Μηχανισμός προσαρμοζόμενης υποστήριξης εκτέλεσης
για συσκευές
στον παραλληλοποιητικό μεταφραστή OMPi

από τον

Κασμερίδη Ηλία
με Α.Μ.: 2264

ως μέρος των Υποχρεώσεων για τη λήψη του

ΔΙΠΛΩΜΑΤΟΣ
ΤΟΥ ΤΜΗΜΑΤΟΣ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΗΣ

ΠΟΛΥΤΕΧΝΙΚΗΣ ΣΧΟΛΗΣ

Πανεπιστήμιο Ιωαννίνων
Οκτώβριος 2018

Επιβλέπων: Βασίλειος Δημακόπουλος

# Περιεχόμενα

# Κατάλογος σχημάτων

# Κεφάλαιο 1

# Εισαγωγή

## 1.1 Η εξέλιξη των Η/Υ

Από την αρχή της δημιουργίας του, ο άνθρωπος συνεχώς αναζητούσε τρόπους που θα βελτιώναν την ποιότητα της ζωής του, αλλά και της κοινωνίας της οποίας αποτελούσε μέλος. Με την πάροδο του χρόνου, προέκυψαν, εκτός από τις βιολογικές, επιπλέον ανάγκες εμπορικών υπολογισμών, οι οποίοι βασίζονταν σε φυσικούς αριθμούς και τις τέσσερις γνωστές αριθμητικές πράξεις.

Στην προσπάθειά του να διευκολύνει τους υπολογισμούς αυτούς, δημιούργησε στα μέσα του $17^{ου}$ αιώνα τις πρώτες μορφές υπολογιστικών συστημάτων, τα οποία χρησιμοποιούταν κυρίως για προσθαφαιρέσεις και υπολογισμό λογαρίθμων. Μιας και το ηλεκτρικό ρεύμα δεν είχε ακόμα ανακαλυφθεί, οι μηχανές αυτές βασίζονταν κυρίως στην μηχανική.

Η πρώτη εμφάνιση ηλεκτρονικού υπολογιστή γενικής χρήσης έγινε την δεκαετία του 1940, με την ονομασία ENIAC και καταλάμβανε 63 τ.μ.. Ο ηλεκτρονικός αυτός υπολογιστής είχε την δυνατότητα να εκτελεί 5000 προσθαφαιρέσεις το δευτερόλεπτο και χρησιμοποιήθηκε μέχρι τα μέσα της δεκαετίας του 1950, κυρίως για στρατιωτικούς σκοπούς.

Οι σημαντικές χωρικές απαιτήσεις του ENIAC σε συνδυασμό με την υψηλή κατανάλωση ρεύματος, στάθηκαν αιτίες για την προσπάθεια του ανθρώπου να βελτιστοποιήσει κάθε χαρακτηριστικό ενός υπολογιστικού συστήματος – το μέγεθός του, την χωρητικότητα της μνήμης του, την ταχύτητά του σε υπολογισμούς, αλλά και την κατανάλωση ισχύος του. Έτσι, φτάσαμε στην σημερινή μορφή ενός ηλεκτρονικού υπολογιστή, ο οποίος έχει την δυνατότητα να εκτελεί δισεκατομμύρια πράξεις το δευτερόλεπτο.

Όσο οι ηλεκτρονικοί υπολογιστές εξελισσόταν, ο άνθρωπος παρατήρησε ότι οι νόμοι της φύσης και το υλικό μπορούν να δράσουν περιοριστικά στην διαδικασία της βελτιστοποίησης τους. Συνεπώς, έπρεπε να προχωρήσει στην εύρεση νέων τεχνικών και μεθόδων για ταχύτερους υπολογισμούς, κάνοντας χρήση του ήδη υπάρχοντος υλικού. Την λύση στο πρόβλημα αυτό έδωσε ο κλάδος της παράλληλης επεξεργασίας, το μοντέλο της οποίας βασίζεται στην ταυτόχρονη εκτέλεση υπολογισμών από το υλικό.

## 1.2 Παράλληλα Συστήματα και Αρχιτεκτονικές

Η παράλληλη επεξεργασία αποβλέπει στην ταυτόχρονη εκτέλεση υπολογισμών και χειρισμών δεδομένων τα οποία ανήκουν σε μία ή περισσότερες διεργασίες, οι οποίες λύνουν το ίδιο πρόβλημα και επομένως στην επίτευξη της συνδρομικότητας (concurrency) σε έναν υπολογισμό. Νόημα στο μοντέλο της παράλληλης επεξεργασίας δίνουν τα παράλληλα συστήματα, τα οποία διαθέτουν περισσότερες από μία επεξεργαστικές μονάδες οι οποίες επικοινωνούν μεταξύ τους σε πραγματικό χρόνο, ώστε να υπολογίσουν ένα αποτέλεσμα ταχύτερα και αναλογικά με το πλήθος τους.

Τα παράλληλα συστήματα κατηγοριοποιούνται σε δύο κατηγορίες, κοινής μνήμης και κατανεμημένης μνήμης. Η υποκείμενη αρχιτεκτονική του παράλληλου συστήματος επηρεάζει άμεσα την τεχνική προγραμματισμού του, η οποία αφορά τον τρόπο με τον οποίο προσπελάζονται τα δεδομένα των διεργασιών που εκτελούνται στο σύστημα.

Από εδώ και στο εξής, χωρίς περιορισμό της γενικότητας, θα αναφερόμαστε στον όρο «επεξεργαστική μονάδα» ως «επεξεργαστής».

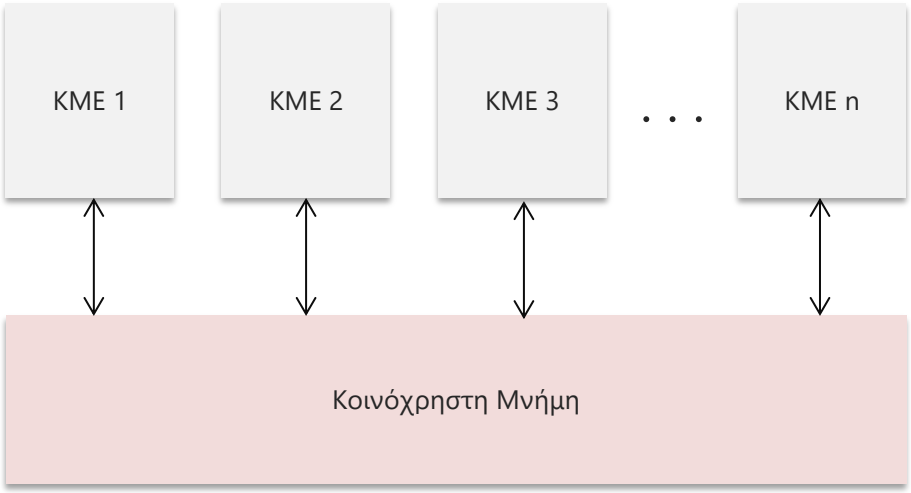
### 1.2.1 Αρχιτεκτονική Κοινής Μνήμης (Shared Memory)

Η βασική ιδέα στην αρχιτεκτονική παράλληλων συστημάτων κοινής μνήμης, είναι η ύπαρξη μιας κύριας μνήμης στην οποία έχουν πρόσβαση όλοι οι επεξεργαστές (Σχήμα 1.1). Τα δεδομένα αποθηκεύονται στην κύρια μνήμη με την μορφή μεταβλητών. Κάθε επεξεργαστής έχει πρόσβαση σε έναν κοινόχρηστο χώρο διευθύνσεων της κύριας μνήμης, τον οποίο χρησιμοποιεί για τροποποίηση ή ανάγνωση μεταβλητών. Μέσω των δύο αυτών ενεργειών, επιτυγχάνεται η επικοινωνία μεταξύ των επεξεργαστών. Ως δομικό στοιχείο για την επικοινωνία αυτή, χρησιμοποιείται ένας δίαυλος.

Η λογική αυτή παρότι μοιάζει αρκετά απλή, στην πραγματικότητα είναι αποτελεσματική μόνο για μικρό πλήθος επεξεργαστών. Εάν αυξήσουμε το πλήθος των επεξεργαστών, συγχρόνως αυξάνεται και ο αριθμός των επικοινωνιών που απαιτούνται για την συνεργασία τους, ενώ η χωρητικότητα του διαύλου παραμένει σταθερή. Συνεπώς, σε μια τέτοια περίπτωση, παρατηρούμε πτώση της απόδοσης

του συστήματος, με το μεγαλύτερο τμήμα του χρόνου εκτέλεσης να αφορά τις επικοινωνίες.

Ο προγραμματισμός συστημάτων κοινής μνήμης ακολουθεί το μοντέλο κοινού χώρου διευθύνσεων, με χρήση διεργασιών ή νημάτων. Οι οντότητες αυτές, κατά την δημιουργία τους, διαμοιράζονται στους επεξεργαστές, ώστε να επιτευχθεί παράλληλη εκτέλεση. Συνήθως προτιμούνται τα νήματα έναντι των διεργασιών. Η προτίμηση αυτή έγκειται στο γεγονός ότι τα νήματα διαθέτουν εγγενώς κοινόχρηστες μεταβλητές, ενώ οι διεργασίες διαθέτουν τον δικό τους ιδιωτικό χώρο διευθύνσεων και η επικοινωνία μεταξύ τους απαιτεί κλήσεις συστήματος, οι οποίες συχνά είναι χρονοβόρες. Επιπλέον, τα νήματα, συνολικά, αποτελούν μια πιο «ελαφριά» οντότητα, συγκριτικά με τις διεργασίες, με την έννοια ότι διαθέτουν μόνο τον μετρητή προγράμματος και κληρονομούν το περιβάλλον της διεργασίας-γονέα. Παρ'όλα αυτά, ο χειρισμός τους κατά την εκτέλεση είναι αρκετά δύσκολος, με αποτέλεσμα ο προγραμματιστής να καταφεύγει, ως επί τω πλείστων, σε έτοιμες βιβλιοθήκες χειρισμού νημάτων, οι οποίες υλοποιούνται σύμφωνα με κάποια πρότυπα παράλληλου προγραμματισμού.



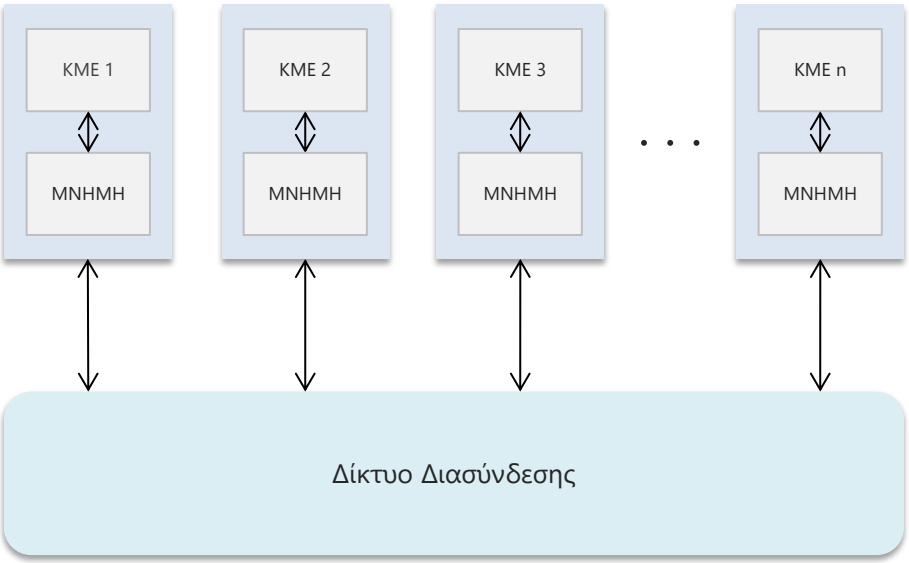

Σχήμα 1.1: Μοντέλο Κοινής Μνήμης

Ένα τέτοιο παράδειγμα αποτελεί το πρότυπο OpenMP [1], το οποίο προορίζεται για προγραμματισμό συστημάτων κοινής μνήμης. με την χρήση νημάτων. Για τις λειτουργίες και τις δομές του OpenMP γίνεται εκτενής αναφορά στο επόμενο κεφάλαιο.

### 1.2.2 Αρχιτεκτονική Κατανεμημένης Μνήμης (Distributed Memory)

Στα συστήματα αρχιτεκτονικής κατανεμημένης μνήμης, κάθε επεξεργαστής διαθέτει τη δική του ιδιωτική μνήμη. Η επικοινωνία των επεξεργαστών γίνεται μέσω ενός

δικτύου διασύνδεσης. Κάθε επεξεργαστής μπορεί να προσπελάσει άμεσα την δική του ιδιωτική μνήμη και έμμεσα την ιδιωτική μνήμη όλων των υπόλοιπων επεξεργαστών. Η έμμεση ανάγνωση και εγγραφή δεδομένων από και προς την ιδιωτική μνήμη ενός άλλου επεξεργαστή, πραγματοποιείται μέσω επικοινωνίας στο υπάρχον δίκτυο, όταν ένας επεξεργαστής θέλει να προσπελάσει ένα δεδομένο το οποίο διαθέτει ένας άλλος επεξεργαστής. Για παράδειγμα, έστω ότι ο επεξεργαστής Α διαθέτει το δεδομένο Χ. Εάν ο επεξεργαστής Β επιθυμήσει να προσπελάσει το Χ, τότε θα αποστείλλει μήνυμα στον Α μέσω του δικτύου διασύνδεσης. Ο επεξεργαστής Α, με την σειρά του, θα αποστείλλει το δεδομένο Χ στον Β.




Σχήμα 1.2: Μοντέλο Κατανεμημένης Μνήμης

Οι έμμεσες προσπελάσεις μνήμης υλοποιούνται με ανταλλαγή μηνυμάτων και μπορούν να εισάγουν σημαντικό φόρτο επικοινωνίας στο δίκτυο, συγκεκριμένα στην περίπτωση όπου το πλήθος των επεξεργαστών είναι μεγάλο και κάθε επεξεργαστής προσπελάζει συνεχώς μεταβλητές άλλων επεξεργαστών. Το γεγονός αυτό μπορεί να προκαλέσει σημαντική μείωση της απόδοσης του συστήματος.

Ο προγραμματισμός των συστημάτων κατανεμημένης μνήμης ακολουθεί το μοντέλο της μεταβίβασης μηνυμάτων. Κάθε διεργασία αντιστοιχεί σε έναν επεξεργαστή του συστήματος και ανταλλάσσει μηνύματα με τις υπόλοιπες, ώστε να επιτευχθεί συγχρονισμός στην εκτέλεση. Η ανταλλαγή μηνυμάτων γίνεται μέσω του δικτύου διασύνδεσης, που, όπως αναφέρθηκε προηγουμένως, αποτελεί δομικό στοιχείο του συστήματος. Ένα παράδειγμα μηχανισμού ανταλλαγής μηνυμάτων είναι το πρότυπο MPI [2]. Το MPI προσφέρει διάφορες λειτουργικότητες, όπως ρουτίνες για την διασύνδεση ομάδων διεργασιών και την επικοινωνία μεταξύ τους, καθώς επίσης και ρουτίνες για συλλογικές επικοινωνίες, όπως η ευρεία μετάδοση μηνύματος. Τέλος, περιλαμβάνει δομές που ορίζουν τον τύπο δεδομένων που θα αποσταλλούν μέσω του καναλιού επικοινωνίας, ώστε να διευκολύνεται η συνεργασία των διεργασιών.

## 1.3 Ετερογένεια

Με την εξέλιξη των υπολογιστικών συστημάτων ξεκίνησαν να υποστηρίζονται, εκτός της κύριας μονάδας επεξεργασίας, και συνοδές υπολογιστικές συσκευές όπως επιταχυντές (accelerators) και GPUs, οι οποίες χρησιμοποιούνται για πράξεις ειδικού σκοπού που η ίδια η CPU μπορεί να εκτελεί βραδύτερα. Η ετερογένεια αυτή εμφανίζεται και στο λογισμικό, το οποίο καλείται να αξιοποιήσει τα ποικίλα υλικά και πολλές φορές απαιτείται να υλοποιηθεί ξεχωριστά. Ως αποτέλεσμα, ο προγραμματισμός ενός συστήματος καθίσταται δυσκολότερος.

Το πρότυπο OpenMP σχεδιάστηκε ώστε να βελτιώσει και να διευκολύνει τον προγραμματισμό ετερογενών υλικών και λογισμικών, υποστηρίζοντας τόσο τον πολυνηματικό προγραμματισμό στο κύριο σύστημα (host), όσο και σε «συσκευές» (devices), αλλά με ενιαίο κώδικα. Με τον όρο συσκευή αναφερόμαστε στα μηχανήματα λογικής εκτέλεσης, φυσικά ή εικονικά, η ύπαρξη των οποίων καθορίζεται από την υλοποίηση. Η χρήση μιας συσκευής γίνεται από το κύριο σύστημα μέσω ειδικών οδηγιών. Η λογική κύριου συστήματος και συσκευών συνυπάρχουν στον ίδιο κώδικα, ώστε να αξιοποιηθεί πλήρως όλο το υλικό και να επιτευχθούν ταχύτεροι υπολογισμοί.

## 1.4 Αντικείμενο της Διπλωματικής Εργασίας

Το αντικείμενο της παρούσας διπλωματικής εργασίας αφορά την βελτιστοποίηση του παραλληλοποιητικού μεταφραστή OMPi (Κεφάλαιο 3) κατά τον χρόνο εκτέλεσης εφαρμογών που περιέχουν τμήματα κώδικα που προορίζονται για συσκευές (devices), όπως αυτές ορίζονται από το πρότυπο OpenMP (Κεφάλαιο 2). Κάθε τμήμα κώδικα κάνει χρήση συγκεκριμένων οδηγιών και δομών του OpenMP, οι οποίες υλοποιούνται από μια ειδική βιβλιοθήκη συσκευής. Από προεπιλογή, η βιβλιοθήκη αυτή περιέχει υλοποιήσεις όλων των απαραίτητων κλήσεων που

απαιτούνται για το περιβάλλον OpenMP, γεγονός το οποίο δικαιολογεί το αυξημένο μέγεθός της.

Το μέγεθος της βιβλιοθήκης συσκευής προσμετράται στον συνολικό όγκο δεδομένων που θα αποθηκευτούν στη μνήμη της συσκευής κατά την εκτέλεση. Έχοντας ως στόχο την μείωση του όγκου αυτού, επιθυμούμε να κάνουμε χρήση μιας μικρότερης σε μέγεθος βιβλιοθήκης, η οποία θα περιέχει μόνο τις απαραίτητες για το κάθε τμήμα κώδικα υλοποιήσεις οδηγιών OpenMP (Κεφάλαιο 3). Μπορούμε να έχουμε πολλαπλές *εκδοχές* (flavors) της βιβλιοθήκής και κάθε μία να υλοποιεί το δικό της υποσύνολο του προτύπου OpenMP. Τίθεται, όμως, το ζήτημα του πώς θα γίνει η αυτόματη επιλογή από τον μεταφραστή της βιβλιοθήκης που αυτός θα χρησιμοποιήσει.

Η λύση που υλοποίησαμε είναι η ανάθεση της εργασίας αυτής σε ένα ξεχωριστό τμήμα της διαδικασίας μετάφρασης, το οποίο ονομάζεται mapper. Κατά την μετάφραση, ο μεταφραστής OMPi συλλέγει τις απαραίτητες μετρικές, με λεπτομερή ανάλυση του κώδικα. ατά το στάδιο της σύνδεσης (linking), οι μετρικές περνούνται στον mapper ο οποίος, γνωρίζοντας τις υπάρχουσες εκδοχές αναλαμβάνει να επιλέξει την πλέον κατάλληλη. Επειδή η επιλογή απαιτεί γνώση των χαρακτηριστικών της εκάστοτε εκδοχής, σχεδιάσαμε και υλοποιήσαμε μία γενική λύση: μία νέα γλώσσα, τη MAL (**Ma**pper **L**anguage), προκειμένου ο σχεδιαστής μίας συσκευής να μπορεί να περιγράψει τη λογική της επιλογής μεταξύ των διάφορων εκδοχών. Συγκεκριμένα, για την προαναφερθείσα βελτιστοποίηση:

- Τροποποιήθηκε η μέθοδος σύνδεσης βιβλιοθηκών υποστήριξης χρόνου εκτέλεσης (runtime support libraries) για τις συσκευές.
- Προστέθηκαν κατάλληλες μετρικές στη διαδικασία συλλογής στατιστικών του τμήματος μεταγλώττισης (compiler).
- Υλοποιήθηκε η MAL, η οποία χρησιμοποιείται για την σύνταξη αρχείων λήψης αποφάσεων με την μορφή διαγράμματος ροής.
- Υλοποιήθηκε ο συντακτικός αναλυτής MAL-parser της γλώσσας MAL, ο οποίος βασίζεται στον λεκτικό αναλυτή MAL-scanner, καθώς επίσης και μια δομή άκυκλου γράφου για την αναπαράσταση των διαγραμμάτων ροής στη μνήμη, το MAL-graph.
- Αναπτύχθηκε το εργαλείο προσαρμοζόμενης υποστήριξης εκτέλεσης σε συσκευές OpenMP MAL-mapper, το οποίο βασίζεται σε διαγράμματα ροής για την λήψη απόφασης. Η υλοποίηση και η λειτουργία του mapper περιγράφεται στα επόμενα κεφάλαια.

## 1.5 Δομή της Διπλωματικής Εργασίας

Η διπλωματική εργασία είναι οργανωμένη με τον εξής τρόπο:

- Κεφάλαιο 2: Παρουσίαση και περιγραφή του προτύπου παράλληλου προγραμματισμού OpenMP. Επεξήγηση των οδηγιών, των δομών και των

κλήσεων συστήματος που αποτελούν το πρότυπο. Παραδείγματα χρήσης. Αναφορά στις νέες δυνατότητες που εισήγαγε η έκδοση 4.5 του OpenMP.

- Κεφάλαιο 3: Παρουσίαση και περιγραφή του παραλληλοποιητικού μεταφραστή OMPi. Αναφορά στις επιμέρους λειτουργίες του μεταφραστή, καθώς και στις δομές που χρησιμοποιεί. Περιγραφή της διαδικασίας σύλλεξης στατιστικών που αφορούν τον πηγαίο κώδικα του χρήστη.
- Κεφάλαιο 4: Περιγραφή του μηχανισμού προσαρμοζόμενης υποστήριξης εκτέλεσης. Περιγραφή της υλοποίησης του εργαλείου mapper. Επεξήγηση του τρόπου χρήσης από τον μεταφραστή OMPi
- Κεφάλαιο 5: Επεξήγηση της προσαρμοζόμενης υποστήριξης εκτέλεσης σε συσκευές OpenMP. Η γλώσσα απόφασης MAL. Περιγραφή της δομής MAL-graph, καθώς και του λεκτικού/συντακτικού αναλυτή. Επισκόπηση του λεκτικού αναλυτή MAL-scanner και του συντακτικού αναλυτή MAL-parser.
- Κεφάλαιο 6: Περιγραφή και ανάλυση των αποτελεσμάτων από την εκτέλεση διάφορων εφαρμογών, για την μέτρηση απόδοσης του OMPi μετά από την εισαγωγή του εργαλείου MAL-mapper, με τη χρήση ενός έτοιμου διαγράμματος ροής.
- Κεφάλαιο 7: Σύνοψη διπλωματικής εργασίας, αναφορά σε μελλοντικές βελτιστοποιήσεις του μεταφραστή OMPi.

