Παραλληλοποίηση μη ορθογωνικών εμφωλευμένων βρόχων

Η Μεταπτυχιακή Διπλωματική Εργασία

υποβάλλεται στην ορισθείσα

από τη Συνέλευση

του Τμήματος Μηχανικών Η/Υ και Πληροφορικής

Εξεταστική Επιτροπή

από τον

Γεώργιο Ζησόπουλο

ως μέρος των υποχρεώσεων για την απόκτηση του

ΔΙΠΛΩΜΑΤΟΣ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΣΤΗ ΜΗΧΑΝΙΚΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΜΕ ΕΙΔΙΚΕΥΣΗ

ΣΤΑ ΠΡΟΗΓΜΕΝΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Ιωάννινα 2026

Εξεταστική Επιτροπή:

- **Βασίλειος Β. Δημακόπουλος**, Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων (Επιβλέπων)
- **Απόστολος Ζάρρας**, Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
- **Γεώργιος Μανής**, Αναπλ. Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων

Περιεχομενα

Κατάλογος Σχημάτων **iii**

Κατάλογος Πινάκων **iv**

Περίληψη **v**

Extended Abstract **vi**

1 Εισαγωγή **1**

1.1 Παράλληλα συστήματα 1

1.1.1 Συστήματα κοινόχρηστης μνήμης 2

1.1.2 Συστήματα κατανεμημένης μνήμης 3

1.2 Βρόχοι επανάληψης 4

1.3 Αντικείμενο εργασίας 5

1.4 Δομή εργασίας 5

2 Μετασχηματισμοί Βρόχων **7**

2.1 Χώροι επανάληψης 8

2.2 Εισαγωγή στους μετασχηματισμούς 13

2.3 Κανονικοποίηση βρόχων 13

2.4 Σύμπτυξη βρόχων 17

2.4.1 Σύμπτυξη ορθογωνικών βρόχων 18

2.4.2 Σύμπτυξη μη ορθογωνικών βρόχων 20

2.4.3 Αποκανονικοποίηση ανάκτησης 24

3 OpenMP και OMPI **25**

3.1 OpenMP 25

3.1.1 Οδηγία parallel 26

3.1.2 Οδηγία for 28
3.2 OMPi 31
3.2.1 Μετασχηματισμοί οδηγιών 32

4 Υλοποίηση στον OMPI **35**
4.1 Υποδομή συμβολικής επεξεργασίας 35
4.2 Σύμπτυξη 38
4.2.1 Σύμπτυξη γενικής περίπτωσης 39
4.2.2 Σύμπτυξη βάθους 2 42
4.2.3 Σύμπτυξη βρόχων με πλασματικές εξαρτήσεις 47
4.2.4 Υλοποιήσεις άλλων μεταφραστών 47

5 Αξιολόγηση **50**
5.1 Έλεγχος ορθότητας 50
5.2 Μετρήσεις 50
5.2.1 Μετρήσεις μεθόδου βάθους 2 52
5.2.2 Μετρήσεις μεθόδου γενικής περίπτωσης 56

6 Επίλογος **62**
6.1 Σύνοψη εργασίας 62
6.2 Μελλοντική εργασία 63

Βιβλιογραφία **64**

Κατάλογος Σχημάτων

2.1 Δομή εμφωλευμένων βρόχων βάθους n 9
2.2 Παράδειγμα μη ορθογωνικού χώρου 11
2.3 Παράδειγμα κανονικοποίησης 16

4.1 Παράδειγμα σύγκρισης ελέγχου αν μια έκφραση ξεκινάει με $(expr1\ bop\ expr2)+expr3$ ή με $expr1\ +(expr2\ bop\ expr3)$, όπου bop αυθαίρετος δυαδικός τελεστής 36
4.2 Παράδειγμα απλοποίησης της έκφρασης $(x+y)(x+y)$ στην έκφραση $x^2+2xy+y^2$ 38
4.3 Παράδειγμα πλαισίου 42
4.4 Παράδειγμα ανάκτησης με τη μέθοδο βάθους 2 48

5.1 Χρόνοι εκτέλεσης για δομή $\mathcal{L}_1$ ($N=8, workiters=300000000$) (sec) . . 53
5.2 Speedup για δομή $\mathcal{L}_1$ ($N=8, workiters=300000000$) 53
5.3 Χρόνοι εκτέλεσης για δομή $\mathcal{L}_2$ ($N=1000, workiters=80000$) (sec) . . . 54
5.4 Speedup για δομή $\mathcal{L}_2$ ($N=1000, workiters=80000$) 55
5.5 Gain για δομή $\mathcal{L}_1$ ($N=8, workiters=300000000$) 55
5.6 Gain για δομή $\mathcal{L}_2$ ($N=1000, workiters=80000$) 56
5.7 Χρόνοι εκτέλεσης για δομή $\mathcal{L}_3$ ($N=1000, workiters=100$) (sec) 59
5.8 Speedup για δομή $\mathcal{L}_3$ ($N=1000, workiters=100$) (sec) 59
5.9 Gain για δομή $\mathcal{L}_3$ ($N=1000, workiters=100$) (sec) 60

Κατάλογος Πινάκων

5.1 Χρόνοι εκτέλεσης για δομή $\mathcal{L}_1$ ($N = 8, workiters = 300000000$) (sec) . . 57
5.2 Speedup για δομή $\mathcal{L}_1$ ($N = 8, workiters = 300000000$) 57
5.3 Gain για δομή $\mathcal{L}_1$ ($N = 8, workiters = 300000000$) (%) 57
5.4 Χρόνοι εκτέλεσης για δομή $\mathcal{L}_2$ ($N = 1000, workiters = 80000$) (sec) . . . 58
5.5 Speedup για δομή $\mathcal{L}_2$ ($N = 1000, workiters = 80000$) 58
5.6 Gain για δομή $\mathcal{L}_2$ ($N = 1000, workiters = 80000$) (%) 58
5.7 Χρόνοι εκτέλεσης για δομή $\mathcal{L}_3$ ($N = 1000, workiters = 100$) (sec) 60
5.8 Speedup για δομή $\mathcal{L}_3$ ($N = 1000, workiters = 100$) 60
5.9 Gain για δομή $\mathcal{L}_3$ ($N = 1000, workiters = 100$) (%) 61

Περιληψη

Γεώργιος Ζησόπουλος, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2026.
Παραλληλοποίηση μη ορθογωνικών εμφωλευμένων βρόχων.
Επιβλέπων: Βασίλειος Β. Δημακόπουλος, Καθηγητής.

Καθώς η πλειονότητα των σύγχρονων υπολογιστικών συστημάτων βασίζεται στους πολυπύρηνους επεξεργαστές, η παραλληλοποίηση του λογισμικού αποτελεί βασική προϋπόθεση για την πλήρη αξιοποίηση των διαθέσιμων υπολογιστικών πόρων. Για την ανάπτυξη παράλληλων εφαρμογών που προορίζονται για συστήματα κοινόχρηστης μνήμης έχει καθιερωθεί ευρέως το πρότυπο OpenMP.

Οι βρόχοι επανάληψης συνιστούν ίσως τη συνηθέστερη πηγή παραλληλισμού, ιδίως σε επιστημονικές εφαρμογές. Το OpenMP προσφέρει ειδικούς μηχανισμούς για την παραλληλοποίηση βρόχων. Με τη φράση *collapse* υποστηρίζει μετά την έκδοση 5.0 τη σύμπτυξη μη ορθογωνικών εμφωλευμένων βρόχων, δηλαδή βρόχων που τα όριά τους μπορεί να περιέχουν εξαρτήσεις.

Στο πλαίσιο της εργασίας υλοποιήθηκε η σύμπτυξη μη ορθογωνικών βρόχων. Αναπτύχθηκαν δύο μέθοδοι, μία για τη γενική περίπτωση και μία για την περίπτωση δομής βρόχων βάθους 2. Η υλοποίηση έχει σχεδιαστεί έτσι ώστε να μπορεί να δεχτεί επιπλέον μεθόδους. Επιπλέον, αναπτύχθηκε μια υποδομή συμβολικής επεξεργασίας, η οποία υποστήριξε την υλοποίηση της σύμπτυξης. Τα πειραματικά αποτελέσματα αναδεικνύουν τις επιδόσεις της υλοποίησης, οι οποίες είναι ανώτερες των άλλων μεταφραστών σε συγκεκριμένες περιπτώσεις.

Extended Abstract

Georgios Zisopoulos, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2026.

Parallelization of non-rectangular loop nests.

Advisor: Vassilios V. Dimakopoulos, Professor.

As the majority of modern computing systems are based on multi-core processors, software parallelization has become essential for fully exploiting the available computational resources. OpenMP has been widely established for the development of parallel applications intended for shared-memory systems.

Iteration loops constitute perhaps the most common source of parallelism, particularly in scientific applications. OpenMP provides specific mechanisms for loop parallelization. Since version 5.0, through the *collapse* clause, it supports the collapsing of non-rectangular nested loops, meaning loops whose bounds may contain dependencies.

In this thesis, the collapsing of non-rectangular loops was implemented. Two methods were developed, one for the general case and one for loop nests of depth 2. The implementation has been designed in a modular manner, allowing additional methods to be incorporated in the future. Furthermore, a symbolic processing infrastructure was developed, which supported the implementation of the loop collapsing. The experimental results demonstrate the proposed implementation, which outperforms other compilers in specific cases.

Κεφαλαιο 1

Εισαγωγη

1.1 Παράλληλα συστήματα

1.2 Βρόχοι επανάληψης

1.3 Αντικείμενο εργασίας

1.4 Δομή εργασίας

1.1 Παράλληλα συστήματα

Οι επιδόσεις είναι από τα σημαντικότερα χαρακτηριστικά ενός υπολογιστικού συστήματος. Καθοριστικό ρόλο για τις επιδόσεις κατέχει ο επεξεργαστής. Για κάποιες δεκαετίες η εξέλιξη των επεξεργαστών βασιζόταν στην αύξηση της συχνότητας ρολογιού, η οποία επιτρεπόταν από τη συνεχή σμίκρυνση των τρανζίστορ. Ωστόσο, από ένα βαθμό σμίκρυνσης και έπειτα, η περαιτέρω αύξηση της συχνότητας κατέστη αδύνατη, καθώς απαιτούνταν υπερβολική αύξηση της κατανάλωσης ενέργειας. Συνεπώς, η εξέλιξη των επεξεργαστών και η αύξηση των επιδόσεών τους συνεχίστηκε με τη δημιουργία πολυπύρηνων επεξεργαστών. Η ευρεία χρήση των επεξεργαστών αυτών ανέδειξε την *παράλληλη επεξεργασία*, δηλαδή τη μέθοδο εκτέλεσης κατά την οποία πολλαπλές υπολογιστικές εργασίες εκτελούνται ταυτόχρονα. Τα συστήματα που διαθέτουν πολλαπλές υπολογιστικές μονάδες και υποστηρίζουν παράλληλη επεξεργασία ονομάζονται *παράλληλα υπολογιστικά συστήματα*.

Η υποστήριξη παράλληλης επεξεργασίας προϋποθέτει τη δημιουργία λογισμικού που είναι ειδικά σχεδιασμένο για παράλληλα συστήματα. Ο παράλληλος προγραμ-

ματισμός διαφέρει από τον κλασικό σειριακό προγραμματισμό και είναι ιδιαίτερα απαιτητικός, διότι για να είναι ωφέλιμη η παραλληλοποίηση, απαιτείται

- Ο λογικός διαχωρισμός μιας εργασίας σε επιμέρους, ανεξάρτητες μεταξύ τους εργασίες
- Ο ορθός και βέλτιστος συντονισμός των επεξεργαστικών μονάδων

Ο συντονισμός των επεξεργαστών απαιτεί γνώση της δομής του συστήματος. Βλέπουμε λοιπόν ότι ο προγραμματισμός πρέπει να γίνει με βάση την αρχιτεκτονική του συστήματος.

Τα παράλληλα συστήματα χωρίζονται σε κατηγορίες με βάση διάφορα κριτήρια. Η πιο κλασική κατηγοριοποίηση κατά τον Flynn γίνεται με βάση τη ροή εντολών και δεδομένων και κατατάσσει τα συστήματα κυρίως σε:

- SISD (Single Instruction, Single Data)
- SIMD (Single Instruction, Multiple Data)
- MIMD (Multiple Instruction, Multiple Data)

Στην αρχιτεκτονική SISD εκτελείται μία εντολή σε ένα δεδομένο και είναι στην ουσία η αρχιτεκτονική των κλασικών σειριακών συστημάτων. Στην SIMD το σύστημα εκτελεί μία εντολή σε πολλά δεδομένα ταυτόχρονα και είναι η αρχιτεκτονική στην οποία βασίζονται οι μονάδες επεξεργασίας γραφικών (GPUs) και αρκετοί άλλοι επιταχυντές υλικού (hardware accelerators). Τέλος, στην MIMD εκτελούνται πολλές εντολές σε πολλά δεδομένα και είναι η αρχιτεκτονική των σύγχρονων παράλληλων συστημάτων.

Μια άλλη κατηγοριοποίηση γίνεται με βάση την οργάνωση μνήμης. Τα συστήματα εδώ χωρίζονται σε συστήματα *κοινόχρηστης μνήμης* και συστήματα *κατανεμημένης μνήμης*.

1.1.1 Συστήματα κοινόχρηστης μνήμης

Στα συστήματα κοινόχρηστης μνήμης οι επεξεργαστές συνδέονται μέσω ενός *μέσου διασύνδεσης* με την κύρια μνήμη. Έτσι, όλοι οι επεξεργαστές έχουν πρόσβαση σε έναν ενιαίο χώρο διευθύνσεων, ο οποίος μπορεί να αποτελείται από μία ή πολλές μνήμες. Η επικοινωνία μεταξύ των επεξεργαστών γίνεται με ανάγνωση και εγγραφή δεδομένων στην κύρια μνήμη.

Η αρχιτεκτονική αυτή δεν έχει καλή κλιμακωσιμότητα ως προς το πλήθος των επεξεργαστών. Όσο αυξάνεται το πλήθος τους, αυξάνεται η πιθανότητα ταυτόχρονων αιτημάτων στη μνήμη, άρα και στο μέσο διασύνδεσης. Συνεπώς, μπορεί να δημιουργηθεί συμφόρηση και η απόδοση του συστήματος μειώνεται.

Η παραλληλοποίηση σε αυτά τα συστήματα συνήθως γίνεται με νήματα. Το γεγονός ότι η μνήμη είναι κοινόχρηστη μπορεί να διευκολύνει κάποιες λειτουργίες, αλλά μπορεί να εισάγει συνθήκες ανταγωνισμού (race conditions), δηλαδή καταστάσεις όπου πολλά νήματα προσπαθούν να κάνουν εγγραφή σε μία θέση μνήμης ταυτόχρονα. Οι συνθήκες ανταγωνισμού μπορούν να αναιρέσουν την ορθότητα των υπολογισμών, συνεπώς επιβάλλεται ο συγχρονισμός νημάτων. Ο χειρισμός και συντονισμός των νημάτων εισάγει προγραμματιστική πολυπλοκότητα. Για το λόγο αυτό, ο προγραμματισμός των συστημάτων αυτών γίνεται με τη διεπαφή OpenMP [1]. Το OpenMP περιλαμβάνει ένα σύνολο οδηγιών για την περιγραφή της παραλληλοποίησης και στόχο έχει την εύκολη συγγραφή κώδικα που υλοποιεί την παραλληλοποίηση.

1.1.2 Συστήματα κατανεμημένης μνήμης

Στα συστήματα κατανεμημένης μνήμης οι επεξεργαστές διαθέτουν τη δική τους τοπική μνήμη και η σύνδεση των επεξεργαστών γίνεται μέσω ενός *δικτύου διασύνδεσης*. Εδώ δεν υπάρχει ενιαίος χώρος διευθύνσεων και κάθε επεξεργαστής μπορεί να προσπελάσει άμεσα μόνο τη δική του μνήμη. Η επικοινωνία των επεξεργαστών γίνεται με ανταλλαγή μηνυμάτων μέσω του δικτύου. Για να προσπελάσει ένας επεξεργαστής μία θέση μνήμης που βρίσκεται σε μνήμη άλλου επεξεργαστή πρέπει να γίνει επικοινωνία μεταξύ τους.

Η αρχιτεκτονική αυτή έχει υψηλότερη κλιμακωσιμότητα, διότι για κάθε προσπέλαση ενός επεξεργαστή σε θέση μνήμης δεν απαιτείται η χρήση του δικτύου. Εδώ όμως, λόγω των δυνατοτήτων του δικτύου διασύνδεσης, η επικοινωνία των επεξεργαστών είναι πιο αργή.

Η παραλληλοποίηση εδώ γίνεται συνήθως με διεργασίες. Αφού εδώ η επικοινωνία των επεξεργαστών είναι αργή, ο βέλτιστος συντονισμός των επεξεργαστών γίνεται πιο δύσκολος. Επίσης, καθώς η επικοινωνία γίνεται με ανταλλαγή μηνυμάτων, εισάγεται αρκετή προγραμματιστική πολυπλοκότητα. Έτσι, για τον προγραμματισμό των συστημάτων αυτών χρησιμοποιείται η διεπαφή MPI. Το MPI απλοποιεί

τους μηχανισμούς επικοινωνίας μευαξύ διεργασιών παρέχοντας έτοιμες λειτουργίες επικοινωνίας.

1.2 Βρόχοι επανάληψης

Οι βρόχοι επανάληψης είναι μία πολύ σημαντική δομή για το λογισμικό και χρησιμοποιείται για την επαναληπτική εκτέλεση εργασιών. Οι βρόχοι συνιστούν ίσως τη συνηθέστερη πηγή παραλληλισμού, ιδίως σε επιστημονικές εφαρμογές. Χωρίζονται κυρίως σε δύο κατηγορίες:

- Βρόχοι με μετρητή: Σε αυτήν την κατηγορία οι επαναλήψεις ελέγχονται από ένα μετρητή, ο οποίος διατρέχει ένα εύρος τιμών με κάποιο βήμα μεταβολής. Το εύρος μπορεί να είναι γνωστό κατά τη μετάφραση ή να καθοριστεί κατά την εκτέλεση. Συνεπώς, το πλήθος των επαναλήψεων είναι προκαθορισμένο πριν την εκτέλεση.

- Βρόχοι με συνθήκη: Εδώ οι επαναλήψεις καθορίζονται από μία λογική συνθήκη, η οποία μπορεί να ελεγχθεί μόνο κατά την εκτέλεση. Άρα, εδώ το πλήθος των επαναλήψεων είναι αδιευκρίνιστο.

Εμείς θα ασχοληθούμε μόνο με βρόχους με μετρητή και πιο συγκεκριμένα με δομές που περιέχουν *εμφωλευμένους βρόχους* (nested loops). Για απλότητα θα αναφερόμαστε σε μια τέτοια δομή ως *δομή βρόχων*. Οι δομές βρόχων που θα ασχοληθούμε θέλουμε να ικανοποιούν τους εξής περιορισμούς:

- Η εμφώλευση της δομής πρέπει να είναι τέλεια, δηλαδή μόνο ο πιο εσωτερικός της βρόχος περιέχει σώμα με εντολές.

- Τα όρια των εσωτερικών βρόχων μπορούν να εξαρτώνται από μετρητές εξωτερικών βρόχων, αλλά τα όρια πρέπει να είναι αφινικές συναρτήσεις των μετρητών, δηλαδή είναι της μορφής $a_n i_n + a_{n-1} i_{n-1} + \cdots + a_1 i_1 + a_0$.

- Τα βήματα των βρόχων πρέπει να είναι σταθεροί ακέραιοι αριθμοί.

1.3 Αντικείμενο εργασίας

Αντικείμενο της εργασίας είναι η υλοποίηση της σύμπτυξης βρόχων στον μεταφραστή OMPi για μη ορθογωνικούς εμφωλευμένους βρόχους, δηλαδή για δομές βρόχων που τα όρια περιέχουν εξαρτήσεις. Σύμπτυξη βρόχων είναι η μετατροπή μιας δομής βρόχων σε έναν ισοδύναμο βρόχο και υποστηρίζεται από το OpenMP με τη φράση `collapse`. Μέχρι το OpenMP 4.5 η φράση αυτή απαιτούσε οι βρόχοι να μην περιέχουν εξαρτήσεις, αλλά από την έκδοση 5.0 πρόσθεσε υποστήριξη και για τους μη ορθογωνικούς.

Η σύμπτυξη βρόχων στον πυρήνα του είναι μαθηματικό πρόβλημα, το οποίο στην περίπτωση των μη ορθογωνικών βρόχων μπορεί να γίνει αρκετά περίπλοκο. Στη βιβλιογραφία υπάρχει μόνο μία δημιοσίευση [2] σχετικά με τη διαδικασία αυτή για δομές με εξαρτήσεις, η οποία για να υλοποιηθεί απαιτεί εξειδικευμένο λογισμικό. Οι υπάρχουσες υλοποιήσεις των GCC και LLVM βασίζονται σε ευρετικές μεθόδους, που είτε διαχειρίζονται το πρόβλημα κατά την εκτέλεση είτε χρησιμοποιούν έμμεσα τη σύμπτυξη ορθογωνικών βρόχων.

Στο πλαίσιο της παρούσας εργασίας υλοποιήθηκαν δύο ευρετικοί μέθοδοι για τη σύμπτυξη μη ορθογωνικών βρόχων. Η υλοποίηση βασίστηκε στο περιβάλλον του παραλληλοποιητικού μεταφραστή OMPi [3]. Επίσης, για την επίτευξη της υλοποίησης ήταν απαραίτητη η δυνατότητα διαχείρισης των ορίων των βρόχων ως συμβολικές εκφράσεις. Συνεπώς, χρειάστηκε να υλοποιηθούν επιπλέον υποστηρικτικές λειτουργίες. Τέλος, έγινε πειραματική αποτίμηση των μεθόδων και συγκριτικη μελέτη με τους GCC και LLVM.

1.4 Δομή εργασίας

Η δομή της παρούσας εργασίας είναι η εξής:

- Κεφάλαιο 1: Εισαγωγή στα παράλληλα συστήματα και στους βρόχους επανάληψης.
- Κεφάλαιο 2: Μαθηματική θεμελίωση των βρόχων και ανάλυση μετασχηματισμών.
- Κεφάλαιο 3: Παρουσίαση της διεπαφής OpenMP και του μεταφραστή OMPi.

- Κεφάλαιο 4: Περιγραφή της υλοποίησης.
- Κεφάλαιο 5: Παρουσίαση μετρήσεων της υλοποίησης και σύγκριση με άλλους μεταφραστές.
- Κεφάλαιο 6: Σύνοψη της εργασίας και ιδέες για μελλοντική εργασία.

Κεφαλαιο 2

Μετασχηματισμοι Βροχων

2.1 Χώροι επανάληψης

2.2 Εισαγωγή στους μετασχηματισμούς

2.3 Κανονικοποίηση βρόχων

2.4 Σύμπτυξη βρόχων

Στο κεφάλαιο αυτό αρχικά θα θεμελιώσουμε μαθηματικά τους βρόχους επανάληψης ορίζοντας τους *χώρους επανάληψης*. Στη συνέχεια θα κάνουμε μια μικρή εισαγωγή στους μετασχηματισμούς βρόχων και θα αναλύσουμε τους σχετικούς με την εργασία μετασχηματισμούς.

Ένας βρόχος μπορεί να περιγραφεί σημασιολογικά με έναν μετρητή $i \in \mathbb{Z}$, δύο όρια $l, u \in \mathbb{Z}$ και $s \in \mathbb{Z} \setminus \{0\}$ το βήμα μεταβολής του μετρητή. Επίσης, αν και συνηθίζεται στο λογισμικό να μην συμπεριλαμβάνεται το u στις υποψήφιες τιμές του i, σε μαθηματικές μοντελοποιήσεις βολεύει να θεωρήσουμε ότι το u συμπεριλαμβάνεται. Έτσι, για ευκολία θα αναφερόμαστε παρακάτω σε τυχόν βρόχο ως $L = (i, l, u, s)$ και σε δομή πολλαπλών βρόχων ως $\mathcal{L} = (L_1, L_2, \ldots, L_n)$, όπου L_1 ο πιο εξωτερικός βρόχος και L_n ο πιο εσωτερικός.

