραμμα πέρασε από τον GCC με υποστήριξη pthreads. Τέλος, όλα τα μετροπρογράμματα είχαν την άδεια να χρησιμοποιήσουν μέχρι και 4 νήματα, καθώς ο επεξεργαστής διαθέτει 4 πυρήνες. Όλες οι εφαρμογές που δοκιμάστηκαν, και αυτές από το PARSEC, και το Larson, επιδεικνύουν εκτενή χρήση δυναμικής ανάθεσης και αποδέσμευσης μνήμης.

Το PARSEC αποτελείται από πολυνηματικές εφαρμογές που εστιάζουν σε αναδυόμενα φορτία εργασίας και είναι σχεδιασμένο να είναι αντιπροσωπευτικό της επόμενης γενιάς προγραμμάτων κοινόχρηστης μνήμης για πολυεπεξεργαστικά συστήματα. Τα προγράμματα αυτά προέρχονται από διαφορετικές περιοχές μεταξύ τους, όπως ρομποτική όραση, κωδικοποίηση βίντεο, οικονομικές αναλύσεις και επεξεργασία εικόνας. Όχι όλα τα μετροπρογράμματα που συμπεριλαμβάνονται χρησιμοποιούν ΔΔΜ. Για τις ανάγκες εκτίμησης επιλέξαμε τις εφαρμογές *bodytrack* και *swaptions*. Πιο συγκεκριμένα, η εφαρμογή *bodytrack* πραγματοποιεί παρακολούθηση του σώματος ενός προσώπου και η *swaptions* αναφέρεται στην τιμολόγηση ενός τύπου χαρτοφυλακίου. Και τα δύο αυτά μετροπρογράμματα εκτελέστηκαν με το αντίστοιχο σετ εισόδου τους *simlarge*, το οποίο σκοπεύει στην ανάλυση απόδοσης με την εξομοίωση των προαναφερθέντων εφαρμογών.

Το μετροπρόγραμμα Larson είναι μία εφαρμογή που εξομοιώνει φορτία εργασίας απαιτητικά σε μνήμη, τα οποία συναντούνται σε εξυπηρετητές: κάθε νήμα αναθέτει και αποδεσμεύει αντικείμενα και μετά μεταφέρει κάποια από τα αντικείμενα αυτά σε άλλα νήματα προκειμένου να ελευθερωθούν. Ένας αριθμός νημάτων δημιουργείται επαναληπτικά για να αναθέσει και να ελευθερώσει σε τυχαία σειρά 10000 μπλοκ, το μέγεθος των οποίων κυμαίνεται από 32 έως 1000 bytes,

ενώ ένας τυχαίος αριθμός από αυτά αφήνεται να ελευθερωθεί από ένα επόμενο νήμα. Η διάρκεια του μετροπρογράμματος είναι 30 δευτερόλεπτα.

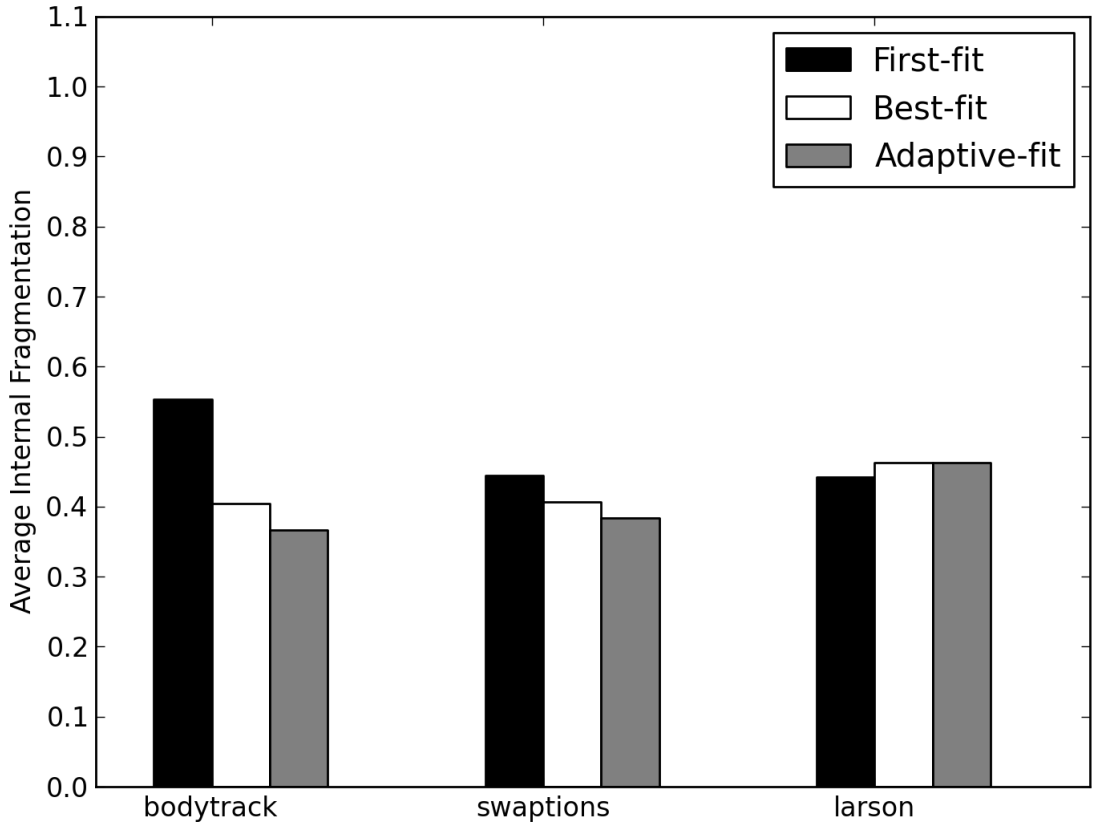
Προκειμένου να πειραματιστούμε με τις διάφορες αποφάσεις ανάθεσης μνήμης και τις σχετικές τους παραμέτρους, χρησιμοποιήσαμε την *dmmlib*. Η *dmmlib* είναι ένα πλαίσιο ανάπτυξης δυναμικών αναθέσεων μνήμης γραμμένο σε C99, το οποίο επιτρέπει όλες τις αποφάσεις που αναφέρθηκαν στην Ενότητα 5.2. Επιτρέπει στους προγραμματιστές να αναπτύξουν προσαρμοσμένους αναθέτες επιλέγοντας τα επιθυμητά χαρακτηριστικά και πολιτικές. Το πλαίσιο παρέχει προσαρμοσμένες υλοποιήσεις για δυναμικό ανάθεση μνήμης, επανάθεση και αποδέσμευση, οι οποίες αντικαθιστούν τις κλασσικές κλήσεις στη κύρια βιβλιοθήκη του συστήματος, δηλαδή τη `malloc()`, τη `realloc()`, τη `calloc()` και τη `free()`. Οι συναρτήσεις που δημιουργούνται μπορεί να είναι εντελώς αυτόνομες αν ο προγραμματιστής γνωρίζει εξαρχής το συνολικό μέγεθος και τη διευθυνσιοδότησης της μνήμης ή της σωρού που πρόκειται να διαχειριστεί ο αναθέτης, ή μπορεί να ενεργοποιήσει κλήσεις (λειτουργικού) συστήματος για να προσπελάσει περισσότερο χώρο μνήμης ή να επιστρέψει στο σύστημα, όπως για παράδειγμα η `sbrk()` ή η `mmap()`. Η υποστήριξη πολυνηματικής δυναμικής διαχείρισης μνήμης χρησιμοποιώντας POSIX mutexes, οι υλοποιήσεις αρκετών οργανώσεων μπλοκ (με μονά ή διπλά συνδεδεμένες λίστες κ.λπ.), η οργάνωση των μπλοκ σε λίστες ανεξάρτητου ή σταθερού μεγέθους [5], μία ποικιλία αλγορίθμων αναζήτησης μπλοκ, η δυνατότητα συγχώνευσης και διαίρεσης μπλοκ μνήμης για την αποτροπή υπερβολικού κατακερματισμού συνθέτουν κάποια από τα χαρακτηριστικά που είναι διαθέσιμα στους προγραμματιστές.