# Κεφάλαιο 2

# Το πρότυπο OpenMP

## 2.1 Εισαγωγή

Καθώς τα παράλληλα συστήματα άρχισαν να αναπτύσσονται με γρήγορους ρυθμούς, προέκυψαν νέες απαιτήσεις που αφορούν τις τεχνικές προγραμματισμού μιας παράλληλης εφαρμογής. Πιο συγκεκριμένα, η διάθεση πολλαπλών επεξεργαστών στον προγραμματιστή, δημιούργησε προβλήματα στην προσπέλαση των δεδομένων, των συναρτήσεων και γενικότερα των συστατικών της εφαρμογής από τις διάφορες οντότητες. Επομένως, ο προγραμματιστής καλείται να χειριστεί με πιο εξειδικευμένες μεθόδους τον συντονισμό των οντοτήτων αυτών. Η ανάγκη αυτή για διευκόλυνση των προγραμματιστών παράλληλων συστημάτων, οδήγησε στην δημιουργία ενός προτύπου-διεπαφής παράλληλου προγραμματισμού, το OpenMP.

Το πρότυπο του OpenMP αποσκοπεί στην εύκολη συγγραφή παράλληλων εφαρμογών σε συστήματα με αρχιτεκτονική κοινής μνήμης. Αποτελείται από:

- **Οδηγίες προς τον μεταφραστή.** Καθορίζουν τον τρόπο με τον οποίο ο μεταφραστής θα χειριστεί ένα τμήμα κώδικα ή κάποια δομή που χρησιμοποιείται στην εφαρμογή. Αναφερόμαστε στις οδηγίες αυτές ως “pragmas” ή “directives”.
- **Ρουτίνες βιβλιοθήκης**. Είναι ρουτίνες διαθέσιμες προς κλήση από τον προγραμματιστή και χρησιμοποιούνται κυρίως για χειρισμό παραμέτρων της εκτέλεσης, ή για διευκόλυνση του συγχρονισμού των νημάτων. Τέτοια παραδείγματα είναι ο ορισμός του πλήθους των νημάτων που θα εκτελέσουν ένα τμήμα κώδικα, ή η χρήση των κλειδαριών κατά την εκτέλεση μιας κρίσιμης περιοχής.
- **Μεταβλητές περιβάλλοντος**, η χρήση των οποίων ρυθμίζει εκ των προτέρων την εκτέλεση ενός παράλληλου προγράμματος που κάνει χρήση του προτύπου OpenMP.

Στόχος του συνόλου των κανόνων και δομών του OpenMP είναι να δώσει την δυνατότητα στον προγραμματιστή να συγγράψει κλιμακωτά μια παράλληλη εφαρμογή, παραλληλοποιώντας τμήματα κώδικα στο ήδη υπάρχον σειριακό πρόγραμμα, χωρίς καμία τροποποίηση στην λογική του. Επιπλέον, καθιστά δυνατή την φορητότητα (portability) των εφαρμογών, με την έννοια ότι το θέμα χειρισμού των νημάτων καλείται να λυθεί από τον εκάστοτε μεταφραστή και όχι από τον ίδιο τον προγραμματιστή, καθιστόντας την συμπεριφορά της παράλληλης εφαρμογής σε διαφορετικές πλατφόρμες πανομοιότυπη.

Ενώ το πρότυπο OpenMP μοιάζει μονόδρομος για τον προγραμματισμό παράλληλων εφαρμογών, συναντούμε ένα μειονέκτημα στην απόκρυψη της λογικής χειρισμού των νημάτων. Αυτό είναι η αδυναμία του προγραμματιστή να καθορίσει εξ ολοκλήρου τον τρόπο συμπεριφοράς τους, ή να χρησιμοποιήσει ένα συγκεκριμένο υποσύνολο νημάτων για την διεκπαιρέωση μιας εργασίας. Παρ'όλα αυτά, προσφέρει άλλες δυνατότητες, όπως ο καθορισμός του κόκκου παραλληλίας (granularity).

## 2.2 Προγραμματιστικό Μοντέλο OpenMP

Το πρότυπο του OpenMP, όπως αναφέρθηκε, αφορά την παραλληλία με χρήση νημάτων σε συστήματα κοινής μνήμης. Το προγραμματιστικό του μοντέλο βασίζεται στο γνωστό μοντέλο fork-join που συναντάται στις διεργασίες.

Πιο συγκεκριμένα, το πρόγραμμα εκτελείται σειριακά από το κύριο νήμα (master thread), το οποίο, αρχικά, ταυτίζεται με την διεργασία. Όταν, κατά την εκτέλεση, συναντηθεί μια *περιοχή παραλληλίας* (parallel region), τότε δημιουργείται μια ομάδα νημάτων, το πλήθος των οποίων προκαθορίζεται από τον προγραμματιστή (δυναμικά) ή από μια μεταβλητή περιβάλλοντος (στατικά). Η ομάδα νημάτων εκτελεί την περιοχή παραλληλίας, ενώ το κύριο νήμα βρίσκεται σε αναμονή έως ότου η ομάδα ολοκληρώσει την εκτέλεση. Στο πέρας της εκτέλεσης, η ομάδα καταστρέφεται και, έπειτα, το κύριο νήμα συνεχίζει την εκτέλεση του υπόλοιπου σειριακού κώδικα. Η διαδικασία αυτή επαναλαμβάνεται όσες φορές χρειαστεί, για κάθε περιοχή παραλληλίας.

Μέσα σε μία περιοχή παραλληλίας, μπορούν να χρησιμοποιηθούν διάφορες οδηγίες προσφερόμενες από την υλοποίηση του OpenMP, ώστε να παραμετροποιηθεί καταλλήλως ο τρόπος εκτέλεσης της. Σε επόμενο υποκεφάλαιο γίνεται μια εκτενής περιγραφή των οδηγιών αυτών.

## 2.3 Οδηγίες OpenMP για C/C++

Στο παρόν κεφάλαιο αναφέρεται ο τρόπος σύνταξης μιας οδηγίας OpenMP, ενώ στην συνέχεια περιγράφονται οι έννοιες των περιοχών *παραλληλίας* (parallel regions) και *διαμοιρασμού* (workshare regions), καθώς επίσης και οι τεχνικές συγχρονισμού των νημάτων, με χρήση *οδηγιών* (directives). Έπειτα, γίνεται μια αναφορά στις μεταβλητές περιβάλλοντος και περιγράφονται συνοπτικά οι νέες δυνατότητες που εισήχθηκαν στο πρότυπο OpenMP 4.5.

### 2.3.1 Σύνταξη οδηγιών OpenMP

Οι οδηγίες του OpenMP για τις γλώσσες προγραμματισμού C/C++ καθορίζονται με την χρήση της οδηγίας προεπεξεργαστή ***pragma***. Η σύνταξη μιας οδηγίας OpenMP είναι η εξής:

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

Ισχύουν οι παρακάτω κανόνες όσον αφορά την σύνταξη της οδηγίας:

- Το όνομα της οδηγίας είναι υποχρεωτικό μετά από την χρήση ενός **#pragma omp**.
- Τα **#pragma omp** κάνουν διάκριση πεζών-κεφαλαιών, όσον αφορά το όνομα της οδηγίας.
- Μετά την οδηγία τοποθετούνται, προαιρετικά, οι φράσεις οδηγιών (χωρίς περιορισμό στην σειρά δήλωσής τους), οποίες ορίζουν τις συνθήκες υπό τις οποίες θα εκτελεστεί η οδηγία. Εάν δεν έχει τοποθετηθεί καμία οδηγία, τότε οι συνθήκες αυτές καθορίζονται στον χρόνο εκτέλεσης του προγράμματος.
- Η σύνταξη μιας οδηγίας OpenMP απαιτεί την ύπαρξη *νέας γραμμής* (new-line) στο τέλος της.

### 2.3.2 Περιοχές Παραλληλίας (Parallel regions)

Μια περιοχή παραλληλίας υποδηλώνει ένα τμήμα κώδικα, το οποίο προορίζεται για εκτέλεση από μια ομάδα νημάτων. Η δήλωση της περιοχής παραλληλίας γίνεται με την οδηγία *parallel*, όπως αυτή περιγράφεται παρακάτω:

**#pragma omp** parallel *[φράση[ [,] φράση] ... ]*
*δομημένο τμήμα*

*(Σημείωση: Ως δομημένο τμήμα (structured block) ορίζουμε το τμήμα κώδικα που πρόκειται να παραλληλοποιηθεί, εγκλεισμένο σε άγκιστρα.)*

Όταν το *κύριο νήμα* (master thread) συναντήσει, κατά την εκτέλεση, την οδηγία parallel, τότε θα δημιουργήσει μια ομάδα νημάτων, στην οποία συμμετέχει το ίδιο ως γονέας. Η ομάδα που δημιουργήθηκε, θα εκτελέσει τον κώδικα που βρίσκεται

μέσα στο *δομημένο τμήμα* που περιγράφηκε στην σύνταξη. Το μέγεθος της ομάδας είναι προκαθορισμένο από τον χρήστη ή το περιβάλλον εκτέλεσης.

Ενδέχεται ένα ή περισσότερα νήματα-μέλη να ολοκληρώσουν την εκτέλεση του τμήματος κώδικα σε μικρότερη χρονική διάρκεια, συγκριτικά με κάποια άλλα μέλη. Το φαινόμενο αυτό οφείλεται στο γεγονός ότι ο κάθε επεξεργαστής για μια χρονική στιγμή μπορεί να εκτελεί διαφορετικού είδους εργασίες, η ύπαρξη των οποίων μεταβάλλει το πότε αυτός θα γίνει διαθέσιμος για το νήμα. Συνεπώς, για τον συγχρονισμό των νημάτων, στο τέλος της περιοχής παραλληλίας υπονοείται ένα φράγμα (barrier), το οποίο αναγκάζει κάθε νήμα που το συναντά να αναμένει τον τερματισμό όλων των υπολοίπων. Κάτα την ολοκλήρωση της εκτέλεσης, το φράγμα λύνεται και η ομάδα καταστρέφεται, ενώ το κύριο νήμα συνεχίζει κανονικά την εκτέλεση του υπόλοιπου προγράμματος.

source_2_1.c

```
1  int main()
2  {
3       int thr_id;
4       omp_set_num_threads(8);
5
6       #pragma omp parallel
7       {
8        thr_id = omp_get_thread_num();
9        printf("Hello. I am thread %d.\n", thr_id);
10      }
11 }
```

Κώδικας 2.1: Παράδειγμα περιοχής παραλληλίας σε OpenMP

Ο χρήστης, όπως προαναφέρθηκε, έχει την δυνατότητα να ορίσει το μέγεθος του τμήματος της ομάδας νημάτων που θα εκτελέσει την περιοχή παραλληλίας. Οι βασικές (ισοδύναμες) μέθοδοι που χρησιμοποιούνται για τον καθορισμό του μεγέθους της ομάδας είναι οι εξής:

- Κλήση της συνάρτησης `omp_set_num_threads()` εκτός της περιοχής παραλληλίας. Η `omp_set_num_threads()` δέχεται ως όρισμα το πλήθος των νημάτων που θα απαρτίσουν την ομάδα που πρόκειται να δημιουργηθεί.
- Εκχώρηση τιμής στην μεταβλητή περιβάλλοντος OMP_NUM_THREADS, πριν την εκτέλεση του παράλληλου προγράμματος.
- Τοποθέτηση της φράσης `num_threads()` μετά από την οδηγία parallel με όρισμα το πλήθος των επιθυμητών νημάτων.

Στον Κώδικα 2.1 αναφέρεται ένα απλό παράδειγμα της οδηγίας parallel. Το κύριο νήμα θα καλέσει την συνάρτηση *omp_set_num_threads()* με όρισμα *8*, συνεπώς η ομάδα θα αποτελείται από 8 νήματα. Το αποτέλεσμα που λαμβάνουμε είναι το εξής:

(Σημείωση: Η συνάρτηση *omp_get_thread_num()* περιγράφεται στην ενότητα 2.4).

### 2.3.3 Περιοχές Διαμοιρασμού Εργασίας (Worksharing regions)

Μια δυνατότητα που προσφέρεται από το OpenMP, είναι ο ορισμός περιοχών διαμοιρασμού εργασίας μέσα στο πρόγραμμα, οι οποίες χρησιμοποιούνται για τον καταμερισμό του φόρτου εργασίας μεταξύ των νημάτων.

Οι περιοχές διαμοιρασμού εργασίας αποκτούν νόημα όταν βρίσκονται εντός μιας περιοχής παραλληλίας. Η βασική διαφορά τους σε σχέση με τις παράλληλες περιοχές, είναι ότι δεν δημιουργούνται νέα νήματα, αλλά, αντιθέτως, χρησιμοποιούνται τα ήδη υπάρχοντα.

Στο τέλος των περιοχών διαμοιρασμού εργασίας υπονοείται ένα φράγμα (barrier), ακριβώς όπως και στην περίπτωση των παράλληλων περιοχών, με σκοπό

**Έξοδος:**

```
Hello. I am thread 0.
Hello. I am thread 1.
Hello. I am thread 2.
Hello. I am thread 3.
Hello. I am thread 4.
Hello. I am thread 5.
Hello. I am thread 6.
Hello. I am thread 7.
```

τον συγχρονισμό των νημάτων. Το φράγμα αυτό μπορεί να παραλειφθεί με την τοποθέτηση της φράσης *nowait* στην οδηγία. Υπάρχουν τρεις τύποι περιοχών διαμοιρασμού εργασίας και ορίζονται, αντίστοιχα, με τις εξής οδηγίες:

- **Οδηγία for.** Αφορά τον καταμερισμό του φόρτου ενός βρόχου επανάληψης for. Αμέσως μετά την δήλωση της συγκεκριμένης οδηγίας, ακολουθεί ο βρόχος for που πρόκειται να παραλληλοποιηθεί. Ο τρόπος με τον οποίο διαμοιράζεται ο φόρτος του βρόχου περιγράφεται αναλυτικά στην συνέχεια του κεφαλαίου. Ο τρόπος σύνταξης της οδηγίας είναι ο εξής:

```
#pragma omp for [φράση[ [,] φράση] ... ] νέα-γραμμή
  for (...) {
    ...
  }
```

- **Οδηγία sections.** Η συγκεκριμένη οδηγία χρησιμοποιείται για την παραλληλοποίηση εργασιών οι οποίες δεν σχετίζονται μεταξύ τους, από τα νήματα εντός μιας περιοχής παραλληλίας. Στο εσωτερικό της οδηγίας τοποθετούνται διαδοχικές δηλώσεις της οδηγίας `#pragma omp section`, συνοδευόμενες από δομημένα τμήματα κώδικα.
  Η δομή της δήλωσης της οδηγίας είναι η εξής:

  **#pragma omp** sections *[φράση[ [,] φράση] ... ] νέα-γραμμή*
  {
  **#pragma omp** section
  *δομημένο τμήμα*
  *...*
  }

- **Οδηγία single**. Το νήμα το οποίο συναντά πρώτο την οδηγία `single` εκτελεί το δομημένο τμήμα κώδικα που ορίζεται μετά από αυτή, ενώ τα υπόλοιπα την προσπερνούν. Η συμπεριφορά των νημάτων, από προεπιλογή, είναι να αναμένουν έως ότου το πρώτο ολοκληρώσει την εκτέλεση του κώδικα, για λόγους συγχρονισμού.
  Παραλλαγή της οδηγίας `single` αποτελεί η οδηγία `master`, η οποία απαιτεί η εκτέλεση του δομημένου τμήματος κώδικα να γίνει από το κύριο νήμα (master thread). Επιπλέον, στην περίπτωση αυτής της οδηγίας δεν υπονοείται φράγμα συγχρονισμού των νημάτων.
  Η σύνταξη των δύο προαναφερθέντων οδηγιών είναι η εξής:

  **#pragma omp** [ single | master ] *[φράση[ [,] φράση] ... ] νέα-γραμμή*
  *δομημένο τμήμα*

### 2.3.5 Οδηγίες Συγχρονισμού

Εξίσου σημαντικό κομμάτι του προτύπου OpenMP αποτελεί το σύνολο των οδηγιών συγχρονισμού. Χρησιμοποιούνται για τον συγχρονισμό των νημάτων κατά την εκτέλεση μιας περιοχής παραλληλίας, ώστε να εξασφαλιστεί συνέπεια των κοινόχρηστων δεδομένων. Οι βασικότερες των οδηγιών αυτών είναι οι παρακάτω:

- **Οδηγία atomic**. Χρησιμοποιείται για εκτέλεση ατομικής πράξης. Οι ατομικές πράξεις εξασφαλίζουν ότι η τροποποίηση μιας μεταβλητής θα πραγματοποιηθεί αδιαίρετα και αδιάκοπα από ένα συγκεκριμένο νήμα τη φορά. Έτσι αποκλείεται το ενδεχόμενο κάποιο άλλο νήμα να παρέμβει στην διαδικασία αυτή και να ανατεθεί κάποια ανεπιθύμητη τιμή στη μεταβλητή ενδιαφέροντος.
- **Οδηγία barrier**. Η οδηγία *barrier* ορίζει ένα φράγμα συγχρονισμού εκτέλεσης στο σημείο εκτέλεσης που τοποθετείται. Τα νήματα που φθάνουν στο σημείο αυτό, αναμένουν έως ότου καταφθάσουν και τα υπόλοιπα νήματα της ομάδας. Όταν το φράγμα λυθεί, έχει εξασφαλιστεί ότι οι τιμές των κοινόχρηστων μεταβλητών είναι οι ίδιες για όλα τα νήματα.
- **Οδηγία critical**. Η συγκεκριμένη οδηγία θεωρείται μια επέκταση της οδηγίας *atomic* για δομημένα τμήματα κώδικα. Ορίζει περιοχές του κώδικα, τις λεγόμενες *κρίσιμες περιοχές*, οι οποίες πρέπει να εκτελεστούν από ένα νήμα τη φορά. Τα υπόλοιπα νήματα της ομάδας αναμένουν έως ότου το επιλεγμένο νήμα ολοκληρώσει την εκτέλεση της κρίσιμης περιοχής.
- **Οδηγία flush**. Η οδηγία flush χρησιμοποιείται για να συγχρονίσει τα νήματα της ομάδας και να εξασφαλίσει συνέπεια της κοινόχρηστης μνήμης των νημάτων, με την έννοια ότι δεν εκκρεμούν εγγραφές σε αυτήν. Αναφερόμαστε στην διαδικασία αυτή ως *write back*.
- **Οδηγία ordered**. Η οδηγία ordered χρησιμοποιείται για την σειριοποίηση της εκτέλεσης ενός τμήματος κώδικα εντός μιας παράλληλης περιοχής. Συνήθως τοποθετείται στο εσωτερικό ενός βρόχου επανάληψης, όταν χρειάζεται να τηρηθεί συγκεκριμένη σειρά στην εκτέλεση των επαναλήψεων.

### 2.3.6 Φράσεις Οδηγιών

Οι φράσεις (ή συνθήκες) οδηγιών τοποθετούνται στο σημείο δήλωσης μιας οδηγίας OpenMP, ακριβώς μετά από το όνομα της οδηγίας και χρησιμοποιούνται για να ρυθμίσουν τον τρόπο εκτέλεσης μιας οδηγίας. Πιο συγκεκριμένα, ορίζουν τις συνθήκες υπό τις οποίες αυτή θα εκτελεστεί, ή καθορίζουν την συμπεριφορά των νημάτων πριν, κατά τη διάρκεια και μετά την εκτέλεση του τμήματος κώδικα που ακολουθεί την οδηγία. Κάποιες βασικές φράσεις που συναντούνται κατά κόρον στον προγραμματισμό με OpenMP είναι οι εξής:

- **if**(*συνθήκη*). Όταν τοποθετήσουμε την φράση *if* στην δήλωση της οδηγίας, γίνεται ο έλεγχος της συνθήκης που αυτή περιέχει. Αν η συνθήκη ισχύει, τότε η οδηγία θα εκτελεστεί κατά τον χρόνο εκτέλεσης, διαφορετικά θα αγνοηθεί.
- **shared**(*λίστα μεταβλητών*). Η φράση *shared* τοποθετείται συνήθως στην δήλωση της οδηγίας *parallel*, και χρησιμοποιείται για να ορίσει τα αντικείμενα της *λίστας μεταβλητών* ως κοινόχρηστα μεταξύ των νημάτων της ομάδας που θα εκτελέσουν την περιοχή παραλληλίας.
- **private**(*λίστα μεταβλητών*). Η φράση *private* χρησιμοποιείται με την δήλωση της οδηγίας *parallel* για να ορίσει ως ιδιωτικά τα αντικείμενα της λίστας μεταβλητών, για κάθε νήμα της ομάδας παραλληλίας. Στην πράξη, οι μεταβλητές της λίστας δηλώνονται εκ νέου στην ιδιωτική μνήμη κάθε νήματος και έχουν ισχύ μέχρι το πέρας της περιοχής παραλληλίας.
- **firstprivate**(*λίστα μεταβλητών*). Οι φράσεις firstprivate και private λειτουργούν με παρόμοιο τρόπο. Η μόνη διαφορά μεταξύ τους είναι ότι στην πρώτη, τα αντικείμενα της λίστας μεταβλητών αρχικοποιούνται με τις τιμές των αντίστοιχων μεταβλητών που βρίσκονται στην εξωτερική περιοχή, σε αντίθεση με την εκ νέου δήλωση που συμβαίνει στην δεύτερη.
- **lastprivate**(*λίστα μεταβλητών*). Οι lastprivate λειτουργεί όπως και η private, με την μόνη διαφορά ότι στο τέλος της περιοχής παραλληλίας οι τιμές των αντικείμενα της λίστας μεταβλητών ενημερώνονται.
- **nowait**. Όταν τοποθετείται η φράση *nowait* στην δήλωση μιας οδηγίας, τότε υπονοείται ότι τα νήματα της ομάδας δεν θα αναμείνουν σε κάποιο φράγμα κατά το τέλος της εκτέλεσης και, συνεπώς, δεν θα συγχρονιστούν.
- **num_threads**(*πλήθος νημάτων*). Η *num_threads* χρησιμοποιείται για να ορίσει το πλήθος των νημάτων που θα εκτελέσουν μια περιοχή παραληλλίας και θα απαρτίσουν μια ομάδα.
- **schedule**(*τύπος[, μέγεθος κόκκου]*). Χρησιμοποιούμε την schedule σε συνδυασμό με την οδηγία *for* για να ορίσουμε τον τρόπο με τον οποίο θα διαμοιραστεί ο φόρτος ενός βρόχου επανάληψης, ανάλογα με τον *τύπο* της χρονοδρομολόγησης. Επιπλέον, προαιρετικά ορίζουμε το μέγεθος του κόκκου παραλληλίας, δηλαδή το πλήθος των επαναλήψεων που θα διαμοιράζεται σε κάθε νήμα τη φορά.