2.1 Χώροι επανάληψης

Οι χώροι επανάληψης είναι ένας φορμαλιστικός τρόπος να αναπαραστήσουμε τις επαναλήψεις μιας δομής βρόχων επανάληψης ως σημεία σε ένα μαθηματικό χώρο. Αυτό επιτρέπει την μοντελοποίηση των δομών αυτών και την εφαρμογή αναλύσεων και βελτιστοποιήσεων.

Θα δούμε τώρα πώς μοντελοποιείται ο χώρος επανάληψης για μεμονωμένο βρόχο και παρακάτω θα ανάγουμε σε δομή πολλαπλών βρόχων. Συνήθως οι ορισμοί δίνονται για βρόχους που έχουν βήμα 1. Εμείς θα δούμε και τη γενική περίπτωση για μεμονωμένο βρόχο, διότι θα χρειαστούμε ένα σημείο του γενικού ορισμού σε επόμενη ενότητα. Πριν δούμε τον ορισμό να υπενθυμίσουμε ότι η σχέση $a \equiv b \pmod{n}$ δηλώνει ότι το $a - b$ είναι ακέραιο πολλαπλάσιο του n.

Ορισμός 2.1 (Χώρος επανάληψης)**.** Έστω βρόχος $L = (i, l, u, s)$. Ο χώρος επανάληψης του L είναι το σύνολο των τιμών που παίρνει ο μετρητής και ορίζεται ως

$$\mathcal{I}_L = \begin{cases} \{i \in \mathbb{Z} \mid l \leq i \leq u,\ i \equiv l\ (\mathrm{mod} s)\}, & s > 0 \\ \{i \in \mathbb{Z} \mid l \geq i \geq u,\ i \equiv l\ (\mathrm{mod} |s|)\}, & s < 0 \end{cases}$$

Αν ισχύει ότι $s = 1$, το σύνολο βλέπουμε ότι απλοποιείται σε

$$\mathcal{I}_L = \{i \in \mathbb{Z} \mid l \leq i \leq u,\ i \equiv l\ (\mathrm{mod} 1)\} = \{i \in \mathbb{Z} \mid l \leq i \leq u\} \tag{2.1}$$

Το πλήθος επαναλήψεων δίνεται από τον τύπο

$$\begin{aligned} N_L = |\mathcal{I}_L| &= \sum_{i \in \mathcal{I}_L} 1 \\ &= \begin{cases} \left\lfloor \frac{u-l}{s} \right\rfloor + 1, & (s > 0\ \wedge\ l \leq u)\ \vee\ (s < 0\ \wedge\ l \geq u) \\ 0, & \text{διαφορετικά} \end{cases} \\ &= \max\left(0, \left\lfloor \frac{u-l}{s} \right\rfloor + 1\right) \end{aligned} \tag{2.2}$$

Ας δούμε τώρα πώς ανάγεται ο ορισμός για δομή βρόχων. Ας υποθέσουμε ότι έχουμε δομή εμφωλευμένων βρόχων $\mathcal{L} = (L_1, L_2, \ldots, L_n)$ όπως τη βλέπουμε στο Σχήμα 2.1, όπου τα όρια του κάθε βρόχου μπορεί να εξαρτώνται από έναν ή περισσότερους μετρητές πιο εξωτερικών βρόχων και το βήμα είναι μοναδιαίο, δηλαδή

```
for (i_1 = l_1; i_1 ≤ u_1; i_1++)
  for (i_2 = l_2(i_1); i_2 ≤ u_2(i_1); i_2++)
    ⋮
      for (i_n = l_n(i_1, i_2, ..., i_{n-1}); i_n ≤ u_n(i_1, i_2, ..., i_{n-1}); i_n++)
        S(i_1, i_2, ..., i_n)
```

Σχήμα 2.1: Δομή εμφωλευμένων βρόχων βάθους n

θα έχουμε

$$\begin{aligned} L_1 &= (i_1, l_1, u_1, 1) \\ L_2 &= (i_2, l_2(i_1), u_2(i_1), 1) \\ &\vdots \\ L_n &= (i_n, l_n(i_1, \ldots, i_{n-1}), u_n(i_1, \ldots, i_{n-1}), 1) \end{aligned}$$

Έτσι, ο χώρος επανάληψης κάθε εσωτερικού βρόχου L_k της $\mathcal{L}$ θα είναι συνάρτηση μετρητών πιο εξωτερικών βρόχων

$$\mathcal{I}_{L_k}(i_1, \ldots, i_{k-1}) = \{i_k \in \mathbb{Z} \mid l_k(i_1, \ldots, i_{k-1}) \leq i_k \leq u_k(i_1, \ldots, {}_{k-1})\}$$

Για τον χώρο της δομής $\mathcal{L}$ λοιπόν μπορούμε να θεωρήσουμε το χώρο ακεραίων $\mathbb{Z}^n$, όπου κάθε μετρητής βρόχου αντιστοιχίζεται σε μια διάσταση. Έτσι, μπορούμε να δούμε την κάθε επανάληψη ως ένα σημείο στο χώρο αυτό, δηλαδή ως μια πλειάδα που απαρτίζεται από τις τιμές των μετρητών. Το σύνολο όλων αυτών των σημείων είναι ο χώρος επανάληψης της δομής βρόχων και θα είναι

$$\begin{aligned} \mathcal{I}_{\mathcal{L}} &= \{(i_1, i_2, \ldots, i_n) \in \mathbb{Z}^n \mid \forall k \in \{1, 2, \ldots, n\},\ i_k \in \mathcal{I}_{L_k}\} \\ &= \{\mathbf{i} \in \mathbb{Z}^n \mid \forall k \in \{1, 2, \ldots, n\},\ l_k(i_1, i_2, \ldots, i_{k-1}) \leq i_k \leq u_k(i_1, i_2, \ldots, i_{k-1})\} \end{aligned}$$

Άμεσα μπορούμε να παρατηρήσουμε ότι το $\mathcal{I}_{\mathcal{L}}$ είναι το σύνολο λύσεων του γραμμικού συστήματος ανισοτήτων

$$\begin{pmatrix} l_1 \\ l_2(i_1) \\ \vdots \\ l_n(i_1, i_2, \ldots, i_{n-1}) \end{pmatrix} \leq \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix} \leq \begin{pmatrix} u_1 \\ u_2(i_1) \\ \vdots \\ u_n(i_1, i_2, \ldots, i_{n-1}) \end{pmatrix}$$

Εδώ βλέπουμε ότι το πλήθος επαναλήψεων ενός εσωτερικού βρόχου βάθους $k \geq 2$ καθορίζεται από τις συναρτήσεις των ορίων του, άρα είναι συνάρτηση εξωτερικών

μετρητών

$$N_{L_k}(i_1,\ldots,i_{k-1}) = |\mathcal{I}_{L_k}(i_1,\ldots,i_{k-1})| = \sum_{i_k=l_k(i_1,\ldots,i_{k-1})}^{u_k(i_1,\ldots,i_{k-1})} 1$$
$$= \max(0, u_k(i_1,\ldots,i_{k-1}) - l_k(i_1,\ldots,i_{k-1}) + 1)$$

Έτσι, το συνολικό πλήθος επαναλήψεων της $\mathcal{L}$ είναι

$$N_{\mathcal{L}} = |\mathcal{I}_{\mathcal{L}}| = \sum_{i_1=l_1}^{u_1} \sum_{i_2=l_2(i_1)}^{u_2(i_1)} \cdots \sum_{i_n=l_n(i_1,i_2,\ldots,i_{n-1})}^{u_n(i_1,i_2,\ldots,i_{n-1})} 1 \tag{2.3}$$

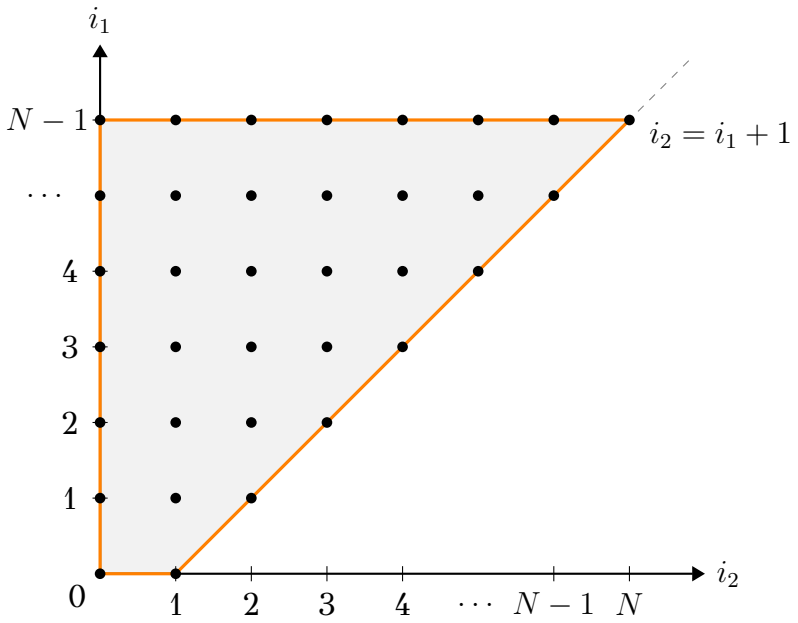
Το περίγραμμα του συνόλου αυτού σχηματίζει ένα γεωμετρικό σχήμα στον n-διάστατο χώρο, το οποίο καθορίζεται από τα όρια των βρόχων. Αν τα όρια των βρόχων είναι σταθεροί αριθμοί, τότε το σύνολο θα είναι ένα n-διάστατο ορθογώνιο πολύτοπο και θα λέμε ότι έχουμε *ορθογωνικούς βρόχους*. Αν κάποιο όριο εξαρτάται από εξωτερικό δείκτη, τότε το σχήμα θα είναι ένα n-διάστατο μη ορθογώνιο πολύτοπο και θα λέμε ότι έχουμε *μη ορθογωνικούς βρόχους*. Αν και το γεωμετρικό σχήμα δεν επιδρά με κάποιον τρόπο στην εκτέλεση των βρόχων, μπορεί να χρησιμοποιηθεί ως πληροφορία για την εύρεση μετασχηματισμού, όπως θα δούμε παρακάτω.

Στο Σχήμα 2.2 μπορούμε να δούμε ένα παράδειγμα γεωμετρικής αναπαράστασης του χώρου επανάληψης. Στα επάνω μέρος του σχήματος βλέπουμε μια δομή βρόχων σε γλώσσα C, όπου ένα όριο περιέχει εξάρτηση. Στο κάτω έχουμε τη γεωμετρική αναπαράσταση του χώρου επανάληψης, όπου οι μαύρες τελείες αποτελούν τις τιμές των μετρητών και το περίγραμμα αναπαριστά το γεωμετρικό σχήμα.

Ιδιαίτερα σημαντικός είναι ο τύπος υπολογισμού του πλήθους των επαναλήψεων της $\mathcal{L}$ (2.3). Έχοντας υποθέσει ότι $|s| = 1$ και τα όρια είναι αφινικές συναρτήσεις με ακέραιους συντελεστές ή $|s| > 1$ και ισχύει για όλους τους συντελεστές των ορίων ότι $(a_{k,u} - a_{k,l}) \cdot s_m$ να διαιρείται με το s_k, ο τύπος αυτός μπορεί να απλοποιηθεί σε κλειστό τύπο. Πιο συγκεκριμένα, αν υποθέσουμε ότι $p_1, p_2, \ldots, p_m$ είναι παράμετροι άγνωστοι κατά τη μετάφραση, τότε ο κλειστός τύπος θα είναι ένα πολυώνυμο $Ehr(p_1, p_2, \ldots, p_m)$ βαθμού n και ονομάζεται πολυώνυμο Ehrhart. Παρατηρούμε ότι για να υπολογιστεί ο τύπος (2.3) χρειάζεται επαναληπτική μέθοδος, ενώ το πολυώνυμο Ehrhart μας επιτρέπει να υπολογίσουμε το πλήθος επαναλήψεων σε $\mathcal{O}(1)$ χρόνο.

Επίσης, χρήσιμη θα φανεί παρακάτω μια διαφορετική μορφή του τύπου όπου μπορούμε να υπολογίσουμε τις επαναλήψεις των $n-k+1$ πιο εσωτερικών βρόχων. Ο

```
for (i1 = 0; i1 <= N-1; i1++)
  for (i2 = 0; i2 <= i1+1; i2++)
```



Σχήμα 2.2: Παράδειγμα μη ορθογωνικού χώρου

τύπος αυτός, επειδή τα όρια μπορεί να εξαρτώνται από πιο εξωτερικούς μετρητές, πρέπει να δέχεται ως παραμέτρους τις τιμές των μετρητών αυτών, οπότε θα είναι

$$N_{\mathcal{L}}^{(k)}(i_1, i_2, \ldots, i_{k-1}) = \sum_{i_k=l_k(i_1,\ldots,i_{k-1})}^{u_k(i_1,\ldots,i_{k-1})} \sum_{i_{k+1}=l_{k+1}(i_1,\ldots,i_k)}^{u_{k+1}(i_1,\ldots,i_k)} \cdots \sum_{i_n=l_n(i_1,\ldots,i_{n-1})}^{u_n(i_1,\ldots,i_{n-1})} 1 \tag{2.4}$$

Αν υποθέσουμε ότι έχουμε ορθογωνικούς βρόχους οι παραπάνω τύποι απλοποιούνται. Συγκεκριμένα, μπορούμε να δούμε ότι στον τύπο (2.3) κάθε άθροισμα μπορεί να απλοποιηθεί, αφού τα όρια είναι σταθερά. Άρα εφαρμόζοντας διαδοχικά την απλοποίηση θα έχουμε

$$N_{\mathcal{L}} = \sum_{i_1=l_1}^{u_1} \sum_{i_2=l_2}^{u_2} \cdots \sum_{i_n=l_n}^{u_n} 1 = N_{L_1} \cdot N_{L_2} \cdot \ldots \cdot N_{L_n} = \prod_{k=1}^{n} N_{L_k} \tag{2.5}$$

Όμοια και ο (2.4) θα γίνει

$$N_{\mathcal{L}}^{(k)}(i_1, i_2, \ldots, i_{k-1}) = N_{\mathcal{L}}^{(k)} = N_{L_k} \cdot N_{L_{k+1}} \cdot \ldots \cdot N_{L_n} = \prod_{k'=k}^{n} N_{L_{k'}}$$

Τέλος, οι δομές βάθους 2 είναι αρκετά συνηθισμένες στην πράξη και μάλιστα οι δομές αυτές διαθέτουν κάποια καλά χαρακτηριστικά. Έτσι, θα δούμε αναλύσουμε

λίγο παραπάνω τις δομές αυτές. Έστω λοιπόν δομή $\mathcal{L} = (L_1, L_2)$ με

$$L_1 = (i_1, l_1, u_1, s_1)$$
$$L_2 = (i_2, l_2(i_1), u_2(i_1), s_2) = (i_2, a_l i_1 + b_l, a_u i_1 + b_u, s_2)$$

Τότε το πλήθος επαναλήψεων του L_2 είναι

$$\begin{aligned} N_{L_2}(i_1) &= \left\lfloor \frac{(u_2 - l_2)(i_1)}{s_2} \right\rfloor + 1 = \left\lfloor \frac{(a_u - a_l)i_1 + (b_u - b_l)}{s_2} \right\rfloor + 1 \\ \Rightarrow N_{L_2}(l_1 + s_1 k) = \bar{N}_{L_2}(k) &= \left\lfloor \frac{(a_u - a_l)(l_1 + s_1 k) + (b_u - b_l)}{s_2} \right\rfloor + 1 \\ &= \left\lfloor \frac{(a_u - a_l)s_1 k}{s_2} + \frac{(a_u - a_l)l_1 + (b_u - b_l)}{s_2} \right\rfloor + 1 \end{aligned} \tag{2.6}$$

Παρατηρούμε ότι αν ισχύει ότι $(a_u - a_l)s_1 \mod s_2 = 0$, τότε το κλάσμα που περιλαμβάνει τη μεταβλητή k μπορεί να βγεί από το $\lfloor \cdot \rfloor$. Αν χρησιμοποιήσουμε τον ισχυρισμό, τότε η (2.6) γίνεται

$$\begin{aligned} \bar{N}_{L_2}(k) &= \left\lfloor \frac{(a_u - a_l)s_1 k}{s_2} + \frac{(a_u - a_l)l_1 + (b_u - b_l)}{s_2} \right\rfloor + 1 \\ &= \frac{(a_u - a_l)s_1}{s_2} k + \left\lfloor \frac{(a_u - a_l)l_1 + (b_u - b_l)}{s_2} \right\rfloor + 1 \\ &= Ak + \lfloor B \rfloor + 1 \end{aligned} \tag{2.7}$$

Διαπιστώνουμε λοιπόν ότι η μεταβολή του πλήθους επαναλήψεων είναι σταθερός ακέραιος αριθμός. Πράγματι, είναι

$$\begin{aligned} \Delta\bar{N}_{L_2}(k) &\coloneqq \bar{N}_{L_2}(k+1) - \bar{N}_{L_2}(k) \\ &= A(k+1) + \lfloor B \rfloor + 1 - Ak - \lfloor B \rfloor - 1 \\ &= \frac{(a_u - a_l)s_1}{s_2} \end{aligned} \tag{2.8}$$

Επομένως, το συνολικό πλήθος επαναλήψεων της $\mathcal{L}$ είναι

$$\begin{aligned} N_{\mathcal{L}} &= \sum_{k=0}^{N_{L_1}-1} \bar{N}_{L_2}(k) \\ &= A \sum_{k=1}^{N_{L_1}-1} k + (\lfloor B \rfloor + 1) \sum_{k=0}^{N_{L_1}-1} 1 \\ &= A \frac{(N_{L_1} - 1)N_{L_1}}{2} + (\lfloor B \rfloor + 1)N_{L_1} \\ &= \frac{(a_u - a_l)s_1}{s_2} \frac{(N_{L_1} - 1)N_{L_1}}{2} + \left(\left\lfloor \frac{(a_u - a_l)l_1 + (b_u - b_l)}{s_2} \right\rfloor + 1 \right) N_{L_1} \end{aligned} \tag{2.9}$$

2.2 Εισαγωγή στους μετασχηματισμούς

Μετασχηματισμός βρόχου είναι η μετατροπή ενός βρόχου ή μιας δομής βρόχων σε κάποια άλλη μορφή. Επειδή οι δομές βρόχων μοντελοποιούνται μέσω των χώρων επανάληψης, ένας μετασχηματισμός βρόχου εκφράζεται μαθηματικά ως ένας *μετασχηματισμός χώρου επανάληψης*.

Ορισμός 2.2 (Μετασχηματισμός χώρου επανάληψης). Έστω $\mathcal{L}$ μια δομή βρόχων και $\mathcal{I}_{\mathcal{L}}$ ο χώρος επανάληψης. Ένας μετασχηματισμός χώρου επανάληψης ορίζεται ως μια συνάρτηση $T : \mathcal{I}_{\mathcal{L}} \to \mathcal{I}_{\mathcal{L}'}$, η οποία απεικονίζει κάθε αρχικό διάνυσμα επανάληψης $\mathbf{i} \in \mathcal{I}_{\mathcal{L}}$ σε ένα νέο διάνυσμα $\mathbf{i}' \in \mathcal{I}_{\mathcal{L}'}$.

Για να είναι οι χώροι να είναι ισοδύναμοι και να είναι δυνατή η ανάκτηση του αρχικού χώρου μέσω της αντίστροφης συνάρτησης T^{-1}, η T πρέπει να είναι 1-1 και επί.

Επίσης, αξίζει να σημειωθεί ότι οι μετασχηματισμοί ορίζονται πάνω σε χώρους επανάληψης. Οι χώροι επανάληψης δεν εμπεριέχουν τη σειρά εκτέλεσης των επαναλήψεων. Οι βρόχοι επανάληψης όμως εκτελούνται πάντα με λεξικογραφική σειρά.

Ορισμός 2.3 (Λεξικογραφική διάταξη). Για ένα χώρο $\mathcal{I}$ μιας δομής κανονικοποιημένων βρόχων και δύο τυχόντα $\mathbf{i}, \mathbf{i}' \in \mathcal{I}$ η λεξικογραφική σειρά ορίζεται ως

$$\begin{aligned}\mathbf{i} \prec_{lex} \mathbf{i}' &\Leftrightarrow \exists k \in \{1, \ldots, n\},\ (\forall m \in \{1, \ldots, k-1\},\ i_m = i'_m\ \wedge\ i_k < i'_k) \\ &\Leftrightarrow \bigvee_{k=1}^{n} (\forall m \in \{1, \ldots, k-1\},\ i_m = i'_m\ \wedge\ i_k < i'_k)\end{aligned}$$

Οι μετασχηματισμοί όπως ορίζονται εδώ μεταβάλλουν μόνο το χώρο επανάληψης, όμως μπορεί έμμεσα να αλλάξει και η σειρά εκτέλεσης.

Οι μετασχηματισμοί εφαρμόζονται από τους μεταφραστές με σκοπό να βελτιώσουν κάποιο χαρακτηριστικό, όπως τον παραλληλισμό, την τοπικότητα στην προσπέλαση μνήμης και άλλα. Εμείς θα αναφέρουμε δύο μετασχηματισμούς που χρειάζονται για το σκοπό της εργασίας, την κανονικοποίηση βρόχων και τη σύμπτυξη βρόχων.

2.3 Κανονικοποίηση βρόχων

Η κανονικοποίηση είναι ίσως ο πιο βασικός μετασχηματισμός. Είναι ο πρώτος μετασχηματισμός που εφαρμόζουν οι προηγμένοι μεταφραστές μόλις ανιχνεύσουν βρόχο,

διότι μετατρέπει τον βρόχο σε μια απλή μορφή κι έτσι διευκολύνεται η ανάλυση και η εφαρμογή άλλων μετασχηματισμών.

Ορισμός 2.4 (Κανονικοποιημένος βρόχος). Ένας βρόχος $L = (i, l, u, s)$ λέγεται κανονικοποιημένος αν ισχύει ότι $i \in \{0, 1, \ldots, u\}$, δηλαδή $l = 0,\ u \geq 0,\ s = 1$.

Θα δούμε τώρα ότι κάθε βρόχος έχει ισοδύναμη κανονικοποιημένη μορφή. Ας υποθέσουμε ότι ο L είναι μη κανονικοποιημένος. Για να τον κανονικοποιήσουμε πρέπει να βρούμε έναν τύπο που να αντιστοιχεί το i σε κανονικοποιημένες τιμές και έναν τύπο ανάκτησης των αρχικών τιμών του. Για τυχόν i και $s > 0$, από τον ορισμό έχουμε

$$\begin{aligned} i \in \mathcal{I}_L &\Rightarrow i \equiv l \ (\mathrm{mod} s) \\ &\Leftrightarrow \exists k_i \in \mathbb{Z},\ i = l + s \cdot k_i \\ &\Leftrightarrow \exists k_i \in \mathbb{Z},\ k_i = \frac{i - l}{s} \end{aligned} \tag{2.10}$$

Παρατηρούμε ότι $k_i \in \{0, 1, \ldots, \lfloor \frac{u-l}{s} \rfloor\}$. Παρόμοια μπορεί να γίνει για την περίπτωση $s < 0$. Έτσι, θέτουμε για αυθαίρετο s τον μετασχηματισμό $T_{norm} : \mathcal{I}_L \to \mathcal{I}_{L'} = \left\{0, 1, \ldots, \lfloor \frac{u-l}{s} \rfloor\right\}$

$$i' = T_{norm}(i) = \frac{i - l}{s} \tag{2.11}$$

και βλέπουμε ότι δημιουργείται ένας νέος, κανονικοποιημένος βρόχος $L' = (i', l' = 0, u' = N_L - 1, s' = 1)$. Επίσης, μπορούμε σε κάθε επανάληψη να ανακτήσουμε τις τιμές του i με τη συνάρτηση

$$i = T^{-1}_{norm}(i') = l + s \cdot i' \tag{2.12}$$

Άρα ο L είναι ισοδύναμος του L'.