Κάποια από τα χαρακτηριστικά αυτά μπορούν να αλλάξουν μόνο κατά το σχεδιασμό, επομένως ο προγραμματιστής θα πρέπει να τα έχει επιλέξει προτού δημιουργήσει τον εκτελέσιμο κώδικα της βιβλιοθήκης. Αντίστοιχα, κάποια χαρακτηριστικά μπορούν να αλλάξουν κατά τη διάρκεια χρήσης της βιβλιοθήκης χωρίς την ανάγκη επανασύνθεσής της. Τα περισσότερα από τα χαρακτηριστικά πάντως ποσοτικοποιούνται και αν ο προγραμματιστής επιλέξει να αλλάζουν οι τιμές αυτές κατά την εκτέλεση, τότε δημιουργούνται παράμετροι που μπορούν να αλλάξουν από την εφαρμογή που χρησιμοποιεί τον αναθέτη. Στο σημείο αυτό θα πρέπει να σημειωθεί ότι υπάρχει μία αντιστάθμιση μεταξύ μεγέθους κώδικα και προσαρμοστικότητας όσο ο προγραμματιστής επιλέγει να εφαρμόσει χαρακτηριστικά και πολιτικές στατικά ή όχι. Τέλος, η βιβλιοθήκη υποστηρίζει ένα πλούσιο σετ από στατιστικά, τα οποία είναι διαθέσιμα κατά τη διάρκεια εκτέλεσης σε βάρος ένα επιπλέον κόστος στις δομές των μεταδεδομένων του αναθέτη.

Για τις ανάγκες μελέτης της προσαρμοστικότητας, δημιουργήσαμε τρεις αναθέτες:

1. **Πρώτης Εφαρμογής** Αυτός είναι ένας πολύ γρήγορος, άπληστος αναθέτης.

2. **Καλύτερης Εφαρμογής** Ο αναθέτης αυτός προσπαθεί να βρει το βέλτιστο μπλοκ για κάθε



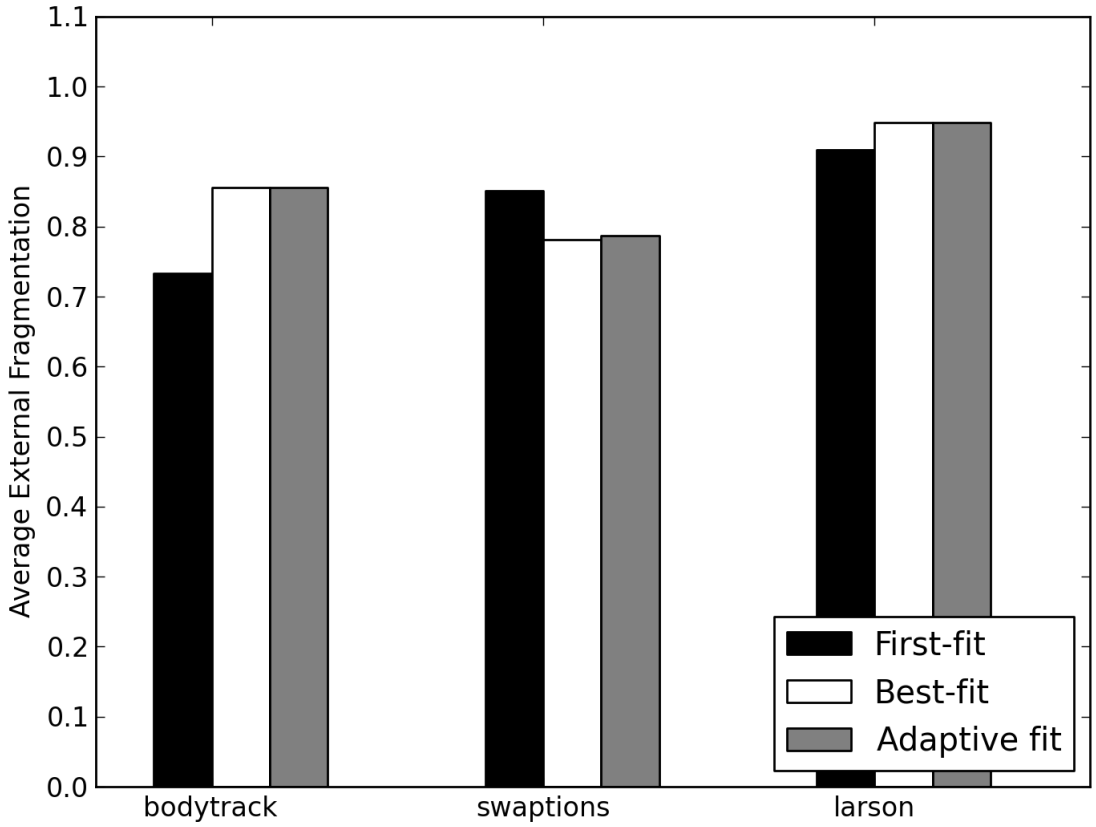
Σχήμα 5.5: Μέσος εσωτερικός κατακερματισμός ανά μετροπρόγραμμα και αναθέτη

αίτημα, αλλά δεν μπορεί να αλλάξει τη διάταξη στη μνήμη μόλις τα μπλοκ δημιουργηθούν.

3. **Προσαρμοσμένης Εφαρμογής** Σε αυτήν την εκδοχή ο αναθέτης θα χρησιμοποιεί την πρόβλεψη που αναφέραμε στις παραπάνω ενότητες, προκειμένου να ρυθμίζει τις παραμέτρους του αναθέτη ενώ οι εφαρμογές εκτελούνται.

Οι τρεις αναθέτες που παρήχθησαν από την dmmlib, έχουν εκτελέσιμο κώδικα παρόμοιου μεγέθους. Η υλοποίηση του ελεγκτή PI και η υποστήριξη αλλαγής παραμέτρων κατά την εκκίνηση δε φάνηκε να δημιουργούν ιδιαίτερο κόστος στο μέγεθος του κώδικα.