Ο *τύπος* μπορεί να είναι ένας εκ των τέσσερις παρακάτω:

- **static**. Ορίζει στατική χρονοδρομολόγηση για τον βρόχο επανάληψης. Ο βρόχος διασπάται σε τμήματα μεγέθους όσου και ο κόκκος παραλληλίας και κάθε τμήμα διαμοιράζεται κυκλικά στα νήματα της ομάδας. Όταν δεν έχει οριστεί μέγεθος κόκκου, τότε ο βρόχος διασπάται σε τόσα τμήματα, όσα και τα νήματα της ομάδας.
- **dynamic**. Ορίζει δυναμική χρονοδρομολόγηση για τον βρόχο επανάληψης, κατά την οποία ο βρόχος διασπάται με τρόπο όμοιο με αυτόν της στατικής χρονοδρομολόγησης, με την μόνη διαφορά ότι ο διαμοιρασμός των τμημάτων γίνεται δυναμικά σε κάθε νήμα. Αν το μέγεθος του κόκκου δεν είναι ορισμένο, τότε το μέγεθος των τμημάτων είναι ίσο με 1.
- **guided**. Στην χρονοδρομολόγηση τύπου *guided*, οι επαναλήψεις διαμοιράζονται κυκλικά στα νήματα, αρχικά σε τμήματα μεγέθους όσο και αυτό του κόκκου παραλληλλίας. Όταν ολοκληρώνεται μια ανάθεση επαναλήψεων σε ένα νήμα, γίνεται εκθετική μείωση στο μέγεθος του επόμενου τμήματος και ανατίθεται δυναμικά σε ένα νήμα. Η διαδικασία επαναλαμβάνεται έως ότου το μέγεθος γίνει ίσο με 1, ή ο βρόχος τερματιστεί.
- **runtime**. Στον συγκεκριμένο τύπο διαμοιρασμού, ο τρόπος χρονοδρομολόγησης ορίζεται από την τιμή της μεταβλητής περιβάλλοντος **OMP_SCHEDULE**, η οποία περιγράφεται στη συνέχεια.

- **reduction(***πράξη : λίστα μεταβλητών***)**. Με τη χρήση της φράσης reduction, επιτυγχάνεται αφαίρεση στα αντικείμενα της λίστας μεταβλητών, μέσω της *πράξης* που έχει δοθεί ως όρισμα. Ουσιαστικά, στο πέρας της περιοχής παραλληλίας, δημιουργείται ένα τοπικό αντίγραφο κάθε μεταβλητής της λίστας για το κάθε νήμα. Έπειτα, για κάθε μεταβλητή, πραγματοποιείται η *πράξη* (+, -, *, /) μεταξύ των αντίστοιχων αντιγράφων των νημάτων. Το αποτέλεσμα της πράξης αποθηκεύεται στην αντίστοιχη μεταβλητή του κυρίου νήματος και, έτσι, εξασφαλίζεται συνέπεια των δεδομένων.

## 2.4 Συσκευές OpenMP (Devices)

Τυπικά, ένα πρόγραμμα περικλείεται από μια υποννοούμενη περιοχή παραλληλίας, γονέα των υπόλοιπων περιοχών, στην οποία συμμετέχει μόνο το κύριο νήμα. Αναφερόμαστε στο κύριο σύστημα το οποίο εκτελεί την περιοχή αυτή ως *host device.*

Η κύρια επεξεργαστική μονάδα (host device) μπορεί ανά πάσα στιγμή να αποστείλει δεδομένα και κώδικα σε μια συσκευή, με τη χρήση οδηγιών OpenMP, ώστε να επιταχύνει την εκτέλεση του προγράμματος. Ειδικότερα, ο κώδικας που δίνεται σε μία συσκευή προς εκτέλεση μπορεί και ο ίδιος να περιέχει παραλληλισμό

εκπεφρασμένο μέσω οδηγιών OpenMP - επομένως μπορεί να γίνει παράλληλη εκτέλεση και εντός της συσκευής, εφόσον υποστηρίζεται.

Οι συσκευές προτιμούνται έναντι ενός συστήματος γενικής χρήσης λόγω της ταχύτητάς τους σε συγκεκριμένες πράξεις ή λόγω άλλων ιδιαιτεροτήτων τους. Ένα παράδειγμα συσκευής είναι μια κάρτα γραφικών, η οποία έχει την ιδιότητα να εκτελεί πράξεις κινητής υποδιαστολής γρήγορα και μαζικά, ή ένα εικονικό μηχάνημα το οποίο έχουμε παραμετροποιήσει με τα επιθυμητά χαρακτηριστικά.

Η υλοποίηση μιας συσκευής πραγματοποιείται με συγκεκριμένο κώδικα, κατά τον οποίο λαμβάνονται υπ'όψη οι ιδιότητες και οι περιορισμοί της, για να επιτευχθεί αποδοτική εκτέλεση. Συνήθως, ο συνολικός κώδικας αναπαριστάται ως βιβλιοθήκη. Η βιβλιοθήκη πρέπει να περιλαμβάνει κατ'ελάχιστον μια στοιχειώδη διεπαφή με το σύστημα φιλοξενίας, καθώς επίσης και την υλοποίηση του συνόλου των ρουτινών της συσκευής που της επιτρέπουν να εκτελεί οδηγίες OpenMP. Λεπτομερέστερη περιγραφή των τεχνικών υλοποίησης των συσκευών γίνεται στο Κεφάλαιο 3.

### 2.4.1 Περιοχές αποστολής κωδικα/δεδομένων (Target regions)

Όπως προαναφέρθηκε, η αποστολή δεδομένων και τμημάτων κώδικα από το κύριο σύστημα προς μία συσκευή, είναι μια συνήθης πρακτική που αποσκοπεί στην επιτάχυνση της εκτέλεσης μιας παράλληλης εφαρμογής.

Η βασική μέθοδος αποστολής κώδικα και δεδομένων σε μια συσκευή είναι η χρήση της οδηγίας **#pragma omp target**. Η σύνταξη της οδηγίας είναι η εξής:

**#pragma omp** target *[φράση[ [,] φράση] ... ]*
*δομημένο τμήμα*

Οι επιτρεπόμενες φράσεις που συνοδεύουν την οδηγία εξετάζονται σε επόμενο υποκεφάλαιο.

Το κύριο νήμα, όταν συναντά την οδηγία target, καλείται να αποστείλει το δομημένο τμήμα κώδικα στη συσκευή, το οποίο αρχικά αποθηκεύεται στην μνήμη του κυρίου νήματός της. Παράλληλα, δημιουργείται για ολόκληρη τη συσκευή ένα *περιβάλλον δεδομένων* (device data environment), το οποίο περιλαμβάνει όλα τα απαραίτητα δεδομένα που κληρονομήθηκαν από το κύριο πρόγραμμα. Αναφερόμαστε στο ενιαίο σύνολο του δομημένου τμήματος και του περιβάλλοντος δεδομένων ως *πυρήνα* (kernel).

Αρχικά, τα νήματα της συσκευής ενεργοποιούνται και όλα εκτός του κυρίου νήματος βρίσκονται σε αναμονή. Η συμπεριφορά του κυρίου νήματος της συσκευής ταυτίζεται με αυτή του κυρίου νήματος στο host device – δημιουργεί και χειρίζεται ομάδες νημάτων, στις οποίες συμμετέχει, ενώ επιπλέον εκτελεί κανονικά σειριακό κώδικα και οδηγίες OpenMP. Συνεπώς, όπως και στο σύστημα φιλοξενίας, τα υπόλοιπα νήματα της συσκευής ενεργοποιούνται όταν συναντάται μια περιοχή παραλληλίας εντός της οδηγίας target.

### 2.4.2 Περιοχές δημιουργίας περιβάλλοντος δεδομένων (Target data regions)

Ο προγραμματιστής συχνά χρειάζεται να εισάγει διαδοχικές περιοχές target στο πρόγραμμά του, η λογική του οποίου τις εμποδίζει να ενοποιηθούν. Όταν αυτές χρησιμοποιούν κοινές μεταβλητές, επιθυμούμε να αποστείλουμε στην συσκευή μόνο το κοινό τους σύνολο δεδομένων, παραλείποντας την εκτέλεση κώδικα. Γι'αυτό το σκοπό, το OpenMP παρέχει την οδηγία **#pragma omp target data**, η σύνταξή της οποίας είναι η εξής:

**#pragma omp** target data *[φράση[ [,] φράση] ... ]*
*δομημένο τμήμα*

*(Σημείωση: η λέξη κλειδί "data" δεν αποτελεί φράση της οδηγίας target.)*

Το κύριο νήμα, όταν συναντά την οδηγία target data, δημιουργεί ένα νέο περιβάλλον δεδομένων στη συσκευή, το οποίο θα κληρονομηθεί από όλα τα περιβάλλοντα δεδομένων των περιοχών target που είναι τοποθετημένα στο εσωτερικό του *δομημένου τμήματος*. Ορίζοντας περιοχές target data στο εσωτερικό του προγράμματός μας, μπορούμε να ελαττώσουμε σημαντικά τον αριθμό των αναγνώσεων/εγγραφών μνήμης που απαιτούνται για την επανειλημμένη δημιουργία περιβάλλοντος δεδομένων.

### 2.4.3 Φράσεις Οδηγιών target/target data

Το πρότυπο OpenMP προσφέρει τη δυνατότητα στον προγραμματιστή να καθορίσει μέσω των φράσεων ποια συσκευή θα χρησιμοποιηθεί, πότε και πώς θα ενημερωθούν τα δεδομένα του περιβάλλοντος της συσκευής, καθώς επίσης και σε ποια κατεύθυνση (προς/από το κύριο πρόγραμμα). Παρακάτω γίνεται μια αναφορά στις επιτρεπτές φράσεις περιοχών target και πού αυτές χρησιμοποιούνται:

- **if(***συνθήκη***)**. Η συμπεριφορά της φράσης if περιγράφηκε στο υποκεφάλαιο 2.3.6. (target/target data)
- **device(***έκφραση ακεραίων***)**. Ορίζει την συσκευή που θα γίνει η αποστολή δεδομένων ή/και κώδικα (target/target data).
- **map(***[τύπος* **:** *] λίστα μεταβλητών***)**. Η φράση map χρησιμοποιείται για να ορίσει τον τρόπο με τον οποίο θα γίνει αντιστοίχιση των αντικειμένων της λίστας μεταβλητών του περιβάλλοντος δεδομένων με το περιβάλλον της αμέσως πιο εξωτερικής περιοχής. Οι βασικότεροι τύποι που χρησιμοποιούνται στην πράξη είναι οι παρακάτω:
  - **to**. Όταν ο τύπος αντιστοίχισης είναι *to*, στο κάθε αντικείμενο της λίστας που θα δημιουργηθεί στο περιβάλλον δεδομένων, ανατίθεται η τιμή του αρχικού αντικειμένου.
  - **from**. Όταν ο τύπος αντιστοίχισης είναι *from*, κατά την έξοδο από την περιοχή target [data], θα ενημερωθεί το κάθε αρχικό αντικείμενο της λίστας με την αντίστοιχη τιμή του αντικειμένου από το περιβάλλον δεδομένων.

- **tofrom** (target/target data). Όταν ο τύπος αντιστοίχισης είναι *tofrom*, ισχύει ο συνδυασμός των περιπτώσεων *to* και *from*. Το κάθε νέο αντικείμενο της λίστας μεταβλητών αρχικά παίρνει την τιμή του αρχικού αντικειμένου, στο οποίο, κατά την έξοδο, περνά την τιμή του.
- **alloc**. Με τον τύπο alloc, η τιμή του κάθε νέου αντικειμένου της λίστας μεταβλητών έχει ακαθόριστη τιμή.

## 2.5 Νέες δυνατότητες του OpenMP v4.5

Το πρότυπο OpenMP αναπτύσσεται συνεχώς, με σκοπό να προσφέρει στον προγραμματιστή όλο και περισσότερες δυνατότητες και να συμβαδίσει με την εξέλιξη των παράλληλων συστημάτων. Στην έκδοση 4.5 εισήχθησαν νέες έννοιες που αφορούν κυρίως την αποστολή σε συσκευές, αλλά και τον χειρισμό τμημάτων πίνακα:

- Υποστήριξη νέων τύπων αντιστοίχισης (map):
  - **release**.
  - **delete**.
- Υποστήριξη νέων φράσεων για τις οδηγίες target/target data:
  - (target) **private(***λίστα μεταβλητών***)**
    (target) **firstprivate(***λίστα μεταβλητών***)**.
    Η συμπεριφορά των οδηγιών private και firstprivate είναι όμοια με αυτή που περιγράφηκε σε προηγούμενο υποκεφάλαιο, με την διαφορά ότι στην προκειμένη περίπτωση η λίστα μεταβλητών αφορά το περιβάλλον δεδομένων της περιοχής target.
  - (target data) **use_device_ptr(***λίστα***)**. Τα αντικείμενα της λίστας μεταβλητών μετατρέπονται σε δείκτες συσκευής, οι οποίοι δείχνουν στα αντίστοιχα αντικείμενα λίστας του περιβάλλοντος δεδομένων.
  - (target) **is_device_ptr(***λίστα***)**. Χρησιμοποιείται για να δηλώσει ότι τα αντικείμενα της λίστας μεταβλητών είναι ήδη υπάρχοντες δείκτες στο περιβάλλον δεδομένων της συσκευής.
  - (target) **defaultmap(tofrom:scalar)**. Τοποθετείται για να ορίσει την αντιστοίχιση *tofrom* για μεταβλητές που έχουν ακέραιο τύπο, ή είναι δείκτες.
  - (target) **nowait**. Η φράση nowait επεξηγήθηκε στο υποκεφάλαιο 2.3.6.
  - (target) **depend(***τύπος : λίστα***)**.
- Νέες οδηγίες συσκευών:
  - **#pragma omp** target enter data *[φράση[ [,] φράση] ... ]*
    Η target enter data είναι μια αυτόνομη οδηγία, η οποία χρησιμοποιείται για να δηλώσει ότι εκείνη την στιγμή ορισμένα δεδομένα πρέπει να εισέλθουν στο περιβάλλον δεδομένων της συσκευής. Συνοδεύεται από τουλάχιστον μία φράση *map* με τύπους *to* ή *alloc.*

- **#pragma omp** target exit data *[φράση[ [,] φράση] ... ]*
  Η target exit data είναι μια αυτόνομη οδηγία, η οποία συνοδεύεται από τουλάχιστον μία φράση *map* με τύπους *from* ή *delete* και χρησιμοποιείται για να δηλώσει ότι εκείνη την στιγμή ορισμένα αντικείμενα πρέπει να εξέλθουν από το περιβάλλον δεδομένων της συσκευής.

# Κεφάλαιο 3

# Ο μεταφραστής OMPi

## 3.1 Δομή του OMPi

Ο μεταφραστής OMPi είναι ένας παραλληλοποιητικός μεταφραστής γλώσσας C για το πρότυπο OpenMP. Η βασική λογική του είναι διαχωρισμένη σε δύο βασικά τμήματα, αυτό του μεταγλωττιστή (compiler) και αυτό της υποστήριξης εκτέλεσης (runtime). Το πρώτο, χρησιμοποιείται για την μετατροπή του πηγαίου κώδικα, που περιλαμβάνει οδηγίες OpenMP, σε πολυνηματικό κώδικα C. Στον παραγόμενο κώδικα όλες οι οδηγίες αντικαθιστώνται από κλήσεις σε ρουτίνες του τμήματος υποστήριξης εκτέλεσης.

Ο πολυνηματικός κώδικας μπορεί να χρησιμοποιεί διαφορετικό είδος νημάτων για παράλληλη εκτέλεση. Αξίζει να σημειωθεί ότι οι οντότητες εκτέλεσης θεωρούνται αφηρημένες, μέχρι τη στιγμή που το σύστημα υποστήριξης εκτέλεσης θα καθορίσει το είδος τους, μέσω των προσφερόμενων βιβλιοθηκών υποστήριξης εκτέλεσης.

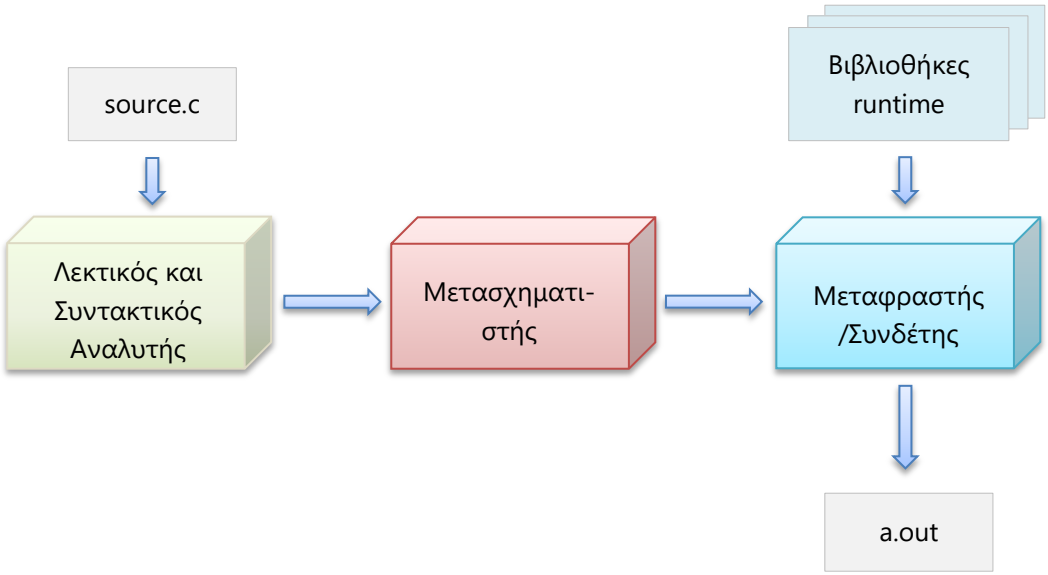
## 3.2 Μετασχηματισμός πηγαίου σε πηγαίο κώδικα (Source-to-source)

Όπως αναφέρθηκε, ο OMPi βασίζεται στον μετασχηματισμό ενός πηγαίου κώδικα γλώσσας προγραμματισμού C σε έναν βελτιστοποιημένο πολυνηματικό κώδικα C. Το αρχικό βήμα της μεταγλώττισης περιλαμβάνει την διαδικασία της συντακτικής ανάλυσης, η οποία είναι κοινή μεταξύ όλων των μεταγλωττιστών. Η συντακτική ανάλυση περιλαμβάνει σε πρώτο στάδιο τη λεκτική ανάλυση του κώδικα και, έπειτα, τη διάσχισή του ώστε να δημιουργηθεί το συντακτικό δέντρο.

Αφού παραχθεί το συντακτικό δέντρο, τότε αυτό διασχίζεται μία φορά ώστε να αντικατασταθεί κάθε οδηγία OpenMP με τις αντίστοιχες κλήσεις του συστήματος υποστήριξης εκτέλεσης. Το αποτέλεσμα των αντικαταστάσεων είναι ένας πηγαίος

κώδικας σε C, ο οποίος βασίζεται μόνο σε βιβλιοθήκες υποστήριξης εκτέλεσης του μεταφραστή.

Το τελευταίο βήμα της διαδικασίας της μεταγλώττισης είναι η μετάφραση του παραγόμενου πηγαίου κώδικα C και η σύνδεση των απαραίτητων βιβλιοθηκών υποστήριξης εκτέλεσης με αυτόν, ώστε να παραχθεί το εκτελέσιμο αρχείο. Η διαδικασία απεικονίζεται στο Σχήμα 3.2.




Σχήμα 3.1: Η διαδικασία της μετάφρασης στον OMPi

## 3.3 Το σύστημα υποστήριξης εκτέλεσης OMPi runtime

Το σύστημα υποστήριξης εκτέλεσης του OMPi είναι περαιτέρω χωρισμένο στις ενότητες *host* και *devices*. Η πρώτη ενότητα αφορά στην υποστήριξη της εκτέλεσης πολυνηματικού κώδικα στο κύριο σύστημα (host). Η ενότητα αυτή περιλαμβάνει το συντονισμό των νημάτων εκτέλεσης, την υλοποίηση των οδηγιών OpenMP με τις φράσεις τους, τον χειρισμό μεταβλητών περιβάλλοντος, καθώς επίσης και τις ρουτίνες βιβλιοθήκης υποστήριξης εκτέλεσης όπως ορίζονται στις προδιαγραφές του προτύπου. Επιπλέον, υποστηρίζεται η επικοινωνία του host με τις συσκευές, όπως αυτή περιγράφηκε στο προηγούμενο κεφάλαιο.

Οι οντότητες εκτέλεσης συντονίζονται από το σύστημα υποστήριξης εκτέλεσης (ORT), το οποίο στηρίζεται σε αυτές, χωρίς, όμως, να είναι υπεύθυνο για την παραγωγή τους. Συγκεκριμένα, η παραγωγή οντοτήτων εκτέλεσης είναι μια διαδικασία που υλοποιείται σε ξεχωριστές βιβλιοθήκες, τις λεγόμενες Βιβλιοθήκες

Οντοτήτων Εκτέλεσης (**E**xecution **E**ntity **Lib**rarie**s** – EELIBS). Η αρχιτεκτονική αυτή του συστήματος υποστήριξης εκτέλεσης του OMPi είναι μοναδική και modular. Επιτρέπει τη διαφανή χρήση διαφορετικών βιλιοθηκών οντοτήτων, ανάλογα με τις ανάγκες του προγραμματιστή, οι οποίες μπορεί, για παράδειγμα, να βασίζονται σε διεργασίες ή νήματα. Το Σχήμα 3.2 διευκολύνει στην κατανόηση της δομής του OMPi.

Η ενότητα *devices* περιλαμβάνει τις βιβλιοθήκες υποστήριξης εκτέλεσης για τις συσκευές, όπως αυτές ορίστηκαν προηγουμένως, στο Κεφάλαιο 2. Κάθε συσκευή διαθέτει τη δική της βιβλιοθήκη, η οποία επιτρέπει την εκτέλεση πολυνηματικού κώδικα στις οντότητές της. Επιπλέον, για κάθε συσκευή ορίζεται η διεπαφή *hostpart*, μέσω της οποίας πραγματοποιείται η επικοινωνία μεταξύ του host και των συσκευών. Εκτενέστερη περιγραφή των συστατικών των συσκευών γίνεται σε παρακάτω υποκεφάλαιο.


Σχήμα 3.2: Η δομή του OMPi runtime για τον host

### 3.3.1 Είσοδος σε περιοχή παραλληλίας

Κατά τη διαδικασία του μετασχηματισμού, οι οδηγίες **#pragma omp parallel** του αρχικού πηγαίου κώδικα μετατρέπονται σε κλήσεις *ort_execute_parallel()*. Η βιβλιοθήκη από την οποία θα κληθεί η ρουτίνα ορίζεται από το σημείο εμφάνισης της οδηγίας. Για παράδειγμα, αν αυτή εμφανίζεται μέσα σε μια περιοχή target η οποία αφορά τη συσκευή Α, τότε η βιβλιοθήκη που θα χρησιμοποιηθεί θα είναι αυτή της υποστήριξης εκτέλεσης της Α.