Η αναγωγή της κανονικοποίησης στη γενική περίπτωση δομής βρόχων $\mathcal{L}$, όπου τα όρια μπορεί να περιέχουν εξαρτήσεις, είναι σχεδόν άμεση και αρκεί ο κάθε βρόχος της δομής να μετατραπεί στη μορφή του ορισμού. Μόνο που επειδή έχουμε εξαρτήσεις, ο βρόχος βάθους k χρειάζεται τις ανακτήσεις των $i_1, i_2, \ldots, i_{k-1}$. Επίσης, επειδή η διαδικασία της κανονικοποίησης περιέχει αλλαγή μεταβλητής, μας βολεύει

να θέσουμε τα όρια ως συναρτήσεις των νέων μετρητών, δηλαδή θα είναι

$$\begin{aligned}
\bar{l}_2(i'_1) &= l_2(l_1 + s_1 \cdot i'_1) \\
\bar{u}_2(i'_1) &= u_2(l_1 + s_1 \cdot i'_1) \\
\bar{l}_3(i'_1, i'_2) &= l_3(l_1 + s_1 \cdot i'_1,\ \bar{l}_2(i'_1) + s_2 \cdot i'_2) \\
\bar{u}_3(i'_1, i'_2) &= u_3(l_1 + s_1 \cdot i'_1,\ \bar{l}_2(i'_1) + s_2 \cdot i'_2) \\
&\vdots \\
\bar{l}_n(i'_1, \ldots, i'_{n-1}) &= l_n(l_1 + s_1 \cdot i'_1, \ldots, \bar{l}_{n-1}(i'_1, \ldots, i'_{n-2}) + s_{n-1} \cdot i'_{n-1}) \\
\bar{u}_n(i'_1, \ldots, i'_{n-1}) &= u_n(l_1 + s_1 \cdot i'_1, \ldots, \bar{l}_{n-1}(i'_1, \ldots, i'_{n-2}) + s_{n-1} \cdot i'_{n-1})
\end{aligned}$$

Συνεπώς, η νέα κανονικοποιημένη δομή $\mathcal{L}'$ θα περιέχει εσωτερικούς βρόχους με άνω όρια

$$\begin{aligned}
u'_1 &= \left\lfloor \frac{u_1 - l_1}{s_1} \right\rfloor \\
u'_2(i'_1) &= \left\lfloor \frac{(\bar{u}_2 - \bar{l}_2)(i'_1)}{s_2} \right\rfloor \\
&\vdots \\
u'_n(i'_1, \ldots, i'_{n-1}) &= \left\lfloor \frac{(\bar{u}_n - \bar{l}_n)(i'_1, \ldots, i'_{n-1})}{s_n} \right\rfloor
\end{aligned}$$

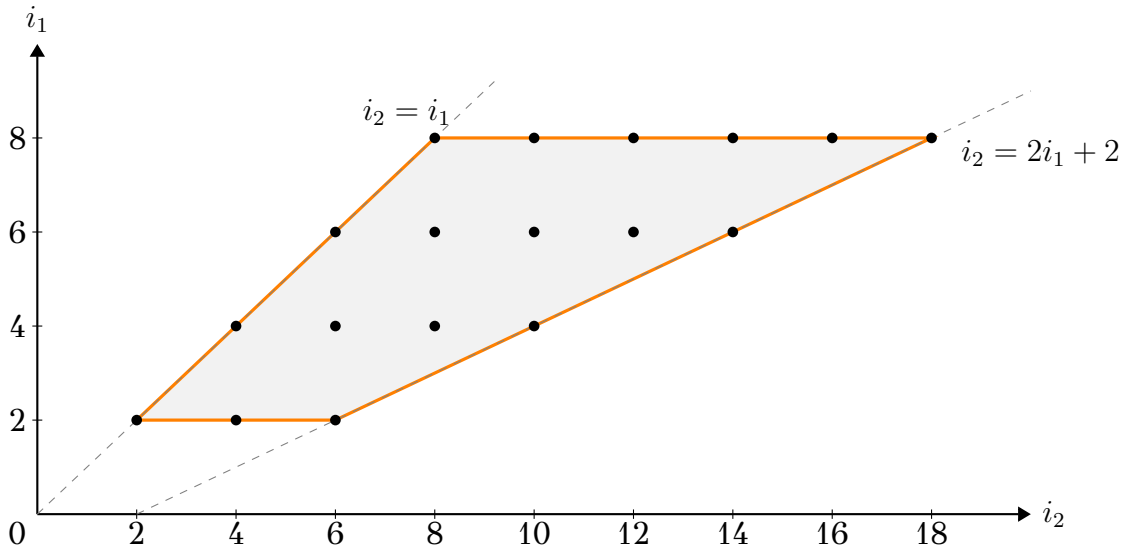
και η συνάρτηση ανάκτησης θα είναι

$$\begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix} = T^{-1}_{norm}(\mathbf{i}') = \begin{pmatrix} l_1 + s_1 \cdot i'_1 \\ \bar{l}_2(i'_1) + s_2 \cdot i'_2 \\ \vdots \\ \bar{l}_n(i'_1, \ldots, i'_{n-1}) + s_n \cdot i'_n \end{pmatrix} \tag{2.13}$$

Η κανονικοποίηση αποτυπώνεται γεωμετρικά ως μετατόπιση προς την αρχή των αξόνων. Ενώ στους ορθογωνικούς η μετατόπιση είναι ενιαία για όλο το σχήμα, στους μη ορθογωνικούς διαφορετικά χωρία του σχήματος θα μετατοπιστούν διαφορετικά ανάλογα με τις εξαρτήσεις.

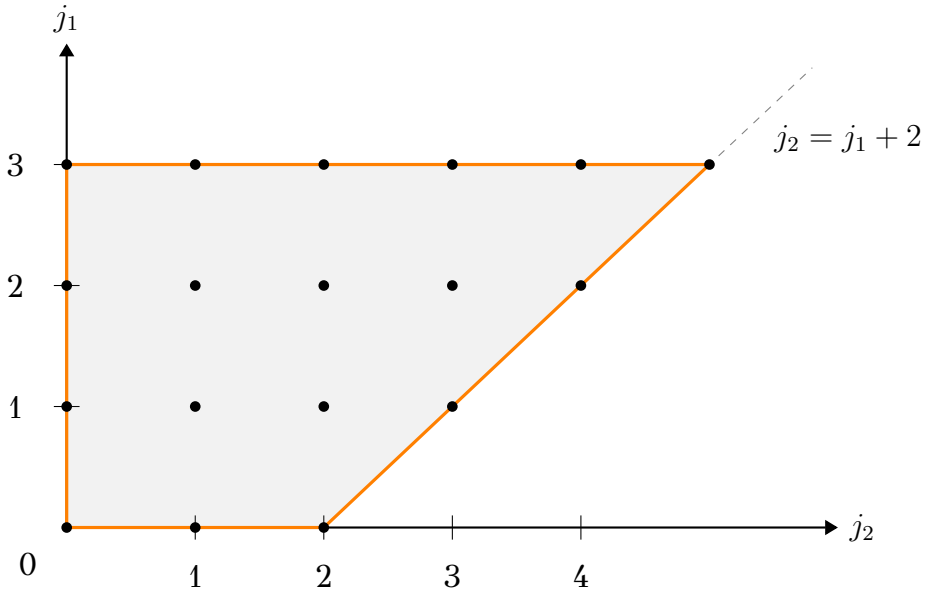
Στο Σχήμα 2.3 μπορούμε να δούμε ένα παράδειγμα κανονικοποίησης και τη γεωμετρική αναπαράστασή της. Το σχήμα αποτελείται από δύο μέρη, το 2.3α περιλαμβάνει την αρχική μορφή και το 2.3β την κανονικοποιημένη μορφή. Κάθε μέρος έχει στο πάνω μέρος τη δομή βρόχων και στο κάτω τη γεωμετρική αναπαράσταση.

```
for (i1 = 2; i1 <= 8; i1+=2)
  for (i2 = i1; i2 <= 2*i1+2; i2+=2)
```



(α) Αρχική μορφή

```
for (j1 = 0; j1 <= (8-2)/2; j1++)
  for (j2 = 0; j2 <= (2*j1+4)/2; j2++)
```



(β) Κανονικοποιημένη μορφή

Σχήμα 2.3: Παράδειγμα κανονικοποίησης

2.4 Σύμπτυξη βρόχων

Σύμπτυξη βρόχων είναι η μετατροπή μιας δομής εμφωλευμένων βρόχων σε έναν ισοδύναμο μεμονωμένο βρόχο. Ο μετασχηματισμός αυτός δύναται να εξάγει παραλληλισμό και να εξισορροπήσει το φόρτο εργασίας που κατανέμεται στα νήματα, κυρίως όταν ο εξωτερικός βρόχος δεν μπορεί να παραλληλοποιηθεί ικανοποιητικά.

Η σύμπτυξη είναι στον πυρήνα του μαθηματικό πρόβλημα. Στη βιβλιογραφία υπάρχουν δύο δημοσιεύσεις [4, 2] που ασχολούνται με το πρόβλημα. Η πρώτη [4] εισήγαγε τη σύμπτυξη ορθογωνικών βρόχων και η δεύτερη [2] παρουσιάζει μία μέθοδο για την περίπτωση των μη ορθογωνικών. Παρόμοιες εργασίες [5, 6] προσπαθούν να επιλύσουν το πρόβλημα του άνισου φόρτου εργασίας που παρουσιάζεται στην παραλληλοποίηση μη ορθογωνικών βρόχων με καλύτερη διαμέριση των επαναλήψεων στα νήματα.

Για να δούμε τη διαδικασία πιο συγκεκριμένα, αν έχουμε μια κανονικοποιημένη δομή $\mathcal{L}$ βάθους n, θέλουμε να την αντικαταστήσουμε με ένα βρόχο $L = (iter, 0, |\mathcal{I}_\mathcal{L}| - 1, 1)$ και στη συνέχεια να κάνουμε ανάκτηση των μετρητών των βρόχων της $\mathcal{L}$ από την τιμή του $iter$. Βλέπουμε λοιπόν ότι για να εφαρμόσουμε τον μετασχηματισμό χρειαζόμαστε

1. Το συνολικό πλήθος επαναλήψεων της $\mathcal{L}$

2. Μία μέθοδο ανάκτησης του αρχικού $\mathbf{i} \in \mathcal{I}_\mathcal{L}$ από τον $iter$

Για το (2) ιδανικά θα θέλαμε να βρούμε μία συνάρτηση $T_{col} : \mathcal{I}_\mathcal{L} \to \{0, \ldots, |\mathcal{I}_\mathcal{L}| - 1\}$ 1-1 και επί και στη συνέχεια αντιστρέφοντάς την να παράξουμε κλειστούς τύπους που υπολογίζουν τους αρχικούς μετρητές συναρτήσει του $iter$. Οι διαδικασίες αυτές λέγονται κατάταξη (ranking) και ανάκτηση (unranking). Επίσης, θέλουμε η συνάρτηση να είναι δομημένη έτσι ώστε να διατηρεί τη λεξικογραφική διάταξη. Συνεπώς, η T_{col} πρέπει για κάθε $\mathbf{i} \in \mathcal{I}_\mathcal{L}$ να μας δίνει τον αριθμό των επαναλήψεων που προη-

γούνται του **i**. Άρα η θεωρητική μορφή της T_{col} θέλουμε να είναι

$$\begin{aligned}
T_{col}(\mathbf{i}) &= |\,\{\mathbf{i}' \in \mathcal{I}_{\mathcal{L}} \mid \mathbf{i}' \prec_{lex} \mathbf{i}\}\,| \\
&= \left| \left\{ \mathbf{i}' \in \mathcal{I}_{\mathcal{L}} \mid \bigvee_{k=1}^{n} (\forall m \in \{1,\ldots,k-1\},\ i'_m = i_m \ \wedge\ i'_k < i_k) \right\} \right| \\
&= \left| \bigcup_{k=1}^{n} \{\mathbf{i}' \in \mathcal{I}_{\mathcal{L}} \mid \forall m \in \{1,\ldots,k-1\},\ i'_m = i_m \ \wedge\ i'_k < i_k\} \right| \\
&= \sum_{k=1}^{n} |\{\mathbf{i}' \in \mathcal{I}_{\mathcal{L}} \mid \forall m \in \{1,\ldots,k-1\},\ i'_m = i_m \ \wedge\ i'_k < i_k\}| \\
&= \sum_{k=1}^{n} \left(\sum_{i'_k=0}^{i_k-1} \sum_{i'_{k+1}=0}^{u_{k+1}(i_1,\ldots,i_{k-1},i'_k)} \cdots \sum_{i'_n=0}^{u_n(i_1,\ldots,i_{k-1},i'_k,\ldots,i'_{n-1})} 1 \right) \\
&= \sum_{k=1}^{n} \left(\sum_{i'_k=0}^{i_k-1} N_{\mathcal{L}}^{(k+1)}(i_1, i_2, \ldots, i_{k-1}, i'_k) \right) \qquad (2.14)
\end{aligned}$$

Η μορφή αυτή αναφέρεται και στο άρθρο [2], απλώς εδώ είδαμε τη γενική περίπτωση και δείξαμε λίγο πιο αναλυτικά πώς παράγεται. Η συνάρτηση αυτή είναι 1-1, αφού είναι γνησίως αύξουσα, και επί στο $\{0,\ldots,|\mathcal{I}_{\mathcal{L}}|-1\}$. Έτσι, δείξαμε ότι υπάρχει συνάρτηση που ικανοποιεί τις προϋποθέσεις. Εύκολα μπορούμε να διακρίνουμε ότι, αφού στη λεξικογραφική διάταξη η επανάληψη $\mathbf{i} = (u_1, u_2(u_1), \ldots, u_n(u_1, u_2, \ldots, u_{n-1}))$ είναι η τελευταία επανάληψη και η αμέσως επόμενη νοητή επανάληψη είναι η $\mathbf{i} = (u_1 + 1, 0, \ldots, 0)$, ισχύει

$$\begin{aligned}
|\mathcal{I}_{\mathcal{L}}| &= T_{col}(u_1, u_2(u_1), \ldots, u_n(u_1, u_2, \ldots, u_{n-1})) + 1 \\
&= T_{col}(u_1 + 1, 0, \ldots, 0)
\end{aligned}$$

Μπορούμε να δούμε ότι αναλόγως των εξαρτήσεων η (2.14) είναι πολυώνυμο έως βαθμού n και η αντιστροφή γίνεται δύσκολη. Η εφαρμογή του μετασχηματισμού λοιπόν είναι διαφορετική μεταξύ *ορθογωνικών* και *μη ορθογωνικών* βρόχων.

2.4.1 Σύμπτυξη ορθογωνικών βρόχων

Στην περίπτωση των ορθογωνικών βρόχων η διαδικασία είναι σχετικά απλή. Αρχικά, για το πλήθος επαναλήψεων έχουμε τον τύπο (2.5). Στη συνέχεια, για την T_{col} η

(2.14) γίνεται

$$\begin{aligned} T_{col}(\mathbf{i}) &= \sum_{k=1}^{n} \left(\sum_{i'_k=0}^{i_k-1} \sum_{i'_{k+1}=0}^{u_{k+1}} \cdots \sum_{i'_n=0}^{u_n} 1 \right) \\ &= \sum_{k=1}^{n} \left(i_k \cdot N_{\mathcal{L}}^{(k+1)} \right) \end{aligned}$$

Για να παράξουμε τώρα τους τύπους που υπολογίζουν τα $i_1, i_2, \ldots, i_n$ από το $iter$ πρέπει να λύσουμε ως προς $i_1, i_2, \ldots, i_n$. Για να γίνει πιο εμφανές το πώς θα γίνει, μπορούμε να γράψουμε την T_{col} ως

$$\forall k \in \{1, \ldots, n\},\ iter = T_{col}(\mathbf{i}) = \sum_{m=1}^{k-1} \left(i_m \cdot N_{\mathcal{L}}^{(m+1)} \right) + i_k \cdot N_{\mathcal{L}}^{(k+1)} + \sum_{m=k+1}^{n} \left(i_m \cdot N_{\mathcal{L}}^{(m+1)} \right)$$

Έτσι βλέπουμε ότι πρέπει να διαιρέσουμε με $N_{\mathcal{L}}^{(k+1)}$

$$\left\lfloor \frac{iter}{N_{\mathcal{L}}^{(k+1)}} \right\rfloor = \left\lfloor \sum_{m=1}^{k-1} \left(i_m \ \frac{N_{\mathcal{L}}^{(m+1)}}{N_{\mathcal{L}}^{(k+1)}} \right) + i_k + \sum_{m=k+1}^{n} \left(i_m \ \frac{N_{\mathcal{L}}^{(m+1)}}{N_{\mathcal{L}}^{(k+1)}} \right) \right\rfloor$$

$$= \lfloor A + i_k + B \rfloor$$

Για το A παρατηρούμε ότι για $m \leq k-1$ το $N_{\mathcal{L}}^{(m+1)}$ ως γινόμενο περιέχει το N_{L_k}, οπότε έχουμε

$$\begin{aligned} A &= \sum_{m=1}^{k-1} \left(i_m N_{L_{m+1}} \cdots N_{L_k} \frac{N_{\mathcal{L}}^{(k+1)}}{N_{\mathcal{L}}^{(k+1)}} \right) \\ &= N_{L_k} \cdot \sum_{m=1}^{k-1} \left(i_m N_{L_{m+1}} \cdots N_{L_{k-1}} \right) \\ &= N_{L_k} \cdot M \end{aligned}$$

Για το B παρατηρούμε ότι μπορεί να εξαλειφθεί, αφού βρίσκεται μέσα σε $\lfloor \cdot \rfloor$ και ισχύει

$$\begin{aligned} 0 \leq B &\leq \sum_{m=k+1}^{n} \left((N_{L_m} - 1) \ \frac{N_{\mathcal{L}}^{(m+1)}}{N_{\mathcal{L}}^{(k+1)}} \right) \\ &= \sum_{m=k+1}^{n} \left(\frac{N_{\mathcal{L}}^{(m)}}{N_{\mathcal{L}}^{(k+1)}} - \frac{N_{\mathcal{L}}^{(m+1)}}{N_{\mathcal{L}}^{(k+1)}} \right) \\ &= \left(1 - \frac{N_{\mathcal{L}}^{(k+2)}}{N_{\mathcal{L}}^{(k+1)}} \right) + \left(\frac{N_{\mathcal{L}}^{(k+2)}}{N_{\mathcal{L}}^{(k+1)}} - \frac{N_{\mathcal{L}}^{(k+3)}}{N_{\mathcal{L}}^{(k+1)}} \right) + \cdots + \left(\frac{N_{\mathcal{L}}^{(n)}}{N_{\mathcal{L}}^{(k+1)}} - \frac{1}{N_{\mathcal{L}}^{(k+1)}} \right) \\ &= 1 - \frac{1}{N_{\mathcal{L}}^{(k+1)}} \\ &< 1 \end{aligned}$$

Άρα, συνολικά από τα παραπάνω έχουμε

$$\forall k \in \{1, \ldots, n\},\ \exists M_k \in \mathbb{Z},\ \left\lfloor \frac{iter}{N_{\mathcal{L}}^{(k+1)}} \right\rfloor = N_{L_k} \cdot M_k + i_k$$
$$\Rightarrow i_k = \left\lfloor \frac{iter}{\prod_{k'=k+1}^{n} N_{L_{k'}}} \right\rfloor \bmod N_{L_k}$$

Συνεπώς, η συνάρτηση ανάκτησης θα είναι

$$\begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix} = T_{col}^{-1}(iter) = \begin{pmatrix} \left\lfloor \frac{iter}{\prod_{k=2}^{n} N_{L_k}} \right\rfloor \bmod N_{L_1} \\ \left\lfloor \frac{iter}{\prod_{k=3}^{n} N_{L_k}} \right\rfloor \bmod N_{L_2} \\ \vdots \\ iter \bmod N_{L_n} \end{pmatrix} \tag{2.15}$$

Έτσι, βλέπουμε ότι για να υλοποιήσουμε σύμπτυξη ορθογωνικών βρόχων από όλα τα προπάνω χρειαζόμαστε μόνο τις (2.5) και (2.15).

2.4.2 Σύμπτυξη μη ορθογωνικών βρόχων

Στην περίπτωση των μη ορθογωνικών βρόχων η διαδικασία γίνεται πιο δύσκολη έως και αδύνατη. Θα παρουσιάσουμε συνοπτικά τη μεθοδολογία που ακολουθείται στο άρθρο [2], αλλά δεν θα τη χρησιμοποιήσουμε στην υλοποίηση. Αρχικά, για το πλήθος επαναλήψεων, όπως αναφέραμε στην Ενότητα 2.1, ο τύπος (2.3) μπορεί να απλοποιηθεί στο πολυώνυμο Ehrhart που υπολογίζεται σε $\mathcal{O}(1)$ χρόνο. Όμως, η απλοποίηση γίνεται ιδιαίτερα δύσκολη όσο το βάθος n μεγαλώνει. Οι συγγραφείς του άρθρου προτείνουν τη χρήση των βιβλιοθηκών PolyLib [7] ή barvinok [8], οι οποίες υλοποιούν εναλλακτικές γεωμετρικές μεθόδους που υπολογίζουν το πολυώνυμο Ehrhart. Στη συνέχεια, για την ανάκτηση των μετρητών δεν μπορούν να βρεθούν καθολικοί τύποι που να λειτουργούν για διαφορετικές δομές $\mathcal{L}$. Έτσι, για κάθε $\mathcal{L}$ είναι υποχρεωτικός ο υπολογισμός της T_{col} και με την αντιστροφή της θα παραχθούν τύποι που λειτουργούν για τη συγκεκριμένη δομή και όλη αυτή η διαδικασία γίνεται κατά τη μετάφραση. Η δυσκολία της αντιστροφής έγκειται στο ότι η T_{col} ανάλογα με τους τύπους των ορίων θα είναι πολυώνυμο βαθμού το πολύ n. Μια πρώτη πληροφορία που έχουμε και θα φανεί χρήσιμη είναι ότι το διάνυσμα $\mathbf{i} = (l_1, l_2(i_1), \ldots, l_n(i_1, i_2, \ldots, i_{n-1})) = (0, 0, \ldots, 0)$ αντιστοιχεί στην πρώτη επανάληψη

της $\mathcal{L}$, άρα ένα πρώτο δεδομένο που έχουμε είναι

$$T_{col}(0,0,\ldots,0)=0$$

Θεωρούμε λοιπόν την άγνωστη παράμετρο $iter$. Για το i_1 παίρνουμε την εξίσωση $T_{col}(i_1,0,\ldots,0)=iter$ και λύνουμε το πολυώνυμο

$$P_1(x)=T_{col}(x,0,\ldots,0)-iter=0$$

Οι λύσεις του P_1 θα είναι συμβολικές λύσεις παραμετροποιημένες ως προς το $iter$. Από τις λύσεις επιλέγουμε την $x_1(iter)$ τέτοια ώστε να ισχύει $\lfloor x_1(0)\rfloor=0$, η οποία σίγουρα υπάρχει και είναι μοναδική, διότι η T_{col} είναι 1-1 και επί, άρα παίρνουμε

$$i_1=\lfloor x_1(iter)\rfloor$$

Στη συνέχεια για το i_2, χρησιμοποιώντας την εύρεση του i_1, σκεπτόμενοι παρόμοια έχουμε

$$T_{col}(i_1,i_2,0,\ldots,0)=iter$$
$$\Rightarrow P_2(x)=T_{col}(i_1,x,0,\ldots,0)-iter=0$$

Οι λύσεις του P_2 τώρα θα είναι παραμετροποιημένες ως προς i_1 και $iter$. Έτσι τώρα επιλέγουμε την $x_2(i_1,iter)$ τέτοια ώστε $\lfloor x_2(0,0)\rfloor=0$, άρα παίρνουμε

$$i_2=\lfloor x_2(i_1,iter)\rfloor$$

Η διαδικασία αυτή επαναλαμβάνεται αναλόγως μέχρι και το i_{n-1}. Για το i_n έχουμε απλώς

$$i_n=iter-T_{col}(i_1,i_2,\ldots,i_{n-1},0)$$

Άρα, συνολικά η συνάρτηση ανάκτησης για τη δοθείσα $\mathcal{L}$ θα είναι

$$\begin{pmatrix} i_1\\ i_2\\ \vdots\\ i_n\end{pmatrix}=T_{col}^{-1}(iter)=\begin{pmatrix} \lfloor x_1(iter)\rfloor\\ \lfloor x_2(i_1,iter)\rfloor\\ \vdots\\ \lfloor x_{n-1}(i_1,i_2,\ldots,i_{n-2},iter)\rfloor\\ iter-T_{col}(i_1,i_2,\ldots,i_{n-1},0)\end{pmatrix} \tag{2.16}$$

Είδαμε λοιπόν ότι η διαδικασία αντιστροφής της T_{col} απαιτεί την επίλυση πολυωνύμων. Οι συγγραφείς του άρθρου χρησιμοποιούν για την επίλυση των πολυωνύμων το λογισμικό συμβολικών υπολογισμών *Maxima* [9]. Αναφέραμε επίσης παραπάνω

ότι γίνεται και χρήση έτοιμων βιβλιοθηκών. Παρατηρούμε λοιπόν ότι η μεθοδολογία του άρθρου προϋποθέτει την ύπαρξη πρόσθετου λογισμικού πέραν του μεταφραστή. Επίσης, γνωρίζουμε από τα μαθηματικά πως μόνο τα πολυώνυμα βαθμού $n \leq 4$ διαθέτουν γενική λύση σε κλειστή αλγεβρική μορφή. Συνεπώς, η μεθοδολογία είναι εφαρμόσιμη μόνο σε δομές $\mathcal{L}$ με βάθος $n \leq 4$.

