Πτυχιακή Εργασία
Σιταράς Φώτιος
Αύγουστος 2009

Αποδοτική εκτέλεση προγραμμάτων OpenMP σε συστάδες Η/Υ

Επιβλέπων:
Βασίλειος Δημακόπουλος

Περιεχόμενα

Κεφάλαιο 1: Εισαγωγή......6
1.1 Η εξέλιξη των Η/Υ......6
1.2 Παράλληλα Συστήματα......6
1.2.1 Σύστημα Κοινής Μνήμης......7
1.2.2 Σύστημα Κατανεμημένης Μνήμης......7
1.3 Αντικείμενο Πτυχιακής Εργασίας......8
1.4 Δομή της Εργασίας......9
Κεφάλαιο 2: Το πρότυπο OpenMP......10
2.1 Εισαγωγή στο OpenMP......10
2.2 Προγραμματιστικό Μοντέλο......10
2.3 Οδηγίες OpenMP......11
2.3.1 Σύνταξη Οδηγιών......11
2.3.2 Παράλληλα Τμήματα......12
2.3.3 Οδηγίες Διαμοίρασης Έργου......13
2.3.4 Οδηγίες Συγχρονισμού......14
2.3.5 Συνθήκες Οδηγιών......14
2.4 Συναρτήσεις Βιβλιοθήκης......16
2.4.1 Συναρτήσεις Περιβάλλοντος Χρόνου Εκτέλεσης......16
2.4.2 Συναρτήσεις Κλειδαριών......17
2.5 Μεταβλητές Περιβάλλοντος......17
Κεφάλαιο 3: OMPi Compiler......19
3.1 Επισκόπηση του OMPi......19
3.2 O Compiler του OMPi......19
3.2.1 Διαδικασία Μετασχηματισμού για Νήματα......20
3.2.2 Διαδικασία Μετασχηματισμού για Διεργασίες......22
3.3 Το Σύστημα Runtime του OMPi......24
3.3.1 Είσοδος σε Παράλληλη Περιοχή......25
3.3.2 Χειρισμός Δομών Διαμοίρασης Έργου......25
3.3.3 Η Διεπαφή με το EELIB......26
3.4 Βιβλιοθήκες Νημάτων......28
3.5 OMPi και Clusters......28
Κεφάλαιο 4: Υποστήριξη Πολλαπλών Νημάτων Ανά Κόμβο......31
4.1 Εισαγωγή......31
4.2 Η Βιβλιοθήκη OPRC......32
4.2.1 Διαδικασία Αρχικοποίησης......32
4.2.2 Το Νήμα Server......34
4.2.3 Εκτέλεση Παράλληλης Περιοχής......35
4.2.4 Συγχρονισμός......38
4.2.5 Διαδικασία Περάτωσης Εκτέλεσης......39
4.3 Διαχείριση της ORT......39
Κεφάλαιο 5: Πειραματικά Αποτελέσματα......41
5.1 Εισαγωγικές Πληροφορίες για τα Πειράματα......41
5.2 EPCC Microbenchmarks......41
5.2.1 Πείραμα 1: Σταθερός αριθμός 4 νημάτων......41

5.2.2 Πείραμα 2: Σύγκριση με τη βιβλιοθήκη διεργασιών του OMPi..........................44
5.2.3 Πείραμα 3: Κλιμακωτή Εκτέλεση..46
5.3 Παράλληλες Εφαρμογές...47
Κεφάλαιο 6: Σύνοψη και Μελλοντική Εργασία...49

Κεφάλαιο 1

Εισαγωγή

1.1 Η εξέλιξη των Η/Υ

Σε όλη την ιστορία του, ο άνθρωπος προσπαθούσε να βελτιώσει και να διευκολύνει την καθημερινότητά του. Επόμενο ήταν λοιπόν, γύρω στον 17ο αιώνα να αρχίσει να κατασκευάζει και μηχανές υπολογισμών. Κάποιες μηχανές υπολόγιζαν λογαρίθμους, άλλες μπορούσαν να προσθαφαιρέσουν και σιγά σιγά με την επινόηση της άλγεβρας Boole και των διάτρητων καρτών, έμπαιναν οι βάσεις για την έναρξη μιας νέας επαναστατικής εποχής.

Την πρώτη περιγραφή ενός Η/Υ με την μορφή που έχει σε γενικές γραμμές σήμερα, την παρατηρούμε σε μια εργασία του *John von Neumann* το 1945 και ο πρώτος Η/Υ κάνει την εμφάνισή του το 1946 με το όνομα ENIAC. Η εργονομία του όμως για τα σημερινά τουλάχιστον δεδομένα είναι πολύ μικρή. Ο ENIAC είχε πολύ μεγάλο όγκο, ήταν θορυβώδης, αποτελούταν από πάρα πολλές λυχνίες που καίγονταν συχνά και κατανάλωνε πολύ μεγάλη ενέργεια. Ήταν όμως αδιαμφισβήτητα ένα μεγάλο βήμα προς το σήμερα.