Ο Κώδικας 3.3 αποτελεί το αποτέλεσμα του μετασχηματισμού του Κώδικα 2.1, ο οποίος περιέχει μία περιοχή παραλληλίας. Στο συγκεκριμένο παράδειγμα η εκτέλεση λαμβάνει χώρα μόνο στην μεριά του host:

source_2_1_ompi.c

```
1  int __original_main(int _argc_ignored, char ** _argv_ignored)
2  # 5 "test1.c"
3  {
4     int thr_id;
5
6       omp_set_num_threads(8);
7       /* (l9) #pragma omp parallel
8       */
9       {
10       struct __shvt__ {
11          int (* thr_id);
12       } _shvars;
13
14       /* byref variables */
15       _shvars.thr_id = &thr_id;
16       ort_execute_parallel(_thrFunc0_, (void *) &_shvars, -1, 0, 0);
17      }
18 }
```

Κώδικας 3.3: Μετασχηματισμός του Κώδικα 2.1

Παρατηρούμε ότι έχουν εισαχθεί τα απαραίτητα σχόλια που υποδεικνύουν την γραμμή στην οποία εμφανίζεται η οδηγία parallel στο αρχικό πρόγραμμα. Επιπλέον, το πρώτο όρισμα της *ort_execute_parallel()*, η συνάρτηση *_thrFunc0_*, αποτελεί το δομημένο τμήμα κώδικα της οδηγίας, ενώ στην δομή _shvars τοποθετούνται οι κοινόχρηστες μεταβλητές της ομάδας που θα εκτελέσει την περιοχή παραλληλίας. Αξίζει να σημειωθεί ότι η μεταβλητή *thr_id* θεωρείται κοινόχρηστη, διότι δεν δηλώθηκε διαφορετικά σε κάποια φράση της οδηγίας.

Η ελάχιστη διαδικασία που ακολουθείται κατά την εκτέλεση της συνάρτησης `ort_execute_parallel()`, περιγράφεται παρακάτω, στην περίπτωση της συσκευής Epiphany.

### 3.3.2 Υποστήριξη συσκευών OpenMP

Ο μεταφραστής OMPi υποστηρίζει την αποστολή κώδικα και δεδομένων σε συσκευές μέσω οδηγιών target. Στην πράξη, η ενέργεια αυτή πραγματοποιείται μέσω μιας καλά ορισμένης διεπαφής μεταξύ host και συσκευών, η οποία διαφέρει ανά εγκατεστημένη συσκευή. Σύντομη περιγραφή της διεπαφής γίνεται στο υποκεφάλαιο 3.3.3.

Πριν την αποστολή του κώδικα, αυτός πρώτα μετασχηματίζεται, προκειμένου να πραγματοποιεί κλήσεις βιβλιοθήκης υποστήριξης εκτέλεσης για την συσκευή και έπειτα εξάγεται σε ένα ξεχωριστό αρχείο, ώστε να μεταγλωττιστεί. Στο εσωτερικό μιας τέτοιας βιβλιοθήκης βρίσκονται υλοποιήσεις όλων των οδηγιών OpenMP που μπορούν να υποστηριχθούν από αυτή. Το σύνολο των υλοποιήσεων που θα αντικαταστήσουν τις οδηγίες ορίζεται ως *devpart* και διαφέρει ανά συσκευή.

Με το πέρας της παραπάνω διαδικασίας, έχει δημιουργηθεί ένα αρχείο πηγαίου κώδικα για κάθε περιοχή target. Για τον διαχωρισμό των αρχείων αυτών, εισάγεται στην ονομασία τους το αναγνωριστικό της περιοχής που αυτά αφορούν. Οι παραγόμενοι πηγαίοι κώδικες πυρήνα μεταγλωττίζονται σε εκτελέσιμα αρχεία, τους *πυρήνες* (kernels), των οποίων η μορφή διαφέρει για την εκάστοτε συσκευή. Ο πυρήνας αποτελεί το πλήρες εκτελέσιμο της περιοχής target που θα αποσταλλεί στη συσκευή.

Ο τρόπος με τον οποίο θα μεταγλωττιστεί ένας πηγαίος κώδικας πυρήνα καθορίζεται από ένα ειδικό Makefile που πραγματοποιεί την σύνδεση με μια συγκεκριμένη βιβλιοθήκη συσκευής. Είναι απαραίτητο το πλήθος των Makefiles να ισούται με το πλήθος των εγκατεστημένων βιβλιοθηκών συσκευής. Συνολικά, αν το πλήθος των περιοχών ειναι X και το πλήθος των συσκευών Υ, παράγονται X*Υ εκτελέσιμα αρχεία.

### 3.3.3 Η διεπαφή host-συσκευών hostpart

Αναπόσπαστο κομμάτι του χειρισμού συσκευών, είναι η διεπαφή *hostpart*, μέσω της οποίας επικοινωνεί ο host με την κάθε συσκευή. Η διεπαφή ορίζεται από την ομώνυμη βιβλιοθήκη, η οποία υλοποείται ανά συσκευή, λόγω των διαφορετικών χαρακτηριστικών που αυτές διαθέτουν.

Η συσκευή host, όταν συναντήσει οδηγία target, κάνει χρήση της διεπαφής για να προετοιμάσει τη συσκευή, αρχικοποιώντας την και ορίζοντας κοινόχρηστες δομές που διευκολύνουν την επικοινωνία. Τελικά, αποστέλλει κώδικα ή/και δεδομένα, τα οποία αποτελούν το αρχείο πυρήνα και τερματίζει τη συσκευή στο πέρας της

περιοχής target. Παρακάτω περιγράφεται το σύνολο των ρουτινών που ορίζονται από τη διεπαφή hostpart:

- **hm_initialize()**. Η συνάρτηση αυτή εκτελεί όλες τις απαραίτητες ενέργειες για να εδραιωθεί η επικοινωνία του host με την συσκευή. Τέτοιες ενέργειες συνήθως αφορούν την αρχικοποίηση των πυρήνων της συσκευής, των κλειδαριών που πρόκειται να χρησιμοποιηθούν, αλλά και των μεταβλητών ελέγχου. Επιπλέον, δεσμεύεται ο κοινόχρηστος χώρος αποθήκευσης των δεδομένων,
- **hm_finalize()**. Η *hm_finalize()* τερματίζει με ασφαλή τρόπο την επικοινωνία του host με την συσκευή. Αυτό επιτυγχάνεται με τον τερματισμό των οντοτήτων της συσκευής, καθώς επίσης και την αποδέσμευση του κοινόχρηστου χώρου μνήμης που αφορά τις επικοινωνίες.
- **hm_offload()**. Από τις πιο βασικές συναρτήσεις, η *hm_offload()* χρησιμοποιείται για να εκτελεστεί από την μεριά του host το αρχείο πυρήνα (kernel file) και, έπειτα, να αποστείλει το αρχείο αυτό στην συσκευή. Στο εσωτερικό της συνάρτησης λαμβάνει χώρα και η επικοινωνία με την συσκευή για την διαχείριση των ομάδων νημάτων που εκτελούν τα αρχεία πυρήνα.
- **hm_dev_alloc()**. Η συνάρτηση αυτή χρησιμοποιείται για να δεσμεύσει μνήμη ορισμένου μεγέθους στην συσκευή. Εκτός από τις περιοχές target, χρησιμοποιείται και στην περίπτωση της οδηγίας target data, όπου απλώς χρειάζεται να δημιουργηθεί ένα περιβάλλον δεδομένων στη συσκευή.
- **hm_dev_free()**. Με την *hm_dev_free()* αποδεσμεύεται η μνήμη η οποία δεσμεύτηκε από την *hm_dev_alloc()*.
- **hm_todev()**. Η μεταφορά (εγγραφή) δεδομένων από τον host στην συσκευή γίνεται, εφ'όσον έχει προηγηθεί δέσμευση μνήμης στην δεύτερη, μέσω της συνάρτησης hm_todev().
- **hm_fromdev()**. Μέσω της συνάρτησης *hm_fromdev()* πραγματοποιείται η ανάγνωση δεδομένων του host από τη συσκευή.
- **hm_imed2umed_addr()**. Η συγκεκριμένη συνάρτηση χρησιμοποιείται για να μετατρέψει μια εσωτερική ενδιάμεση διεύθυνση της συσκευής, σε μια χρησιμοποιήσιμη από τον host ενδιάμεση διεύθυνση.
- **hm_umed2imed_addr()**. Λειτουργώντας αντίστροφα από την *hm_imed2umed_addr()*, η *hm_umed2imed_addr()* μετατρέπει μια χρησιμοποιήσιμη ενδιάμεση διεύθυνση σε εσωτερική ενδιάμεση διεύθυνση της συσκευής.

## 3.4 Ο επιταχυντής Epiphany

Ένα παράδειγμα συσκευής OpenMP που υποστηρίζεται από τον μεταφραστή OMPi είναι ο επιταχυντής (accelerator) Epiphany-III. Σε αυτήν την ενότητα γίνεται αρχικά μια μικρή αναφορά στην αρχιτεκτονική της πλατφόρμας Parallella-16. Έπειτα, περιγράφεται η δομή των βιβλιοθηκών υποστήριξης εκτέλεσης στο εσωτερικό του

μεταφραστή OMPi, καθώς επίσης και ο τρόπος με τον οποίο επικοινωνούν τα δύο επίπεδα της πλατφόρμας.

### 3.4.1 Επισκόπηση της Parallella-16

Η πλατφόρμα Parallella-16 [3] είναι ένας μινι-υπολογιστής 18 πυρήνων και μνήμης 1 GiB, με δύο επίπεδα επεξεργασίας. Στο πρώτο επίπεδο βρίσκεται ο διπύρηνος επεξεργαστής ARMv7l (host), ο οποίος είναι υλοποιημένος μέσα στο FPGA Zynq 7010 ή 7020, διαθέτοντας 32 KB μνήμης cache L1 και 512 KB κοινόχρηστης μνήμης cache L2. Στον επεξεργαστή ARM εκτελείται το λειτουργικό σύστημα Linux, ακριβώς όπως και σε έναν προσωπικό υπολογιστή. Στο δεύτερο επίπεδο διατίθεται ο συνεπεξεργαστής Epiphany με μόλις 32 KB τοπικής μνήμης ανά πυρήνα, ο οποίος δεν εκτελεί κάποιο λειτουργικό σύστημα και διαθέτει μνήμη χωρίς προστασία.

Η αρχιτεκτονική του Epiphany είναι σχεδιασμένη βάσει ενός πλέγματος μεγέθους 64 x 64, συνεπώς θεωρητικά μπορούν να συνδυαστούν κυκλώματα 16 και 64 πυρήνων Epiphany (III & IV, αντίστοιχα), τα λεγόμενα eCores, ώστε να σχηματιστεί ένα δίκτυο διασύνδεσης 4096 κόμβων. Στην πράξη, στην πλατφόρμα Parallella-16 γίνεται χρήση μόνο ενός υποπλέγματος 4 x 4 του εικονικού πλέγματος 64 x 64, όπου κάθε κόμβος είναι και ένας πυρήνας eCore.

Το υλικό του επιταχυντή Epiphany διαθέτει 4 συνδέσμους eLinks (ανατολικά, δυτικά, βόρεια και νότια) που του επιτρέπουν να συνδέεται με άλλα κυκλώματα. Πρακτικά, στη συγκεκριμένη πλακέτα γίνεται χρήση μόνο του ανατολικού συνδέσμου για την διασύνδεση με τον host επεξεργαστή.

### 3.4.2 Η βιβλιοθήκη υποστήριξης εκτέλεσης για το Epiphany-III

Ο επεξεργαστής Zynq στο πρώτο επίπεδο της Parallella-16 θεωρείται γενικού σκοπού και συνεπώς επιτρέπει τη χρήση βιβλιοθηκών υποστήριξης εκτέλεσης του μεταφραστή OMPi στην πλήρη μορφή τους (ενότητα 3.3). Επιπλέον, έχει την δυνατότητα να επικοινωνεί με τον συνεπεξεργαστή Epiphany, μέσω του δικτύου διασύνδεσης. Όπως αναφέρθηκε προηγουμένως, ο Epiphany-III διαθέτει περιορισμένη μνήμη. Ως αποτέλεσμα, για την υποστήριξη του προτύπου OpenMP από τα eCores, έχει υλοποιηθεί μια «μικρογραφία» της βιβλιοθήκης υποστήριξης εκτέλεσης του host, όσον αφορά το μέγεθος, η οποία παρ'όλα αυτά αξιοποιεί τις ιδιαιτερότητές τους.

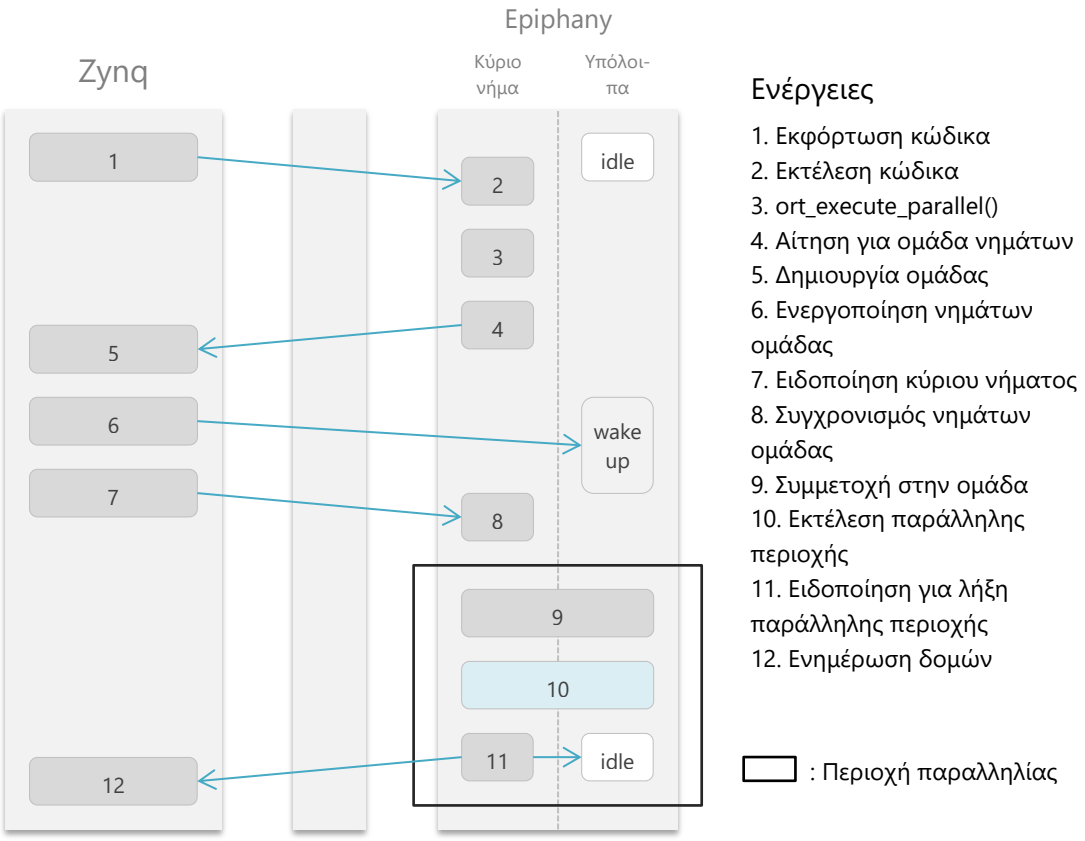
Η βιβλιοθήκη αυτή κάνει χρήση της έτοιμης βιβλιοθήκης ανάπτυξης λογισμικού eSDK που προσφέρεται από την πλατφόρμα. Το eSDK περιέχει υλοποιήσεις συναρτήσεων για:

- αρχικοποίηση και τερματισμό των πυρήνων eCores (`e_open()` και `e_close()`),
- φόρτωση και εκτέλεση αρχείων πυρήνα στα eCores (`e_load()` και `e_start()`),
- ομαδοποιημένη φόρτωση πυρήνων σε eCores (`e_load_group()`),

- ανάγνωση και εγγραφή του host από και πρός την μνήμη των eCores, αντίστοιχα (*e_read()* και *e_write()*).

Το κύριο πρόγραμμα εκτελείται από τον Zynq, ο οποίος κάνει χρήση της διεπαφής hostpart για να επικοινωνήσει με το Epiphany-III. Μια τυπική διαδικασία που ακολουθείται για την εκτέλεση κώδικα στους πυρήνες, όταν συναντάται οδηγία target, είναι η εξής:

- κλήση της *hm_initialize()*, η οποία αρχικοποιεί τα eCores με κλήσεις *e_open()*
- αρχικοποίηση της κοινόχρηστης μνήμης καλώντας τη συνάρτηση *e_write()*
- κλήση των *hm_dev_alloc()* και *hm_to_dev()* για την δημιουργία περιβάλλοντος δεδομένων στην συσκευή
- φόρτωση του κώδικα πυρήνα μέσω της *hm_offload()* – η συγκεκριμένη συνάρτηση περιλαμβάνει κλήσεις *e_load()/e_load_group()*, κλήσεις εγγραφής και ανάγνωσης *e_write()/e_read()*, καθώς και *e_start()*. Επιπλέον, εκτελεί έναν επαναληπτικό βρόχο για να ικανοποιεί τα αιτήματα της συσκευής έως ότου αυτή τερματίσει.
- κλήση της συνάρτησης *hm_fromdev()* για την ανάγνωση του αποτελέσματος που υπολογίστηκε στα eCores (κλήση *e_read()*)
- αποδέσμευση των eCores με χρήση της *hm_finalize()* μέσω κλήσεων *e_close()*




Σχήμα 3.4: Περιοχή target με παράλληλη περιοχή στην Parallella-16

Στο Σχήμα 3.4 απεικονίζεται η ακολουθία ενεργειών κατά την εκτέλεση μιας περιοχής target με μία περιοχή παραλληλίας, στην πλατφόρμα Parallella-16. Η Parallella-16 χρησιμοποιήθηκε για τους σκοπούς της διπλωματικής εργασίας, διότι αποτελεί παράδειγμα συσκευής που επιδέχεται βελτιώσεων, ώστε να αξιοποιηθούν περαιτέρω οι δυνατότητές της. Ιδιαίτερο ενδιαφέρον παρουσιάζει ο περιορισμός των eCores σε μνήμη, γεγονός που αποτέλεσε κίνητρο για την επιλογή της εργασίας αυτής.

# Κεφάλαιο 4

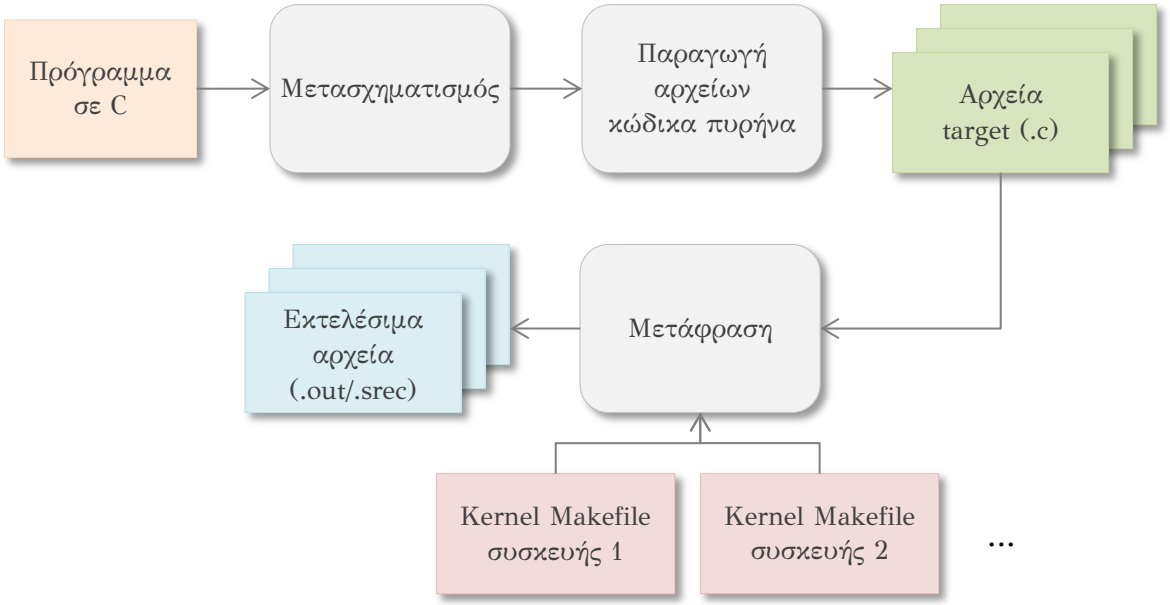
# Προσαρμοζόμενη υποστήριξη εκτέλεσης για συσκευές

## 4.1 Προσαρμοζόμενη υποστήριξη εκτέλεσης

Η σύνδεση βιβλιοθηκών υποστήριξης εκτέλεσης είναι το τελευταίο στάδιο της μεταγλώττισης ενός προγράμματος. Το πρόγραμμα μεταφράζεται, και, έπειτα, εντοπίζονται οι γενικές βιβλιοθήκες που απαιτούνται, οι οποίες και εισάγονται στο τελικό εκτελέσιμο. Ανάμεσα στις βιβλιοθήκες αυτές, βρίσκεται και η βιβλιοθήκη host, η οποία υποστηρίζει την εκτέλεση στο κύριο σύστημα. Όταν πρόκειται για συστήματα τα οποία δεν έχουν εμφανείς περιορισμούς στη μνήμη (π.χ. προσωπικοί υπολογιστές), ή στην χωρητικότητα δίσκου (αναλογικά με το μέγεθος μιας εφαρμογής C), το μέγεθος της βιβλιοθήκης host (π.χ. σε γραμμές κώδικα) επηρεάζει ελάχιστα επιπτώσεις την συνολική επίδοση του παράλληλου προγράμματος.

Καθώς, όμως, οι συσκευές ξεκίνησαν να εισέρχονται ως έννοια στον παράλληλο προγραμματισμό με OpenMP, ήταν απαραίτητη η συγγραφή ειδικά σχεδιασμένων βιβλιοθηκών υποστήριξης εκτέλεσης, ανά κάθε εγκατεστημένη συσκευή. Οι βιβλιοθήκες αυτές συνεργάζονται με τη βιβλιοθήκη host, ώστε να υποστηρίξουν την εκτέλεση στις συσκευές. Από προεπιλογή, ο OMPi χρησιμοποιεί την πλήρη βιβλιοθήκη για την ενέργεια αυτή. Η πλήρης βιβλιοθήκη συσκευής περιέχει την υλοποίηση όλων των οδηγιών OpenMP και συνεπώς μπορεί να είναι και αυτή αρκετά μεγάλη σε μέγεθος, συγκριτικά με την μνήμη της συσκευής (για παράδειγμα, 32KB μνήμη στο Epiphany-III).

Στο Σχήμα 4.1 παρουσιάζεται η τρέχουσα διαδικασία υποστήριξης εκτέλεσης σε συσκευές.