Μια εναλλακτική μέθοδος είναι να θεωρήσουμε όλες τις σταθερές των ορίων ως άγνωστες παραμέτρους και για κάθε βάθος $n \leq 4$ ξεχωριστά να υπολογίσουμε συμβολικά το πολυώνυμο Ehrhart και να παράξουμε τη συνάρτηση ανάκτησης για συγκεκριμένα βάθη $n \leq 4$ ξεχωριστά κρατώντας όλες τις σταθερές των ορίων ως άγνωστες παραμέτρους. Έτσι, για κάθε n ξεχωριστά μπορούμε να έχουμε προκατασκευασμένες συναρτήσεις ανάκτησης, οι οποίες θα δέχονται ως παραμέτρους όλες τις άγνωστες παραμέτρους.

Εφόσον γνωρίζουμε ότι τα πολυώνυμα βαθμού 2 επιλύονται εύκολα και επειδή η μέθοδος αυτή γίνεται αρκετά πολύπλοκη για $n \geq 3$, θα παρουσιάσουμε τη περίπτωση $n = 2$. Υποθέτουμε ότι η κανονικοποιημένη $\mathcal{L}$ έχει βάθος 2. Το άνω όριο του εσωτερικού βρόχου ως αφινική συνάρτηση του i_1 θα έχει τη μορφή $u_2(i_1) = a_1 i_1 + a_0$. Αρχικά, η T_{col} θα είναι

$$\begin{aligned}
T_{col}(i_1, i_2; a_1, a_0) &= \sum_{k=1}^{2} \left(\sum_{i'_k=0}^{i_k-1} N_{\mathcal{L}}^{(k+1)}(i_1, i_2, \ldots, i_{k-1}, i'_k) \right) \\
&= \sum_{i'_1=0}^{i_1-1} \sum_{i'_2=0}^{a_1 i'_1 + a_0} 1 + \sum_{i'_2=0}^{i_2-1} 1 \\
&= \sum_{i'_1=0}^{i_1-1} (a_1 i'_1 + a_0 + 1) + i_2 \\
&= a_1 \sum_{i'_1=0}^{i_1-1} i'_1 + a_0 \sum_{i'_1=0}^{i_1-1} 1 + \sum_{i'_1=0}^{i_1-1} 1 + i_2 \\
&= a_1 \frac{(i_1 - 1) i_1}{2} + a_0 i_1 + i_1 + i_2 \qquad (2.17)
\end{aligned}$$

Επομένως, για το i_1 θα έχουμε

$$\begin{aligned} T_{col}(x,0;a_1,a_0) - iter &= 0 \\ \Rightarrow a_1\frac{(x-1)x}{2} + a_0x + x - iter &= 0 \\ \Rightarrow \frac{a_1}{2}x^2 + (-\frac{a_1}{2} + a_0 + 1)x - iter &= 0 \\ \Rightarrow a_1x^2 + (-a_1 + 2a_0 + 2)x - 2iter &= 0 \\ \Rightarrow x(iter;a_1,a_0) = \frac{-(-a_1+2a_0+2) \pm \sqrt{(-a_1+2a_0+2)^2 + 8a_1\cdot iter}}{2a_1} & \end{aligned}$$

Αφού θέλουμε να ισχύει $\lfloor x_1(0;a_1,a_0)\rfloor = 0$, επιλέγουμε

$$\begin{aligned} i_1 &= \lfloor x_1(iter;a_1,a_0)\rfloor \\ &= \left\lfloor \frac{-(-a_1+2a_0+2) + \sqrt{(-a_1+2a_0+2)^2 + 8a_1\cdot iter}}{2a_1} \right\rfloor \end{aligned}$$

Και για το i_2 θα έχουμε

$$\begin{aligned} i_2 &= x_2(iter;a_1,a_0) \\ &= iter - T_{col}(i_1,0;a_1,a_0) \\ &= iter - \left(a_1\frac{(i_1-1)i_1}{2} + a_0i_1 + i_1\right) \end{aligned}$$

Συνεπώς, συνολικά η συνάρτηση ανάκτησης είναι

$$\begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = T_{col}^{-1}(iter;a_1,a_0) = \begin{pmatrix} \left\lfloor \frac{-(-a_1+2a_0+2) + \sqrt{(-a_1+2a_0+2)^2 + 8a_1\cdot iter}}{2a_1} \right\rfloor \\ iter - \left(a_1\frac{(i_1-1)i_1}{2} + a_0i_1 + i_1\right) \end{pmatrix} \tag{2.18}$$

Συνεπώς, μπορούμε να υλοποιήσουμε τις συναρτήσεις $T_{col}(i_1,i_2,a_1,a_0),\ x_1(iter,a_1,a_0)$ και $x_2(iter,a_1,a_0)$ και όταν ο μεταφραστής κληθεί να εφαρμόσει σύμπτυξη σε δομή βάθους 2, να τις χρησιμοποιήσει. Βέβαια, η υλοποίηση της συνάρτησης x_1 δεν είναι τετριμμένη υπόθεση, διότι απαιτείται ο υπολογισμός τετραγωνικής ρίζας. Ο πιο άμεσος τρόπος υλοποίησης του υπολογισμού αυτού είναι να γίνει με τύπο κινητής υποδιαστολής. Τότε όμως όλος ο τύπος πρέπει να υπολογιστεί με αριθμητική κινητής υποδιαστολής. Συνεπώς, μπορεί να έχουμε απώλεια ακρίβειας. Για το λόγο αυτό, το τελικό αποτέλεσμα πρέπει να επαληθευτεί και, αν χρειαστεί, να αναπροσαρμοστεί με κάποια μικρή αύξηση ή μείωση. Η επαλήθευση μπορεί να γίνει με την την ανισότητα

$$T_{col}(i,0;a_1,a_0) \le iter < T_{col}(i+1,0;a_1,a_0)$$

Ένας πιο σύνθετος αλλά ίσως καλύτερος τρόπος για τον υπολογισμό της τετραγωνικής ρίζας είναι η υλοποίηση κάποιου αλγορίθμου υπολογισμού της χρησιμοποιώντας μόνο αριθμητική ακεραίων.

Οι παραπάνω μέθοδοι βασίζονται σε αλγεβρική προσέγγιση και για έναν τυπικό μεταφραστή μόνο η εναλλακτική μέθοδος για βάθος 2 είναι ρεαλιστικά υλοποιήσιμη. Η δική μας υλοποίηση, που θα παρουσιάσουμε παρακάτω, χρησιμοποιεί ευρετική μέθοδο για τη γενική περίπτωση και για βάθος 2 βασίζεται σε γεωμετρική προσέγγιση.

2.4.3 Αποκανονικοποίηση ανάκτησης

Τέλος, όλα τα παραπάνω έγιναν με την υπόθεση ότι έχουμε κανονικοποιημένους βρόχους. Σε περίπτωση μη κανονικοποιημένων βρόχων η διαδικασία δεν αλλάζει. Πρέπει όμως να σημειωθεί ότι η σύμπτυξη που εφαρμόζει η T_{col} δημιουργεί κανονικοποιημένο βρόχο. Έτσι, οι μετρητές που ανακτώνται αντιστοιχούν πάντα σε κανονικοποιημένη μορφή του αρχικού χώρου. Συνεπώς, αν ο αρχικός χώρος δεν ήταν κανονικοποιημένος, πρέπει να χρησιμοποιήσουμε την T_{norm}^{-1} στους ανακτημένους μετρητές που μας δίνει η T_{col}^{-1} για να πάρουμε τις σωστές τιμές. Άρα, οι αρχικοί μετρητές θα είναι

$$\begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix} = T_{norm}^{-1}\left(T_{col}^{-1}(iter)\right) \tag{2.19}$$

Κεφαλαιο 3

OpenMP και OMPI

3.1 OpenMP

3.2 OMPi

3.1 OpenMP

Το OpenMP [1] είναι ένα μια διεπαφή προγραμματισμού που αποτελείται από μια συλλογή από οδηγίες (directives), συναρτήσεις χρόνου εκτέλεσης και μεταβλητές περιβάλλοντος. Στοχεύει στην απλοποίηση της παραλληλοποίησης εφαρμογών κυρίως για συστήματα κοινόχρηστης μνήμης. Έχει γίνει το πλέον διαδεδομένο πρότυπο και υποστηρίζεται από τους περισσότερους σύγχρονους μεταφραστές C, C++ και Fortran.

Πριν την καθιέρωση τέτοιων διεπαφών, για την παραλληλοποίηση μιας εφαρμογής ο προγραμματιστής έπρεπε να διαχειρίζεται άμεσα τη δημιουργία και τον συγχρονισμό νημάτων εκτέλεσης χρησιμοποιώντας χαμηλού επιπέδου βιβλιοθήκες. Η προσέγγιση αυτή, αν και προσφέρει απόλυτο έλεγχο, αυξάνει δραματικά την πολυπλοκοτητα και την πιθανότητα σφαλμάτων. Το OpenMP προσφέρει ένα επίπεδο αφαίρεσης, μέσω του οποίου ο προγραμματιστής περιγράφει ποια τμήματα του προγράμματος μπορούν να εκτελεστούν παράλληλα και πού απαιτείται συγχρονισμός, χωρίς να χρειάζεται η τροποποίηση της σειριακής λογικής του προγράμματος. Έπειτα, ο μεταφραστής αναλαμβάνει την παραγωγή του απαιτούμενου χαμηλού επιπέδου κώδικα.

Το OpenMP βασίζεται στο μοντέλο διακλάδωσης-ένωσης (fork-join model) για την παράλληλη εκτέλεση. Πιο συγκεκριμένα, αρχικά το πρόγραμμα εκτελείται σειριακά από το *αρχικό νήμα* (initial thread). Όταν η ροή εκτέλεσης συναντήσει μια *παράλληλη περιοχή* (parallel region), η οποία ορίζεται μέσω μιας οδηγίας, το κύριο νήμα δημιουργεί μια ομάδα από νήματα (fork) και το κάθε νήμα εκετελεί παράλληλα και αυτόνομα τον κώδικα που βρίσκεται στην παράλληλη περιοχή. Στο τέλος της παράλληλης περιοχής τα νήματα συγχρονίζονται και τερματίζονται (join) και το κύριο νήμα συνεχίζει τη σειριακή εκτέλεση του προγράμματος.

Οι οδηγίες αποτελούν το μεγαλύτερο και σημαντικότερο μέρος του OpenMP, καθώς μέσω αυτών εκφράζεται η παραλληλοποίηση του προγράμματος. Οι κυριότερες λειτουργίες που προσφέρουν οι οδηγίες είναι ο ορισμός παράλληλης περιοχής, ο διαμοιρασμός εργασίας, το περιβάλλον δεδομένων και ο συγχρονισμός νημάτων. Επίσης, κάθε οδηγία περιλαμβάνει φράσεις (clauses) για τη ρύθμιση συγκεκριμένων επιλογών. Η σύνταξη μιας οδηγίας γίνεται με την οδηγία προεπεξεργαστή *pragma*, τη λέξη *omp*, το όνομα της οδηγίας και φράσεις που θα χρησιμοποιηθούν, δηλαδή

```
#pragma omp όνομα-οδηγίας [φράση[ [,] φράση] ... ] νέα-γραμμή
```

Παρακάτω θα αναφέρουμε τις οδηγίες που χρειάζονται στο πλαίσιο της εργασίας.

3.1.1 Οδηγία parallel

Η οδηγία parallel είναι ίσως η κυριότερη οδηγία και χρησιμοποιείται για να προσδιοριστεί μια παράλληλη περιοχή. Η σύνταξη είναι

```
#pragma omp parallel [φράση[ [,] φράση] ... ] νέα-γραμμή
{
  Σύνολο-εντολών
}
```

Αν το *Σύνολο-εντολών* περιλαμβάνει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν. Όταν το *αρχικό νήμα* συναντήσει μία παράλληλη περιοχή, δημιουργείται μια ομάδα από νήματα και το αρχικό νήμα γίνεται το *κύριο νήμα* (master thread) της ομάδας αυτής. Σε κάθε νήμα ανατίθεται ένας αναγνωριστικός ακέραιος αριθμός νήματος (thread number) και το κύριο νήμα έχει πάντα τον αριθμό νήματος 0. Επίσης, στο τέλος της παράλληλης περιοχής εισάγεται αυτόματα ένα υπονοούμενο *φράγμα* (barrier), το οποίο αναγκάζει όποιο νήμα το συναντήσει να αναμένει στο συγκεκριμένο σημείο μέχρι να καταφθάσει όλη η υπόλοιπη ομάδα. Έτσι, όλη η

ομάδα θα εκτελέσει παράλληλα το *Σύνολο-εντολών* και όταν κάθε νήμα ολοκληρώσει την εκτέλεση της περιοχής, το αρχικό νήμα θα συνεχίσει τη σειριακή εκτέλεση.

Ο προεπιλεγμένος αριθμός νημάτων που θα εκτελέσουν τις παράλληλες περιοχές είναι συνήθως ο αριθμός των λογικών πυρήνων που είναι ορατός από το Λειτουργικό Σύστημα. Αν ο χρήστης θέλει να τροποποιήσει τον αριθμό αυτό, μπορεί να χρησιμοποιήσει κάποιον από τους εξής τρόπους:

1. Εισάγοντας τη φράση num_threads(*number_of_threads*) δίπλα στην οδηγία parallel.

2. Καλώντας τη συνάρτηση omp_set_num_threads(*number_of_threads*) πριν τη παράλληλη περιοχή. Προφανώς αν υπάρχουν παράλληλες περιοχές πριν την κλήση της συνάρτησης, για αυτές θα χρησιμοποιηθεί η προεπιλεγμένη τιμή.

3. Θέτοντας τη μεταβλητή περιβάλλοντος OMP_NUM_THREADS.

Τα παραπάνω δίνονται σε φθίνουσα σειρά προτεραιότητας, δηλαδή αν κάποιος χρησιμοποιήσει και τους τρεις τρόπους δίνοντας διαφορετική τιμή, θα υπερισχύσει ο πρώτος.

Σημαντικό ρόλο έχει και η κοινοχρησία που έχουν τα νήματα στα δεδομένα. Από προεπιλογή όλες οι μεταβλητές που δηλώνονται πριν τη δημιουργία των νημάτων θεωρούνται κοινόχρηστες. Υπάρχουν φράσεις που επιτρέπουν στο χρήστη να αλλάξει την προεπιλεγμένη συμπεριφορά και να ορίσει επακριβώς την κοινοχρησία των μεταβλητών. Κυριότερες τέτοιες φράσεις είναι οι shared(*list*) και private(*list*).

Στον Κώδικα 3.1 βλέπουμε ένα παράδειγμα της οδηγίας parallel. Στην αρχή ορίζουμε το πλήθος των νημάτων μέσω της συνάρτησης που προσφέρει το OpenMP. Στη συνέχεια δημιουργείται μια παράλληλη περιοχή και το κάθε νήμα αποθηκεύει το αναγνωριστικό του σε μια μεταβλητή. Τέλος, κάθε νήμα θα τυπώσει το αναγνωριστικό του και τη φράση "Hello world!".

Κώδικας 3.1 Παράδειγμα χρήσης parallel

```
#include <stdio.h>
#include <omp.h>

#define NUM_THREADS 4

int main()
{
  omp_set_num_threads(NUM_THREADS);
  #pragma omp parallel
  {
    int thread_id = omp_get_thread_num();
    printf("Thread %d: Hello world!\n", thread_id);
  }
  return 0;
}
```

3.1.2 Οδηγία for

Η οδηγία for είναι μία οδηγία διαμοιρασμού εργασίας. Επιτρέπει τον εύκολο διαμοιρασμό των επαναλήψεων ενός βρόχου επανάληψης στην ομάδα νημάτων. Η οδηγία πρέπει να βρίσκεται μέσα σε παράλληλη περιοχή και αμέσως μετά από αυτή να ακολουθεί δομή βρόχων. Η σύνταξη είναι

```
#pragma omp parallel [ ... ] νέα-γραμμή
{
  Σύνολο-εντολών
  #pragma omp for [φράση[ [,] φράση] ... ] νέα-γραμμή
    Βρόχοι-for
}
```

Επειδή αρκετές φορές χρειάζεται να παραλληλοποιήσουμε μόνο μία δομή βρόχων, το OpenMP μας δίνει τη δυνατότητα να παραλληλοποιήσουμε μόνο μια δομή βρόχων με την πιο σύντομη σύνταξη

```
#pragma omp parallel for [φράση[ [,] φράση] ... ] νέα-γραμμή
  Βρόχοι-for
```

Ο διαμοιρασμός της εργασίας γίνεται διαμερίζοντας τις επαναλήψεις του πιο εξωτερικού βρόχου. Έμμεσα όμως διαμοιράζονται και οι εσωτερικές επαναλήψεις. Παρατηρούμε λοιπόν ότι αν έχουμε μη ορθογωνικούς βρόχους, τότε το κάθε νήμα θα εκτελέσει διαφορετικό αριθμό επαναλήψεων.

Το OpenMP απαιτεί ο βρόχος στον οποίο θα εφαρμοστεί διαμοιρασμός να είναι σε κανονική μορφή. Η κανονική μορφή ορίζει τις πιθανές συντακτικές μορφές που μπορεί να έχει ο βρόχος. Ένα χαρακτηριστικό της κανονικής μορφής που είναι σχετικό με την εργασία είναι ότι τα όρια του βρόχου πρέπει να είναι της μορφής $a_1 \cdot var\_outer + a_2$, όπου var_outer ο μετρητής του εξωτερικού βρόχου και οι μεταβλητές a_1, a_2 είναι τύπου ακεραίων. Επίσης, οι μεταβλητές a_1 και a_2 δεν είναι απαραίτητο να είναι γνωστές σε χρόνο μετάφρασης. Επίσης, ένα ακόμη χαρακτηριστικό είναι ότι επιτρέπει μόνο μία συνθήκη τερματισμού.

Η οδηγία for, εκτός από τις φράσεις κοινοχρησίας δεδομένων, υποστηρίζει επιπρόσθετες φράσεις που έχουν να κάνουν με χαρακτηριστικά βρόχων. Εμείς θα αναφερθούμε φράσεις ordered, schedule και collapse.

Η φράση ordered χρησιμοποιείται όταν θέλουμε κάποιες εντολές του σώματος της δομής βρόχων να εκτελεστούν με σειρά σειριακής εκτέλεσης. Η φράση schedule(*kind[, chunk_size]*) ρυθμίζει με το όρισμα *kind* τον τρόπο δρομολόγησης με τον οποίο οι επαναλήψεις κατανέμονται στα νήματα και με το *chunk_size* το μέγεθος των υποσυνόλων στο οποίο θα διαιρεθούν οι επαναλήψεις. Οι κυριότερες επιλογές του *kind* είναι:

- static: Τα υποσύνολα διανέμονται ομοιόμορφα στα νήματα. Είναι ο πιο απλός τρόπος δρομολόγησης και κατάλληλος αν ο φόρτος εργασίας είναι ισορροπημένος.

- dynamic: Τα υποσύνολα διανέμονται στα νήματα με δυναμικό τρόπο. Σε κάθε αδρανές νήμα διανέμεται ένα υποσύνολο επαναλήψεων. Όταν ένα νήμα εκτελέσει το υποσύνολο αυτό, τότε του δίνεται κάποιο επόμενο υποσύνολο. Η διαδικασία επαλαναμβάνεται έως ότου τελειώσουν τα διαθέσιμα υποσύνολα. Αυτός ο τρόπος επιτρέπει τη χρονική ελαχιστοποίηση της κατάστασης όπου κάποια νήματα έχουν τελειώσει τις εκτελέσεις τους, ενώ κάποια άλλα ακόμα εκτελούν. Ο τρόπος αυτός είναι κατάλληλος όταν έχουμε μη ισορροπημένο

φόρτο εργασίας σε διαφορετικές επαναλήψεις. Όμως, η δυναμική κατανομή εργασιών απαιτεί παραπάνω επεξεργαστική ισχύ, συνεπώς η λανθασμένη χρήση του μπορεί να δώσει χειρότερους χρόνους εκτέλεσης.

Οι ordered και schedule δε θα μας χρησιμεύσουν κάπου στο πλαίσιο της εργασίας, αλλά τις περιγράψαμε συνοπτικά, γιατί θα χρειαστεί να τις αναφέρουμε και παρακάτω.

Η φράση collapse(*depth*) αιτείται την εφαρμογή σύμπτυξης βρόχων σε βάθος *depth* και είναι το κύριο θέμα της υλοποίησής μας. Το OpenMP υποστήριζε μέχρι και την έκδοση 4.5 τη σύμπτυξη μόνο ορθογωνικών βρόχων. Από την έκδοση 5.0 προστέθηκε η υποστήριξη της σύμπτυξης μη ορθογωνικών βρόχων. Η υποστήριξη όμως λόγω του ότι η διαδικασία της εφαρμογής είναι δύσκολη, έρχεται με περιορισμούς:

- Κάθε βρόχος βάθους $k \leq depth$ μπορεί να εξαρτάται από έναν μόνο εξωτερικό βρόχο. Στο Κεφάλαιο 2 περιγράψαμε τη διαδικασία για την πιο γενική περίπτωση, όπου ένας εσωτερικός βρόχος μπορεί να εξαρτάται από οποιουσδήποτε από τους $k-1$ εξωτερικούς βρόχους. Ο περιορισμός αυτός εξασφαλίζει ότι ο τύπος υπολογισμού του πλήθους επαναλήψεων του βρόχου είναι συνάρτηση μιας μεταβλητής.

- Αν ένας βρόχος εξαρτάται από εξωτερικό βρόχο και έχει όρια $l = a_l \cdot var\_outer + b_l$, $u = a_u \cdot var\_outer + b_u$ και βήμα s_{inner} και ο εξωτερικός βρόχος έχει βήμα s_{outer}, τότε το $(a_u - a_l) \cdot s_{outer}$ πρέπει να είναι πολλαπλάσιο του s_{inner}, δηλαδή πρέπει να ισχύει $(a_u - a_l) \cdot s_{outer} \bmod s_{inner} = 0$. Ο περιορισμός αυτός εξασφαλίζει ότι η μεταβολή του πλήθους επαναλήψεων του εσωτερικού βρόχου είναι σταθερός ακέραιος αριθμός.

Επιπλέον, αν σε μια οδηγία for χρησιμοποιείται η collapse και οι βρόχοι είναι μη ορθογωνικοί, τότε δεν μπορούν να χρησιμοποιηθούν οι schedule και ordered.

Στον Κώδικα 3.2 βλέπουμε ένα παράδειγμα της οδηγίας for, όπου με τη φράση `collapse` εφαρμόζεται σύμπτυξη σε μια δομή ορθογωνικών βρόχων βάθους 2. Εδώ δημιουργείται μια ομάδα 64 νημάτων, όπου το κάθε νήμα αποθηκεύει το αναγνωριστικό του και αρχικοποιεί έναν μετρητή. Στη συνέχεια, επειδή έχουμε τη φράση `collapse`, η δομή βρόχων θα αντικατασταθεί από έναν βρόχο με πλήθος επαναλήψεων 64 και οι απαναλήψεις του θα ισομοιραστούν στα 64 νήματα. Έτσι, κάθε νήμα θα εκτελέσει μία επανάληψη. Αν δεν είχαμε τη φράση `collapse`, τότε μόνο οι

επαναλήψεις του εξωτερικού βρόχου θα μοιράζονταν στα νήματα, οπότε 8 νήματα θα εκτελούσαν 8 επαναλήψεις και τα υπόλοιπα νήματα δεν θα αξιοποιούνταν.