Το Σχήμα 5.5 δείχνει τον μέσο εσωτερικό κατακερματισμό που παρατηρείται στα τρία μετροπρογράμματα για τους τρεις διαφορετικούς αναθέτες. Σε όλες σχεδόν τις περιπτώσεις ο αναθέτης προσαρμοσμένης εφαρμογής προκάλεσε το μικρότερο εσωτερικό κατακερματισμό. Η μόνη εξαίρεση ήταν στο μετροπρόγραμμα Larson, όπου εμφανίστηκαν ίδια ποσοστά με τον αναθέτη καλύτερης εφαρμογής και ο αναθέτης πρώτης εφαρμογής νίκησε εξαιτίας του μεγάλου αριθμού αιτημάτων που όλοι οι αναθέτες όφειλαν να εξυπηρετήσουν. Σε τέτοιες περιπτώσεις, αν το

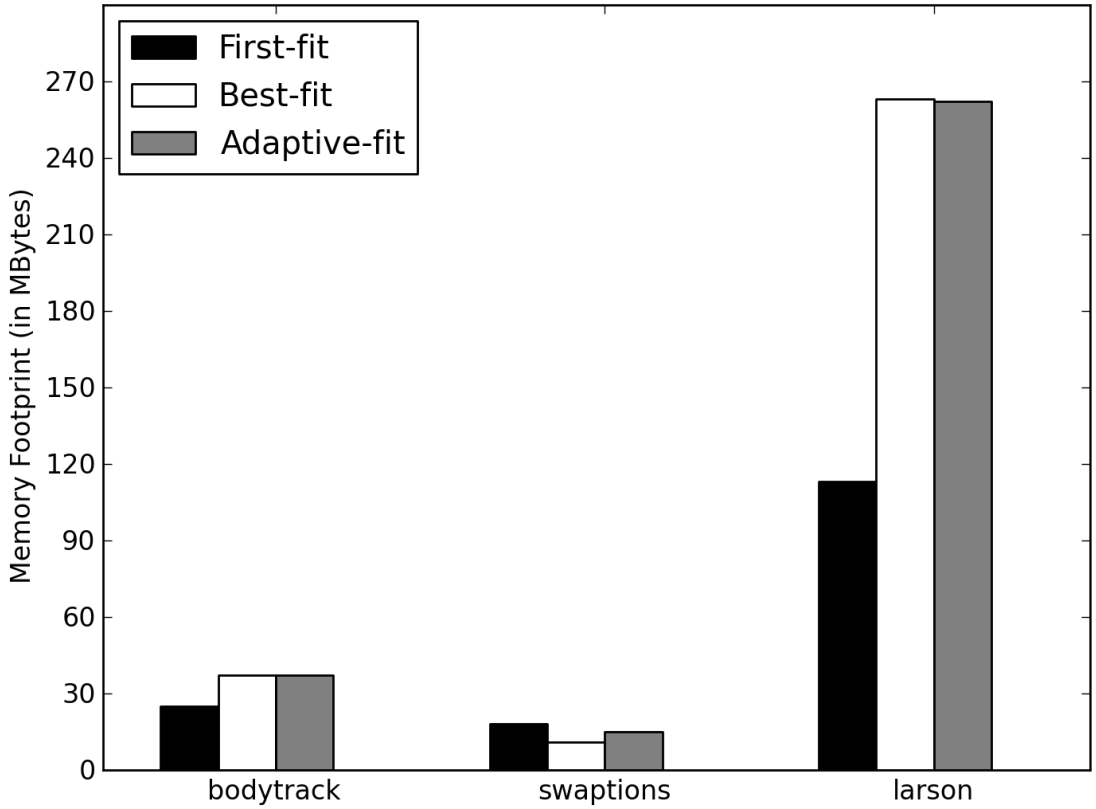


Σχήμα 5.6: Μέσος εξωτερικός κατακερματισμός ανά μετροπρόγραμμα και αναθέτη

καλύτερο μπλοκ χρησιμοποιηθεί αντί του πρώτου διαθέσιμου, είναι πιθανό να προκληθεί μεγαλύτερος εσωτερικός κατακερματισμός στο επόμενο αίτημα μνήμης, καθώς τα μελλοντικά μπλοκ μπορεί να είναι καταλληλότερα από το μπλοκ που ήταν μόλις ανατέθηκε στο προηγούμενο αίτημα μνήμης. Παρόλα αυτά, η διαφορά αναφορικά του εσωτερικού κατακερματισμού μεταξύ των διαφορετικών αναθέσεων σε τέτοιες περιπτώσεις θεωρείται ελάχιστη.

Το Σχήμα 5.6 δείχνει το μέσο εξωτερικό κατακερματισμό, όπου η προσαρμοσμένη εφαρμογή συμβαδίζει με την καλύτερη εφαρμογή. Θα πρέπει να σημειωθεί ότι η πολιτική πρώτης εφαρμογής φαίνεται να είναι πιο αποτελεσματική σε περιπτώσεις όπου τα απαιτούμενα μεγέθη είναι παρόμοια. Σε αυτήν την περίπτωση, ο αναθέτης δεν προκαλεί τεράστιο εξωτερικό κατακερματισμό, επειδή ο ίδιος μπορεί να επαναχρησιμοποιήσει τα μπλοκ που έχουν προηγουμένως αποδεσμευτεί. Έτσι, ο μηχανισμός διαίρεσης συνήθως δεν επικαλείται, ο οποίος έχει μεγάλη συμβολή στον εξωτερικό κατακερματισμό.

Το Σχήμα 5.7 δείχνει την κατανάλωση μνήμης κάθε αναθέτη. Ο αναθέτης πρώτης εφαρμογής



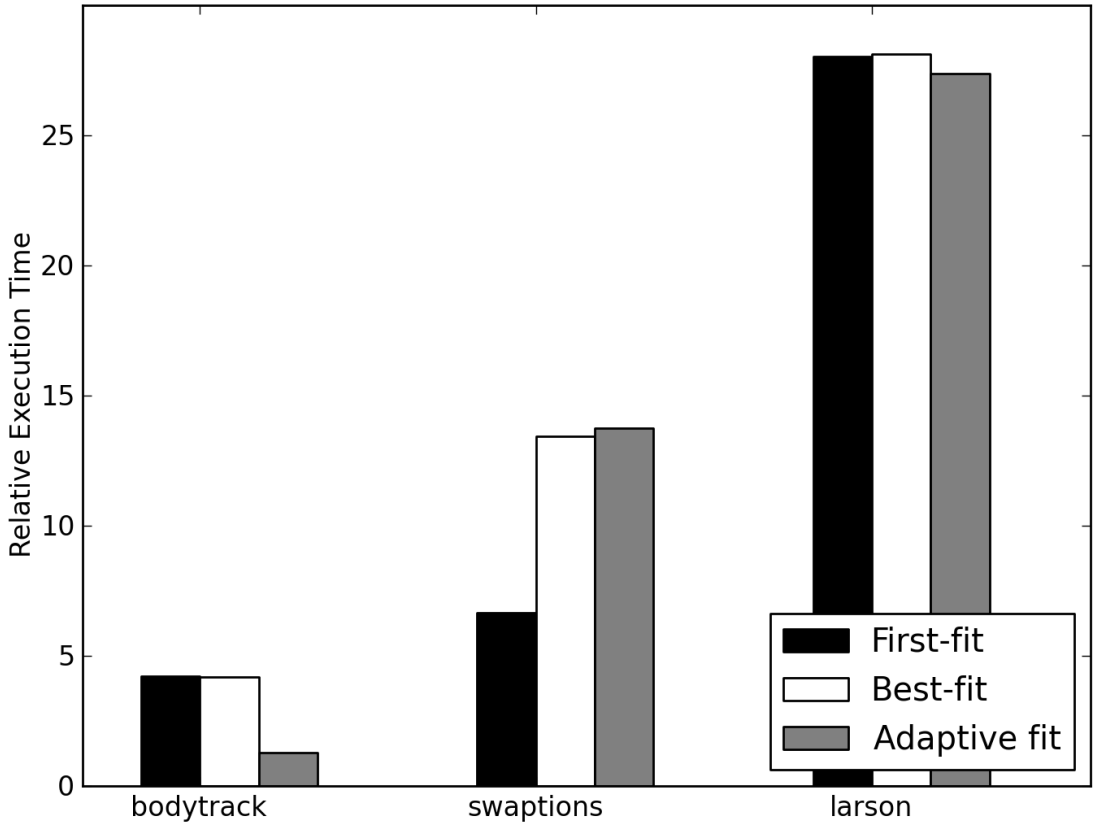
Σχήμα 5.7: Κατανάλωση μνήμης ανά μετροπρόγραμμα και αναθέτη