Σχήμα 4.1: Αρχική υποστήριξη εκτέλεσης στον OMPi (ανά συσκευή)

Η ιδέα που αποτέλεσε κίνητρο για την συγγραφή της διπλωματικής εργασίας, είναι ότι με την εκ των προτέρων γνώση κάποιων μετρικών που αφορούν τις περιοχές target, μπορεί κάποιος να καταλάβει ότι δεν υπάρχει ανάγκη πλήρους υποστήριξης OpenMP για αυτές. Εναλλακτικά, μπορούν να χρησιμοποιούνται πιο «ελαφριές» βιβλιοθήκες ανά περίπτωση target, οι λεγόμενες *εκδοχές* (flavors) βιβλιοθήκης, οι οποίες παραλείπουν κάποιες υλοποιήσεις οδηγιών OpenMP. Επιλέγοντας κάθε φορά την κατάλληλη βιβλιοθήκη, υποστηρίζεται στο μέγιστο η εκτέλεση μιας περιοχής target και ταυτόχρονα εξοικονομούνται πόροι της συσκευής. Αυτή είναι η λογική πίσω από το C.A.R.S. (Compiler-assisted Adaptive Runtime System) [4]. Προς το παρόν, ο μεταφραστής OMPi συλλέγει κάποιες μετρικές που αφορούν τις περιοχές target. Περιγραφή της ανάλυσης περιοχών target γίνεται στην υποενότητα 4.1.1, ενώ ο τρόπος με τον οποίο αυτοματοποιήσαμε την διαδικασία περιγράφεται στην υποενότητα 4.1.2.

### 4.1.1 Ανάλυση περιοχών target

Ο μηχανισμός της προσαρμοζόμενης υποστήριξης εκτέλεσης ξεκινά από τη διαδικασία της συντακτικής ανάλυσης. Αρχικά, ο κώδικας των πυρήνων (kernels) είναι τοποθετημένος λεκτικά στο ενιαίο αρχείο του προγράμματος. Επομένως, ο μεταφραστής OMPi μπορεί να αναλύσει την καθεμία ξεχωριστά και να εξάγει κάποια στατιστικά που αφορούν το πλήθος εμφανίσεων κάθε οδηγίας OpenMP, καθώς επίσης και άλλες ιδιότητες τους. Από τα εξαγόμενα χαρακτηριστικά, μπορούμε να συμπεράνουμε ποιες λειτουργικότητες του OpenMP είναι απαραίτητες

για την εκτέλεση ενός πυρήνα και ποιες όχι, εντοπίζοντας με αυτόν τον τρόπο την ιδανική εκδοχή βιβλιοθήκης.

Η ανάλυση μιας περιοχής target γίνεται με την επίσκεψη όλων των συναρτήσεων που καλούνται στο εσωτερικό της και την αναγνώριση των οδηγιών και κλήσεων OpenMP που αυτές περιλαμβάνουν. Όταν κάθε κώδικας πυρήνα μετασχηματιστεί και εξαχθεί σε ξεχωριστό αρχείο, ο OMPi θα τοποθετήσει τις μετρικές που υπολογίστηκαν στην αρχή του κάθε αρχείου, μέσα σε σχόλια. Η ενέργεια αυτή πραγματοποιείται μόνο όταν δοθεί το ειδικό όρισμα `--carstats` σε συνδυασμό με το όρισμα `-k`, ώστε να διατηρηθούν όλα τα ενδιάμεσα αρχεία.

Οι μετρικές που εξάγει ο μεταφραστής OMPi έχουν την εξής μορφή:

*όνομα μετρικής 1 = τιμή μετρικής 1*

*όνομα μετρικής 2 = τιμή μετρικής 3*

...

Το σύνολο των μετρικών που εξήγαγε ο OMPi στην αρχική μορφή του [4] [5], είναι το παρακάτω (η παρένθεση περιέχει το όνομα της μετρικής):

- *Περιοχές παραλληλίας* (“parallel”). Το πλήθος των παράλληλων περιοχών είναι απαραίτητο για να γνωρίζουμε εάν θα δημιουργηθούν ομάδες νημάτων ή όχι, στο εσωτερικό του πυρήνα.
- *Μέγιστο επίπεδο παραλληλίας* (“max_par_lev”). Γνωρίζοντας το μέγιστο επίπεδο παραλληλίας, μπορούμε να παραλείψουμε τις υλοποιήσεις εμφωλευμένου παραλληλισμού, ο οποίος εισάγει σημαντικό κόστος εκτέλεσης.
- *Περιοχές διαμοιρασμού* (“single”, “for”, “sections”). Η υλοποίηση οδηγιών που αφορούν περιοχές διαμοιρασμού (single, for, sections) αποτελεί ένα μεγάλο κομμάτι της βιβλιοθήκης συσκευής. Γνωρίζοντας ποιες από αυτές χρησιμοποιούνται, μπορούμε να επιλέξουμε εκδοχή με τους κατάλληλους συνδυασμούς τους.
- *Κρίσιμες περιοχές* (“uncritical”, “criticals”). Ισούται με το πλήθος οδηγιών *critical* χωρίς ονομασία.
- *Πλήθος tasks* (“tasks”). Ο μηχανισμός για την υποστήριξη tasking κατά την εκτέλεση έχει μεγάλες απαιτήσεις σε μνήμη. Γνωρίζοντας ότι δεν χρησιμοποιούνται tasks σε μια περιοχή target, μπορούμε να αποφύγουμε αχρείαστες επιβαρύνσεις της μνήμης της συσκευής.
- *Χρήση δείκτη σε συνάρτηση* (“hasfuncptr”). Στη περίπτωση της χρήσης δείκτη σε συνάρτηση, δεν είναι δυνατή η εξέταση αυτής, μιας και στον χρόνο μετάφρασης η διεύθυνση του δείκτη δεν είναι γνωστή.
- *Χρήση αναδρομής* (“hasrecursion”). Η κλήση σε αναδρομική συνάρτηση στην περιοχή target, μας εμποδίζει από το να γνωρίζουμε το μέγιστο επίπεδο παραλληλίας που υποστηρίζεται.

### 4.1.2 Υποστήριξη νέων μετρικών

Για την ολοκλήρωση και την πληρότητα του τμήματος ανάλυσης του μεταφραστή, απαραίτητη ήταν η εισαγωγή νέων μετρικών. Οι μετρικές αυτές είναι απαραίτητες για την γενίκευση της διαδικασίας της επιλογής εκδοχής, ώστε αυτή να μπορεί να εφαρμοστεί και σε άλλες συσκευές. Παρακάτω αναφέρονται οι μετρικές που υλοποιήθηκαν:

- *Πλήθος reduction* ("reduction"). Η τιμή της μετρικής αυτής ισούται με το πλήθος των φράσεων reduction.
- *Πλήθος ordered* ("ordered"). Ισούται με το πλήθος των οδηγιών *ordered* της περιοχής.
- *Χρήση κλειδαριών* ("locks"). Ισούται με 1 αν πραγματοποιούνται κλήσεις ρουτινών που κλειδώνουν/ξεκλειδώνουν κάποια κλειδαριά στο εσωτερικό της περιοχής target, μέσω των συναρτήσεων `omp_set_lock()/omp_unset_lock()`, αντίστοιχα.
- *Πλήθος ατομικών πράξεων* ("atomic"). Ισούται με το πλήθος οδηγιών *atomic* που συναντώνται στην περιοχή.
- *Πλήθος οδηγιών χωρίς αποκλεισμό* ("nowait"). Ισούται με το πλήθος των οδηγιών που συνοδεύονται από *nowait*.
- *Χρήση επιπλέον ICV* ("extraicvs"). Ισούται με 1 αν χρησιμοποιούνται ειδικές συναρτήσεις που σχετίζονται με προσπέλαση μιας ICV. Συγκεκριμένα, ελέγχεται η κλήση των συναρτήσεων `omp_set_dynamic()/omp_get_dynamic()`, `omp_set_nested()/omp_get_nested()`, αλλά και των `omp_set_schedule()` & `omp_get_schedule()`. Οι πρώτες αφορούν τον δυναμικό ορισμό του πλήθους των νημάτων μιας περιοχής παραλληλίας, οι δεύτερες την ενεργοποίηση/απενεργοποίηση εμφωλευμένου παραλληλισμού, ενώ οι τελευταίες το προεπιλεγμένο είδος χρονοδρομολόγησης σε περιοχές διαμοιρασμού *for*.
- *Πλήθος χρονοδρομολογήσεων ανα είδος* ("schedstatic", "scheddynamic", "schedguided"). Ισούται με το πλήθος των περιοχών διαμοιρασμού *for* που χρησιμοποιούν το αντίστοιχο είδος δρομολόγησης επαναλήψεων.
- *Υποστήριξη OpenMP* ("openmp"). Ισούται με 1 αν γίνεται χρήση του προτύπου OpenMP σε ολόκληρη την περιοχή, διαφορετικά ισούται με 0. Η μετρική αυτή είναι ιδιαίτερα χρήσιμη και συνήθως ελέγχεται στο πρώτο στάδιο της διαδικασίας επιλογής εκδοχής βιβλιοθήκης, ώστε να επιλεχθεί η τεττριμένη.
- *Συνολικό πλήθος μετρικών* ("totalmetrics"). Η τιμή της μετρικής αυτής ισούται με το πλήθος των μετρικών που διαθέτουν τιμή διαφορετική του 0. Για παράδειγμα, έστω ότι στο διάγραμμα ροής έχει εξεταστεί ότι μια περιοχή target περιλαμβάνει περιοχές παραλληλίας και περιοχές διαμοιρασμού *for*, ενώ επιπλέον η μετρική totalmetrics έχει τιμή 2. Συνεπώς,

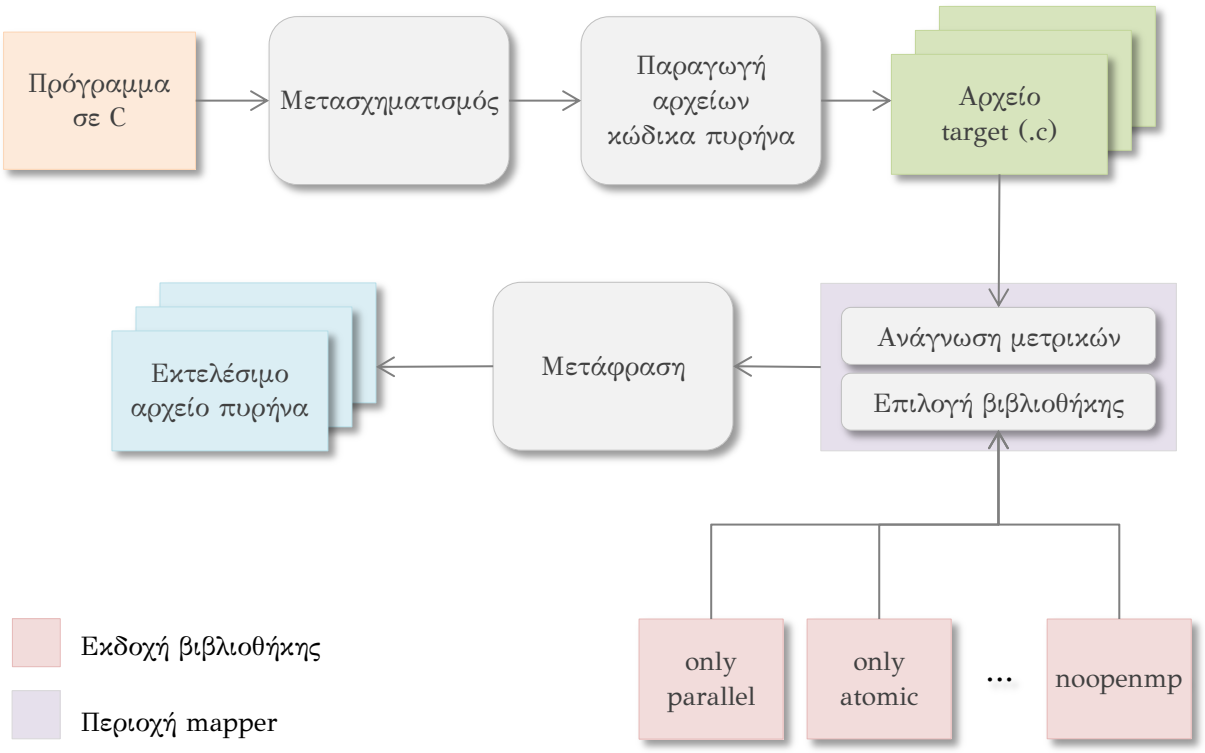
η πληροφορία αυτή μας επιτρέπει να επιλέξουμε απευθείας κάποια εκδοχή που υλοποιεί μόνο τις αντίστοιχες οδηγίες.

### 4.1.3 Αυτοματοποίηση διαδικασίας

Η αρχική ιδέα της προσαρμοζόμενης υποστήριξης εκτέλεσης είχε εφαρμοστεί χειροκινητα εξ'ολοκλήρου στην πλατφόρμα Parallella-16, όπου ο προγραμματιστής διατηρούσε ένα σύνολο εκδοχών βιβλιοθήκης συσκευής, εξέταζε τις μετρικές για την εκάστοτε περιοχή target και αποφάσιζε την κατάλληλη εκδοχή, γνωρίζοντας εκ των προτέρων τα χαρακτηριστικά της καθεμίας. Τυπικά, η απόφαση της εκδοχής βιβλιοθήκης, περιλάμβανε εκ νέου εγκατάσταση της βιβλιοθήκης ώστε να χρησιμοποιηθεί στην θέση της πλήρους.

Η λύση στο πρόβλημα της χειροκίνητης επιλογής εκδοχής δόθηκε με την υλοποίηση του μηχανισμού προσαρμοζόμενης υποστήριξης εκτέλεσης, ο οποίος αυτοματοποιεί την παραπάνω διαδικασία. Η λογική πίσω από τον μηχανισμό αυτό, είναι η προεγκατάσταση των υπάρχοντων εκδοχών βιβλιοθηκών και η επιλογή της μικρότερης σε μέγεθος εκδοχής, που μπορεί να ικανοποιήσει πλήρως τις απαιτήσεις της εφαρμογής. Συγκεκριμένα, τα βήματα που ακολουθούνται είναι τα εξής:

1) Ο προγραμματιστής αρχικά σχεδιάζει και υλοποιεί τις εκδοχές βιβλιοθηκών, με τέτοιο τρόπο ώστε να μπορεί εύκολα να γίνει διαχωρισμός των περιπτώσεων βάσει ενός διαγράμματος ροής, και συνεπώς να είναι σαφής η διαδικασία με την οποία θα επιλεχθεί η πλέον κατάλληλη βιβλιοθήκη.
2) Χρησιμοποιώντας μια νέα δική μας γλώσσα περιγραφής (MAL), ο προγραμματιστής περιγράφει την εν λόγω διαδικασία επιλογής, μέσω κανόνων, οι οποίοι αναπαριστούν ένα διάγραμμα ροής. Απαραίτητη προϋπόθεση είναι η απουσία κύκλων στο διάγραμμα ροής.
3) Ο μεταφραστής αποφασίζει αυτόματα την κατάλληλη εκδοχή βιβλιοθήκης με την οποία θα γίνει η σύνδεση της περιοχής target. Η απόφαση ανατίθεται σε ένα νέο εργαλείο που υλοποίησαμε, τον mapper. Ο mapper λαμβάνει την απόφαση αναλύοντας τις μετρικές και ακολουθώντας τους κανόνες που συντάχθηκαν από τον προγραμματιστή.




Σχήμα 4.2: Προσαρμοζόμενη υποστήριξη εκτέλεσης στον OMPi (ανά συσκευή)

## 4.2 Ο MAL-mapper

Για την αξιοποίηση των δομών που περιγράφηκαν προηγουμένως, υλοποιήθηκε το εργαλείο MAL-mapper. Ο MAL-mapper αναλαμβάνει την επιλογή των κατάλληλων εκδοχών βιβλιοθήκης με τις οποίες θα συνδεθούν οι πυρήνες.

Οι βασικές λειτουργίες του MAL-mapper περιλαμβάνουν την ανάγνωση των στατιστικών που έχουν συλλεχθεί ανά πηγαίο κώδικα αρχείου πυρήνα και την προσπέλαση της δομής που φυλάει το σύνολο των αποφάσεων και την λίστα των υποστηριζομενων εκδοχών. Αναφερόμαστε στη δομή αυτή ως MAL-graph, και η περιγραφή της γίνεται στην ενότητα 5.2. Επιπλέον, στο εσωτερικό του υλοποιούνται τα ερωτήματα των κόμβων (4.2.2) καθώς και άλλες διαχειριστικές λειτουργίες που σχετίζονται με τις εγκατεστημένες συσκευές του OMPi.

### 4.2.1 Επιλογή βιβλιοθήκης συσκευής

Στο πρώτο στάδιο της επιλογής της βιβλιοθήκης συσκευής με την οποία θα γίνει η σύνδεση μίας περιοχής target, γίνεται η ανάγνωση των μετρικών που συλλέχθηκαν από τον μεταγλωττιστή. Οι μετρικές αποθηκεύονται στον πίνακα `metrics`, με τη μορφή της δομής `metric_t`. Το όνομα και η τιμή της μετρικής αποθηκεύονται στα πεδία `name` και `value` της δομής αυτής. Για την εξαγωγή των δεδομένων,

χρησιμοποιείται η βιβλιοθήκη `keyval`, η οποία επιτρέπει την ανάγνωση ζευγών κλειδιού-τιμής από ένα αρχείο, με την εξής μορφή:

*key = value*

Έπειτα, για κάθε συσκευή και για κάθε αρχείο πυρήνα, πραγματοποιείται η εξής διαδικασία:

- παραμετροποίηση και εκτέλεση του συντακτικού αναλυτή της γλώσσας MAL, ο οποίος παράγει το γράφημα MAL-graph,
- διαπέραση της δομής MAL-graph,
- επιστροφή συμβολοσειράς με το όνομα της επιλεγμένης βιβλιοθήκης.

Το αποτέλεσμα που θα επιστραφεί από τον mapper διαφέρει ανά εγκατεστημένη συσκευή, μιας η κάθε μία διαθέτει διαφορετικό αρχείο κανόνων. Στην επόμενη ενότητα περιγράφεται αναλυτικότερα το σημείο στο οποίο παρεμβαίνει ο mapper, κατά τη διάρκεια της μεταγλώττισης από τον OMPi.

### 4.2.2 Υλοποίηση ερωτημάτων

Στο εσωτερικό του MAL-mapper, βρίσκονται οι υλοποιήσεις των ερωτημάτων που πραγματοποιούνται κατά την επίσκεψη ενός κόμβου. Προς το παρόν, υποστηρίζονται τα ερωτήματα *query_has, query_hasonly* και *query_num*. Τα ερωτήματα αυτά βασίζονται στον πίνακα *metrics* που δημιουργείται στο στάδιο της ανάγνωσης μετρικών.

Περισσότερες λεπτομέρειες όσον αφορά τα ερωτήματα αναφέρονται στην ενότητα 5.1.3.

## 4.3 Ενσωμάτωση mapper στον μεταφραστή OMPi

Προκειμένου να υποστηριχθεί η προσαρμοζόμενη υποστήριξη εκτέλεσης σε συσκευές, απαραίτητη ήταν η τροποποίηση της διαδικασίας με την οποία μεταγλωττίζονται οι πηγαίοι κώδικες που αφορούν περιοχές target. Στην ενότητα 4.3.1 περιγράφεται ο υπάρχων μηχανισμός υποστήριξης εκτέλεσης. Η ενότητα 4.3.2 προτείνει τις τροποποιήσεις στον μηχανισμό αυτό.

### 4.3.1 Αρχική μεταγλώττιση των περιοχών target

Για τη μεταγλώττιση των αρχείων πυρήνων ενός προγράμματος, ο OMPi, αρχικά, διατηρούσε εγκατεστημένο ένα Makefile πυρήνα για κάθε βιβλιοθήκη συσκευής. Μετά τον μετασχηματισμό και διαχωρισμό των πυρήνων σε ξεχωριστά αρχεία, χρησιμοποιούσε τα Makefiles για να παράξει τους εκτελέσιμους πυρήνες κάθε συσκευής.

Η επιλογή ενός Makefile πυρήνα στην παραπάνω διαδικασία είναι τεττριμένη. Με το πέρας της, κάθε αρχείο πυρήνα έχει μεταφραστεί σε εκτελέσιμους πυρήνες, βάσει της πλήρους βιβλιοθήκης συσκευής, ή όποιας βιβλιοθήκης έχει επιλεχθεί χειροκίνητα από τον προγραμματιστή.

### 4.3.2 Τροποποιήσεις για προσαρμοζόμενη υποστήριξη εκτέλεσης

Για τους σκοπούς αυτής της διπλωματικής εργασίας, τροποποιήθηκε η διαδικασία της υποστήριξης εκτέλεσης σε συσκευές. Συγκεκριμένα, επιθυμούμε να επιλέγουμε το κατάλληλο makefile πυρήνα, για την κάθε περιοχή target. Βάσει του αρχείου αυτού, γίνεται η μετάφραση της περιοχής σε εκτελέσιμο αρχείο πυρήνα. Αναφερόμαστε στα makefiles πυρήνα ως *αρχεία MakeKernel*. Απαραίτητη ήταν η τροποποίηση των ίδιων των αρχείων MakeKernel, ώστε να μεταγλωττίζουν *μόνο έναν* πηγαίο κώδικα target και όχι *όλους*. Επιπλέον, για κάθε βιβλιοθήκη συσκευής, απαραίτητη ήταν η μετονομασία των MakeKernel βάσει της βιβλιοθήκης που αυτά αντιπροσωπεύουν.

Η κλήση του mapper πραγματοποιείται με την συνάρτηση `mapper_run()`, η οποία αρχικοποιεί τον mapper με την διαδρομή του προγράμματος, το όνομα συσκευής και άλλες παραμέτρους για την ορθή λειτουργία του. Το στάδιο στο οποίο ενσωματώνεται ο Mapper, είναι αυτό της επιλογής των κατάλληλων Makefiles πυρήνα.

# Κεφάλαιο 5

# Σχεδιασμός και υλοποίηση του μηχανισμού MAL-mapper

## 5.1 Η γλώσσα MAL (Mapper Language)

Η γλώσσα MAL σχεδιάστηκε στο πλαίσιο της διπλωματικής εργασίας για την σύνταξη αρχείων κανόνων από τον προγραμματιστή βιβλιοθηκών υποστήριξης εκτέλεσης συσκευών. Η σύνταξή της είναι απλή και γενική, γεγονός που την καθιστά χρήσιμη σε κάθε είδους συσκευή η οποία μπορεί να παραμετροποιηθεί βάσει των εξαγόμενων μετρικών.

Για τη σύνταξη αρχείων κανόνων, αρχικά ενδείκνυται (για λόγους ευκολίας) η δημιουργία ενός διαγράμματος ροής από τον προγραμματιστή, το οποίο περιγράφει πλήρως τη διαδικασία της επιλογής κατάλληλης εκδοχής βιβλιοθήκης (flavor). Έπειτα, ακολουθώντας την γραμματική MAL (Παράρτημα Α), ο προγραμματιστής εισάγοντας τους τελικούς κόμβους-εκδοχές και τα ενδιάμεσα σημεία απόφασης, αποτυπώνει, τελικά, το διάγραμμα αυτό σε αρχείο κανόνων. Απαραίτητη προϋπόθεση είναι το διάγραμμα ροής να μην περιέχει κύκλους μεταξύ των κόμβων του.