Κώδικας 3.2 Παράδειγμα χρήσης for

```
#include <stdio.h>
#include <omp.h>

#define NUM_THREADS 64
#define NUM_ITERS_I 8
#define NUM_ITERS_J 8

int main()
{
  omp_set_num_threads(NUM_THREADS);
  #pragma omp parallel
  {
    int thread_id = omp_get_thread_num();
    int count = 0;
    #pragma omp for collapse(2)
      for (int i = 0; i < NUM_ITERS_I; i++)
        for(int j = 0; j < NUM_ITERS_J; j++)
          count++;
    printf("Thread %d executed %d iterations.\n", thread_id, count);
  }
  return 0;
}
```

3.2 OMPi

Ο OMPi [3] είναι ένας μεταφραστής OpenMP πηγαίου-σε-πηγαίο κώδικα και σύστημα χρόνου εκτέλεσης για τη γλώσσα C. Αναπτύσσεται από την Ομάδα Παράλληλης Επεξεργασίας του Πανεπιστημίου Ιωαννίνων και είναι ανοιχτού κώδικα. Υποστηρίζει τη φράση collapse μόνο για την έκδοση 4.5 του OpenMP και η εργασία

αυτή προσθέτει υποστήριξη για την έκδοση 5.0.

Σκοπός του OMPi είναι η υλοποίηση του προτύπου OpenMP. Ο μεταφραστής αναλαμβάνει να ανγνωρίσει τις διάφορες οδηγίες και φράσεις του OpenMP και να παράγει τον κατάλληλο κώδικα. Το σύστημα χρόνου εκτέλεσης αναλαμβάνει όλες τις απαραίτητες διαδικασίες των νημάτων που απαιτούνται κατά την εκτέλεση. Εμείς θα επικεντρωθούμε στο μέρος του μεταφραστή.

Η μετάφραση που αναλαμβάνει ο OMPi αποτελείται από τρία στάδια, την ανάλυση, τον μετασχηματισμό και την παραγωγή κώδικα. Κατά το στάδιο της ανάλυσης ο κώδικας αναλύεται λεκτικά, συντακτικά και σημασιολογικά και κατασκευάζεται το συντακτικό δέντρο για την αναπαράστασή του. Στη συνέχεια, το δέντρο αυτό αναλύεται για να αναγνωριστούν οι λειτουργίες που περιγράφονται με το OpenMP και τροποποιείται κατάλληλα ώστε να γίνουν οι κατάλληλοι μετασχηματισμοί στον κώδικα. Τέλος, το δέντρο μετατρέπεται σε κώδικα κι έτσι έχουμε ένα νέο πηγαίο κώδικα σε C.

3.2.1 Μετασχηματισμοί οδηγιών

Ο OMPi κατά το στάδιο του μετασχηματισμού διατρέχει το συντακτικό δέντρο και μόλις συναντήσει κάποια οδηγία του OpenMP, αποθηκεύει όλο το υποδέντρο που αναπαριστά το *Σύνολο-εντολών* που περιλαμβάνει η οδηγία αυτή. Μετά αντικαθιστά τις οδηγίες του OpenMP με κώδικα που περιλαμβάνει ρουτίνες του συστήματος χρόνου εκτέλεσης και τοποθετεί το σημασιολογικά αμετάβλητο υποδέντρο σε κατάλληλο σημείο στον νέο κώδικα. Επίσης, ο OMPi μεταφέρει τα περιεχόμενα της `main` σε μια καινούργια συνάρτηση `__original_main` και δημιουργεί καινούργια `main` στην οποία γίνονται οι απαραίτητες αρχικοποιήσεις του συστήματος χρόνου εκτέλεσης και καλείται η `__original_main`. Παρακάτω θα περιγράψουμε συνοπτικά τον κώδικα που αντικαθιστά τις οδηγίες parallel και for.

Στην περίπτωση της οδηγίας parallel ο OMPi αρχικά δημιουργεί μία συνάρτηση για κάθε παράλληλη περιοχή με όνομα `_thrFunc<`*`parallel_id`*`>_`, όπου *parallel_id* μια νοητή αρίθμηση της περιοχής. Η συνάρτηση αυτή περιέχει κώδικα που διαχειρίζεται την κοινοχρησία δεδομένων και το *Σύνολο-εντολών* της οδηγίας. Στη θέση της οδηγίας parallel εισάγεται κώδικας ιδαχείρισης των δεδομένων και στη συνέχεια ακολουθεί η συνάρτηση `_ort_execute_parallel`, η οποία δέχεται ως παράμετρο τη συνάρτηση `_thrFunc<`*`parallel_id`*`>_`. Η `_ort_execute_parallel` δημιουργεί την ομάδα

νημάτων και κάθε νήμα θα εκτελέσει τη _thrFunc<*parallel_id*>_.

Για την οδηγία for η διαδικασία μεταβάλλεται λίγο ανάλογα με το αν έχει δοθεί η φράση collapse(*depth*), όπου από προεπιλογή ισχύει $depth = 1$. Υπενθυμίζουμε ότι αν δεν έχει δοθεί η φράση, τότε ισχύει $depth = 1$. Εδώ εκτός από την οδηγία αφαιρούνται και οι *Βρόχοι-for* βάθους $depth$ και αντικαθίστανται από έναν μεμονωμένο βρόχο με μετρητή *iter*. Πριν το βρόχο αυτό τοποθετείται κώδικας που περιλαμβάνει τη δήλωση των μετρητών και τον υπολογισμό του πλήθους επαναλήψεων των βρόχων που αφαιρέθηκαν. Επίσης, εισάγεται η συνάρτηση _ort_entering_for, η οποία ενημερώνει το νήμα ότι εισέρχεται σε περιοχή διαμοιρασμού εργασίας και πραγματοποιεί κάποιες ρυθμίσεις σχετικές με τις φράσεις που περιείχε η οδηγία for. Επιπλέον, ο μεμονωμένος βρόχος πλαισιώνεται από κατάλληλο κώδικα που αναλαμβάνει να διαμοιράσει τις επαναλήψεις ανάλογα με τη ρύθμιση του schedule και τα όριά του θα είναι τιμές που υπολογίζονται από κατάλληλη συνάρτηση. Στο εσωτερικό του βρόχου γίνονται οι ανακτήσεις των αρχικών μετρητών από την τιμή του *iter* και ακολουθεί το σώμα των βρόχων. Τέλος, μετά το σώμα ακολουθεί η _ort_leaving_for, που ενημερώνει το νήμα ότι η περιοχή for τελείωσε, και η _ort_barrier_me, η οποία διαχειρίζεται το συγχρονισμό του φράγματος που υπάρχει πάντοτε στο τέλος της οδηγίας for.

Στον Κώδικα 3.3 μπορούμε να δούμε ένα παράδειγμα όλων των παραπάνω. Πιο συγκεκριμένα, βλέπουμε τις συναρτήσεις __ort_original_main και _thrFunc0_ που παράγει ο OMPi μεταφράζοντας τον Κώδικα 3.2.

Κώδικας 3.3: Παράδειγμα μετασχηματισμών του OMPi

```
int __original_main(int _argc_ignored, char ** _argv_ignored)
{
  omp_set_num_threads(4);
  /* (l10) #pragma omp parallel */
  {
    _ort_execute_parallel(_thrFunc0_, (void *) 0, -1, 0, 0);
  }
  return (0);
}

static void * _thrFunc0_(void * __arg)
```

```
{
  /* (l10) #pragma omp parallel -- body moved below */
  {
    int thread_id = omp_get_thread_num();
    int count = 0;
    {
      /* (l14) #pragma omp for */
      int i;
      unsigned long niters_ = 0, iter_ = 0, fiter_, liter_ = 0;
      niters_ = (long) (((10) >= (0)) ? ((10) - (0)) : 0);
      _ort_entering_for(0, 0);
      if (_ort_get_static_default_chunk(niters_, &fiter_, &liter_))
        {
          for (iter_ = fiter_, i = (0) + fiter_ * 1; iter_ < liter_; iter_++,
          ↪  i += 1)
            count++;
        }
      CANCEL_for_14 :
        ;
      _ort_leaving_for();
      if (_ort_barrier_me(1) == 1)
        goto CANCEL_parallel_10;
    }
    printf("Thread %d executed %d iterations.\n", thread_id, count);
  }
  CANCEL_parallel_10 :
    _ort_taskwait(2);
  return ((void *) 0);
}
```

Κεφαλαιο 4

Υλοποιηση στον OMPI

4.1 Υποδομή συμβολικής επεξεργασίας

4.2 Σύμπτυξη

Η υλοποίηση σύμπτυξης βασίστηκε στο περιβάλλον του OMPi και περιλαμβάνει δύο μεθόδους. Η μία μέθοδος εφαρμόζεται στη γενική περίπτωση δομής βρόχων βάθους n και η άλλη σε βάθος 2. Επίσης, στο πλαίσιο της κύριας υλοποίησης χρειάστηκε η ανάπτυξη κάποιων πρόσθετων λειτουργιών.

4.1 Υποδομή συμβολικής επεξεργασίας

Ο OMPi υποστήριζε ήδη τη φράση `collapse` για ορθογωνικούς βρόχους. Έτσι, υπήρχε ήδη ο βασικός μηχανισμός αναγνώρισης και εφαρμογής της φράσης. Όμως, στην περίπτωση των μη ορθογωνικών βρόχων έχουμε όρια που είναι συναρτήσεις εξωτερικών μετρητών. Στον OMPi τα όρια των βρόχων αναπαρίστανται ως εκφράσεις μέσω της δομής *αφηρημένου συντακτικού δέντρου*. Για την υλοποίηση της σύμπτυξης χρειάστηκαν λειτουργίες ανάλυσης και χειρισμού των ορίων. Για το λόγο αυτό, υλοποιήθηκε μια υποδομή συμβολικής επεξεργασίας εκφράσεων. Η υλοποίηση των συναρτήσεων έγινε, όπου ήταν εφικτό, με την `ast_expr_traverse`, η οποία παρέχεται από τον OMPi, διαφορετικά με τη μέθοδο της αναδρομής, αφού οι εκφράσεις είναι δυαδικά δέντρα. Παρακάτω θα περιγράψουμε κάποιες από τις λειτουργίες που

υποστηρίζει.

Η βασικότερη λειτουργία της υποδομής είναι το *συντακτικό ταίριασμα προτύπων* (structural pattern matching), η οποία είναι μια τεχνική όπου εξετάζεται αν μια έκφραση ταιριάζει συντακτικά σε ένα προκαθορισμένο πρότυπο. Η λειτουργία αυτή είναι θεμελιώδης μηχανισμός που διευκολύνει την ανάλυση εκφράσεων και μεταξύ άλλων χρησιμοποιείται και για τη δόμηση των υπόλοιπων λειτουργιών της υποδομής. Υλοποιείται με τη συνάρτηση μεταβλητού πλήθους ορισμάτων `expr_match`, η οποία δέχεται ως όρισμα μια έκφραση και μια συμβολοσειρά μορφοποίησης (format string). Στον παρακάτω κώδικα μπορούμε να δούμε ένα παράδειγμα ταιριάσματος της έκφρασης $var\_name + 1$, όπου var_name μεταβλητή συμβολοσειράς.

```
if (expr_match(expr, "+ var:%s num:1", var_name))
```

Στο Σχήμα 4.1 έχουμε ένα άλλο συγκρητικό παράδειγμα, όπου φαίνεται ξεκάθαρα η διευκόλυνση που προσφέρει η `expr_match`.

```
if (expr->opid == BOP_add &&
    ((expr->left->type == UOP && expr->left->opid == UOP_paren &&
      expr->left->left->type == BOP && expr->left->left->opid ==
         BOP_add) ||
     (expr->right->type == UOP && expr->right->opid == UOP_paren
        &&
      expr->right->right->type == BOP && expr->right->right->opid
         == BOP_add)))
```

(α) Χειροκίνητος έλεγχος

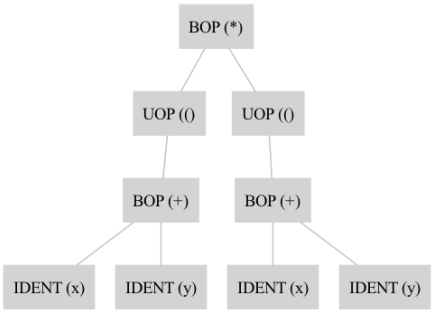
```
if (expr_match(expr, "+ ( bop") || expr_match(expr, "+ _ ( bop"))
```

(β) Έλεγχος με `expr_match`

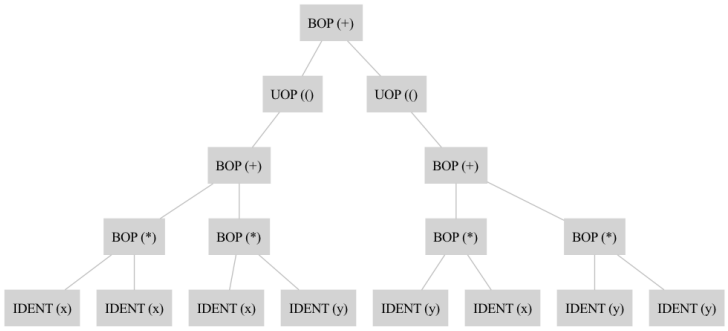
Σχήμα 4.1: Παράδειγμα σύγκρισης ελέγχου αν μια έκφραση ξεκινάει με $(expr1\ bop\ expr2) + expr3$ ή με $expr1\ + (expr2\ bop\ expr3)$, όπου bop αυθαίρετος δυαδικός τελεστής

Έχοντας τώρα δημιουργήσει έναν μηχανισμό που μας επιτρέπει να αναλύουμε εύκολα τη συντακτική δομή των εκφράσεων, υλοποιήθηκαν λειτουργίες για την μαθηματικής φύσεως ανάλυση των ορίων των βρόχων, ώστε να μπορεί να ελεγχθεί αν τα όρια πληρούν τον κανονισμό του OpenMP. Αρχικά, προστέθηκε η λειτουργία που ελέγχει το πλήθος των μετρητών που περιέχονται σε μια έκφραση. Η λειτουργία αυτή προσφέρεται με τη συνάρτηση `expr_has_loopnest_indices`, η οποία δέχεται ως ορίσματα μια έκφραση και έναν πίνακα από εκφράσεις των μετρητών της δομής βρόχων και επιστρέφει το πλήθος των μετρητών που περιέχει η έκφραση. Το πλήθος όμως από μόνο του δεν εγγυάται την πληρότητα του κανονισμού. Έτσι, έχουμε και τη λειτουργία που εξετάζει αν μια μεταβλητή εμφανίζεται γραμμικά σε μια έκφραση και υλοποιείται με τη συνάρτηση `expr_has_linear_var`, που δέχεται μια έκφραση και το όνομα μιας μεταβλητής.

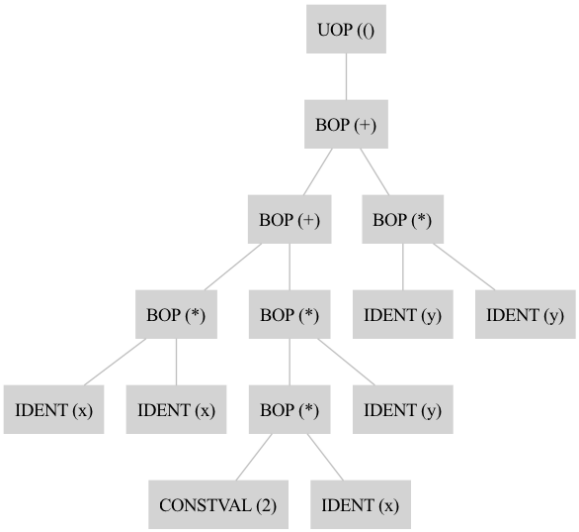
Επίσης, υλοποιήθηκαν λειτουργίες που επιτρέπουν τον χειρισμό των ορίων ως μαθηματικές εκφράσεις. Μία τέτοια είναι η αντικατάσταση μεταβλητών με νέες εκφράσεις. Για τη λειτουργία αυτή χρειαστηκε η δημιουργία ενός `struct expr_binding`, που περιέχει ως στοιχεία τη μεταβλητή που θέλουμε να αντικαταστήσουμε και την νέα έκφραση. Έτσι, έχουμε τη συνάρτηση `expr_substitute`, που παίρνει ορίσματα την έκφραση που περιέχει μεταβλητές και έναν πίνακα με στιγμιότυπα της δομής `expr_binding`, όπου το πλήθος του πίνακα θα είναι το πλήθος των μεταβλητών που θέλουμε να αντικαταστήσουμε, και επιστρέφει τη νέα έκφραση που προκύπτει από τις αντικαταστάσεις. Επιπλέον, μια άλλη λειτουργία χειρισμού που υλοποιήθηκε είναι η μαθηματική απλοποίηση πολυωνυμικών εκφράσεων με τη συνάρτηση `expr_simplify`, η οποία δέχεται μια έκφραση και επιστρέφει τη νέα απλοποιημένη έκφραση. Η απλοποίηση δεν είναι μία καλώς ορισμένη διαδικασία. Η καταλληλότερη απλοποιημένη μορφή μιας έκφρασης εξαρτάται από τη χρήση για την οποία προορίζεται. Στη δική μας περίπτωση, η λειτουργία υποστηρίζει μόνο πολυωνυμικές εκφράσεις, όπου συγχωνεύει τα όμοια μονώνυμα μεταξύ τους. Στο Σχήμα 4.2 απεικονίζονται, με τη μορφή συντακτικού δέντρου, τα διαδοχικά στάδια απλοποίησης της έκφρασης $(x + y)(x + y)$.



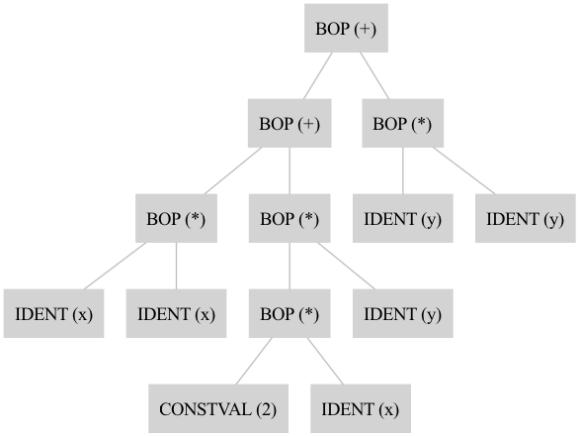
(α) Αρχική μορφή



(β) Εφαρμογή επιμεριστικής ιδιότητας



(γ) Συγχώνευση όμοιων μονωνύμων



(δ) Αφαίρεση εξωτερικής παρένθεσης

Σχήμα 4.2: Παράδειγμα απλοποίησης της έκφρασης $(x + y)(x + y)$ στην έκφραση $x^2 + 2xy + y^2$

4.2 Σύμπτυξη

Η υλοποίηση σύμπτυξης μη ορθογωνικών βρόχων, όπως είδαμε στο Κεφάλαιο 2, είναι ιδιαίτερα πολύπλοκη διαδικασία. Η μέθοδος που παρουσιάζεται στο άρθρο [2] έχει περιορισμούς και απαιτεί τη χρήση εξειδικευμένου λογισμικού. Συνεπώς, δεν αποτελεί καλή επιλογή για έναν μεταφραστή ευρείας χρήσης. Οι GCC και LLVM, οι δύο πιο διαδεδομένοι μεταφραστές, δεν τη χρησιμοποιούν. Για την υλοποίηση σύμπτυξης στον OMPi χρησιμοποιήθηκαν ευρετικές μέθοδοι που χρησιμοποιούν τη σύμπτυξη ορθογωνικών βρόχων και εφαρμόζονται κατά τη μετάφραση. (Η μία μέθοδος αφορά τη σύμπτυξη της γενικής περίπτωσης βάθος n και επιλέχθηκε για την απλότητα της υλοποίησής της. Η άλλη μέθοδος αφορά τη σύμπτυξη βάθους 2, η

οποία επίσης έχει εύκολη υλοποίηση, αλλά βασίζεται σε γεωμετρική προσέγγιση.)

4.2.1 Σύμπτυξη γενικής περίπτωσης

Η μέθοδος για τη γενική περίπτωση βάθους n επιλέχθηκε για την απλότητα της υλοποίησής της. Κάθε χώρος επανάληψης, οποιασδήποτε διάστασης και οποιουδήποτε σχήματος, περικλείεται πάντα από ένα ορθογωνικό πλαίσιο, το οποίο ονομάζεται *ελάχιστο περικλείον πλαίσιο* (minimum bounding box). Μπορούμε λοιπόν να εφαρμόσουμε σύμπτυξη σε αυτό το πλαίσιο και έτσι οι ανακτήσεις θα γίνονται με τον απλό ορθογωνικό τρόπο. Επειδή όμως οι μετρητές που θα ανακτηθούν αναφέρονται σε υπερσύνολο του πραγματικού χώρου, οι τιμές των μετρητών φιλτράρονται και οι εντολές του σώματος εκτελούνται μόνο όταν οι τιμές ανήκουν στον πραγματικό χώρο.

Είναι προφανές ότι η μέθοδος αυτή είναι υπερπροσέγγιση του χώρου και απαιτεί την εκτέλεση περιττών επαναλήψεων. Συνεπώς, εισάγει υπολογιστική επιβάρυνση (overhead). Επίσης, οι περιττές επαναλήψεις πιθανότατα δεν κατανέμονται ομοιόμορφα στα νήματα, οπότε καταλήγουμε και σε ανισορροπία φόρτου εργασίας μεταξύ των νημάτων. Μπορεί όμως να εξάγει παραλληλισμό, αφού όλες οι επαναλήψεις της δομής μεταφέρονται στον βρόχο που παραλληλοποιείται.

Για τη μέθοδο αυτή πρέπει λοιπόν να κατασκευάσουμε ένα ορθογωνικό πλαίσιο που περιέχει όλο το χώρο επανάληψης και είναι με το ελάχιστο δυνατό μέγεθος, ώστε οι περιττές επαναλήψεις να είναι όσο το δυνατόν λιγότερες. Για να το κάνουμε αυτό πρέπει για κάθε βρόχο που περιέχει εξάρτηση να υπολογίσουμε το μέγιστο αριθμό επαναλήψεων που εκτελεί ως προς όλες τις τιμές του εξωτερικού μετρητή. Έτσι, το μέγεθος του πλαισίου θα είναι το γινόμενο του αριθμού επαναλήψεων των σταθερών βρόχων και του μέγιστου αριθμού επαναλήψεων των εξαρτημένων βρόχων.

Θα δούμε τώρα τη διαδικασία πιο συγκεκριμένα. Ας υποθέσουμε ότι έχουμε μια δομή $\mathcal{L}$ βάθους n. Η διαδικασία εφαρμόζεται διαδοχικά από τον εξωτερικό βρόχο L_1 προς τον εσωτερικό L_n και για κάθε βρόχο ξεχωριστά υπολογίζονται:

1. Ο μέγιστος αριθμός επαναλήψεων του βρόχου
2. Η ελάχιστη και μέγιστη τιμή του μετρητή του βρόχου, οι οποίες τιμές χρησιμοποιούνται από πιο εσωτερικούς βρόχους που μπορεί να εξαρτώνται από αυτόν για τον υπολογισμό του μέγιστου αριθμού επαναλήψεων.

Έστω λοιπόν ότι εφαρμόζεται η διαδικασία στον L_k βρόχο. Τα παραπάνω υπολογίζονται διαφορετικά αν ο βρόχος έχει σταθερά όρια ή αν τα όρια περιέχουν εξάρτηση, άρα πρέπει να διακρίνουμε περιπτώσεις.