έχει σαφές πλεονέκτημα στην περίπτωση του μετροπρογράμματος Larson εξαιτίας της επιτυχημένης αναζήτησης που πραγματοποιεί. Και πάλι αυτό συμβαίνει χάρη της ιδιαιτερότητας των αιτημάτων.

Τέλος, οι χρόνοι εκτέλεσης σε δευτερόλεπτα απεικονίζονται στο Σχήμα 5.8. Το σχήμα αυτό δείχνει ότι η προσαρμοστική τεχνική είναι πολύ ελαφριά, επιφέροντας αμελητέο επιπλέον κόστος. Οι χρόνοι εκτέλεσης είναι όλοι τους πολύ κοντά σε όλες τις εφαρμογές που δοκιμάστηκαν με εξαίρεση την εφαρμογή *swaptions*, όπου ο αναθέτης πρώτης εφαρμογής είναι ο ταχύτερος, αλλά με κόστος τον υπερβολικό κατακερματισμό και την κατανάλωση μνήμης, όπως φαίνονται στα Σχήματα 5.5 και 5.7.

5.4 Συμπεράσματα

Στην παρούσα εργασία παρουσιάσαμε έναν τρόπο να προβλέπει κανείς τα μεγέθη σε μελλοντικά αιτήματα μνήμης. Δοκιμάσαμε έναν αναθέτη που να χρησιμοποιεί την τεχνική αυτή και



Σχήμα 5.8: Χρόνος εκτέλεσης ανά μετροπρόγραμμα και αναθέτη

τον συγκρίναμε με δύο άλλους τυπικούς σε μετροπρογράμματα που χρησιμοποιούν δυναμικό ανάθεση μνήμης. Η προσέγγισή μας εισήγαγε ελάχιστο επιπλέον κόστος, ενώ κατάφερε να διατηρήσει τον κατακερματισμό της μνήμης στο ελάχιστο.

6 Συμπεράσματα και Μελλοντικές Επεκτάσεις

Στο παρόν κεφάλαιο συνοψίζονται τα αποτελέσματα από την ερευνητική δραστηριότητα που παρουσιάστηκε στα προηγούμενα κεφάλαια και παρουσιάζονται ορισμένες κατευθύνσεις επέκτασης των αποτελεσμάτων αυτών.

6.1 Συμπεράσματα

Η διατριβή αυτή ασχολήθηκε με το κρίσιμο ζήτημα σχεδιασμού και ανάπτυξης δυναμικών αναθετών μνήμης. Ο πρωταρχικός μας στόχος ήταν η ανάπτυξη ενός πλαισίου λογισμικού για αναθέτες μνήμης. Πάνω στην dmmlib, οι πολιτικές και οι μηχανισμοί από διάφορους άλλους αναθέτες μπορούν να υλοποιηθούν και να συνδυαστούν. Οι υλοποιήσεις των παραγόμενων αναθετών μπορούν να αξιολογηθούν και να συγκριθούν μεταξύ τους πάνω σε μια πιο δίκαια και πιο οργανωμένη βάση. Τα κριτήρια αξιολόγησης των αναθετών παραμένουν η κατανάλωση μνήμης και η απόδοση των προγραμμάτων που χρησιμοποιούν τους αναθέτες. Υποστηρίζουμε ότι για την περαιτέρω βελτιστοποίηση των εφαρμογών είναι σημαντικό να εκθέτουμε τον αναθέτη μνήμης στην εφαρμογή αντί να κρύβουμε αποφάσεις για τη διαχείριση του φόρτου εργασίας στο λειτουργικό σύστημα. Με την dmmlib μπορούμε τότε να σκιαγραφήσουμε το προφίλ της εφαρμογής και να κατασκευάσουμε, δοκιμάζοντας διάφορες επιλογές, τον κατάλληλο αναθέτη μνήμης.

Παρουσιάσαμε, επίσης, την dmmlib να τρέχει πάνω σε αρχιτεκτονικές ενσωματωμένων συστημάτων και να εξετάζει εκεί την αποδοτικότητα διαφόρων αναθετών μνήμης. Δείξαμε ότι ένας σύνθετος, εξελιγμένος αναθέτης μνήμης μπορεί να μην ταιριάζει πάντα στον φόρτο εργασίας και τον τύπο μιας πλατφόρμας. Υποστηρίξαμε ότι είναι εξαιρετικά σημαντικό να δοκιμάζονται απλές πολιτικές και μηχανισμοί, οι οποίοι μπορούν να καλύπτουν μια πλατφόρμα προσφέροντας αύξηση στην απόδοση ή χρόνο για επιμέρους διαχείριση των πόρων. Έχοντας καλή γνώση της πλατφόρμας, παρέχεται η δυνατότητα να γραφούν σημεία στον αναθέτη σε πιο χαμηλό επίπεδο, τα οποία να στοχεύουν σε συγκεκριμένο υλισμικό, για ακόμη καλύτερες επιδόσεις. Μάλιστα, μία υβριδική λύση, υλοποιημένη σε υλισμικό και λογισμικό, μπορεί να έχει καλύτερη συμπεριφορά

από μία λύση αμιγώς υλοποιημένη σε υλισμικό.

Τέλος, παρουσιάσαμε έναν τρόπο πρόβλεψης των αιτημάτων που δέχεται ένας αναθέτης μνήμης. Προσαρμόσαμε ορισμένες πολιτικές στην dmmlib, ώστε με καινούριους μηχανισμούς να μπορούν να παραμετροποιηθούν οι πολιτικές αυτές κατά την διάρκεια εκτέλεσης. Με την αρχική πρόβλεψη επιτρέπουμε στις πολιτικές αυτές να προσαρμοστούν στο φορτίο της εφαρμογής. Η εμπειρία μας και η αξιολόγηση του σχεδιασμού αυτού μάς δείχνουν ότι η προσέγγιση αυτή είναι ένας αποτελεσματικός τρόπος να δημιουργηθούν νέοι δυναμικοί αναθέτες μνήμης.