### 5.1.1 Βασική σύνταξη

Η σύνταξη ενός αρχείου κανόνων είναι η εξής:

```
{
    flavors = [ λίστα εκδοχών ],
    nodes = [ λίστα κόμβων ]
}
```

Αρχικά γίνεται η δήλωση της *λίστας βιβλιοθηκών*, η οποία περιλαμβάνει όλες τις δυναμικές βιβλιοθηκές υποστήριξης εκτέλεσης συσκευής που υποστηρίζονται, χωρισμένες με κόμμα, ως εξής:

*"όνομα βιβλιοθήκης" [, "όνομα βιβλιοθήκης" ...]*

Τη δήλωση των δυναμικών βιβλιοθηκών ακολουθεί η δήλωση της *λίστας* των ενδιάμεσων *κόμβων* απόφασης. Κατ'ελάχιστο απαιτείται η ύπαρξη ενός κόμβου, ενώ ένας κόμβος διαχωρίζεται από τον επόμενο με κόμμα (,) ως εξής:

*κόμβος [, κόμβος ... ]*

Κάθε στοιχείο-κόμβος έχει την παρακάτω μορφή:

*όνομα κόμβου* = {
*ερώτημα*(*μετρική*) ,
*συνθήκη*
[, *συνθήκη* ... ]
}

Αρχικά ορίζεται το όνομα του κόμβου. Συνηθίζεται ο κόμβος να παίρνει την ονομασία του από το είδος της απόφασης που λαμβάνει. Έπειτα, δηλώνεται το ερώτημα που αφορά μια συγκεκριμένη μετρική, καθώς επίσης και το σύνολο των συνθηκών. Ανάλογα με τις ερωτήσεις που επιδέχεται το ερώτημα, το ελάχιστο πλήθος των συνθηκών μπορεί να είναι είτε 1 είτε 2.

### 5.1.2 Συνθήκες

Οι συνθήκες ορίζουν το σύνολο των ακμών που ξεκινούν από έναν κόμβο. Το αποτέλεσμα του ερωτήματος χρησιμοποιείται ώστε να συμμετάσχει σε μια ειδική έκφραση, της οποίας η αλήθεια θα καθορίσει τον (επόμενο) κόμβο μετάβασης. Μια συνθήκη έχει την εξής μορφή:

*συγκριτικός-τελεστής ακέραιος* : *όνομα-γειτονικού-κόμβου* ($1^{η}$ μορφή)

ή

*τιμή αληθείας* : *όνομα-γειτονικού-κόμβου* (true ή false) ($2^{η}$ μορφή)

Ο συγκριτικός τελεστής μπορεί να είναι ένας από τους παρακάτω:

>=, <=, >, <, = ή !=

Για παράδειγμα, οι συνθήκες `> 5 : "devpart-full"` και `true : "devpart-parallel"` είναι ορθές συνθήκες.

### 5.1.3 Ερωτήματα

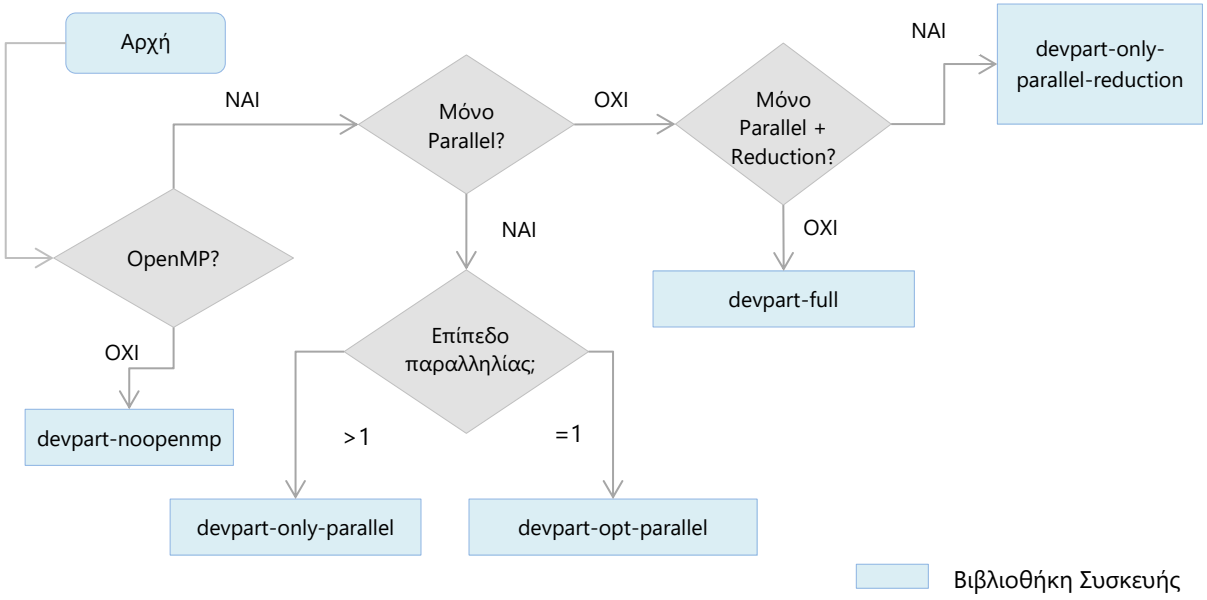
Για τη λήψη απόφασης χρησιμοποιούμε ερωτήματα σχετικά με την συλλογή στατιστικών. Οι απαντήσεις στα ερωτήματα επιστρέφονται από το εργαλείο

Mapper, η υλοποίηση του οποίου περιγράφεται σε επόμενη ενότητα. Το είδος ενός ερωτήματος μπορεί να είναι ένα εκ των παρακάτω:

- **has(*μετρική*)**. Το ερώτημα has επιστρέφει TRUE αν η τιμή της *μετρικής* είναι μεγαλύτερη ή ίση του 1. Αν η τιμή της μετρικής είναι ίση με το 0, τότε επιστρέφει FALSE, ενώ αν αυτή δεν βρεθεί στα εξαγόμενα στατιστικά, επιστρέφεται -1. Σημειώνεται ότι το ερώτημα has πρέπει να συνοδεύεται από ακριβώς δύο συνθήκες, μία true και μία false $2^{ης}$ μορφής.
- **hasonly(*μετρική*)**. Το ερώτημα hasonly επιστρέφει TRUE αν η *μετρική* είναι η μοναδική στο σύνολο των στατιστικών με τιμή μεγαλύτερη ή ίση του 1. Αν υπάρχουν και άλλες μετρικές με τιμή μεγαλύτερη ή ίση του 1, τότε επιστρέφεται FALSE. Στην περίπτωση που η μετρική δεν βρεθεί, επιστρέφεται -1. Σημειώνεται ότι το ερώτημα has πρέπει να συνοδεύεται από ακριβώς δύο συνθήκες, μία true και μία false $1^{ης}$ μορφής.
- **num(*μετρική*)**. Το ερώτημα num επιστρέφει την τιμή της *μετρικής*. Αν η μετρική δεν βρεθεί, τότε επιστρέφεται -1. Σημειώνεται ότι το ερώτημα num χρησιμοποιείται μόνο με συνθήκες της $1^{ης}$ μορφής.

### 5.1.4 Αποτύπωση διαγράμματος ροής σε αρχείο κανόνων MAL

Για την κατανόηση της γλώσσας MAL, παρατίθεται ένα παράδειγμα διαγράμματος ροής (Σχήμα 5.1) και η διαδικασία που ακολουθείται για να αποτυπωθεί αυτό σε ένα αρχείο κανόνων.




Σχήμα 5.1: Παράδειγμα διαγράμματος ροής για επιλογή βιβλιοθήκης

Αρχικά, παρατηρούμε ότι στο διάγραμμα ροής υπάρχουν 5 τερματικοί κόμβοι-βιβλιοθήκες συσκευής. Συνεπώς, ξεκινούμε με τη δήλωσή τους στο αρχείο κανόνων:

```
{
    flavors = [
     "devpart-noopenmp",
     "devpart-only-parallel",
     "devpart-opt-parallel",
     "devpart-only-parallel-reduction",
     "devpart-full"
    ]
    ...
}
```

Έπειτα, στο διάγραμμα περιλαμβάνονται 4 σημεία απόφασης:

i. *OpenMP?* Αντιστοιχεί στο ερώτημα has(openmp), δηλαδή αν υπάρχει οδηγία OpenMP μέσα στην περιοχή target.

ii. *Μόνο parallel?* Αντιστοιχεί στο ερώτημα hasonly(parallel), δηλαδή αν η οδηγία parallel είναι η μοναδική σε ολόκληρη την περιοχή.

iii. *Επίπεδο παραλληλίας;* Αντιστοιχεί στο ερώτημα num(max_par_lev), όπου εξετάζεται το μέγιστο εμφανιζόμενο επίπεδο παραλληλίας στην περιοχή.

iv. *Μόνο parallel + reduction?* Αντιστοιχεί σε σύνθετο ερώτημα, το οποίο για να εκφραστεί στη γλώσσα MAL απαιτούνται 3 υποερωτήματα:

   a. *has(parallel).* Αρχικά ελέγχουμε αν υπάρχει οδηγία parallel στην περιοχή target. Αν το ερώτημα επιστρέψει «1», προχωρούμε στο επόμενο ερώτημα. Διαφορετικά, επιλέγουμε την ακμή «ΟΧΙ» του σημείου απόφασης, ο οποίος οδηγεί στη βιβλιοθήκη "Full".

   b. *has(reduction).* Με το ερώτημα αυτό ελέγχουμε την ύπαρξη φράσης reduction στο εσωτερικό της περιοχής target. Αν λάβουμε ως αποτέλεσμα «1», τότε προχωρούμε στο επόμενο (και τελευταίο) ερώτημα c. Διαφορετικά, επιλέγουμε την ακμή «ΟΧΙ» του σημείου απόφασης.

   c. *num(total).* Η τιμή της μετρικής total ισούται με το πλήθος των οδηγιών OpenMP που χρησιμοποιούνται στην περιοχή target. Αν το πλήθος των συνολικών μετρικών είναι 2, τότε με βεβαιότητα μπορούμε να πούμε ότι αυτές είναι οι parallel και reduction και άρα να επιλέξουμε την βιβλιοθήκη devpart-parallel-reduction. Διαφορετικά, επιλέγουμε τη βιβλιοθήκη devpart-full. Συνεπώς, εισάγουμε δύο συνθήκες "=2" και "!=2" στον κόμβο, οι οποίες συσχετίζονται με τους κόμβους devpart-parallel-reduction και devpart-full.

Με τις παραπάνω τροποποιήσεις, προκύπτει το τελικό αρχείο κανόνων:

```
1  {
2        flavors = [
3         “devpart-noopenmp”,
4         “devpart-parallel”,
5         “devpart-opt-parallel”,
6         “devpart-parallel-reduction”,
7         “devpart-full”
8        ],
9        nodes = [
11        hasopenmp: {
12            has(openmp), true: “onlyparallel”, false: “devpart-noopenmp”
13        },
14        onlyparallel: {
15            num(parallel), >1: “devpart-parallel”, =1: “devpart-opt-parallel”
16        },
17        onlyparreduction1: {
18            has(parallel), true: “onlyparreduction2”, false: “devpart-full”
19        },
20        onlyparreduction2: {
21            has(reduction), true: “onlyparreduction3”, false: “devpart-full”
22        },

23        onlyparreduction3: {
24            num(totalmetrics), =2: “devpart-parallel-reduction”, !=2: “devpart-full”
25        }
26       ]
27 }
```

Σχήμα 5.2: Το αρχείο κανόνων MAL για το Σχήμα 5.1

## 5.2 Η δομή MAL-graph

Η εσωτερική αναπαράσταση των κανόνων που χρησιμοποιούμε στον mapper είναι ένα είδος άκυκλου γραφήματος, το MAL-graph. Για το γράφημα αναπτύχθηκαν όλες οι απαραίτητες λειτουργίες όπως αυτές της εισαγωγής και διαγραφής νέων κόμβων, της διατήρησης γειτονικών κόμβων με την μορφή λιστών γειτνιάσεων ανά κόμβο, αλλά και της διαπέρασης του γράφου για τη λήψη απόφασης.

### 5.2.1 Πεδία

Η δομή MAL-graph περιέχει έναν αριθμό πεδίων, για την αποθήκευση της πληροφορίας που αποτυπώνεται στο αρχείο κανόνων. Τα πεδία αυτά είναι τα εξής:

- **set(graph_st)** *graph*. Το πεδίο *graph* αποτελεί ένα hash set το οποίο διατηρεί τους κόμβους του γραφήματος. Οι κόμβοι αναπαριστώνται από την δομή *mg_node_t*, η οποία διατηρεί τα εξής πεδία:
  - **mgquery_t** *query*. Η δομή *mgquery_t* συμβολίζει το ερώτημα, όπως αυτό περιγράφηκε στην ενότητα 4.2.3.
  - **mgadjnode_t** *adjacents[]*. Ο πίνακας *adjacents* είναι η λίστα γειτνίασης του κόμβου. Η δομή *mgadjnode_t* συγκρατεί το αναγνωριστικό του γειτονικού κόμβου, καθώς επίσης και την συνθήκη (**mgcond_t** *cond*) με την οποία αυτός συσχετίζεται, σύμφωνα με το αρχείο κανόνων.
  - Το αναγνωριστικό (*id*) του κόμβου, το όνομά του (*name*) και μεταβλητές ελέγχου που αφορούν πλήθος των γειτονικών του κόμβων (*nneighbours*) και το αν αυτός είναι ο αρχικός κόμβος του γραφήματος (*isinitial*).
- **mgnode_t** *pending_nodes[]*. Ο πίνακας *pending_nodes* χρησιμοποιείται για την αποθήκευση κόμβων που εκκρεμεί η εισαγωγή τους στο γράφημα. Μέσω του μηχανισμού αυτού γνωρίζουμε ποιοι κόμβοι εντοπίστηκαν ως γείτονες ενός άλλου και δεν έχουν δηλωθεί ρητά στο αρχείο κανόνων.
- **int** *npending*. Ο ακέραιος αυτός διατηρεί το πλήθος των εκκρεμών κόμβων.
- **int** *lastid*. Ο ακέραιος *lastid* διατηρεί το αναγνωριστικό που ανατέθηκε στον κόμβο που εισήχθηκε πιο πρόσφατα στο γράφημα.

### 5.2.2 Ρουτίνες

Οι λειτουργίες που υποστηρίζονται αυτή τη στιγμή πάνω στη δομή MAL-graph, είναι οι εξής:

- **void** mal_graph_init**(mal_graph_t*** m**,** **int** (*q1)(char* metric)**,** **int** (*q2)(char* metric)**,** **int** (*q3)(char* metric)**)**.
  Η συνάρτηση αυτή χρησιμοποιείται για να αρχικοποιήσει μια δομή MAL-graph, δίνοντας ως όρισμα 3 δείκτες στις συναρτήσεις που υλοποιούν τα ερωτήματα has, hasonly και num. Αυτό συμβαίνει γιατί η συλλογή στατιστικών και τα ερωτήματα διατηρούνται σε άλλο σημείο κατά την εκτέλεση.
- **void** mal_graph_init_neighbours**(mal_graph_t*** m**,** **int** id**)**.
  Η συνάρτηση αυτή αρχικοποιεί τον πίνακα γειτνίασης ενός κόμβου με αναγνωριστικό id.
- **void** mal_graph_set_qtype**(mal_graph_t*** m**,** **int** id**,** **int** type**)**.

Χρησιμοποιούμε την *mal_graph_set_node_qtype()* για να ορίσουμε το είδος του ερωτήματος που πραγματοποιείται κατά την επίσκεψη του κόμβου με αναγνωριστικό id.

- **void** mal_graph_set_node_constr(**mal_graph_t*** m, **int** id, **char*** construct).
  Χρησιμοποιούμε την συνάρτηση *mal_graph_set_node_constr()* για να ορίσουμε τη μετρική την οποία θα αφορά το ερώτημα ενός κόμβου με αναγνωριστικό id.
- **int** mal_graph_is_term(**mal_graph_t*** m, **int** id).
  Η συνάρτηση *mal_graph_is_term()* επιστρέφει 1 αν ένας κόμβος του γραφήματος *m* είναι τερματικός. Διαφορετικά, επιστρέφει 0 αν ο κόμβος είναι ενδιάμεσος, ή -1 αν ο κόμβος δεν βρεθεί στο γράφημα.
- **int** mal_graph_search(**mal_graph_t*** m, **char*** nodename).
  Η *mal_graph_search()* χρησιμοποιείται για την αναζήτηση ενός κόμβου με όνομα *nodename* στο γράφημα *m*. Αν ο κόμβος βρεθεί, επιστρέφεται το αναγνωριστικό του. Διαφορετικά, επιστρέφεται -1.
- **void** mal_graph_add_node(**mal_graph_t*** m, **char*** nodename).
  Χρησιμοποιούμε την *mal_graph_add_node()* για την εισαγωγή ενός κόμβου με όνομα *nodename* στο γράφημα *m*.
- **void** mal_graph_add_neighbour(**mal_graph_t*** m, **int** id, **char*** nodename, **int** condtype, **int** condvalue).
  Μέσω της *mal_graph_add_neighbour()* μπορούμε να εισάγουμε έναν κόμβο στην λίστα γειτνίασης ενός κόμβου με αναγνωριστικό *id*. Αρχικά, γίνεται αναζήτηση του γειτονικού κόμβου στο γράφημα *m*. Αν αυτός δεν βρεθεί, τότε πριν την προσθήκη του στην λίστα γειτνίασης, τοποθετείται στην λίστα εκκρεμών κόμβων. Έπειτα, προστίθεται η συνθήκη ελέγχου, με την οποία ο κόμβος αυτός συσχετίζεται.
- **int** mal_graph_get_first(**mal_graph_t*** m).
  Η συνάρτηση *mal_graph_get_first()* επιστρέφει τον αρχικό κόμβο του γραφήματος *m*.
- **void** mal_graph_visit(**mal_graph_t*** m, **int** id).
  Η συνάρτηση *mal_graph_visit()* καλείται αναδρομικά κατά την διαπέραση του γραφήματος *m*. Χρησιμοποιείται για την επίσκεψη ενός κόμβου με αναγνωριστικό *id*.
- **void** mal_graph_traverse(**mal_graph_t*** m, **char*** retvalue).
  Η *mal_graph_traverse()* χρησιμοποείται για την εκκίνηση της διαπέρασης του γραφήματος *m*. Ουσιαστικά, επισκέπτεται ο αρχικός κόμβος του γραφήματος.

Λεπτομέρειες για τη διαπέραση της δομής MAL-graph αναφέρονται στο επόμενο υποκεφάλαιο, όπου εξετάζεται η μέθοδος επιλογής ενός κόμβου, καθώς επίσης και η συνθήκη τερματισμού της διαπέρασης.

### 5.2.3 Διαπέραση του γραφήματος

Η κύρια λειτουργία που αξιοποιεί τη δομή MAL-graph είναι αυτή της διαπέρασης. Κατά τη διαπέραση, επισκεπτόμαστε διαδοχικά κόμβους ώστε να καταλήξουμε στην κατάλληλη εκδοχή βιβλιοθήκης υποστήριξης εκτέλεσης. Ανάλογα με τη δομή του γραφήματος, η οποία εξαρτάται από την σύνταξη του αρχείου κανόνων, η διαπέραση μπορεί να είναι επιτυχής ή ανεπιτυχής. Στην δεύτερη περίπτωση, αποφασίζουμε την χρήση μιας προεπιλεγμένης βιβλιοθήκης (π.χ. την πλήρη βιβλιοθήκη).

Για τη διαπέραση έχουν υλοποιηθεί οι προαναφερθείσες ρουτίνες `mal_graph_traverse()` και `mal_graph_visit()`. Η πρώτη εκκινεί τη διαπέραση, καλώντας τη δεύτερη με όρισμα τον αρχικό κόμβο του γραφήματος. Η `mal_graph_visit()` καλεί τον εαυτό της αναδρομικά, με συνθήκη τερματισμού την εύρεση τερματικού κόμβου.

Αρχικά ανακτούμε τον κόμβο βάσει του αναγνωριστικού του και πραγματοποιούμε το ερώτημα με το οποίο αυτός συσχετίζεται. Αφού αποθηκεύσουμε το αποτέλεσμα του ερωτήματος, διατρέχουμε την λίστα γειτνίασης του κόμβου. Όπως αναφέρθηκε, τα στοιχεία της λίστας είναι ζεύγη συνθηκών με αναγνωριστικά γειτονικού κόμβου. Για κάθε στοιχείο, γίνεται ο έλεγχος αληθείας της συνθήκης, στις οποίες συμμετέχει το αποτέλεσμα. Αν η συνθήκη είναι αληθής, πραγματοποιούμε επίσκεψη στον κόμβο με τον οποίο συσχετίζεται. Παρακάτω αναλύεται η διαδικασία που ακολουθείται για τον Κώδικα 4.2 και απεικονίζεται στο Σχήμα 4.3 (με μετρικές openmp=1, parallel=2 και όλες τις υπόλοιπες 0):

➔ Κόμβος `hasopenmp`

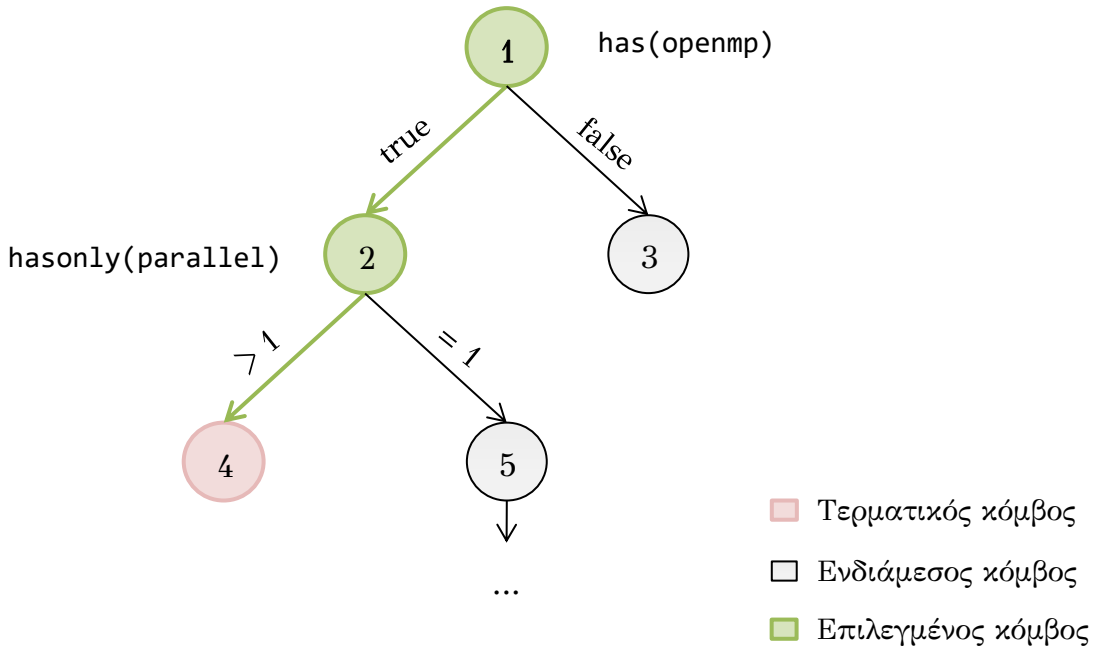
Τερματικός; **ΟΧΙ**
`has(openmp)` -> 1
1 == 1 ≡ **ΑΛΗΘΕΙΑ** ==> Επίσκεψη κόμβου `onlyparallel`
1 == 0 ≡ **ΨΕΜΑ**

➔ Κόμβος `onlyparallel`

Τερματικός; **ΟΧΙ**
`hasonly(parallel)` -> 2
2 > 1 ≡ **ΑΛΗΘΕΙΑ** ==> Επίσκεψη κόμβου `devpart-parallel`
1 == 1 ≡ **ΨΕΜΑ**

➔ Κόμβος `devpart-parallel`

Τερματικός; **ΝΑΙ**
∴ Επιλογή βιβλιοθήκης `devpart-parallel`




Σχήμα 5.3: Παράδειγμα υπό συνθήκη μετάβασης σε κόμβο
(για openmp = 1, parallel = 2)

## 5.3 Ο λεκτικός αναλυτής MAL-scanner

Η λεκτική ανάλυση του αρχείου κανόνων γίνεται από το MAL-scanner, που υλοποιήσαμε στο πλαίσιο της εργασίας μας. Ο MAL-scanner είναι ένας απλά δομημένος, αλλά γενικός λεκτικός αναλυτής, ασφαλής για πολυνηματική χρήση. Έχει την δυνατότητα να αναγνωρίζει σύμβολα και δεσμευμένες λέξεις από ένα αρχείο ή μια συμβολοσειρά.