Αν ο L_k έχει σταθερά όρια, τότε ο μέγιστος αριθμός επαναλήψεων είναι απλά το πλήθος επαναλήψεων, δηλαδή θα είναι

$$N_{L_k}^{(max)} = N_{L_k} = \max\left(0, \left\lfloor \frac{u_k - l_k}{s_k} \right\rfloor + 1\right)$$

και η ελάχιστη και μέγιστη τιμή του i_k θα είναι

$$i_k^{(min)} = \min \mathcal{I}_{L_k} = \begin{cases} l_k, & s_k > 0 \\ u_k + (l_k - u_k) \bmod |s_k|, & s_k < 0 \end{cases} \tag{4.1}$$

$$i_k^{(max)} = \max \mathcal{I}_{L_k} = \begin{cases} u_k - (u_k - l_k) \bmod s_k, & s_k > 0 \\ l_k, & s_k < 0 \end{cases} \tag{4.2}$$

Αν τώρα ο L_k έχει όρια με εξάρτηση από έναν εξωτερικό μετρητή i_m, τότε για να υπολογιστούν τα παραπάνω θα χρειαστούν η μέγιστη και ελάχιστη τιμή του i_m, οι οποίες τιμές έχουν ήδη υπολογιστεί, αφού η διαδικασία εφαρμόζεται διαδοχικά. Επίσης, θα χρησιμοποιήσουμε τη μονοτονία που έχουν τα όρια ως αφινικές συναρτήσεις μιας μεταβλητής. Έτσι, ο μέγιστος αριθμός επαναλήψεων της L_k θα είναι η μέγιστη τιμή της $N_{L_k}(i_m)$ για τις ακραίες τιμές του i_m, δηλαδή θα είναι

$$N_{L_k}^{(max)} = \max_{i_m \in \mathcal{I}_{Lm}} N_{L_k}(i_m) = \max\left(N_{L_k}\left(i_m^{(min)}\right), N_{L_k}\left(i_m^{(max)}\right)\right)$$

και η ελάχιστη και μέγιστη τιμή του i_k θα είναι η μέγιστη και ελάχιστη τιμή των ορίων για τις ακραίες τιμές του i_m, δηλαδή θα είναι

$$\begin{aligned} i_k^{(min)} &= \min_{i_m \in \mathcal{I}_{Lm}} \mathcal{I}_{L_k}(i_m) \\ &= \begin{cases} \min\limits_{i_m \in \mathcal{I}_{L_m}} l_k(i_m), & s_k > 0 \\ \min\limits_{i_m \in \mathcal{I}_{L_m}} u_k(i_m), & s_k < 0 \end{cases} \\ &= \begin{cases} \min\left(l_k\left(i_m^{(min)}\right), l_k\left(i_m^{(max)}\right)\right), & s_k > 0 \\ \min\left(u_k\left(i_m^{(min)}\right), u_k\left(i_m^{(max)}\right)\right), & s_k < 0 \end{cases} \end{aligned} \tag{4.3}$$

$$
\begin{aligned}
i_k^{(max)} &= \max_{i_m \in \mathcal{I}_{L_m}} \mathcal{I}_{L_k}(i_m) \\
&= \begin{cases} \max\limits_{i_m \in \mathcal{I}_{L_m}} u_k(i_m), & s_k > 0 \\ \max\limits_{i_m \in \mathcal{I}_{L_m}} l_k(i_m), & s_k < 0 \end{cases} \\
&= \begin{cases} \max\left(u_k\left(i_m^{(min)}\right), u_k\left(i_m^{(max)}\right)\right), & s_k > 0 \\ \max\left(l_k\left(i_m^{(min)}\right), l_k\left(i_m^{(max)}\right)\right), & s_k < 0 \end{cases}
\end{aligned} \tag{4.4}
$$

Αξίζει να σημειωθεί ότι το όριο u δεν ανήκει σίγουρα στις τιμές του μετρητή. Για το λόγο αυτό στους τύπους (4.1), (4.2), όπου τα όρια είναι σταθερά, μπορούν να χρησιμοποιηθούν τύποι που υπολογίζουν την πραγματική τιμή του μετρητή. Οι τύποι αυτοί όμως μετατρέποντάς τους σε συναρτήσεις δεν είναι αφινικοί και παύουν να είναι μονότονοι. Οπότε δεν μπορούμε να τους χρησιμοποιήσουμε στους (4.3), (4.4).

Εφαρμόζοντας λοιπόν τη διαδικασία σε ολόκληρη τη δομή, θα έχουμε τις τιμές $N_{L_1}^{(max)}, \ldots, N_{L_n}^{(max)}$. Έτσι, παράγεται ο τύπος που υπολογίζει το μέγεθος του περικλείοντος πλαισίου και είναι

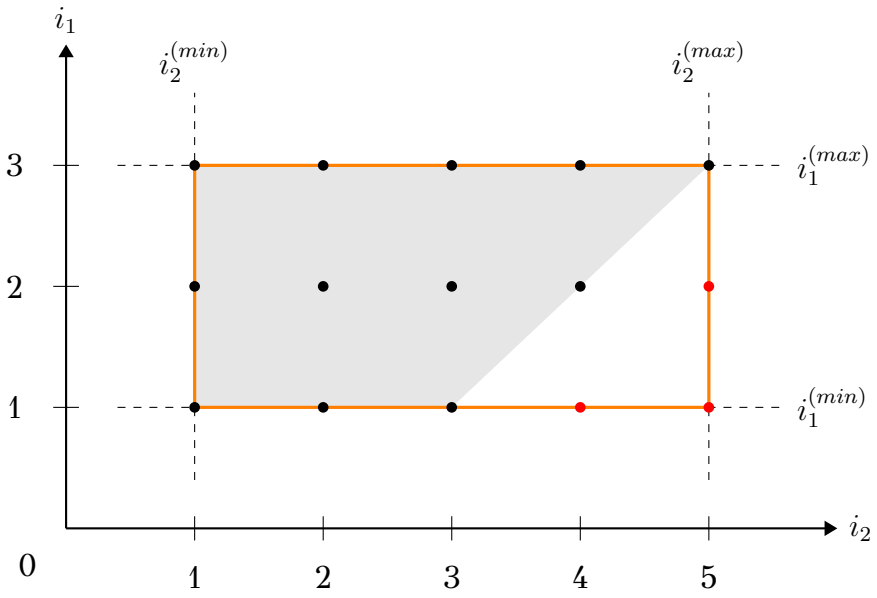
$$
N_{\mathcal{L}}^{(\mathcal{B})} = N_{L_1}^{(max)} \cdot N_{L_2}^{(max)} \cdot \ldots \cdot N_{L_n}^{(max)} = \prod_{k=1}^{n} N_{L_k}^{(max)}
$$

Άρα, δημιουργούμε βρόχο $L' = (iter, 0, N_{\mathcal{L}}^{(\mathcal{B})} - 1, 1)$ και για την ανάκτηση των αρχικών μετρητών χρησιμοποιούμε τις συναρτήσεις (2.15), (2.13) όπως εδώ (2.19). Στη συνέχεια, για να εκτελεστεί το σώμα της $\mathcal{L}$ πρέπει να ελεγχθεί αν οι μετρητές που ανήκουν σε βρόχους με εξάρτηση έχουν τιμή μεταξύ του πραγματικού εύρους του βρόχου. Δηλαδή αν $\mathcal{I}_{dep}$ είναι το σύνολο αυτών των μετρητών, οι εντολές του σώματος θα εκτελεστούν αν ισχύει η συνθήκη

$$
\bigwedge_{i_k \in \mathcal{I}_{dep}} \begin{cases} l(i_{k,outer}) \leq i_k \leq u(i_{k,outer}), & s_k > 0 \\ l(i_{k,outer}) \geq i_k \geq u(i_{k,outer}), & s_k < 0 \end{cases}
$$

Στο Σχήμα 4.3 μπορούμε να δούμε ένα παράδειγμα πλαισίου. Αρχικά, τα διακεκομμένα ευθύγραμμα τμήματα δείχνουν τις ελάχιστες και μέγιστες τιμές των μετρητών. Το πορτοκαλί περίγραμμα σχηματίζει το περικλείον πλαίσιο που δημιουργείται από τη μέθοδο και οι μαύρες και κόκκινες τελείες αναπαριστούν τις πραγματικές και περιττές επαναλήψεις αντίστοιχα.

```
for (i1 = 1; i1 <= 3; i1++)
  for (i2 = 1; i2 <= i1+2; i2++)
```



Σχήμα 4.3: Παράδειγμα πλαισίου

Τέλος, επειδή η μέθοδος αυτή επιτρέπει την ανάκτηση των μετρητών σαν οι βρόχοι να είναι ορθογωνικοί, μπορούμε να χρησιμοποιήσουμε συνδυαστικά ως βελτιστοποίηση τη μέθοδο βάθους 2 που θα δούμε παρακάτω.

4.2.2 Σύμπτυξη βάθους 2

Για την ανάπτυξη της μεθόδου αυτής βασιστήκαμε σε γεωμετρική προσέγγιση. Εφαρμόζει ορθογωνιοποίηση μιας δομής ορθογωνικών βρόχων βάθους 2 χρησιμοποιώντας τη μετατροπή τραπεζίου σε ισοδύναμο ορθογώνιο παραλληλόγραμμο. Έτσι, μπορεί στη συνέχεια να χρησιμοποιηθεί η σύμπτυξη ορθογωνικών βρόχων. Πρόκειται για μια μέθοδο που, από όσο γνωρίζουμε, δεν υπάρχει στην ανοιχτή βιβλιογραφία.

Αρχικά, ο γεωμετρικός χώρος που αντιστοιχεί στο χώρο επανάληψης μιας δομής βάθους 2 είναι το χωρίο που ορίζεται από τα ευθύγραμμα τμήματα που σχηματίζουν τα όρια της δομής. Επομένως, τα πιθανά γεωμετρικά σχήματα που μπορεί να σχηματιστούν είναι τραπέζιο ή παραλληλόγραμμο ή τρίγωνο. Η μέθοδος αυτή υποστηρίζει όλες τις κατηγορίες αυτές, αφού το παραλληλόγραμμο και το τρίγωνο

μπορούν να ειδωθούν ως ειδικές κατηγορίες τραπεζίου. Βέβαια, η περίπτωση του παραλληλόγραμμου είναι ειδική περίπτωση όπου έχουμε σταθερό αριθμό επαναλήψεων και τη διαχειριζόμαστε με καλύτερη μέθοδο, όπως θα δούμε παρακάτω. Ένας χώρος επανάληψης που σχηματίζει τραπέζιο ονομάζεται *τραπεζοειδής* (trapezoidal), ενώ αν σχηματίζει τρίγωνο ονομάζεται *τριγωνικός* (triangular).

Γνωρίζουμε από τη γεωμετρία ότι αν έχουμε ένα τραπέζιο με παράλληλες πλευρές b_1, b_2 και μεταξύ τους απόσταση h, τότε το εμβαδόν του θα είναι

$$\frac{(b_1 + b_2)h}{2}$$

Βλέπουμε ότι το τραπέζιο έχει το ίδιο εμβαδόν με ορθογώνιο πλευρών $\frac{b_1+b_2}{2}, h$ ή $(b_1 + b_2), \frac{h}{2}$. Μπορούμε λοιπόν με μία αντιστοίχηση σημείων να μετατρέψουμε ένα τραπέζιο σε ένα ορθογώνιο και αντίστροφα ένα ορθογώνιο σε ένα τραπέζιο. Επομένως, μπορούμε να διαχειριστούμε μία μη ορθογωνική δομή βάθους 2 ως ορθογωνική. Η διαφορά με τη μέθοδο γενικής περίπτωσης είναι ότι εδώ το πλήθος επαναλήψεων υπολογίζεται ακριβώς και δεν έχουμε περιττές επαναλήψεις.

Θα δούμε τώρα πιο αναλυτικά τη διαδικασία. Έστω δομή $\mathcal{L} = (L_1, L_2)$, όπου

$$L_1 = (i_1, l_1, u_1, s_1)$$

$$L_2 = (i_2, l_2(i_1), u_2(i_1), s_2) = (i_2, a_l i_1 + b_l, a_u i_1 + b_u, s_2)$$

Τα b_1, b_2, όπως αναφέραμε, πρέπει να είναι τα παράλληλα ευθύγραμμα τμήματα, οπότε προφανώς θα είναι τα ευθύγραμμα τμήματα που περιέχουν τις επαναλήψεις του εσωτερικού βρόχου που δημιουργούνται από την πρώτη και τελευταία τιμή του i_1. Επίσης, προφανώς η απόσταση μεταξύ των b_1, b_2 είναι το πλήθος επαναλήψεων του εξωτερικού βρόχου. Άρα θέτουμε

$$b_1 = N_{L_2}(l_1) = \bar{N}_{L_2}(0)$$

$$b_2 = N_{L_2}(l_1 + s_1(N_{L_1} - 1)) = \bar{N}_{L_2}(N_{L_1} - 1)$$

$$h = N_{L_1}$$

Το OpenMP απαιτεί να ισχύει $(a_u - a_l)s_1 \bmod s_2 = 0$, οπότε χρησιμοποιώντας τον

τύπο (2.7) και θέτοντας $A = \frac{(a_u - a_l)s_1}{s_2}$, $B = \left\lfloor \frac{(a_u - a_l)l_1 + (b_u - b_l)}{s_2} \right\rfloor$, θα έχουμε

$$b_1 = \left\lfloor \frac{(a_u - a_l)l_1 + (b_u - b_l)}{s_2} \right\rfloor + 1 = \lfloor B \rfloor + 1 \tag{4.5}$$

$$b_2 = \frac{(a_u - a_l)s_1}{s_2}(N_{L_1} - 1) + \left\lfloor \frac{(a_u - a_l)l_1 + (b_u - b_l)}{s_2} \right\rfloor + 1 = A(N_{L_1} - 1) + \lfloor B \rfloor + 1 \tag{4.6}$$

$$h = N_{L_1} \tag{4.7}$$

Θέτουμε λοιπόν τον νοητό ορθογωνικό χώρο

$$\mathcal{I}_{\mathcal{L}}^{(rect)} = \left(\left(\left[0, \left\lfloor \frac{h}{2} \right\rfloor \right) \times [0, b_1 + b_2) \right) \cup \left(\left\{ \left\lfloor \frac{h}{2} \right\rfloor + h \bmod 2 - 1 \right\} \times \left[0, \frac{b_1 + b_2}{2} \right) \right) \right) \cap \mathbb{Z}^2$$

Το πλήθος επαναλήψεων του χώρου αυτού θα είναι

$$\begin{aligned} N_{\mathcal{L}}^{(rect)} &= \begin{cases} \dfrac{(b_1 + b_2)h}{2}, & h \text{ άρτιος} \\ \dfrac{(b_1 + b_2)(h-1)}{2} + \dfrac{b_1 + b_2}{2}, & h \text{ περιττός} \end{cases} \\ &= \frac{(b_1 + b_2)h}{2} \end{aligned} \tag{4.8}$$

Αρχικά, πρέπει να δούμε ότι το πλήθος επαναλήψεων του $\mathcal{I}_{\mathcal{L}}$ παράγει το πλήθος του $N_{\mathcal{L}}^{(rect)}$. Πράγματι, χρησιμοποιώντας τους τύπους (2.9), (4.5), (4.6), (4.7) και (4.8), θα έχουμε

$$\begin{aligned} N_{\mathcal{L}} &= \frac{(a_u - a_l)s_1}{s_2} \frac{(N_{L_1} - 1)N_{L_1}}{2} + \left(\left\lfloor \frac{(a_u - a_l)l_1 + (b_u - b_l)}{s_2} \right\rfloor + 1 \right) N_{L_1} \\ &= A\frac{(N_{L_1} - 1)N_{L_1}}{2} + (\lfloor B \rfloor + 1) N_{L_1} \\ &= (A(N_{L_1} - 1) + 2\lfloor B \rfloor + 2) \frac{N_{L_1}}{2} \\ &= (\lfloor B \rfloor + 1 + A(N_{L_1} - 1) + \lfloor B \rfloor + 1) \frac{N_{L_1}}{2} \\ &= \left(\bar{N}_{L_2}(0) + \bar{N}_{L_2}(N_{L_1} - 1) \right) \frac{N_{L_1}}{2} \\ &= \frac{\left(\bar{N}_{L_2}(0) + \bar{N}_{L_2}(N_{L_1} - 1) \right) N_{L_1}}{2} \\ &= \frac{(b_1 + b_2)h}{2} \\ &= N_{\mathcal{L}}^{(rect)} \end{aligned} \tag{4.9}$$

Επίσης, μπορούμε να διακρίνουμε στη σχέση (4.9) ότι το γινόμενο του αριθμητή του τύπου του εμβαδού παράγει πάντα άρτιο αριθμό, οπότε προγραμματιστικά το μόνο που χρειάζεται να προσέξουμε για να μην έχουμε πρόβλημα με ακέραιες διαιρέσεις είναι να γίνει πρώτα η πράξη του γινομένου.

Πρέπει να δούμε τώρα γιατί η μέθοδος λειτουργεί. Η μετατροπή ενός τραπεζίου σε ορθογώνιο επιτυγχάνεται με αναδιάταξη των σημείων του τραπεζίου. Αρχικά, το τραπέζιο διαχωρίζεται σε δύο τμήματα με μια νοητή οριζόντια τομή στο μέσο του ύψους του. Στη συνέχεια, εφαρμόζεται στο κάτω τμήμα κατοπτρισμός ως προς την ευθεία της τομής και οριζόντια μετατόπιση. Έτσι, το πάνω τμήμα του τραπεζίου συμπληρώνεται κατάλληλα με το κάτω μέρος και μαζί αποτελούν μία σειρά του ορθογωνίου. Για να δούμε ότι αυτό που παράγεται είναι ορθογώνιο πρέπει να δούμε ότι σε κάθε ύψος έχει σταθερό μήκος. Μεταφέροντας αυτή τη λογική στο χώρο επανάληψης, πρέπει να δείξουμε ότι το άθροισμα των επαναλήψεων δύο συμπληρωματικών γραμμών είναι πάντα σταθερό. Κάθε σειρά k του τραπεζοειδή χώρου συμπληρώνεται με τη σειρά $N_{L_1}-1-k$, οπότε χρησιμοποιώντας πάλι τον τύπο (2.7) και παρατηρώντας τον τύπο του εμβαδού, βλέπουμε ότι για $k \in \{0, \dots, \lfloor \frac{h}{2} \rfloor - 1\}$ το άθροισμα των επαναλήψεων των σειρών αυτών μας δίνει τις επαναλήψεις του ορθογωνικού χώρου. Πράγματι, έχουμε

$$\begin{aligned}
\bar{N}_{L_2}(k) + \bar{N}_{L_2}(N_{L_1}-1-k) &= \frac{(a_u-a_l)s_1}{s_2}k + \left\lfloor \frac{(a_u-a_l)l_1+(b_u-b_l)}{s_2} \right\rfloor + 1 \\
&\quad + \frac{(a_u-a_l)s_1}{s_2}(N_{L_1}-1-k) + \left\lfloor \frac{(a_u-a_l)l_1+(b_u-b_l)}{s_2} \right\rfloor + 1 \\
&= Ak + \lfloor B \rfloor + 1 + A(N_{L_1}-1-k) + \lfloor B \rfloor + 1 \\
&= A(N_{L_1}-1) + 2\lfloor B \rfloor + 2 \\
&= b_1 + b_2 \\
&= N_{L_2^{(rect)}}
\end{aligned}$$

Μπορούμε λοιπόν να εφαρμόσουμε σύμπτυξη της $\mathcal{L}$ δημιουργώντας νέο βρόχο $L = (iter, 0, N_{\mathcal{L}}-1, 1)$. Πρέπει όμως τώρα να δούμε πώς θα γίνει η ανάκτηση. Ακολουθώντας την αντίστροφη διαδικασία που ακολουθήθηκε παραπάνω, πρέπει από τον L' να ανακτήσουμε το νοητό ορθογώνιο και να το μετασχηματίσουμε στον αρχικό χώρο. Επειδή η ανάκτηση μας δίνει κανονικοποιημένο χώρο, ο μετασχηματισμός του ορθογωνίου στον αρχικό χώρο περιλαμβάνει στην ουσία δύο διαδικασίες, το

μετασχηματισμό του ορθογωνίου σε κανονικοποιημένο τραπέζιο και την αποκανονικοποίηση του τραπεζίου αυτού.

Αρχικά, η ανάκτηση του ορθογωνίου γίνεται με τη συνάρτηση (2.15) και έχουμε

$$\begin{pmatrix} i_{rect} \\ j_{rect} \end{pmatrix} = T^{-1}_{col,\mathcal{L}^{(rect)}}(iter) = \begin{pmatrix} \left\lfloor \dfrac{iter}{b1+b2} \right\rfloor \\ iter \bmod (b1+b2) \end{pmatrix} \tag{4.10}$$

Στη συνέχεια, ο μετασχηματισμός του χώρου αυτού σε κανονικοποιημένη μορφή του αρχικού χώρου γίνεται με τη συνάρτηση $\mathcal{G}_{\mathcal{L}} : \mathcal{I}_{\mathcal{L}^{(rect)}} \to N_{\mathcal{L}'}$

$$\begin{pmatrix} i'_1 \\ i'_2 \end{pmatrix} = \mathcal{G}_{\mathcal{L}}(i_{rect}, j_{rect}) = \begin{cases} (i_{rect}, j_{rect}), & j_{rect} < \bar{N}_{L_2}(i_{rect}) \\ (N_{L_1} - 1 - i_{rect}, j_{rect} - \bar{N}_{L_2}(i_{rect})), & j_{rect} \geq \bar{N}_{L_2}(i_{rect}) \end{cases} \tag{4.11}$$

Η συνάρτηση αυτή ακολουθεί την αντίστροφη διαδικασία της μετατροπής τραπεζίου σε ορθογώνιο. Κάθε σειρά k του ορθογωνίου περιέχει τις σειρές k και $N_{L_1} - 1 - k$ του τραπεζίου. Έτσι, αν η ανακτημένη τιμή j_{rect} είναι μικρότερη από το πλήθος επαναλήψεων που έχει το τραπέζιο στη σειρά i_{rect}, τότε σημαίνει ότι οι i_{rect}, j_{rect} έχουν τις πραγματικές τιμές του κανονικοποιημένου τραπεζίου και αφήνονται ως έχουν. Αν όμως το j_{rect} είναι μεγαλύτερο από το πλήθος επαναλήψεων της σειράς, τότε η συνάρτηση αντιστοιχίζει τις τιμές i_{rect}, j_{rect} στις πραγματικές κανονικοποιημένες τιμές τους. Τέλος, η αποκανονικοποίηση γίνεται με τον τύπο (2.13)

$$\begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = T^{-1}_{norm,\mathcal{L}}(i'_1, i'_2) = \begin{pmatrix} l_1 + s_1 \cdot i'_1 \\ l_2(l_1 + s_1 \cdot i'_1) + s_2 \cdot i'_2 \end{pmatrix} \tag{4.12}$$

Έτσι, έχουμε τις πραγματικές αρχικές τιμές και η ανάκτηση ολοκληρώνεται.

Στο Σχήμα 4.4 μπορούμε να δούμε ένα παράδειγμα ανάκτησης με τη μέθοδο που περιγράψαμε. Αρχικά, όταν χρησιμοποιήσουμε τους τύπους (2.13) θα πάρουμε τις τιμές του ορθογωνικού χώρου που απεικονίζεται στο Σχήμα 4.4α. Στη συνέχεια, στο Σχήμα 4.4β μπορούμε να δούμε την αντιστοίχιση των σημείων στην πραγματική τους τιμή. Τα σημεία που απεικονίζονται με πορτοκαλί κύκλους θα αντιστοιχιθούν στα σημεία με πορτοκαλί τελείες βάσει της συνάρτησης (4.11). Τέλος, στο Σχήμα 4.4γ έχουμε τον αρχικό χώρο σε κανονικοποιημένη μορφή.

Η μέθοδος αυτή έχει πλεονέκτημα και μειονέκτημα. Το πλεονέκτημα είναι ότι, λόγω του ότι χρησιμοποιεί την ανάκτηση ορθογωνικών βρόχων, η οποία έχει σχετικά μικρό overhead, επιτρέπει την ανάκτηση των αρχικών τιμών σε κάθε επανάληψη.

Επομένως, είναι πρακτικά εφικτή η δρομολόγηση των επαναλήψεων μέσω της φράσης `schedule`, το οποίο είναι επιθυμητό και πιθανώς επιτραπεί από μία μελλοντική έκδοση του OpenMP. Οι υλοποιήσεις άλλων μεταφραστών κάνουν ανάκτηση των τιμών μία φορά ανά νήμα για να ελαχιστοποιήσουν το overhead. Από την άλλη το μειονέκτημα είναι ότι η σειρά εκτέλεσης του αρχικού χώρου αλλάζει έμμεσα με την αντιστοίχιση τιμών που εφαρμόζεται. Αυτό δεν αποτελεί πρόβλημα, διότι το OpenMP δεν επιτρέπει τη φράση `ordered` αν η δομή βρόχων είναι μη ορθογωνική. Όμως, μπορεί να έχει αρνητική επίπτωση στην προσπέλαση μνήμης, δηλαδή να έχουμε περισσότερες *αστοχίες cache* (cache misses).