6.2 Μελλοντικές Επεκτάσεις

Ένα ευρύ φάσμα ανοικτών ερωτημάτων παραμένει στο χώρο της δυναμικής διαχείρισης μνήμης. Αισθανόμαστε ότι τα σημαντικότερα θέματα έχουν να κάνουν με την επέκταση της μελέτης των δυναμικών αναθετών μνήμης, παρά με την περαιτέρω αξιολόγηση των επιδόσεων, η οποία διαφέρει ανά πλατφόρμα και αρχιτεκτονική. Μερικές από τις ενδιαφέρουσες προκλήσεις που θέτει το πλαίσιο δημιουργίας δυναμικών αναθετών μνήμης dmmlib περιγράφονται παρακάτω:

- **Εξερεύνηση περισσότερων πολιτικών και μηχανισμών** Υποστηρίξαμε ότι με την dmmlib μπορεί κανείς να υλοποιήσει εύκολα οποιαδήποτε πολιτική και μηχανισμό για αναθέτη μνήμης θέλει. Αυτό αντίκειται στις συμβατικές προσεγγίσεις που στηρίζονται σε σταθερές, στατικές μεθόδους, οι οποίες δεν μπορούν να αλλάξουν κατά την μεταγλώττισή του αναθέτη τους. Αν και κάποιοι αναθέτες εκθέτουν παραμέτρους ρύθμισης κατά την εκτέλεση των εφαρμογών, πιστεύουμε ότι χρειάζεται η δυνατότητα να εξερευνηθούν πιο βασικές επιλογές, όπως για παράδειγμα ο τρόπος οργάνωσης του χώρου. Η dmmlib θα μπορούσε να φιλοξενήσει τις υλοποιήσεις αυτές.

- **Ασφαλέστερος προγραμματισμός αναθετών** Η πλειονότητα των αναθετών αναπτύσσονται στην γλώσσα προγραμματισμού C. Αν και πολλές υλοποιήσεις είναι ήδη ώριμες αρκετά, Θα μπορούσαν να προταθούν ασφαλέστερες εκδοχές τους γραμμένες σε άλλες γλώσσες επιπέδου συστήματος. Για παράδειγμα με την γλώσσα Rust θα μπορούσαν να αποφευχθούν αρκετές προβληματικές δηλώσεις μεταβλητών, με τον μεταγλωττιστή να παραπονιέται ήδη για λάθη, τα οποία κανένας μεταγλωττιστής της C θα αναγνώριζε.

- **Μεταφορά της dmmlib στην περιοχή του πυρήνα (kernel space)** Η μεθοδολογία της dmmlib θα μπορούσε να δοκιμαστεί και στην περιοχή του πυρήνα. Στο σύστημα αυτό συγκαταλέγονται οι μηχανισμοί με τους οποίους τα επιμέρους τμήματα του πυρήνα (kernel modules) ζητούν από το σύστημα μνήμη και ο μηχανισμός διαχείρισης των σελίδων. Αν και

τα περισσότερα γίνονται σχεδόν με στατικό τρόπο, θα υπήρχε ενδιαφέρον να δοκιμαστούν τουλάχιστον και άλλες δομές δεδομένων και να αξιολογηθούν τυχόν βελτιώσεις.

- **Δυναμική ανάθεση μνήμης και garbage collection** Γλώσσες υψηλού επιπέδου όπως η Java, η C# και η Go, δεν επιτρέπουν στους προγραμματιστές τους να διαχειριστούν οι ίδιοι την μνήμη. Ανά τακτά χρονικά διαστήματα χρησιμοποιούν προγράμματα που ονομάζονται garbage collectors προκειμένου να επιστρέψουν στο σύστημα την μνήμη που δεν χρησιμοποιούν οι εφαρμογές πλέον. Στα πλαίσια αυτά θα είχε ενδιαφέρον να μελετηθούν περαιτέρω αναθέτες όπως οι προσαρμόσιμοι που παρουσιάζονται στο κεφάλαιο 5, ώστε να εξεταστεί η συμπεριφορά τους σε συνθήκες υψηλού φόρτου εργασίας που μεταβάλλεται στο πεδίο του χρόνου.

Βιβλιογραφία

[1] Lord Abbett. (19 Νοέ. 2015). Consolidation vs. Innovation in the Global Semiconductor Industry, διεύθν.: https://www.lordabbett.com/content/lordabbett/en/perspectives/equityperspectives/consolidation-vs-innovation-in-global-semiconductor-industry.html.

[2] D. Lash. (Σεπτ. 2007). Computer Memory Hierarchy, διεύθν.: https://commons.wikimedia.org/wiki/File:ComputerMemoryHierarchy.svg.

[3] C. Scott. (28 Αύγ. 2015). Numbers Every Programmer Should Know By Year, διεύθν.: http://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html.

[4] W. A. Wulf και S. A. McKee, «Hitting the Memory Wall: Implications of the Obvious», *SIGARCH Comput. Archit. News*, τόμ. 23, αρθμ. 1, σσ. 20–24, Μαρ. 1995, ISSN: 0163-5964. DOI: 10.1145/216585.216588. διεύθν.: http://doi.acm.org/10.1145/216585.216588.

[5] P. R. Wilson, M. S. Johnstone, M. Neely και D. Boles, «Dynamic Storage Allocation: A Survey and Critical Review», στο *IWMM*, 1995, σσ. 1–116.

[6] E. D. Berger, K. S. McKinley, R. D. Blumofe και P. R. Wilson, «Hoard: a scalable memory allocator for multithreaded applications», *SIGPLAN Not.*, τόμ. 35, αρθμ. 11, σσ. 117–128, Νοέ. 2000, ISSN: 0362-1340. DOI: http://doi.acm.org/10.1145/356989.357000. διεύθν.: http://doi.acm.org/10.1145/356989.357000.

[7] ISO, *International Standard ISO/IEC 9899:1999: Technical Corrigendum 3*. pub-ISO: pub-ISO, Νοέ. 2007, 10 σσ. διεύθν.: http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=43485;%20http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf.

[8] D. Detlefs, A. Dosser και B. Zorn, «Memory allocation costs in large C and C++ programs», *Software: Practice and Experience*, τόμ. 24, αρθμ. 6, σσ. 527–542, Ιούν. 1994, ISSN: 00380644, 1097024X. DOI: 10.1002/spe.4380240602. διεύθν.: http://doi.wiley.com/10.1002/spe.4380240602.

[9] Niall Douglas. (5 Δεκ. 2013). nedmalloc, διεύθν.: http://www.nedprod.com/programs/portable/nedmalloc/.

[10] D. Kaiser, «Intel Advisor XE - Threading Design & Prototyping Vectorization Assistant», Ιούλ. 2014.

[11] NVIDIA, «Whitepaper: NVIDIA GeForce GTX 1080», 14 Μάι. 2016. διεύθν.: http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf.