Κύρια συνάρτηση του MAL-scanner είναι η `mal_scan_nexttoken()`. Υποστηρίζει την αναγνώριση αριθμών, συμβολοσειρών που ορίζονται με εισαγωγικά (“”), λέξεων (δεσμευμένων και μη) και ειδικών συμβόλων, όπως οι τελεστές σύγκρισης. Προς το παρόν αναγνωρίζονται ξεχωριστά οι λέξεις-κλειδιά “flavors”, “nodes”, “true”, “false”, “has”, “hasonly” και “num”. Οι δεσμευμένες λέξεις ορίζονται στον πίνακα `keyword[]` ο οποίος αποτελείται από ζεύγη λέξεων με αναγνωριστικά.

Για την χρήση του scanner, αρκεί η δήλωση του scanner (τύπου *scanif_t*), καθώς επίσης και η δήλωση ενός ακεραίου στο οποίο θα αποθηκεύεται το αναγνωρισμένο σύμβολο. Αρχικοποιούμε τον scanner με την *mal_scan_file_init()*, δίνοντας ως όρισμα ένα δείκτη αρχείου, ή την *mal_scan_string_init()* με όρισμα μια συμβολοσειρά. Έπειτα καλούμε επανηλειμμένα την συνάρτηση *mal_scan_nexttoken()* με όρισμα τον scanner, για να λάβουμε το επόμενο σύμβολο.

Σε κάθε κλήση της *mal_scan_nexttoken()* έχουμε την δυνατότητα να λάβουμε διάφορες πληροφορίες που αφορούν το πιο πρόσφατα αναγνωρισμένο σύμβολο,

μέσω συναρτήσεων. Έτσι, μπορούμε να χειριστούμε καταλλήλως τα σύμβολα από ένα εξωτερικό πρόγραμμα, όπως αυτό του MAL-parser (περιγράφεται στο επόμενο υποκεφάλαιο). Αυτές είναι οι παρακάτω:

- `int mal_scan_get_position(scanif_t s, int *linepos)`. Επιστρέφει τον αριθμό της γραμμής στην οποία ανήκει το αναγνωρισμένο σύμβολο.
- `char *mal_scan_get_toktext(scanif_t s)`. Επιστρέφει μια συμβολοσειρά που περιέχει το αναγνωρισμένο σύμβολο.
- `int mal_scan_get_toklen(scanif_t s)`. Επιστρέφει το μήκος του αναγνωρισμένου συμβόλου.
- `int mal_scan_get_intval(scanif_t s)`. Όταν το αναγνωρισμένο σύμβολο είναι ένας ακέραιος, χρησιμοποιούμε τη συνάρτηση `mal_scan_get_intval()` για να λάβουμε την τιμή του.

## 5.4 Ο συντακτικός αναλυτής MAL-parser

Για την συντακτική ανάλυση της γλώσσας MAL, αναπτύχθηκε ο MAL-parser. Λόγω της σχετικά μικρής πολυπλοκότητας της γλώσσας, ο MAL-parser υλοποιήθηκε με τη τεχνική της αναδρομικής κατάβασης (recursive descent) [6], ώστε να υποστηρίξει πλήρως την γραμματική της γλώσσας MAL. Η γραμματική της MAL σχεδιάστηκε ώστε να μην περιέχει αριστερή αναδρομή. Πρακτικά, αυτό σημαίνει ότι οι κανόνες παράγονται μόνο από τα δεξιά και συνεπώς συνεπώς είναι εύκολη η αντιστοίχιση κάθε κανόνα σε μία συνάρτηση που τον αναγνωρίζει.

Η συνάρτηση που διατίθεται για εξωτερική κλήση είναι η `mal_parse()`, η οποία δέχεται ως όρισμα τον δείκτη αρχείου για το αρχείο κανόνων, καθώς επίσης και τη δομή MAL-graph. Πρώτα γίνεται η αρχικοποίηση του λεκτικού αναλυτή MAL-scanner και λαμβάνεται το πρώτο αναγνωρισμένο σύμβολο. Έπειτα, καλείται η συνάρτηση που αφορά τον αρχικό κανόνα της γραμματικής, η `mal()`.

Στο εσωτερικό της κάθε συνάρτησης κανόνα, εκτός από την αναγνώριση των συμβόλων που αυτός αφορά, πραγματοποιούνται παράλληλα όλες οι κλήσεις που αφορούν τις λειτουργίες του γραφήματος MAL-graph. Αυτές περιλαμβάνουν την προσθήκη κόμβων (τερματικών ή ενδιάμεσων), ή των γειτονικών αυτών στις λίστες γειτνίασης, καθώς και τον ορισμό του ερωτήματος που πραγματοποιείται σε κάθε κόμβο.

Τέλος, δίνεται η δυνατότητα παραμετροποίησης του MAL-parser, μέσω της συνάρτησης `mal_parse_set_opts()`. Ο προγραμματιστής δημιουργεί μια δομή τύπου `parseopts_t` και θέτει τις τιμές των εξής πεδίων της:

- `int nolibcheck`. Για την τιμή 0, ο MAL-parser θα ελέγξει την ύπαρξη των δυναμικών βιβλιοθηκών που ορίζονται στον πίνακα flavors του αρχείου κανόνων.
- `int parsetime`. Οι επιτρεπόμενες τιμές του πεδίου αυτού είναι `PARSE_ALL` και `PARSE_FLAVORS_ONLY` και ορίζουν την συμπεριφορά του parser (ανάλυση

ολόκληρου του αρχείου κανόνων ή μόνο των δηλωμένων βιβλιοθηκών, αντίστοιχα).

- `int (*flavlib_ok)(char *)`. Ο δείκτης συνάρτησης `flavlib_ok` πρέπει να δείχνει σε μια συνάρτηση που πραγματοποιεί τον έλεγχο της βιβλιοθήκης και επιστρέφει 0 ή 1 για αποτυχία ή επιτυχία, αντίστοιχα. Χρησιμοποιείται στην περίπτωση όπου το `nolibcheck` είναι ίσο με 0.

Για την ομαλή λειτουργία του parser, η παραμετροποίηση του πρέπει να γίνεται πριν την κλήση της συνάρτησης `mal_parse()`.

# Κεφάλαιο 6

# Εφαρμογή του μηχανισμού προσαρμοζόμενης εκτέλεσης στο Epiphany-III

## 6.1 Εισαγωγή

Για τον έλεγχο της υποδομής του Mapper, εφαρμόσαμε τον μηχανισμό στη συσκευή Epiphany-III. Πραγματοποιήθηκαν τόσο πειράματα ορθότητας όσο και πειράματα επιδόσεων. Στα πρώτα, μελετάται η ορθή επιλογή βιβλιοθήκης από το εργαλείο Mapper, για διάφορες μετρικές. Στην δεύτερη συλλογή πειραμάτων, εξετάζονται οι επιδόσεις που επιτυγχάνονται κατά την εκτέλεση κώδικα πυρήνων στις συσκευές, με την αυτόματη επιλογή της κατάλληλης βιβλιοθήκης.

Συγκεκριμένα, τα πειράματα εκτελέστηκαν στην πλατφόρμα Parallella-16, με λειτουργικό σύστημα Linaro 14.04 (*GNU/Linux 3.14.12-parallella-xilinx-g40a90c3 armv7l*). Η πλατφόρμα επιλέχθηκε διότι ο συνεπεξεργαστής Epiphany-III που διαθέτει, προκάλεσε μεγάλο ενδιαφέρον, μιας και η προσαρμοζόμενη υποστήριξη εκτέλεσης έχει νόημα, λόγω του μικρού μεγέθους μνήμης που διατίθεται ανά επεξεργαστική μονάδα (32KB).

Χωρίσαμε τα πειράματα σε δύο κατηγορίες: αυτά της ορθότητας, μέσω των οποίων διαπιστώνεται η ορθότητα της λειτουργίας του Mapper και αυτά των επιδόσεων, ώστε να διαπιστώσουμε αν επιτυγχάνονται οι βελτιώσεις που υπόσχεται η ιδέα της διπλωματικής εργασίας, όσον αφορά το μέγεθος και το χρόνο εκτέλεσης προγραμμάτων. Συγκεκριμένα, για τη δεύτερη κατηγορία πειραμάτων, χρησιμοποιήθηκαν τα EPCC Benchmarks, μια οικογένεια δοκιμαστικών προγραμμάτων για την μέτρηση της απόδοσης προγραμμάτων σε OpenMP. Όλα τα

προγράμματα εκτελέστηκαν με το σύνολο των εκδοχών βιβλιοθήκης που περιγράφεται στη συνέχεια (6.1.3).

### 6.1.1 Εκδοχές βιβλιοθήκης

Το σύνολο των εκδοχών βιβλιοθηκών που χρησιμοποιήθηκε για την δοκιμή της διαδικασίας είναι το εξής:

- *Full*. Η εκδοχή αυτή είναι η πλήρης βιβλιοθήκη συσκευής.
- *NoOpenMP*. Στην εκδοχή αυτή δεν υποστηρίζεται καμία λειτουργικότητα του OpenMP. Κάθε eCore εκτελεί κώδικα με σειριακό τρόπο.
- *OnlyTasksSingle*. Η εκδοχή αυτή υποστηρίζει την εκτέλεση tasks από ομάδες eCores, τα οποία διαμοιράζονται τον φόρτο της εκτέλεσης μέσω μιας οδηγίας single. Η εκδοχή κατασκευάστηκε βάσει της πρακτικής που αφορά στην δημιουργία όλων των tasks από ένα νήμα μέσω της οδηγίας single και στην εκτέλεση τους από τα υπόλοιπα νήματα της ομάδας.
- *BlockingWS*. Στην περίπτωση της εκδοχής αυτής, υποστηρίζεται πλήρως το σύνολο του OpenMP, με την μόνη διαφορά ότι η υλοποίηση της φράση *nowait* έχει αφαιρεθεί, μιας και μπορεί να εισάγει σημαντικές επιβαρύνσεις κατά την εκτέλεση.
- *NoExplTasks-BlockingWS*. Η εκδοχή αυτή είναι όμοια με την προηγούμενη, αλλά δεν διαθέτει μηχανισμό για tasks, μιας και η υλοποίηση αυτών μπορεί να εισάγει σημαντικό κόστος όσον αφορά τη μνήμη.
- *NoExplTasks*. Στην εναλλακτική εκδοχή αυτή, υποστηρίζεται το σύνολο του OpenMP, με εξαίρεση τον μηχανισμό για tasks.
- *OnlyParallel*. Η εκδοχή αυτή επιτρέπει μόνο την δημιουργία ομάδων παραλληλίας μεταξύ των eCores και την επικοινωνία μεταξύ τους. Καμία άλλη λειτουργικότητα δεν υποστηρίζεται.
- *OnlyParallelAtomic*. Όντας επέκταση της *OnlyParallel*, η εκδοχή αυτή υποστηρίζει μόνο την δημιουργία ομάδων παραλληλίας στα νήματα και τον συγχρονισμό τους μέσω οδηγιών atomic.
- *OnlyCritical*. Μια εναλλακτική επέκταση της *OnlyParallel*, η εκδοχή *OnlyCritical* επιτρέπει την δημιουργία ομάδων παραλληλίας στα eCores και τον συγχρονισμό τους μέσω κρίσιμων περιοχών.
- *OnlyParallelLocks*. Η *OnlyParallelLocks* υποστηρίζει, εκτός από την δημιουργία ομάδων παραλληλίας των eCores, τον συγχρονισμό τους μέσω κλειδαριών.
- *OnlyParallelReduction*. Στην εκδοχή αυτή επεκτείνουμε την *OnlyParallel* υποστηρίζοντας επιπλέον τη χρήση reductions.
- *OnlySingle*. Η εκδοχή αυτή υποστηρίζει μόνο την δημιουργία παράλληλων ομάδων μεταξύ των νημάτων και την επικοινωνία μεταξύ τους με χρήση reductions.

- *OnlyStaticFor*. Η εκδοχή αυτή υποστηρίζει μόνο το διαμοιρασμό φόρτου με χρήση της οδηγίας for με στατική χρονοδρομολόγηση και τη δημιουργία παράλληλων περιοχών.
- *OnlyStaticOrderedFor*. Επεκτείνοντας την *OnlyStaticFor*, η εκδοχή αυτή υποστηρίζει επιπλέον τη χρήση οδηγιών ordered για την οδηγία for.
- *OnlyTasks*. Η εκδοχή αυτή επιτρέπει μόνο την δημιουργία ομάδων παραλληλίας μεταξύ των eCores και την δημιουργία tasks από αυτά.
- *OnlyTasksICVS.* Η εκδοχή αυτή αποτελεί επέκταση της εκδοχής *OnlyTasks*, υλοποιώντας επιπλέον τις ρουτίνες υποστήριξης εκτέλεσης του OpenMP.

### 6.1.2 Δοκιμαστικά προγράμματα

Για την υποστήριξη της ιδέας που παρουσιάζεται στην διπλωματική εργασία, εκτελέστηκαν μετρήσεις επιδόσεων, οι οποίες αφορούν τις απαιτήσεις διαφόρων εφαρμογών σε χρόνο και μνήμη. Αρχικά γίνεται η παρουσίαση των αποτελεσμάτων που επιτυγχάνει η χρήση της πλήρους βιβλιοθήκης συσκευής για το Epiphany-III. Έπειτα τα αποτελέσματα συγκρίνονται με αυτά που επιτυγχάνει η προσαρμοζόμενη υποστήριξη εκτέλεσης, όπου η επιλογή βιβλιοθήκης συσκευής γίνεται βέλτιστα. Εκτελέστηκαν 2 είδη εφαρμογών, οι γενικές και αυτές της συλλογής δοκιμαστικών EPCC.

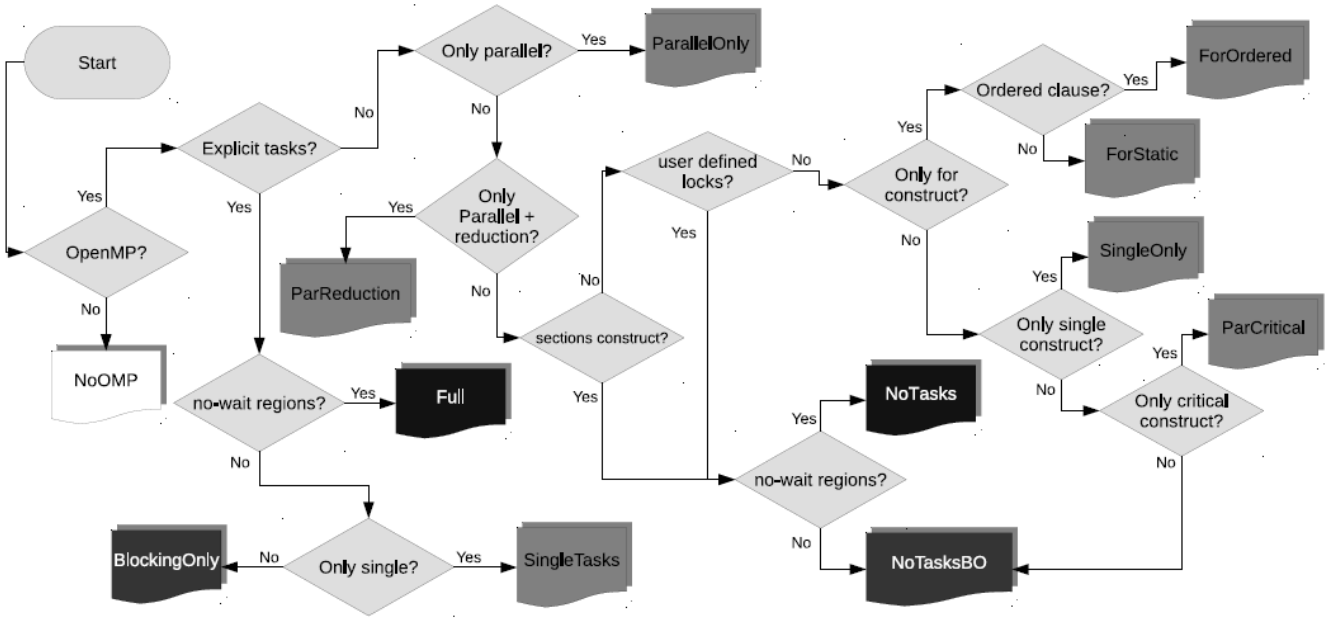
Η πρώτη γενική εφαρμογή που χρησιμοποιήθηκε ήταν η περίπτωση του κενού πυρήνα, όπου εκτελούμε μια περιοχή target στη συσκευή, η οποία δεν περιλαμβάνει καμία οδηγία OpenMP. Η δεύτερη εφαρμογή αφορά τον υπολογισμό του π=3.14159.. βάσει του κανόνα τραπεζίου, παράγοντας μία ομάδα παραλληλίας 16 νημάτων. Η τρίτη εφαρμογή αφορά τον υπολογισμό όλων των λύσεων του προβλήματος N-queens, για μέγεθος σκακιέρας N x N, έτσι ώστε καμία βασίλισσα να μην απειλεί κάποια άλλη. Συγκεκριμένα, η εφαρμογή N-queens τροποποιήθηκε καταλλήλως ώστε να τερματίζει μετά από κάποιο χρονικό διάστημα, λόγω της αδυναμίας εκτέλεσης ολόκληρου του πειράματος στο Epiphany-III. Τέλος, η τέταρτη εφαρμογή αφορά την εφαρμογή Game of life, το οποίο προσομοιώνει μια διαδικασία εξέλιξης και βασίζεται σε έναν πίνακα με ορισμένο πλήθος κελιών.

Για την εφαρμογή του κενού πυρήνα, ο μηχανισμός της προσαρμοζόμενης εκτέλεσης επέλεξε την εκδοχή βιβλιοθήκης *NoOpenMP*, λόγω της απουσίας οδηγιών OpenMP, επιτυγχάνοντας την μεγαλύτερη εξοικονόμιση σε μνήμη. Η εφαρμογή υπολογισμού του π παράγει μία ομάδα παραλληλίας, η οποία κάνει υπολογισμούς και τελικά αυτοί αθροίζονται με χρήση οδηγιών reduction. Συνεπώς, η εκδοχή που επιλέχθηκε αυτόματα ήταν η *OnlyParallelReduction*. Για τη λύση του προβλήματος N-queens, χρησιμοποιούνται περιοχές παραλληλίας, tasks, και οδηγίες single, συνεπώς ο mapper αποφάνθηκε ότι η κατάλληλη εκδοχή είναι η *OnlyTasksSingle*. Τέλος, η εφαρμογή Game of life κάνει χρήση ομάδων παραλληλίας και οδηγιών συγχρονισμού barrier, συνεπώς η κατάλληλη εκδοχή η οποία επιλέχθηκε είναι η *OnlyParallel*.

Για το δεύτερο είδος εφαρμογών, εκτελέστηκαν δοκιμαστικά προγράμματα της συλλογής EPCC. Τα προγράμματα αυτά χρησιμοποιούνται για την μέτρηση των επιβαρύνσεων που σχετίζονται με συγκεκριμένες οδηγίες OpenMP. Επομένως, ο μηχανισμός για κάθε οδηγία επέλεξε την εκδοχή βιβλιοθήκης που υλοποιεί μόνο αυτήν.

## 6.2 Πειράματα ορθότητας

Τα πειράματα ορθότητας ελέγχουν την σωστή απόφαση του μηχανισμού υποστήριξης εκτέλεσης. Εκτελέσαμε τα προγράμματα της ενότητας 6.1.2, αρχικά για να εξασφαλίσουμε ότι η απόφαση του Mapper είναι ορθή. Το αρχείο κανόνων




Σχήμα 6.1: Mapper decision flow

που χρησιμοποιήθηκε βασίζεται σε τροποποίηση του διαγράμματος ροής Mapper decision flow (Σχήμα 6.1) [5].

Βάσει των αποτελεσμάτων που λάβαμε κατά την εκτέλεση των πειραμάτων ορθότητας, αποδείχθηκε ότι επιτυγχάνεται σε ποσοστό 100% η αυτόματη επιλογή βιβλιοθήκης συσκευής από τον συνολικό μηχανισμό για το σύνολο των προγραμμάτων.

## 6.3 Πειράματα επιδόσεων

### 6.3.1 Μέγεθος κώδικα

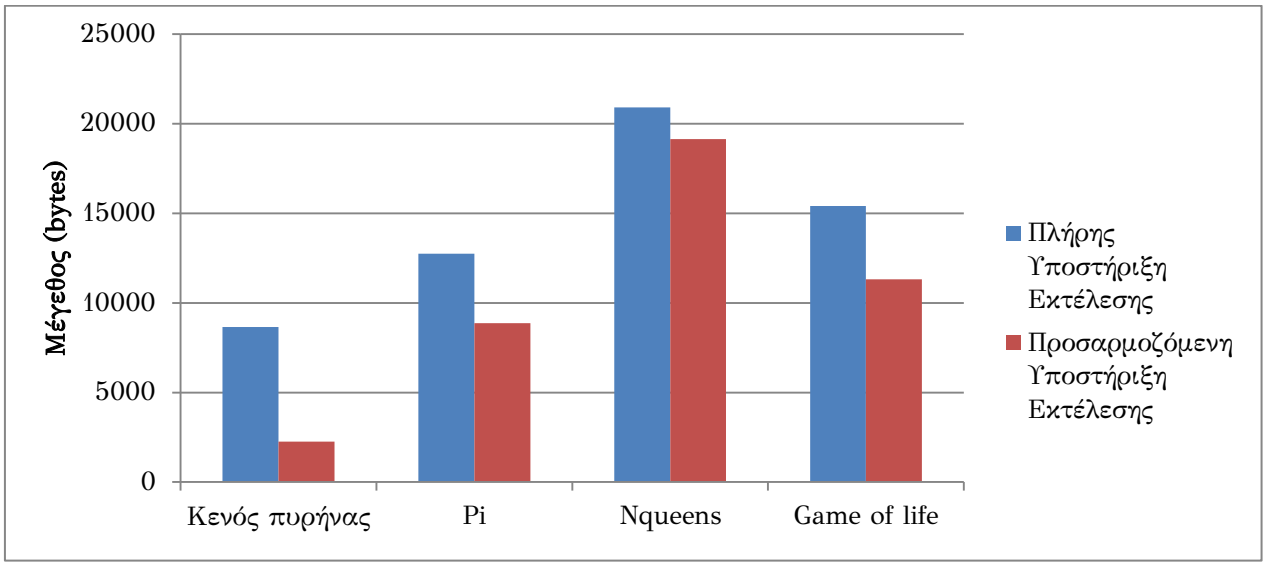
Στο Σχήμα 6.2 αναφέρονται τα αποτελέσματα που λάβαμε για το μέγεθος των εκτελέσιμων πυρήνων που παράγονται από τον κώδικα πυρήνα, με χρήση της πλήρους βιβλιοθήκης υποστήριξης εκτέλεσης και με χρήση της κατάλληλης εκδοχής που αποφάσισε ο mapper.