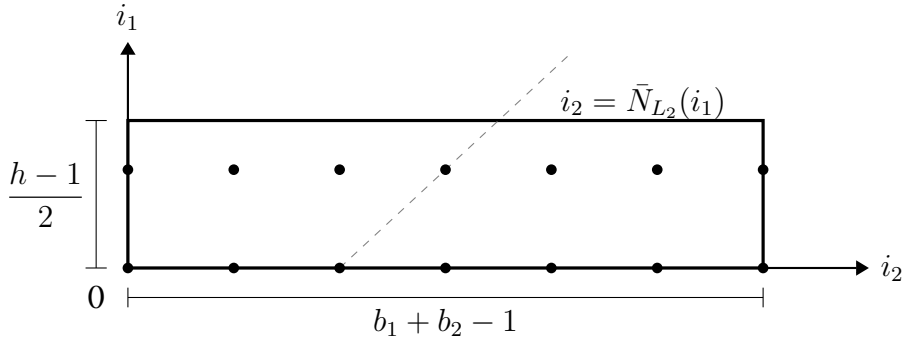
4.2.3 Σύμπτυξη βρόχων με πλασματικές εξαρτήσεις

Όταν ένας βρόχος περιέχει όρια με εξάρτηση, μπορούμε να παρατηρήσουμε ότι δημιουργούνται περιπτώσεις όπου το πλήθος επαναλήψεων είναι σταθερό. Αυτό συμβαίνει όταν για τους συντελεστές μετρητών ισχύει $a_u - a_l = 0$. Έτσι, κατά τη μετάφραση δημιουργείται η έκφραση της διαφοράς των ορίων, η οποία καθορίζεται από την κατεύθυνση του βήματος, χρησιμοποιείται η `expr_simplify` για να απλοποιηθεί η έκφραση αυτή και το αποτέλεσμα ελέγχεται για το αν περιέχει ακόμα εξωτερικό μετρητή. Αν δεν τον περιέχει, τότε εφαρμόζεται η σύμπτυξη ορθογωνικών, που θεωρητικά έχει μικρότερο overhead.

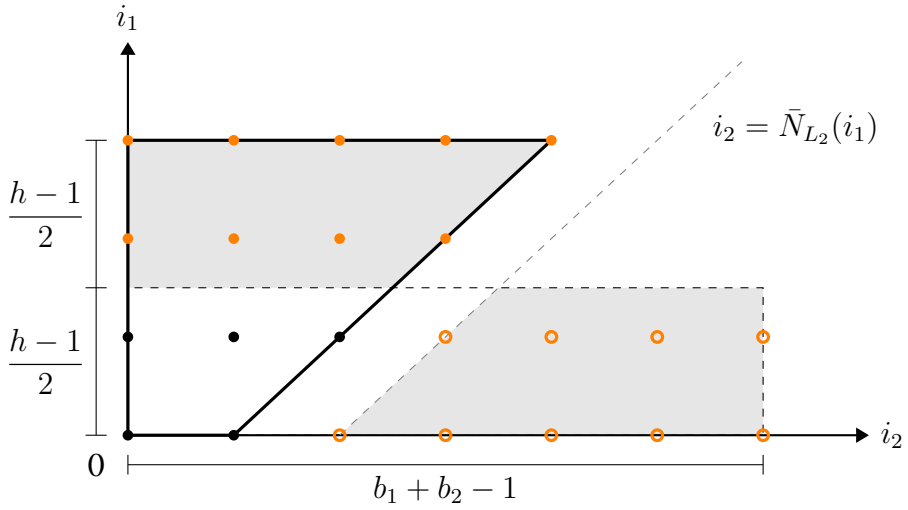
4.2.4 Υλοποιήσεις άλλων μεταφραστών

Εδώ θα κάνουμε μία περιγραφή των υλοποιήσεων σύμπτυξης που παρέχουν οι GCC και LLVM, οι δύο πιο ευρέως διαδεδομένοι μεταφραστές. Όπως είναι λογικό λόγω της φύσης του προβλήματος σύμπτυξης, κι εδώ οι υλοποιήσεις χωρίζονται στις δύο κύριες κατηγορίες γενικής περίπτωσης και δομής βάθους 2.

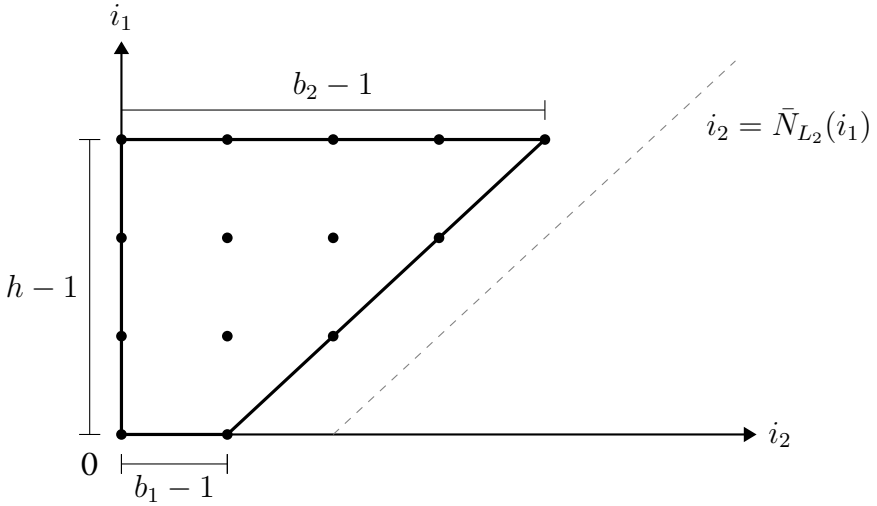
Ο GCC για τη γενική περίπτωση χρησιμοποιεί μία μέθοδο χρόνου εκτέλεσης. Πριν την εκτέλεση της δομής βρόχων κάνει καταμέτρηση των επαναλήψεων όλης της δομής εκτελώντας στην ουσία μία φορά όλη τη δομή χωρίς τις εντολές του σώματος. Ύστερα, κατανέμει το πλήθος επαναλήψεων ομοιόμορφα στα νήματα. Έτσι, κάθε νήμα θα λάβει μία αρχική και τελική τιμή $first\_iter\_thread$ και $last\_iter\_thread$ που δηλώνουν το μέρος του ενιαίου βρόχου που θα εκτελέσει. Στη συνέχεια, κάθε νήμα στην πρώτη εκτέλεση μόνο θα ανακτήσει τις αρχικές τιμές από την τιμή του



(α) Ανάκτηση ορθογωνικού χώρου



(β) Μετασχηματισμός χώρου



(γ) Αρχικός κανονικοποιημένος χώρος

Σχήμα 4.4: Παράδειγμα ανάκτησης με τη μέθοδο βάθους 2

$first\_iter\_thread$ και για τις υπόλοιπες επαναλήψεις απλώς αυξάνει ή μειώνει κατάλληλα τις αρχικές τιμές με κατάλληλους ελέγχους.

Για την περίπτωση δομής βάθους 2 ο GCC χρησιμοποιεί τη μέθοδο επίλυσης πολυωνύμου που περιγράψαμε στην Ενότητα 2.4. Αρχικά, υπολογίζει το πλήθος επαναλήψεων με έναν αναλυτικό τύπο παρόμοιο με τον (2.17) και κατανέμει το πλήθος αυτό ομοιόμορφα στα νήματα. Στη συνέχεια, παρόμοια με τη γενική περίπτωση, κάθε νήμα μόνο στην πρώτη εκτέλεση θα ανακτήσει τις αρχικές τιμές με τους τύπους που είχαμε δει.

Ο LLVM τώρα για τη γενική περίπτωση χρησιμοποιεί τη μέθοδο του ελάχιστου πλαισίου, τη μέθοδο που χρησιμοποιήσαμε κι εμείς. Επίσης, αξίζει να σημειωθεί ότι, αν και όπως είπαμε το OpenMP έχει τον περιορισμό της μη χρήσης του `schedule` στους μη ορθογωνικούς βρόχους, ο LLVM επιτρέπει τη χρήση της φράσης.

Για δομές βάθους 2 ο LLVM χρησιμοποιεί δύο μεθόδους ανάλογα με τον τύπο της δομής. Για τη γενική περίπτωση χρησιμοποιεί πάλι το ελάχιστο πλαίσιο. Αν η δομή είναι τριγωνική, τότε μόνο χρησιμοποιεί τη μέθοδο επίλυσης πολυωνύμου. Επίσης, στη συγκεκριμένη περίπτωση, και ο LLVM κατανέμει ομοιόμορφα τις επαναλήψεις και η ανάκτηση των αρχικών τιμών γίνεται μόνο μία φορά ανά νήμα.

Κεφαλαιο 5

Αξιολογηση

5.1 Έλεγχος ορθότητας

5.2 Μετρήσεις

5.1 Έλεγχος ορθότητας

Για να ελεγχθεί η ορθότητα της υλοποίησης χρησιμοποιήθηκε η σουίτα των τεστ για το `collapse` που προσφέρει το OpenMP. Βέβαια, η σουίτα αυτή περιλαμβάνει πολύ απλές διαδικασίες και δεν είναι κατάλληλη για μετρήσεις. Επίσης, περιλαμβάνει αρκετά απλές δομές μη ορθογωνικών βρόχων, και έτσι χρειάστηκε η ανάπτυξη πιο ειδικών εργαλείων για την ανάλυση των μετρητών που παρήγαγε η υλοποίηση κατά την ανάπτυξη.

5.2 Μετρήσεις

Οι πειραματικές μετρήσεις γίνονται κυρίως χρησιμοποιώντας *μετροπρογράμματα* (benchmarks), τα οποία περιέχουν υπολογισμούς που ανταποκρίνονται σε πραγματικές εφαρμογές και συνήθως είναι σχεδιασμένες έτσι ώστε να στοχεύουν στην αξιολόγηση συγκεκριμένων διαδικασιών. Στην περίπτωσή μας δεν βρέθηκε κάποιο μετροπρόγραμμα που να είναι σχεδιασμένο για σύμπτυξη βρόχων. Το PolyBench [10] είναι ένα μετροπρόγραμμα που περιλαμβάνει διαδικασίες που χρησιμοποιούν βρό-

χους, αλλά οι δομές δεν είναι τέλεια εμφωλευμένες, συνεπώς δεν γίνεται να χρησιμοποιηθεί η φράση `collapse`. Για να μετρήσουμε λοιπόν τις δύο κύριες μεθόδους που υλοποιήθηκαν, αναπτύχθηκαν δύο προγράμματα με απλές δομές βρόχων βάθους 2 και 3. Οι δομές αυτές εκτελούν απλές αριθμητικές πράξεις, όπου το πλήθος των πράξεων αυτών ελέγχεται με τη μεταβλητή *workiters*. Με τον τρόπο αυτό μπορούμε να ελέγχξουμε τον φόρτο εργασίας του σώματος της δομής.

Οι εκτελέσεις πραγματοποιήθηκαν στον υπολογιστή *parade*, που ανήκει στην Ομάδα Παράλληλης Επεξεργασίας. Το πειραματικό περιβάλλον έχει τα εξής χαρακτηριστικά:

- Επεξεργαστής: 4 × Intel Xeon Gold 6130
- Κύρια μνήμη: 256 GiB
- Λειτουργικό Σύστημα: AlmaLinux 9.4
- Μεταφραστές: GCC 11.4.1, Clang 17.0.6

Χρησιμοποιήθηκαν οι μεταβλητές περιβάλλοντος του OpenMP `OMP_PROC_BIND = true` και `OMP_PLACES = cores`. Επίσης, κατά τη μετάφραση χρησιμοποιήθηκε το όρισμα `-O2`. Τέλος, ο OMPi χρησιμοποιεί ως τελικό μεταφραστή τον GCC.

Οι μετρήσεις χωρίζονται κυρίως σε δύο κατηγορίες, μία για τη μέθοδο γενικής περίπτωσης και μία για βάθος 2. Για κάθε κατηγορία και για καθέναν από τους μεταφραστές GCC, LLVM και OMPi μετρήθηκε ο χρόνος εκτέλεσης του προγράμματος για πλήθος επεξεργαστών $p \in \{1, 2, 4, 8, 16, 32, 64\}$, όπου για $p > 1$ η παραλληλοποίηση έγινε με το OpenMP, με και χωρίς τη φράση `collapse`. Κάθε πείραμα επαναλήφθηκε 10 φορές και ως τελική τιμή καταγράφηκε ο μέσος όρος. Θα χρησιμοποιήσουμε δύο μετρικές, το speedup και το gain. Το speedup εκφράζει την επιτάγχυνση που επιτυγχάνεται μέσω της παραλληλοποίησης και δίνεται από τον τύπο

$$Speedup(p) = \frac{T_{serial}}{T_{parallel}(p)}$$

Το gain δείχνει την ποσοστιαία βελτίωση του χρόνου εκτέλεσης χωρίς και με τη χρήση `collapse` και υπολογίζεται ως

$$Gain(p) = \frac{T_{nocol}(p) - T_{col}(p)}{T_{nocol}(p)} \times 100\%$$

Έτσι, παρέχονται γραφήματα και πίνακες με τις τιμές χρόνου εκτέλεσης, speedup και gain.

5.2.1 Μετρήσεις μεθόδου βάθους 2

Για την περίπτωση βάθους 2 πειραματιστήκαμε με δύο διαφορετικές δομές. Οι δομές επιλέχθηκαν έτσι ώστε ο εξωτερικός βρόχος να έχει πλήθος επαναλήψεων μικρότερο του αριθμού νημάτων στη μία περίπτωση και αρκετά μεγαλύτερο στην άλλη περίπτωση, με σκοπό τον προσδιορισμό της χρησιμότητας της φράσης `collapse`. Επίσης, χρησιμοποιήσαμε τη μεταβλητή *workiters*, ώστε να προσαρμόσουμε το φόρτο εργασίας. Και οι δύο δομές που επιλέχθηκαν ενεργοποιούν τις ίδιες μεθόδους σύμπτυξης σε κάθε μεταφραστή. Ο OMPi θα χρησιμοποιήσει τη μέθοδο γεωμετρικού μετασχηματισμού, ο GCC τη μέθοδο επίλυσης πολυωνύμου δευτέρου βαθμού και ο LLVM τη μέθοδο περικλείοντος πλαισίου.

Θα δούμε αρχικά τη δομή $\mathcal{L}_1$ με βρόχους

$$L_1 = (i, 0, N-1, 1)$$

$$L_2 = (j, 0, 10i + N - 1, 1)$$

η οποία εκτελέστηκε με παραμέτρους $N = 8, workiters = 300000000$. Βλέπουμε ότι η δομή αυτή έχει εξωτερικό βρόχο με 8 επαναλήψεις, συνεπώς αν παραλληλοποιηθεί χωρίς `collapse`, μπορούμε να εκμεταλευτούμε το πολύ 8 νήματα. Αν επιπλέον συνυπολογίσουμε την ανισοκατανομή του φόρτου εργασίας που προκαλεί το μη σταθερό όριο, αναμένουμε η σύμπτυξη να έχει αρκετά θετική επίδραση στην εκτέλεση. Στους Πίνακες 5.1, 5.2 και 5.3 μπορούμε να δούμε αναλυτικά τους χρόνους εκτέλεσης και τις τιμές των μετρικών που παράγονται.

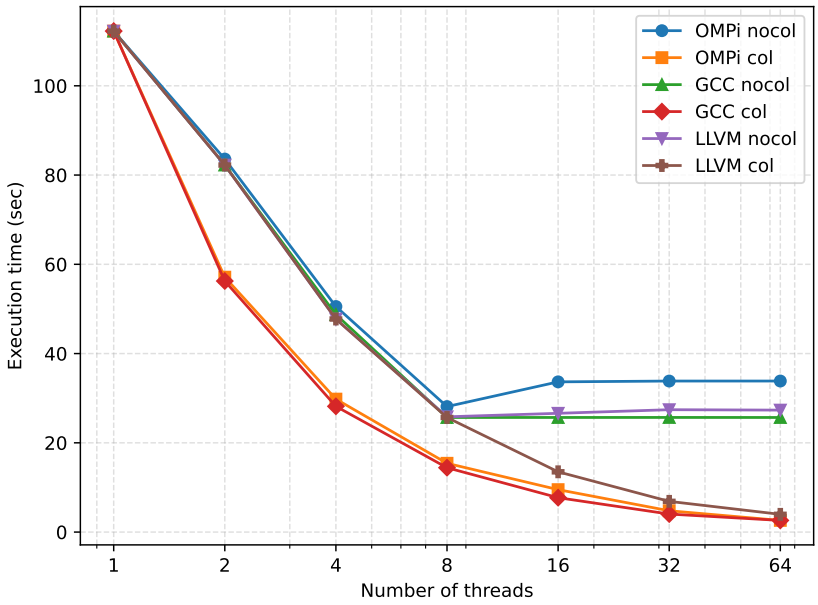
Στα Σχήματα 5.1 και 5.2 έχουμε τα γραφήματα των χρόνων εκτέλεσης και speedup ως προς τον αριθμό νημάτων. Εύκολα μπορούμε να διαπιστώσουμε, ειδικά στο γράφημα του speedup, ότι στις εκτελέσεις χωρίς `collapse` πολλά νήματα μένουν ανεκμετάλλευτα. Εδώ φαίνεται η χρησιμότητα του `collapse`, αφού μπορούμε να παρατηρήσουμε ότι οι εκτελέσεις που εφαρμόζουν σύμπτυξη έχουν καλύτερη αξιοποίηση των νημάτων και καλύτερη επιτάχυνση λόγω παραλληλοποίησης.

Στη συνέχεια, θα δούμε τη δομή $\mathcal{L}_2$ με βρόχους

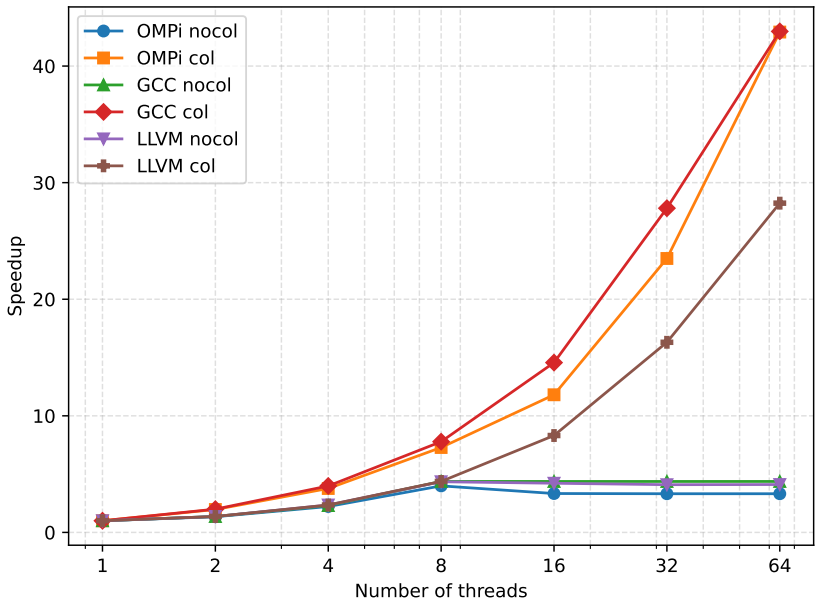
$$L_1 = (i, 0, N-1, 1)$$

$$L_2 = (j, 0, i + N - 1, 1)$$

η οποία εκτελέστηκε με παραμέτρους $N = 1000, workiters = 80000$. Εδώ βλέπουμε ότι η δομή αυτή έχει εξωτερικό βρόχο με αρκετές επαναλήψεις, συνεπώς η παραλ-



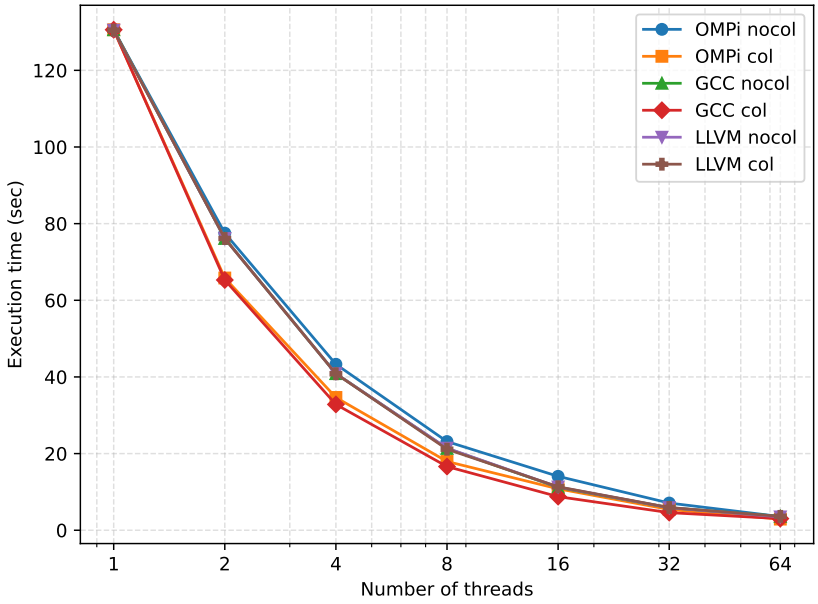
Σχήμα 5.1: Χρόνοι εκτέλεσης για δομή $\mathcal{L}_1$ $(N = 8, workiters = 300000000)$ (sec)



Σχήμα 5.2: Speedup για δομή $\mathcal{L}_1$ $(N = 8, workiters = 300000000)$

ληλοποίηση χωρίς `collapse` θα αξιοποιήσει όλα τα νήματα. Σε αυτές τις περιπτώσεις υπάρχει μόνο το πρόβλημα της ανισοκατανομής του φόρτου εργασίας στα νήματα λόγω του ορίου και αυτό επιδιώκει να επιλύσει η σύμπτυξη. Στους Πίνακες 5.4, 5.5 και 5.6 μπορούμε να δούμε αναλυτικά τις μετρήσεις.

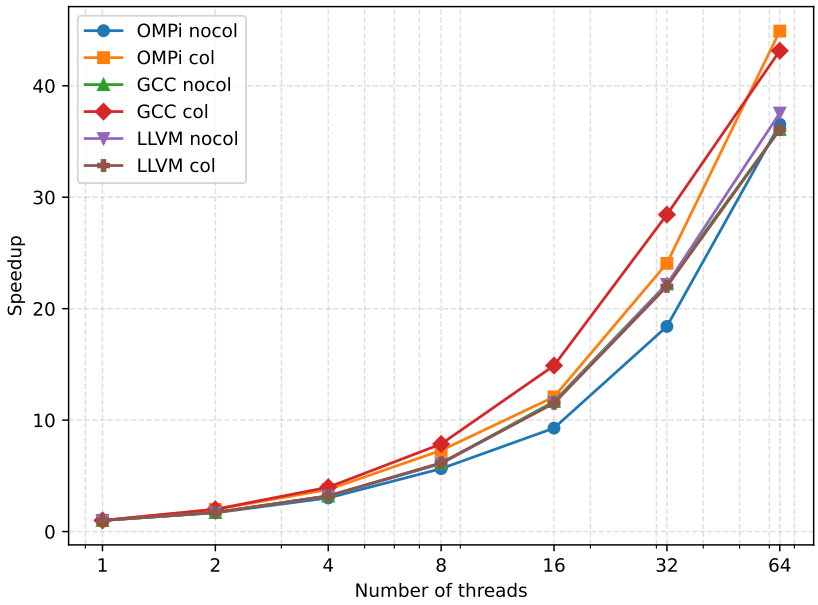
Πράγματι, από τα Σχήματα 5.3 και 5.4 μπορούμε να παρατηρήσουμε ότι η παραλληλοποίηση χωρίς `collapse` εκμεταλεύεται όλα τα νήματα και έχει καλούς χρόνους. Όμως, πάλι το οι εκτελέσεις με `collapse` παραλληλοποιούν καλύτερα τη δομή. Επίσης, η διαφορά μπορεί να μεταβάλλεται ανάλογα με τη μορφή των ορίων.