[12] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn και T. J. Purcell, «A Survey of General-Purpose Computation on Graphics Hardware», *Computer Graphics Forum*, τόμ. 26, αρθμ. 1, σσ. 80–113, 2007, ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2007.01012.x. διεύθν.: http://dx.doi.org/10.1111/j.1467-8659.2007.01012.x.

[13] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal και P. Dubey, «Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU», στο *Proceedings of the 37th Annual International Symposium on Computer Architecture*, σειρά ISCA '10, New York, NY, USA: ACM, 2010, σσ. 451–460, ISBN: 978-1-4503-0053-7. DOI: 10.1145/1815961.1816021. διεύθν.: http://doi.acm.org/10.1145/1815961.1816021.

[14] A. Heinecke, M. Klemm και H.-J. Bungartz, «From GPGPU to Many-Core: Nvidia Fermi and Intel Many Integrated Core Architecture», *Computing in Science & Engineering*, τόμ. 14, αρθμ. 2, σσ. 78–83, Μαρ. 2012, ISSN: 1521-9615. DOI: 10.1109/MCSE.2012.23. διεύθν.: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6159201.

[15] STMicroelectronics και CEA, «Platform 2012: A Many-core programmable accelerator for Ultra-Efficient Embedded Computing in Nanometer Technology», Whitepaper, 2011.

[16] B. Zorn, «Custo-Malloc: efficient synthesized memory allocators», Technical Report CU-CS-602-92, Computer Science Department, University of Colorado, 1992.

[17] A. Iyengar, «Parallel dynamic storage allocation algorithms», στο *Parallel and Distributed Processing, 1993. Proceedings of the Fifth IEEE Symposium on*, Δεκ. 1993, σσ. 82–91. DOI: 10.1109/SPDP.1993.395547.

[18] S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris και K. Pekmestzi, «Custom multi-threaded Dynamic Memory Management for Multiprocessor System-on-Chip platforms», στο *Embedded Computer Systems (SAMOS), 2010 International Conference on*, Ιούλ. 2010, σσ. 102–109. DOI: 10.1109/ICSAMOS.2010.5642078.

[19] L. Doug. (20 Απρ. 2000). A Memory Allocator, διεύθν.: http://gee.cs.oswego.edu/dl/html/malloc.html.

[20] J. M. Richter, *Advanced Windows: The Developer's Guide to the WIN32 API for Windows NT 3.5 and Windows 95*. Redmond, WA, USA: Microsoft Press, 1995, ISBN: 1-55615-677-4.

[21] M. S. Johnstone και P. R. Wilson, «The Memory Fragmentation Problem: Solved?», *SIGPLAN Not.*, τόμ. 34, αρθμ. 3, σσ. 26–36, Οκτ. 1998, bibtex: Johnstone:1998:MFP:301589.286864 bibtex[issue_date=March 1999;numpages=11;acmid=286864], ISSN: 0362-1340. DOI: 10.1145/301589.286864. διεύθν.: http://doi.acm.org/10.1145/301589.286864.

[22] M. R. Krishnan, «Heap: Pleasures and pains», *Microsoft Developer Newsletter*, 1999.

[23] J. M. Chang και E. F. Gehringer, «A high performance memory allocator for object-oriented systems», *Computers, IEEE Transactions on*, τόμ. 45, αρθμ. 3, σσ. 357–366, 1996.

[24] M. Shalan και V. J. Mooney III, «Hardware support for real-time embedded multiprocessor system-on-a-chip memory management», στο *Proceedings of the tenth international symposium on Hardware/software codesign*, σειρά CODES '02, New York, NY, USA: ACM, 2002, σσ. 79–84, ISBN: 1-58113-542-4. DOI: 10.1145/774789.774806. διεύθν.: http://doi.acm.org/10.1145/774789.774806.

[25] M. Monchiero, G. Palermo, C. Silvano και O. Villa, «Exploration of distributed shared memory architectures for NoC-based multiprocessors», *Journal of Systems Architecture*, τόμ. 53, αρθμ. 10, σσ. 719–732, 2007, ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2007.01.008. διεύθν.: http://www.sciencedirect.com/science/article/pii/S1383762107000203.

[26] K. Bai και A. Shrivastava, «Heap data management for limited local memory (LLM) multicore processors», στο *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, σειρά CODES/ISSS '10, New York, NY, USA: ACM, 2010, σσ. 317–326, ISBN: 978-1-60558-905-3. DOI: http://doi.acm.org/10.1145/1878961.1879015. διεύθν.: http://doi.acm.org/10.1145/1878961.1879015.

[27] ——, «A Software-only Scheme for Managing Heap Data on Limited Local Memory(LLM) Multicore Processors», *ACM Trans. Embed. Comput. Syst.*, τόμ. 13, αρθμ. 1, 5:1–5:18, Σεπτ. 2013, ISSN: 1539-9087. DOI: 10.1145/2501626.2501632. διεύθν.: http://doi.acm.org/10.1145/2501626.2501632.

[28] K. Bai και A. Shrivastava, «Automatic and efficient heap data management for Limited Local Memory multicore architectures», στο *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, Μαρ. 2013, σσ. 593–598. DOI: 10.7873/DATE.2013.130.

[29] K.-P. VO, «Vmalloc: A General and Efficient Memory Allocator», *Software: Practice and Experience*, τόμ. 26, αρθμ. 3, σσ. 357–374, 1996, ISSN: 1097-024X. DOI: 10.1002/(SICI)1097-024X(199603)26:3<357::AID-SPE15>3.0.CO;2-\#.

[30] P.-Å. Larson και M. Krishnan, «Memory allocation for long-running server applications», στο *Proceedings of the 1st international symposium on Memory management*, σειρά ISMM '98, New York, NY, USA: ACM, 1998, σσ. 176–185, ISBN: 1-58113-114-3. DOI: 10.1145/286860.286880. διεύθν.: http://doi.acm.org/10.1145/286860.286880.

[31] V.-Y. Vee και W.-J. Hsu, «A scalable and efficient storage allocator on shared-memory multiprocessors», IEEE Comput. Soc, 1999, σσ. 230–235, ISBN: 0-7695-0231-8. DOI: 10.1109/ISPAN.1999.778944. διεύθν.: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=778944.

[32] E. D. Berger, B. G. Zorn και K. S. McKinley, «Composing high-performance memory allocators», *ACM SIGPLAN Notices*, τόμ. 36, αρθμ. 5, σσ. 114–124, 1 Μάι. 2001, ISSN: 03621340. DOI: 10.1145/381694.378821. διεύθν.: http://portal.acm.org/citation.cfm?doid=381694.378821.

[33] ——, «Reconsidering custom memory allocation», στο *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ACM Press, 2002, σ. 1, ISBN: 1-58113-471-1. DOI: 10.1145/582419.582421. διεύθν.: http://portal.acm.org/citation.cfm?doid=582419.582421.

[34] M. M. Michael, «Scalable lock-free dynamic memory allocation», *ACM SIGPLAN Notices*, τόμ. 39, αρθμ. 6, σ. 35, 9 Ιούν. 2004, ISSN: 03621340. DOI: 10.1145/996893.996848. διεύθν.: http://portal.acm.org/citation.cfm?doid=996893.996848.