Αρχικά, στην περίπτωση του κενού πυρήνα συνταντούμε μια εξοικονόμιση 73,95% στην απαιτούμενη μνήμη. Αυτό συμβαίνει επειδή έχουν παραληφθεί όλες οι υλοποιήσεις για το OpenMP. Έτσι, επιτρέπεται στον προγραμματιστή να εκμεταλλευτεί τα ~6.5KB για δεδομένα της εφαρμογής. Έπειτα, στην περίπτωση του υπολογισμού του π, παρατηρούμε ότι εξοικονομήθηκαν περίπου 3.5KB, ενώ για την εφαρμογή N-queens η μείωση σε μέγεθος ήταν μικρή. Αυτό συμβαίνει γιατί ο μηχανισμός του tasking εισάγει ίσως από τις μεγαλύτερες επιβαρύνσεις χώρου σε μια βιβλιοθήκη. Τέλος, για την εφαρμογή Game of Life αλλά και για τα υπόλοιπα δοκιμαστικά προγράμματα, η μείωση στο μέγεθος κυμαίνεται μεταξύ 25% και 32%, εξοικονομώντας ~3.5KB.

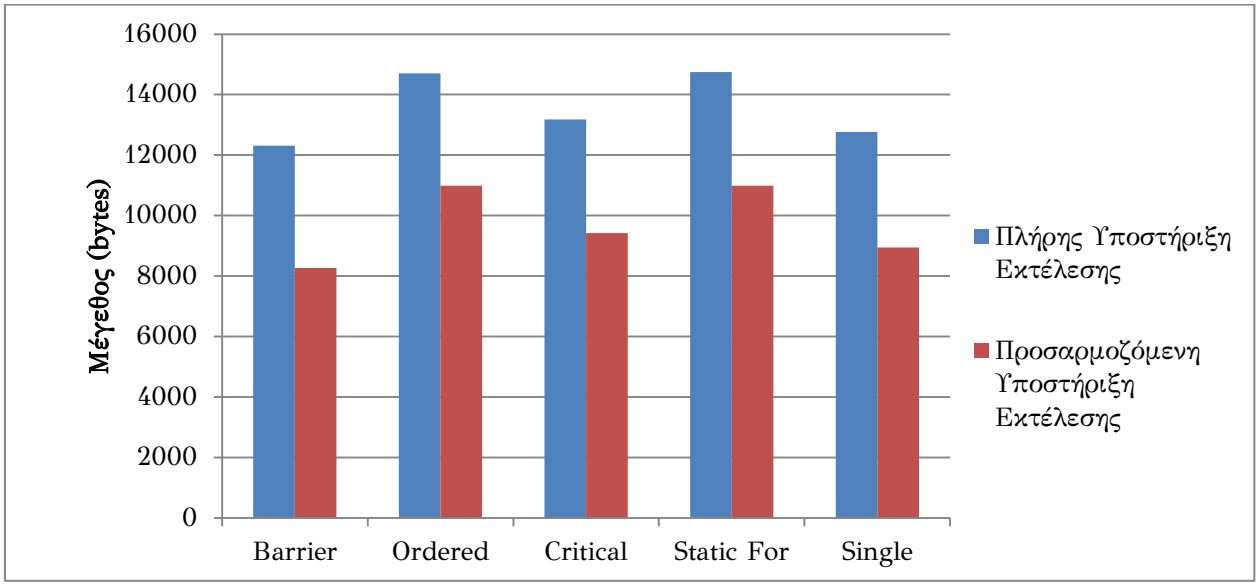
Οι μειώσεις είναι σημαντικές, αν ληφθεί υπ'όψη το γεγονός ότι η μνήμη του συνεπεξεργαστική Epiphany-III είναι πολύ περιορισμένη. Μειώνοντας την επιβάρυνση που εισάγει μια πλήρης βιβλιοθήκη συσκευής, επιτρέπουμε στον προγραμματιστή να εκμεταλλευθεί όσον το δυνατόν καλύτερα την διαθέσιμη μνήμη.

| Εφαρμογή | Βιβλιοθήκη πλήρους OpenMP (full) | Προσαρμοζόμενη Υποστήριξη Εκτέλεσης | Εξοικονόμιση |
|---|---|---|---|
| Κενός πυρήνας | 8648 | 2252 | 73,95% |
| Pi | 12744 | 8864 | 30,44% |
| N-queens | 20908 | 19148 | 8,41% |
| Game of life | 15412 | 11320 | 26,55% |
| EPCC Barrier | 12316 | 8268 | 32,86% |
| EPCC Ordered | 14704 | 10992 | 25,24% |
| EPCC Critical | 13184 | 9420 | 28,54% |
| EPCC Static For | 14744 | 10992 | 25,44% |
| EPCC Single | 12768 | 8944 | 29,94% |

Σχήμα 6.2: Μεγέθη εκτελέσιμων πυρήνων (σε bytes)




Σχήμα 6.3: Γράφημα μεγέθους κώδικα για γενικές εφαρμογές




Σχήμα 6.4: Γράφημα μεγέθους κώδικα για τις δοκιμές EPCC

### 6.3.2 Χρονικές επιδόσεις

Εκτός από την μέτρηση των μεγεθών των εφαρμογών, μετρήθηκε και ο χρόνος της εκτέλεσής τους. Οι εκδοχές οι οποίες χρησιμοποιήθηκαν προέκυψαν με αφαίρεση δομών και κώδικα, χωρίς κάποια άλλη ουσιαστική αλλαγή. Παρ'όλα αυτά, η μείωση αυτή του μεγέθους των εκδοχών πρέπει να είναι αρκετή για να επιτύχουμε καλύτερες επιδόσεις. Σε κάποιες περιπτώσεις το αποτέλεσμα είναι φανερό, ενώ σε

άλλες η επιλογή κατάλληλης εκδοχής βιβλιοθήκης δεν επηρέασε τις χρονικές επιδόσεις του προγράμματος. Τα αποτελέσματα που επιτεύχθηκαν με τη χρήση βιβλιοθήκης πλήρους OpenMP και προσαρμοζόμενης υποστήριξης εκτέλεσης, παρουσιάζονται παρακάτω.

| Εφαρμογή | Βιβλιοθήκη πλήρους OpenMP (full) | Προσαρμοζόμενη Υποστήριξη Εκτέλεσης | Μείωση |
|---|---|---|---|
| Κενός πυρήνας | 0,252071 | 0,212093 | 15,85% |
| Pi | 0,266510 | 0,257710 | 3,30% |
| N-queens | 1,824243 | 1,813211 | 0,60% |
| Game of life | 2,084800 | 1,374700 | 34,06% |

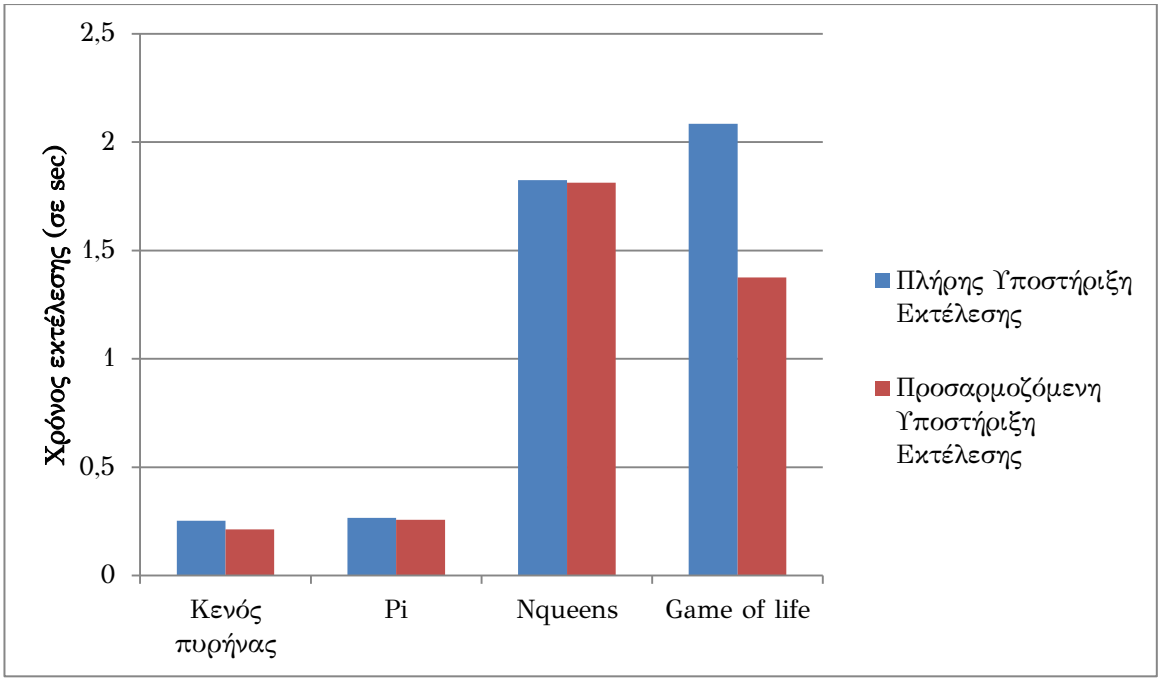
Σχήμα 6.5: Χρόνοι εκτέλεσης των γενικών εφαρμογών (σε sec)

Το Σχήμα 6.5 απεικονίζει τους χρόνους εκτέλεσης των 4 βασικών προγραμμάτων. Στην περίπτωση του κενού πυρήνα επιτυγχάνουμε μια μείωση 15,85% στον χρόνο εκτέλεσης, διότι απαιτείται λιγότερος χρόνος για την αποστολή του συνολικού κώδικα kernel σε καθένα από τα eCores, μιας και το μέγεθος της βιβλιοθήκης είναι μικρότερο. Στον υπολογισμό του π, η μείωση οφείλεται στο γεγονός ότι χρησιμοποιείται ένας ελαφρύτερος barrier. Γενικά, υποστηρίζονται δύο κατηγορίες barrier, αυτός που χρησιμοποιείται στις εκδοχές βιβλιοθηκών που υποστηρίζουν tasking, και ένας ελαφρύτερος για τον συγχρονισμό των νημάτων μετά από περιοχές παραλληλίας και διαμοιρασμού.

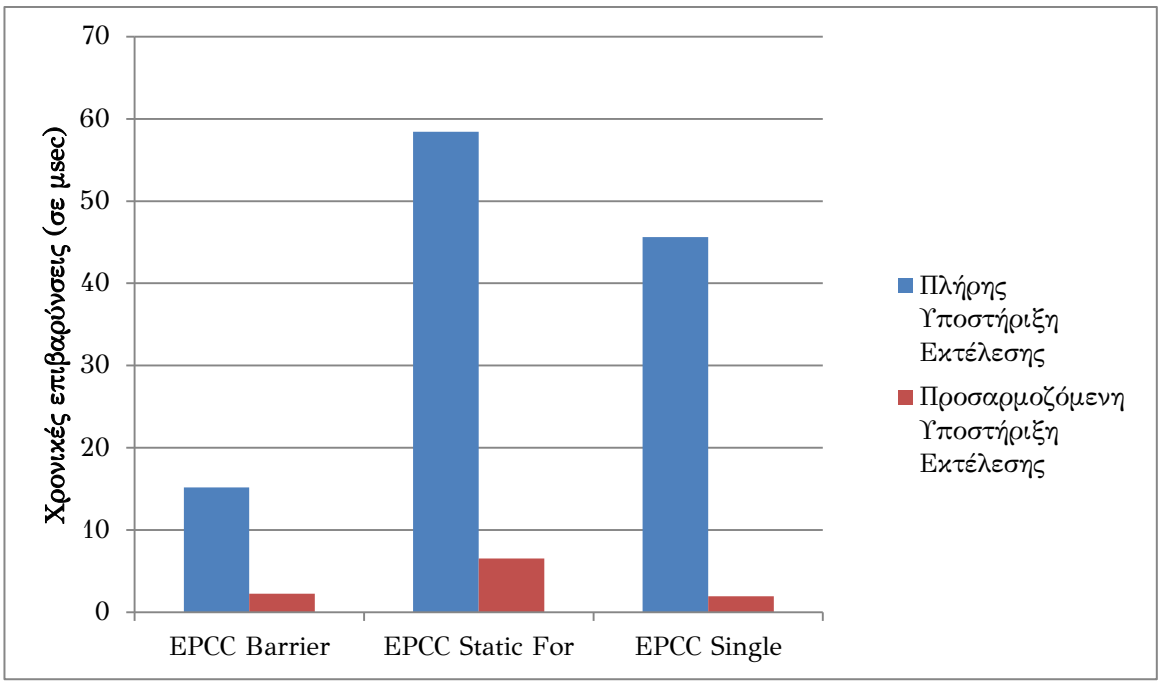
Επιπλέον, στο Game of Life, παρατηρείται μια μείωση 34,06% για τον ίδιο ακριβώς λόγο. Τέλος, στην περίπτωση του N-queens προβλήματος, η μείωση είναι αμελητέα.

| Εφαρμογή | Βιβλιοθήκη πλήρους OpenMP (full) | Προσαρμοζόμενη Υποστήριξη Εκτέλεσης | Μείωση |
|---|---|---|---|
| EPCC Barrier | 15,174295 | 2,284415 | 84,94% |
| EPCC Static For | 58,43120 | 6,545626 | 88,79% |
| EPCC Single | 45,61402 | 1,958293 | 95,70% |

Σχήμα 6.6: Χρονικές επιβαρύνσεις για τις δοκιμές EPCC (σε μsec)




Σχήμα 6.7: Γράφημα χρόνου εκτέλεσης των γενικών εφαρμογών




Σχήμα 6.8: Γράφημα χρονικών επιβαρύνσεων για τις δοκιμές EPCC

Οι χρονικές επιβαρύνσεις των μικροδοκιμών EPCC απεικονίζονται στο Σχήμα 6.6. Ορισμένα αποτελέσματα παραλείπονται (EPCC Ordered & EPCC Critical) διότι η μείωση του χρόνου ήταν αμελητέα. Στις περιπτώσεις Barrier, Static For και Single, παρατηρείται μια σημαντική μείωση στον απαιτούμενο χρόνο εκτέλεσης, διότι χρησιμοποιείται μια βελτιστοποιημένη έκδοση του barrier ο οποίος δεν περιλαμβάνει υποστήριξη για tasking. Τα πειράματα, συμφωνούν και επιβεβαιώνουν τα αποτελέσματα του [5], όπου, βέβαια, δεν γινόταν χρήση του αυτόματου μηχανισμού επιλογής flavor.

# Κεφάλαιο 7

# Επίλογος

## 7.1 Σύνοψη

Ο OMPi είναι ένας παραλληλοποιητικός μεταφραστής πηγαίου σε πηγαίο κώδικα (Source-to-source), που υποστηρίζει το πρότυπο OpenMP για την γλώσσα προγραμματισμού C. Υποστηρίζεται σε όλα τα συστήματα με αρχιτεκτονική συμβατή με το POSIX και προσφέρει ευελιξία στη χρήση του. Επιπλέον, είναι δομημένος με τέτοιο τρόπο ώστε να επιτρέπει την ανάπτυξη των ενοτήτων του χωρίς να υπάρχουν επικαλύψεις στον προγραμματισμό τους. Ο διαχωρισμός της λογικής του στις ενότητες compiler και runtime, καθώς και η υποστήριξη πολλαπλών βιβλιοθηκών για τις ανάγκες του προγραμματιστή, είναι τα βασικότερα πλεονεκτήματά του.

Η παρούσα διπλωματική εργασία, αν θα έπρεπε να τοποθετηθεί σε μια ενότητα, αυτή θα ήταν μεταξύ της μετάφρασης και της υποστήριξης εκτέλεσης. Το ζήτημα που παρουσιάστηκε αφορά στις βιβλιοθήκες υποστήριξης εκτέλεσης συσκευής, οι οποίες προκειμένου να υλοποιήσουν το σύνολο της υποδομής που είναι απαραίτητο για το OpenMP, μπορεί να προσθέτουν σημαντικές επιβαρύνσεις μνήμης και χρόνου στην ίδια τη συσκευή. Εξετάσαμε το γεγονός αυτό και καταλάβαμε πως τέτοιες βιβλιοθήκες είναι συνήθως αχρείαστες για εφαρμογές που χρησιμοποιούν ένα μικρό υποσύνολο των οδηγιών OpenMP. Επομένως, υλοποιήσαμε την ιδέα του μηχανισμού προσαρμοζόμενης εκτέλεσης, ώστε να επιτρέψουμε στον μεταφραστή να επιλέγει τις βέλτιστες εκδοχές των βιβλιοθηκών υποστήριξης εκτέλεσης, εξοικονομώντας απαιτούμενη μνήμη και χρόνο.

Από τα αποτελέσματα των δοκιμών που πραγματοποιήθηκαν, αποδείχθηκε ότι:

- Με την επιλογή κατάλληλης βιβλιοθήκης ανά πυρήνα συσκευής, είναι δυνατόν να επιτύχουμε σημαντική μείωση σε χώρο, μιας και οι εκδοχές περιέχουν υλοποιήσεις ενός μικρού υποσυνόλου του OpenMP.

- Μπορούμε να εξοικονομήσουμε απαιτούμενο χρόνο για την εκτέλεση του προγράμματος, μιας και κάθε εκδοχή είναι βελτιστοποιημένη για το εκάστοτε υποσύνολο οδηγιών που υποστηρίζει.

## 7.2 Μελλοντική εργασία

Οι δοκιμές πραγματοποιήθηκαν στη συσκευή Epiphany-III, η οποία αποτέλεσε ένα είδος πρόκλησης, λόγω του μικρού μεγέθους μνήμης που διαθέτει ανά επεξεργαστική μονάδα (32KB). Όμως, είναι σαφές ότι ο μηχανισμός μας είναι εντελώς γενικός και εφαρμόσιμος σε οποιαδήποτε άλλη συσκευή. Συνεπώς, ως μελλοντική εργασία είναι σημαντικό να συμπεριληφθεί η περεταίρω εξέλιξη του μηχανισμού, για την εφαρμογή του σε περισσότερες συσκευές. Αυτό πιθανώς μπορεί να γίνει είτε με την εύρεση νέων μετρικών, είτε με τον εμπλουτισμό της γλώσσας MAL ώστε να υποστηρίζει πολυπλοκότερα ερωτήματα.

Μια επιπλέον ιδέα που θα μπορούσε να υλοποιηθεί στο πλαίσιο της μελλοντικής εργασίας, είναι η διατήρηση στην διαδρομή εγκατάστασης του OMPi όλων των πηγαίων αρχειών της πλήρους βιβλιοθήκης συσκευής. Έτσι, μπορεί να επιτραπεί η επί τόπου παραμετροποίησή τους και η σύνδεσή τους με τον κώδικα πυρήνα. Ένα παράδειγμα παραμέτρου της εν λόγω διαδικασίας είναι είναι το πλήθος των κλειδαριών που χρησιμοποιούνται, βάσει του οποίου μπορούμε να αποφύγουμε σημαντικές επιβαρύνσεις σε δέσμευση μνήμης. Θεωρητικά αναμένεται να λάβουμε ακόμα καλύτερα αποτελέσματα στις επιδόσεις. Όμως, κρίνεται απαραίτητο η ιδέα αυτή να δοκιμαστεί και στην πράξη.

# Βιβλιογραφία


[1] OpenMP Architecture Review Board, "OpenMP Application Programming Interface", November 2015.

[2] M. Snir, J. Dongara, J. S. Kowalik, S. Huss-Lederman, S. W. Otto και D. W. Walker, MPI: The Complete Reference, The MIT Press, 1998.

[3] Adapteva, "Parallella Referance Manual", Sept. 2014.

[4] S.N. Agathos, V.V. Dimakopoulos, "Adaptive OpenMP Runtime System for Embedded Multicores", 16th Int'l Conference on Embedded and Ubiquitous Computing (EUC-2018), pp. 174--181, Bucharest, Romania, Oct. 2018

[5] S.N. Agathos, *Efficient OpenMP Runtime Support for General-Purpose and Embedded Multi-Core Platforms*, PhD Thesis, University of Ioannina, Apr. 2016.

[6] Michael L. Scott, Programming Language Pragmatics (Fourth Edition), Elsevier Inc., 2016, pp. 73-79.

# Παράρτημα Α

⟨*S*⟩ ::= ‘{’ ⟨*flavor-list*⟩ ‘,’ ⟨*node-list*⟩ ‘}’

⟨*flavor-list*⟩ ::= **flavors** ‘=’ ‘{’ ⟨*flavor*⟩ ‘}’

⟨*flavor*⟩ ::= ⟨*string*⟩ ‘,’ ⟨*flavor*⟩
| ⟨*string*⟩

⟨*node-list*⟩ ::= **nodes** ‘=’ ‘{’ ⟨*nodes*⟩ ‘}’

⟨*nodes*⟩ ::= ⟨*node*⟩ ‘,’ ⟨*nodes*⟩
| ⟨*node*⟩

⟨*node*⟩ ::= id. ‘=’ ‘{’ ⟨*node-body*⟩ ‘}’

⟨*node-body*⟩ ::= ⟨*bool-query*⟩ ‘,’ ⟨*bool-adj-list*⟩
| **num** ( id. ) ‘,’ ⟨*num-adj-list*⟩

⟨*bool-query*⟩ ::= **has** ‘(’ id. ‘)’ | **hasonly** ‘(’ id. ‘)’

⟨*bool-adj-list*⟩ ::= ⟨*bool-edge*⟩ ‘,’ ⟨*bool-edge*⟩

⟨*bool-edge*⟩ ::= ⟨*boolean*⟩ ‘:’ ⟨*string*⟩

⟨*num-adj-list*⟩ ::= ⟨*num-edge*⟩ ‘,’ ⟨*num-adj-list*⟩
| ⟨*num-edge*⟩

⟨*num-edge*⟩ ::= ⟨*relop*⟩ int. ‘:’ ⟨*string*⟩

⟨*string*⟩ ::= ‘"’ id. ‘"’ | id.

⟨*boolean*⟩ ::= **true** | **false**

⟨*relop*⟩ ::= ‘>’ | ‘<’ | ‘=’ | ‘>=’ | ‘<=’ | ‘!=’