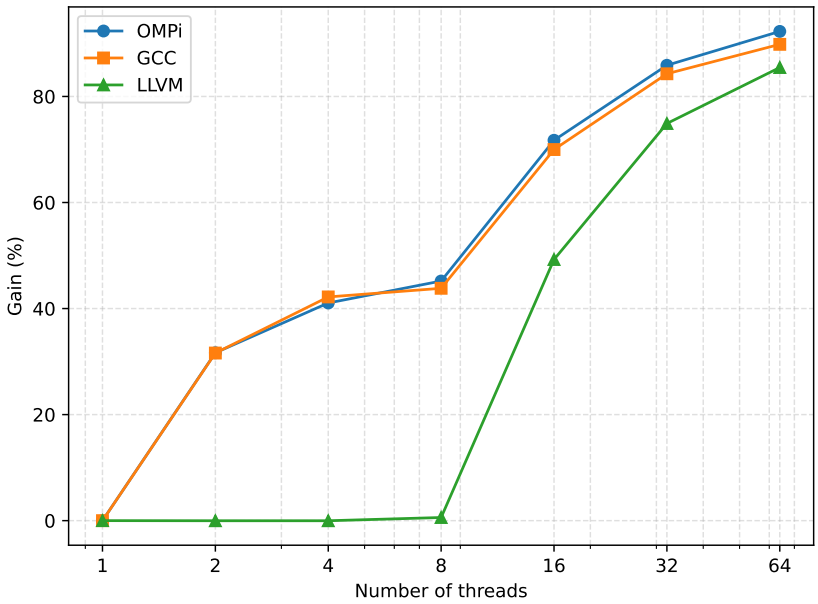
Σχήμα 5.3: Χρόνοι εκτέλεσης για δομή $\mathcal{L}_2$ ($N = 1000, workiters = 80000$) (sec)

Συνολικά για το speedup, μπορούμε να δούμε στα Σχήματα 5.2 και 5.4 ότι ο GCC έχει τις καλύτερες τιμές. Θα μπορούσαμε να πούμε ότι ακολουθεί ο OMPi, ο οποίος ενώ με τη `nocollapse` εκτέλεσή του εμφανίζει τις χαμηλότερες τιμές για κάποιους αριθμούς νημάτων, με την `collapse` εκτέλεσή του εμφανίζει τιμές κοντά σε αυτές του GCC. Μάλιστα, στην $\mathcal{L}_2$ για $p = 64$ εμφανίζει υψηλότερη τιμή από του GCC. Τέλος, ο LLVM με τη μέθοδο πλαισίου μόνο στην $\mathcal{L}_1$ καταφέρνει να εμφανίσει καλύτερη εκτέλεση με `collapse`.

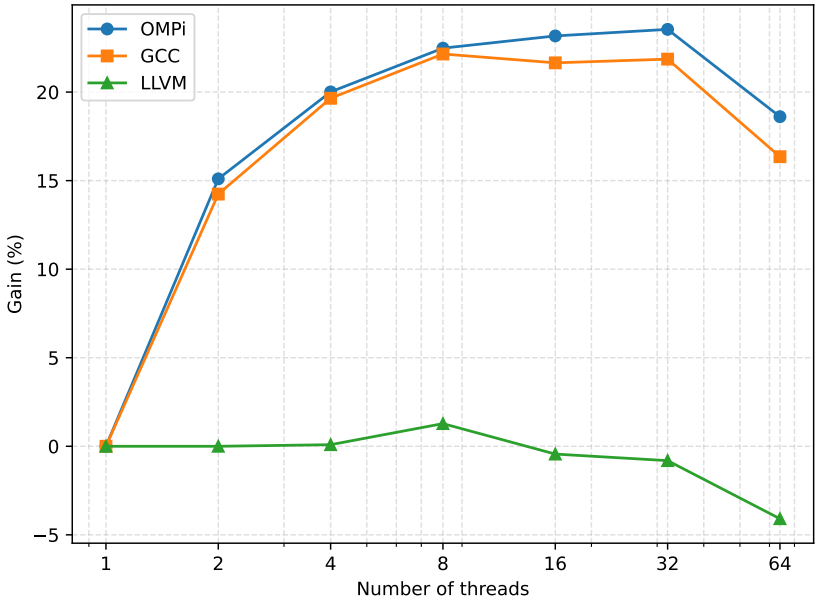
Όσον αφορά το gain, βλέπουμε στα Σχήματα 5.5 και 5.6 ότι ο OMPi εμφανίζει τις καλύτερες τιμές, ειδικά στην $\mathcal{L}_2$ που ήταν και η δύσκολη περίπτωση. Ακολουθεί με μικρή διαφορά ο GCC. Για τον LLVM εδώ είναι πιο εμφανές ότι μόνο στην $\mathcal{L}_1$ εμφανίζει καλύτερες εκτελέσεις με το `collapse`.



Σχήμα 5.4: Speedup για δομή $\mathcal{L}_2$ $(N = 1000, workiters = 80000)$



Σχήμα 5.5: Gain για δομή $\mathcal{L}_1$ $(N = 8, workiters = 300000000)$



Σχήμα 5.6: Gain για δομή $\mathcal{L}_2$ ($N = 1000, workiters = 80000$)

5.2.2 Μετρήσεις μεθόδου γενικής περίπτωσης

Για την περίπτωση αυτή εκτελέσαμε τη δομή $\mathcal{L}_3$ με βρόχους

$$L_1 = (i, 0, N-1, 1)$$

$$L_2 = (j, 0, i+N-1, 1)$$

$$L_3 = (k, 0, i-1, 1)$$

η οποία εκτελέστηκε με παραμέτρους $N = 1000, workiters = 100$. Η δομή περιέχει εξωτερικό βρόχο με αρκετά μεγάλο πλήθος επαναλήψεων, οπότε οι εκτελέσεις χωρίς `collapse` θα έχουν ικανοποιητικούς χρόνους. Επίσης, η δομή αυτή δεν ανήκει σε κάποια περίπτωση που επιδέχεται ιδιαίτερη διαχείριση, συνεπώς αναγκάζει τους μεταφραστές να χρησιμοποιήσουν τις γενικής περίπτωσης μεθόδους τους. Ο OMPi θα εφαρμόσει το περικλείον πλαίσιο συνδυαστικά με τη μέθοδο βάθους 2, ο GCC τη μέθοδο χρόνου εκτέλεσης και ο LLVM το περικλείον πλαίσιο. Στους Πίνακες 5.7, 5.8 και 5.9 έχουμε αναλυτικά όλες τις μετρήσεις.

Αρχικά, βλέποντας του χρόνους εκτέλεσης στο Σχήμα 5.7 συμπεραίνουμε ότι ο GCC έχει τους καλύτερους χρόνους. Αναμενόμενα ο OMPi, λόγω της μεθόδου του πλαισίου, είναι πίσω από τον GCC, αλλά μπροστά από τον LLVM, αφού χρησιμοποιεί συνδυαστικά και τη μέθοδο βάθους 2. Τέλος, ο LLVM αποτελεί έκπληξη, καθώς

Πίνακας 5.1: Χρόνοι εκτέλεσης για δομή $\mathcal{L}_1$ ($N = 8, workiters = 300000000$) (sec)

	OMPi		GCC		LLVM	
Νήματα	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη
1	112.263	112.263	112.263	112.263	112.234	112.234
2	83.584	57.118	82.252	56.252	82.228	82.244
4	50.540	29.795	48.742	28.177	47.734	47.742
8	28.124	15.413	25.679	14.426	25.829	25.676
16	33.660	9.510	25.681	7.708	26.613	13.505
32	33.842	4.778	25.687	4.037	27.422	6.882
64	33.845	2.616	25.679	2.612	27.330	3.974

Πίνακας 5.2: Speedup για δομή $\mathcal{L}_1$ ($N = 8, workiters = 300000000$)

	OMPi		GCC		LLVM	
Νήματα	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη
1	1.000	1.000	1.000	1.000	1.000	1.000
2	1.343	1.965	1.365	1.995	1.365	1.365
4	2.221	3.768	2.303	3.984	2.351	2.351
8	3.991	7.284	4.372	7.782	4.345	4.371
16	3.335	11.805	4.371	14.564	4.217	8.310
32	3.317	23.496	4.370	27.809	4.093	16.308
64	3.317	42.914	4.371	42.979	4.106	28.241

Πίνακας 5.3: Gain για δομή $\mathcal{L}_1$ ($N = 8, workiters = 300000000$) (%)

Νήματα	OMPi	GCC	LLVM
1	0.00	0.00	0.00
2	31.66	31.61	-0.02
4	41.05	42.19	-0.02
8	45.19	43.82	0.59
16	71.74	69.99	49.25
32	85.88	84.28	74.90
64	92.27	89.83	85.46

Πίνακας 5.4: Χρόνοι εκτέλεσης για δομή $\mathcal{L}_2$ ($N = 1000, workiters = 80000$) (sec)

	OMPi		GCC		LLVM	
Νήματα	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη
1	130.592	130.592	130.592	130.592	130.469	130.469
2	77.517	65.809	76.138	65.295	76.127	76.126
4	43.310	34.643	40.860	32.828	40.863	40.825
8	23.144	17.942	21.348	16.621	21.420	21.146
16	14.068	10.808	11.191	8.769	11.287	11.336
32	7.096	5.426	5.877	4.592	5.884	5.932
64	3.573	2.908	3.619	3.027	3.474	3.616

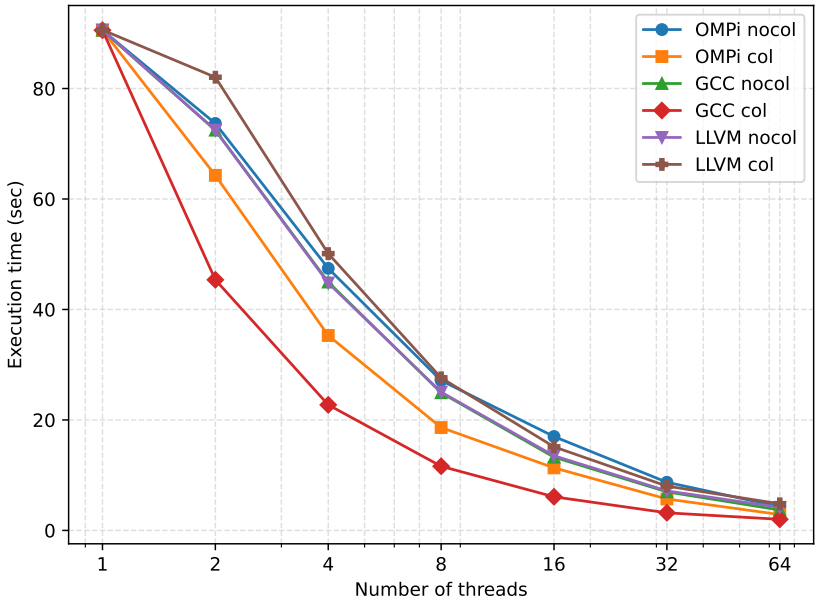
Πίνακας 5.5: Speedup για δομή $\mathcal{L}_2$ ($N = 1000, workiters = 80000$)

	OMPi		GCC		LLVM	
Νήματα	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη
1	1.000	1.000	1.000	1.000	1.000	1.000
2	1.685	1.984	1.715	2.000	1.713	1.714
4	3.015	3.770	3.196	3.978	3.193	3.196
8	5.643	7.279	6.117	7.857	6.091	6.170
16	9.283	12.083	11.670	14.894	11.559	11.509
32	18.404	24.068	22.221	28.439	22.173	21.994
64	36.550	44.908	36.085	43.142	37.556	36.081

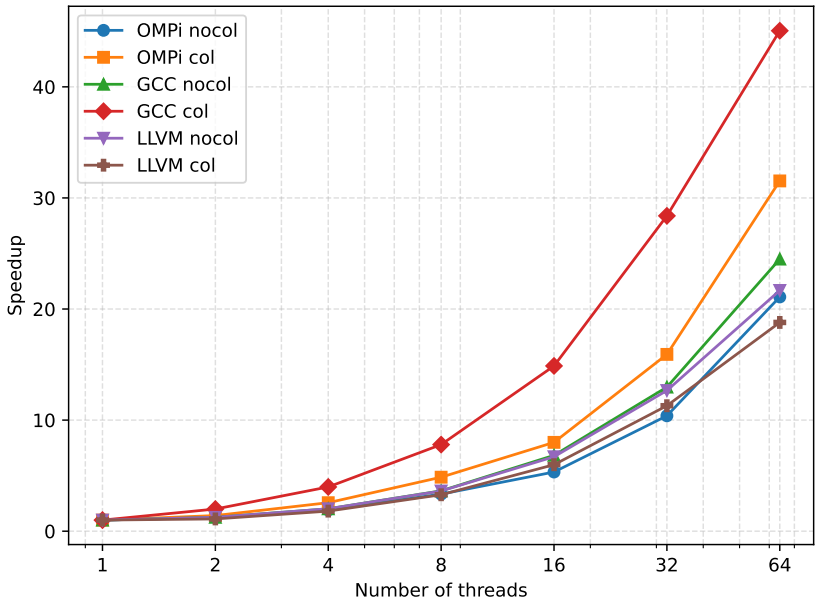
Πίνακας 5.6: Gain για δομή $\mathcal{L}_2$ ($N = 1000, workiters = 80000$) (%)

Νήματα	OMPi	GCC	LLVM
1	0.00	0.00	0.00
2	15.10	14.24	0.00
4	20.01	19.65	0.09
8	22.48	22.15	1.28
16	23.17	21.65	-0.44
32	23.54	21.86	-0.81
64	18.61	16.35	-4.09

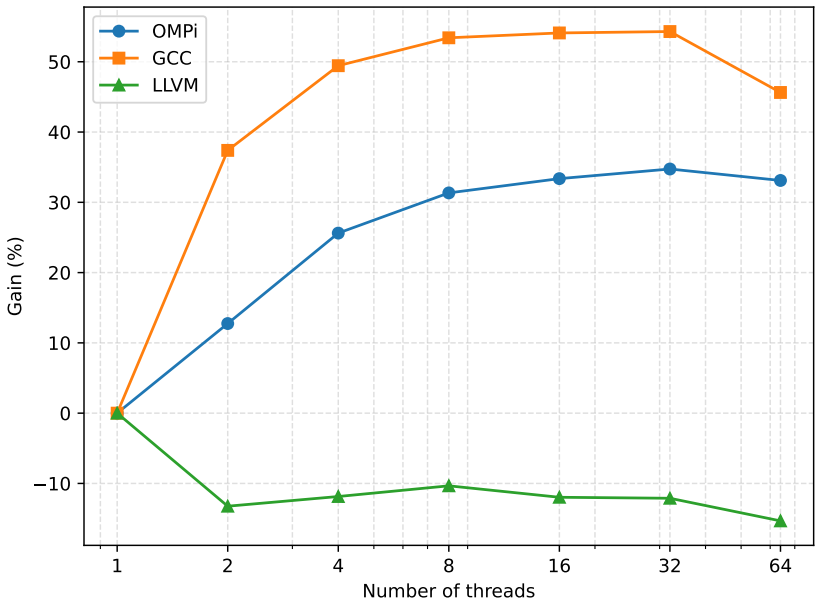
η `collapse` εκτέλεσή του έχει χειρότερους χρόνους από την `nocollapse` εκτέλεση. Παρόμοια εικόνα αποτυπώνεται και στα γραφήματα των speedup και gain στα Σχήματα 5.8 και 5.9.



Σχήμα 5.7: Χρόνοι εκτέλεσης για δομή $\mathcal{L}_3$ $(N = 1000, workiters = 100)$ (sec)



Σχήμα 5.8: Speedup για δομή $\mathcal{L}_3$ $(N = 1000, workiters = 100)$ (sec)



Σχήμα 5.9: Gain για δομή $\mathcal{L}_3$ $(N = 1000, workiters = 100)$ (sec)

Πίνακας 5.7: Χρόνοι εκτέλεσης για δομή $\mathcal{L}_3$ $(N = 1000, workiters = 100)$ (sec)

	OMPi		GCC		LLVM	
Νήματα	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη
1	90.551	90.551	90.551	90.551	90.597	90.597
2	73.673	64.276	72.460	45.371	72.448	82.043
4	47.441	35.289	44.963	22.734	44.798	50.112
8	27.133	18.629	24.926	11.615	25.034	27.623
16	16.997	11.325	13.263	6.088	13.516	15.134
32	8.719	5.690	6.982	3.190	7.156	8.022
64	4.294	2.872	3.697	2.010	4.180	4.821

Πίνακας 5.8: Speedup για δομή $\mathcal{L}_3$ $(N = 1000, workiters = 100)$

	OMPi		GCC		LLVM	
Νήματα	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη	Χωρίς σύμπτυξη	Με σύμπτυξη
1	1.000	1.000	1.000	1.000	1.000	1.000
2	1.229	1.409	1.250	1.996	1.251	1.104
4	1.909	2.566	2.014	3.983	2.022	1.808
8	3.337	4.861	3.633	7.796	3.619	3.280
16	5.328	7.996	6.827	14.874	6.702	5.986
32	10.385	15.914	12.970	28.386	12.663	11.294
64	21.088	31.529	24.493	45.050	21.673	18.792

Πίνακας 5.9: Gain για δομή $\mathcal{L}_3$ ($N = 1000, workiters = 100$) (%)

Νήματα	OMPi	GCC	LLVM
1	0.00	0.00	0.00
2	12.75	37.39	-13.24
4	25.62	49.44	-11.86
8	31.34	53.41	-10.34
16	33.37	54.10	-11.97
32	34.74	54.30	-12.11
64	33.11	45.63	-15.33

Κεφαλαιο 6

Επιλογος

6.1 Σύνοψη εργασίας

6.2 Μελλοντική εργασία

6.1 Σύνοψη εργασίας

Στην εργασία αυτή αρχικά είδαμε τη μαθηματική θεμελίωση για δομές βρόχων επανάληψης αυθαίρετου βάθους n, αναλύσαμε τους τύπους που χρησιμοποιούνται σε θεωρητικό επίπεδο και εισάγαμε την έννοια των μετασχηματισμών. Στη συνέχεια, περιγράψαμε την κανονικοποίηση βρόχων, καταγράψαμε τους τύπους ανάκτησης και δείξαμε τη γεωμετρική αποτύπωση της διαδικασίας. Ύστερα, είδαμε τη σύμπτυξη βρόχων και αναπτύξαμε τη θεωρητική του υπόσταση. Διαπιστώσαμε τη σαφή διαφοροποίηση στη διαδικασία μεταξύ ορθογωνικών και μη ορθογωνικών βρόχων και, χρησιμοποιώντας και το άρθρο της βιβλιογραφίας, είδαμε τις μεθόδους που δίνει η αλγεβρική προσέγγιση.

Στο πλαίσιο της εργασίας υλοποιήθηκε η σύμτπυξη μη ορθογωνικών βρόχων, η οποία βασίστηκε στον μεταφραστή OMPi. Υλοποιήθηκαν δύο μέθοδοι, η ευρετική μέθοδος του περικλείον πλαισίου και η μέθοδος με γεωμετρική μετατροπή του χώρου. Η πρώτη υλοποιήθηκε για να καλύψει την γενική περίπτωση και ο OMPi να υποστηρίξει την έκδοση 5.0 του OpenMP. Η δεύτερη υλοποιήθηκε ως μια εναλλακτική μέθοδος που δεν χρησιμοποιείται σε κάποιον από τους GCC και LLVM και είναι για ειδική αλλά συνηθισμένη περίπτωση δομής βρόχων βάθους 2.

Οι μετρήσεις δείχνουν ο OMPi να ανταγωνίζεται τους άλλους μεταφραστές. Συγκεκριμένα, στις περιπτώσεις που είδαμε φαίνεται να πλησιάζουμε σε χρόνους τον GCC, ο οποίος παρότι χρησιμοποιεί μεθόδους χρόνου εκτέλεσης, στα συγκεκριμένα πειράματα που κάναμε φάνηκε να έχει εν γένει πολύ καλή συμπεριφορά.

6.2 Μελλοντική εργασία

Η μέθοδος βάθους 2 κάνει ανάκτηση των τιμών σε κάθε επανάληψη του ενιαίου βρόχου. Αυτό, όπως αναφέραμε, επιτρέπει την επιλογή δρομολόγησης που θα εφαρμοστεί στις εντολές. Μια βελτίωση θα ήταν να τροποποιηθεί έτσι ώστε, αν το $chunksize$ που προκύπτει από το `schedule` είναι μεγαλύτερο του 1, τότε η μέθοδος να μην κάνει ανάκτηση σε κάθε επανάληψη, αλλά να κάνει ανάκτηση μία φορά στην αρχή του $chunk$ και να αυξομειώνει τους μετρητές μέχρι το τέλος του $chink$.

Για να βελτιωθούν γενικά οι επιδόσεις της σύμπτυξης του OMPi πρέπει να υλοποιηθεί διαφορετική μέθοδος για τη γενική περίπτωση. Ο μόνος γνωστός τρόπος υλοποίησης που παρότι δεν ήταν αναμενόμενο, στην πράξη δείχνει να αποδίδει αρκετά καλά, είναι του GCC. Επίσης, βασιζόμενοι στην ιδέα της γεωμετρικής προσέγγισης, θα μπορούσε να μελετηθεί η δυνατότητα παρόμοιων μετασχηματισμών για μεγαλύτερο βάθος δομής.

Επίσης, θα ήταν ιδιαίτερα χρήσιμη η υλοποίηση συνθετικών μετροπρογραμμάτων, τα οποία να είναι συμβατά με τους περιορισμούς της σύμπτυξης μη ορθογωνικών βρόχων. Μια ιδέα είναι να χρησιμοποιηθεί το PolyBench [10] και να μετασχηματιστούν, όπου αυτό είναι δυνατό, οι βρόχοι που περιέχει σε ισοδύναμους με τέλεια εμφώλευση, ώστε να μπορεί να χρησιμοποιηθεί η φράση `collapse`.

Τέλος, στο άρθρο [2] που είδαμε, οι βιβλιοθήκες [7, 8] που χρησιμοποιούνται αποτελούν υλοποιήσεις του *Πολυεδρικού μοντέλου* (Polyhedral model). Το Πολυεδρικό μοντέλο είναι ένα μαθηματικό μοντέλο που χρησιμοποιείται για την αναπαράσταση βρόχων, ώστε να είναι δυνατή η εφαρμογή βελτιστοποιήσεων. Μια καλή ιδέα θα ήταν η έρευνά του για εύρεση εργαλείων.

Βιβλιογραφια

[1] OpenMP Architecture Review Board, "OpenMP application programming interface version 5.0," OpenMP Architecture Review Board, Tech. Rep., November 2018. [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf

[2] P. Clauss, E. Altintas, and M. Kuhn, "Automatic collapsing of non-rectangular loops," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 778–787.

[3] V. V. Dimakopoulos, E. Leontiadis, and G. Tzoumas, "A portable C compiler for OpenMP V.2.0," in *Proc. EWOMP 2003, 5th European Workshop on OpenMP*, Aachen, Germany, September 2003, pp. 5–11.

[4] C. D. Polychronopoulos, "Loop coalesing: A compiler transformation for parallel machines," in *International Conference on Parallel Processing*, 1987. [Online]. Available: https://api.semanticscholar.org/CorpusID:10135795

[5] R. Sakellariou, "A compile-time partitioning strategy for non-rectangular loop nests," in *Proceedings 11th International Parallel Processing Symposium*, 1997, pp. 633–637.

[6] A. Kejariwal, P. D'Alberto, A. Nicolau, and C. D. Polychronopoulos, "A geometric approach for partitioning n-dimensional non-rectangular iteration spaces," in *Languages and Compilers for High Performance Computing*, R. Eigenmann, Z. Li, and S. P. Midkiff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 102–116.

[7] P. Clauss and V. Loechner, "Parametric analysis of polyhedral iteration spaces," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 19, no. 2, pp. 179–194, 1998.

[8] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe, "Counting integer points in parametric polytopes using Barvinok's rational functions," *Algorithmica*, vol. 48, no. 1, pp. 37–66, 2007.

[9] Maxima. Maxima, a computer algebra system. https://maxima.sourceforge.io/. [Online]. Available: https://maxima.sourceforge.io/

[10] L.-N. Pouchet, "Polybench: The polyhedral benchmark suite," 2012. [Online]. Available: https://www.cs.colostate.edu/~pouchet/software/polybench/

Σύντομο Βιογραφικό

Ο Γεώργιος Ζησόπουλος είναι απόφοιτος του Τμήματος Μαθηματικών του Πανεπιστημίου Ιωαννίνων. Είναι επίσης κάτοχος μεταπτυχιακού τίτλου στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων από το Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής του ίδιου πανεπιστημίου. Τα ερευνητικά του ενδιαφέροντα περιλαμβάνουν κυρίως την παράλληλη υπολογιστική, την υπολογιστική υψηλών επιδόσεων και τις βελτιστοποιήσεις μεταφραστών.