[35] S. Schneider, C. D. Antonopoulos και D. S. Nikolopoulos, «Scalable locality-conscious multithreaded memory allocation», ACM Press, 2006, σ. 84, ISBN: 1-59593-221-6. DOI: 10.1145/1133956.1133968. διεύθν.: http://portal.acm.org/citation.cfm?doid=1133956.1133968.

[36] J. Evans, «A scalable concurrent malloc (3) implementation for FreeBSD», στο *Proc. of the BSDCan Conference, Ottawa, Canada*, 2006.

[37] ——, «Scalable memory allocation using jemalloc», *Notes by Facebook Engineering*, 2011. διεύθν.: https://www.facebook.com/notes/facebook-engineering/scalable-memory-allocation-using-jemalloc/480222803919.

[38] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele και A. Vandecappelle, «Data Transfer and Storage Exploration Methodology», στο *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*, Boston, MA: Springer US, 1998, σσ. 29–54, ISBN: 978-1-4757-2849-1. διεύθν.: http://dx.doi.org/10.1007/978-1-4757-2849-1_3.

[39] S. Mamagkakis, D. Atienza, C. Poucet, F. Catthoor και D. Soudris, «Energy-efficient dynamic memory allocators at the middleware level of embedded systems», στο *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, ACM, 2006, σσ. 215–222.

[40] D. Atienza, J. M. Mendias, S. Mamagkakis, D. Soudris και F. Catthoor, «Systematic dynamic memory management design methodology for reduced memory footprint», *ACM Trans. Des. Autom. Electron. Syst.*, τόμ. 11, αρθμ. 2, σσ. 465–489, Απρ. 2006, ISSN: 1084-4309. DOI: 10.1145/1142155.1142165. διεύθν.: http://doi.acm.org/10.1145/1142155.1142165.

[41] A. Dominguez, S. Udayakumaran και R. Barua, «Heap Data Allocation to Scratch-pad Memory in Embedded Systems», *J. Embedded Comput.*, τόμ. 1, αρθμ. 4, σσ. 521–540, Δεκ. 2005, ISSN: 1740-4460. διεύθν.: http://dl.acm.org/citation.cfm?id=1233791.1233799.

[42] D. Cho, S. Pasricha, I. Issenin, N. D. Dutt, M. Ahn και Y. Paek, «Adaptive Scratch Pad Memory Management for Dynamic Behavior of Multimedia Applications», *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, τόμ. 28, αρθμ. 4, σσ. 554–567, Απρ. 2009, ISSN: 0278-0070. DOI: 10.1109/TCAD.2009.2014002.

[43] I. Anagnostopoulos, S. Xydis, A. Bartzas, Z. Lu, D. Soudris και A. Jantsch, «Custom Microcoded Dynamic Memory Management for Distributed On-Chip Memory Organizations», *IEEE Embedded Systems Letters*, τόμ. 3, αρθμ. 2, σσ. 66–69, Ιούν. 2011, ISSN: 1943-0663. DOI: 10.1109/LES.2011.2146228. διεύθν.: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5755171.

[44] P.-H. Kamp, «malloc (3) Revisited.», στο *USENIX Annual Technical Conference*, 1998.

[45] Y. Feng και E. D. Berger, «A locality-improving dynamic memory allocator», στο *Proceedings of the 2005 workshop on Memory system performance*, σειρά MSP '05, New York, NY, USA: ACM, 2005, σσ. 68–77, ISBN: 1-59593-147-3. DOI: 10.1145/1111583.1111594. διεύθν.: http://doi.acm.org/10.1145/1111583.1111594.

[46] S. Ghemawat και P. Menage, *Tcmalloc: Thread-caching malloc*. 2009.

[47] B. C. Kuszmaul, «SuperMalloc: A Super Fast Multithreaded Malloc for 64-bit Machines», στο *Proceedings of the 2015 International Symposium on Memory Management*, σειρά ISMM '15, New York, NY, USA: ACM, 2015, σσ. 41–55, ISBN: 978-1-4503-3589-8. DOI: 10.1145/2754169.2754178. διεύθν.: http://doi.acm.org/10.1145/2754169.2754178.

[48] S. Borkar, «Thousand core chips: a technology perspective», στο *Proceedings of the 44th annual Design Automation Conference*, σειρά DAC '07, New York, NY, USA: ACM, 2007, σσ. 746–749, ISBN: 978-1-59593-627-1. DOI: http://doi.acm.org/10.1145/1278480.1278667. διεύθν.: http://doi.acm.org/10.1145/1278480.1278667.

[49] Minalogic, *STMicroelectronics Platform 2012 users' Community [Restricted Access]*. Νοέ. 2011. διεύθν.: https://minalogic.net/projects/p2012comm/.

[50] S. Xydis, I. Stamelakos, A. Bartzas και D. Soudris, «Runtime tuning of dynamic memory management for mitigating footprint-fragmentation variations», στο *Architecture of Computing Systems (ARCS) 2011*, 2011.

[51] X. Chen, Z. Lu, A. Jantsch και S. Chen, «Supporting Distributed Shared Memory on multi-core Network-on-Chips using a dual microcoded controller», στο *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, Μαρ. 2010, σσ. 39–44.

[52] X. Chen, Z. Lu, A. Jantsch, S. Chen, S. Chen και H. Gu, «Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property», *Computers & Electrical Engineering*, τόμ. 39, αρθμ. 2, σσ. 596–612, 2013, ISSN: 0045-7906. DOI: http://dx.doi.org/10.1016/j.compeleceng.2012.04.009. διεύθν.: http://www.sciencedirect.com/science/article/pii/S004579061200081X.

[53] M. Millberg, E. Nilsson, R. Thid, S. Kumar και A. Jantsch, «The Nostrum backbone-a communication protocol stack for Networks on Chip», στο *VLSI Design, 2004. Proceedings. 17th International Conference on*, 2004, σσ. 693–696. DOI: 10.1109/ICVD.2004.1261005.

[54] I. Anagnostopoulos, J.-M. Chabloz, I. Koutras, A. Bartzas, A. Hemani και D. Soudris, «Power-aware dynamic memory management on many-core platforms utilizing DVFS», *ACM Transactions on Embedded Computing Systems (TECS)*, τόμ. 13, αρθμ. 1, σ. 40, 2013.

[55] S. Yalamanchili κ.ά., «From Adaptive to Self-Tuned Systems», στο *Proc. of The Future of Computing, essay in memory of Stamatis Vassiliadis*, 2007.

[56] C. Bienia, «Benchmarking Modern Multiprocessors», Διδακτορική διατρ., Princeton University, Ιαν. 2011.

[57] N. Rafique, W.-T. Lim και M. Thottethodi, «Architectural Support for Operating System-driven CMP Cache Management», στο *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, σειρά PACT '06, New York, NY, USA: ACM, 2006, σσ. 2–12, ISBN: 1-59593-264-X. DOI: 10.1145/1152154.1152160. διεύθν.: http://doi.acm.org/10.1145/1152154.1152160.

[58] A. Sharifi, H. Zhao και M. Kandemir, «Feedback control for providing QoS in NoC based multicores», στο *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, Μαρ. 2010, σσ. 1384–1389. DOI: 10.1109/DATE.2010.5457029.

[59] K. Aisopos, J. Moses, R. Illikkal, R. Iyer και D. Newell, «PCASA: Probabilistic control-adjusted Selective Allocation for shared caches», στο *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, Μαρ. 2012, σσ. 473–478.

[60] I. Koutras, A. Bartzas και D. Soudris, «Efficient memory allocations on a many-core accelerator», στο *ARCS Workshops (ARCS), 2012*, IEEE, 2012, σσ. 1–6.

[61] D. Simon, *Optimal state estimation: Kalman, H [infinity] and nonlinear approaches*. John Wiley και Sons, 2006.

[62] K. Kim και R. C. Schaefer, «Tuning a PID controller for a digital excitation control system», *IEEE Transactions on Industry Applications*, τόμ. 41, αρθμ. 2, σσ. 485–492, Μαρ. 2005, ISSN: 0093-9994. DOI: 10.1109/TIA.2005.844368.

Γλωσσάρι

ανάθεση *Allocation*. Σε συστήματα με λειτουργικό σύστημα, οι περισσότερες εφαρμογές εκτελούνται σε Λειτουργία Χρήστη (User Mode) και δεν έχουν τη δυνατότητα να προσπελάζουν απευθείας τη μνήμη. Προκειμένου, λοιπόν, να αποκτήσουν πρόσβαση σε περισσότερη μνήμη, χρησιμοποιούν την ΠΔΕ του συστήματος και αιτούνται συγκεκριμένης ποσότητας μνήμης. Η διαδικασία αυτή για το σύστημα ονομάζεται "ανάθεση" και ολοκληρώνεται με την επιστροφή της διεύθυνσης μνήμης στην εφαρμογή ή κάποιου σφάλματος σε περίπτωση προβλήματος.. 22, 125

Ανομοιόμορφη Προσπέλαση Μνήμης *Non-Uniform Memory Access (NUMA)*. Σχεδιασμός μνήμης που χρησιμοποιείται στην πολυεπεξεργασία, όπου ο χρόνος προσπέλασης στην μνήμη εξαρτάται από την επεξεργαστική μονάδα και την τοποθεσία της μνήμης αναφορικά σε αυτή. Στον σχεδιασμό αυτό, ένας επεξεργαστής μπορεί να προσπελάζει την δική του μνήμη ταχύτερα από μία μνήμη που δεν είναι τοπική σε αυτόν (είτε την μοιράζεται με πολλούς άλλους επεξεργαστές ή είναι τοπική σε άλλον επεξεργαστή).. 127

αντιστάθμιση *Trade-off*. Κάθε κατάσταση στην οποία ένα χαρακτηριστικό ή μία ποσότητα ενός πράγματος πρέπει να αλλάξει ή να μειωθεί, ώστε να αλλάξει ή να αυξηθεί το άλλο.. 37, 96

αποδέσμευση *Deallocation*. Πρόκειται για την αντίστροφη διαδικασία της ανάθεσης: Η εφαρμογή δεν επιθυμεί να χρησιμοποιήσει πλέον μία περιοχή μνήμης που το σύστημα τής είχε αναθέσει και σηματοδοτεί την απελευθέρωσή της, ώστε το σύστημα να μπορέσει να την επαναχρησιμοποιήσει σε μελλοντικές αναθέσεις.. 22

Δυναμική Διαχείριση Μνήμης *Dynamic Memory Management*. Η διαδικασία ανάθεσης και απελευθέρωσης χώρου μνήμης.. 7, 21, 30, 127

μαγικός αριθμός *Magic number*. Στα υπολογιστικά συστήματα ο μαγικός αριθμός αναφέρεται συνήθως σε μία σταθερή αριθμητική ή αλφαριθμητική τιμή, η οποία χρησιμοποιείται για να διακρίνει μορφές δεδομένων, αρχείων, πρωτοκόλλων κ.λπ. Οι τιμές αυτές θα πρέπει να είναι διακριτές και μοναδικές, ώστε να είναι απίθανο να μπερδευτούν.. 58, 60

Μονάδα Διαχείρισης Μνήμης *Memory Management Unit*. Αποκαλούμενη ορισμένες φορές και *μονάδα διαχείρισης σελιδοποιημένης μνήμης*, η μονάδα αυτή ανήκει στο υλισμικό υπολογιστών και περνάει το σύνολο των αναφορών της μνήμης από αυτή, κυρίως για να μεταφράσει τις εικονικές διευθύνσεις μνήμης σε φυσικές διευθύνσεις.. 38, 127

Προγραμματιστική Διεπαφή Εφαρμογής *Application Programming Interface*.Ένα σύνολο από ρουτίνες, πρωτόκολλα και εργαλεία για την ανάπτυξη λογισμικού και εφαρμογών.. 25, 76, 127

σελίδα μνήμης *Memory page*. Μία σελίδα, ή σελίδα μνήμης, ή εικονική σελίδα, είναι ένα σταθερού μήκους, συνεχές μπλοκ εικονικής μνήμης, το οποίο αποτελεί μία ξεχωριστή εγγραφή στον πίνακα σελιδοποίησης. Είναι η μικρότερη μονάδα δεδομένων στη διαχείριση εικονικής μνήμης στο επίπεδο λειτουργικού συστήματος ή υλισμικού.. 24, 38, 43, 48, 49, 67

τμήμα δεδομένων *Data segment*. Το μέρος ενός αρχείου αντικειμένου ή ο αντίστοιχος εικονικός χώρος διευθύνσεων ενός προγράμματος, το οποίο περιέχει τις αρχικοποιημένες στατικές μεταβλητές, δηλαδή τις καθολικές και τις στατικές τοπικές μεταβλητές. Το μέγεθος του τμήματος αυτού καθορίζεται από το μέγεθος των τιμών στον πηγαίο κώδικα του προγράμματος. Συνήθως μπορεί να επεκταθεί από την οικογένεια κλήσεων της `brk()` ώστε να διευρυνθεί το μέγεθος της σωρού, αλλά η τακτική αυτή αποφεύγεται από τα σύγχρονα λειτουργικά συστήματα, όπου συνίσταται η χρήση της `mmap()` για τον ίδιο σκοπό.. 65

Συντομογραφίες

ΑΠΜ Ανομοιόμορφη Προσπέλαση Μνήμης. 85, 127, *Γλωσσάρι:* Ανομοιόμορφη Προσπέλαση Μνήμης

ΔΔΜ Δυναμική Διαχείριση Μνήμης. 7, 21, 30, 31, 36, 87, 90, 127, *Γλωσσάρι:* Δυναμική Διαχείριση Μνήμης

ΛΣ Λειτουργικό Σύστημα. 11, 21--28, 48, 66, 70

ΜΔΜ Μονάδα Διαχείρισης Μνήμης. 38, 67, 127, *Γλωσσάρι:* Μονάδα Διαχείρισης Μνήμης

ΠΔΕ Προγραμματιστική Διεπαφή Εφαρμογής. 25, 28, 55, 61, 70, 76, 85, 86, 89, 90, 125, 127, *Γλωσσάρι:* Προγραμματιστική Διεπαφή Εφαρμογής