Έκτοτε ο άνθρωπος επένδυσε αρκετή φαιά ουσία στο να βελτιστοποιήσει κάθε τομέα αυτού του εγχειρήματος. Είτε αυτό ήταν το μέγεθος, είτε η κατανάλωση ενέργειας, είτε η χωρητικότητα μνήμης, είτε ακόμα σημαντικότερα η επεξεργαστική ισχύς. Ενώ ο ENIAC μπορούσε να εκτελέσει 5000 προσθέσεις ανά δευτερόλεπτο, ένας σημερινός υπολογιστής μπορεί να εκτελέσει δισεκατομμύρια τέτοιες πράξεις. Ο άνθρωπος ξεκίνησε ένα κυνήγι με την ταχύτητα προσπαθώντας να φτιάξει όλο και πιο γρήγορους ηλεκτρονικούς υπολογιστές δημιουργώντας όλο και πιό γρήγορους επεξεργαστές. Η εξέλιξη αυτή όμως, βρήκε ένα τοίχο μπροστά της. Ο άνθρωπος πλέον περιοριζόταν από τους νόμους της φύσης και το υλικό και έτσι θα έπρεπε να βρει νέους τρόπους.

Ας το δούμε ως μια καθημερινή δραστηριότητα. Αν σε ένα σπίτι έχουμε να κάνουμε 3 δουλειές. Τότε θα ξεκινήσει ένα άτομο (επεξεργαστής) και θα τις τελειώσει σε 3 χρονικές στιγμές. Αν όμως είχαμε 3 άτομα (επεξεργαστές), τότε οι δουλειές θα τελείωναν σε 1 χρονική στιγμή αν αναθέταμε σε κάθε άτομο μια δουλειά και αυτά την εκτελούσαν παράλληλα. Η λύση λοιπόν δόθηκε από τα παράλληλα συστήματα και επόμενο ήταν να αναπτυχθούν νέοι κλάδοι στην πληροφορικοί που έχουν ως αντικείμενο την παράλληλη επεξεργασία και τον παράλληλο προγραμματισμό.

1.2 Παράλληλα Συστήματα

Όταν μιλάμε για παράλληλα συστήματα ουσιαστικά αναφερόμαστε σε ένα σύστημα με πολλές επεξεργαστικές μονάδες (CPUs) που συνεργάζονται και επικοινωνούν για την επίτευξη του αποτελέσματος. Οι επεξεργαστές δηλαδή αναλαμβάνουν διάφορα μέρη του προβλήματος και στη συνέχεια επικοινωνούν για να συγκεντρώσουν και να συνδυάσουν τα επιμέρους αποτελέσματα τους ώστε να συμπληρώσουν το τελικό επιθυμητό αποτέλεσμα.

Η ανάπτυξη των παράλληλων συστημάτων έχει δημιουργήσει νέες απαιτήσεις στους προγραμματιστές. Για να προγραμματίσει κάποιος ένα παράλληλο σύστημα χρειάζεται να λάβει υπόψιν του την αρχιτεκτονική και να διαχειριστεί με ειδικό τρόπο τα συστατικά του προγράμματος (συναρτήσεις, μεταβλητές κ.α.) καθώς πλέον θα είναι κοινόχρηστα για περισσότερες υπολογιστικές οντότητες (νήματα, διεργασίες). Υπάρχουν δύο βασικά συστήματα παράλληλων αρχιτεκτονικών. Το σύστημα κοινής μνήμης και το σύστημα κατανεμημένης μνήμης. Ανάλογα με το σύστημα που έχουμε, ο προγραμματιστής θα πρέπει να ακολουθήσει διαφορετικές προγραμματιστικές λογικές και μεθόδους. Το ποιο είναι πιο αποδοτικό εξαρτάται από το πρόβλημα που έχουμε να αντιμετωπίσουμε.

1.2.1 Σύστημα Κοινής Μνήμης

Στα συστήματα κοινής μνήμης οι επεξεργαστές έχουν πρόσβαση σε ένα κοινό χώρο διευθύνσεων. Η επικοινωνία μεταξύ των επεξεργαστών γίνεται μέσω αυτής της κοινής μνήμης αλλάζοντας τις τιμές των μεταβλητών. Οι μεταβλητές καθώς και οι αλλαγές που γίνονται σε αυτές, είναι ορατές από όλους τους επεξεργαστές που έχουν πρόσβαση στην κοινή μνήμη.

Μπορεί όλη αυτή η λογική να φαίνεται αποτελεσματική και απλά δομημένη και πράγματι είναι, όμως για μικρό αριθμό επεξεργαστών. Αυτό συμβαίνει για το λόγο οτι σε τέτοια συστήματα η μορφή του δικτύου διασύνδεσης μεταξύ των επεξεργαστών είναι ένας δίαυλος. Αν αυξήσουμε τον αριθμό των επεξεργαστών, θα αυξηθεί και ο αριθμός των επικοινωνιών, αλλά η χωρητικότητα του διαύλου μένει σταθερή με αποτέλεσμα η απόδοση να πέφτει. Έτσι δεν μπορούμε να έχουμε τέτοια συστήματα με μεγάλο αριθμό επεξεργαστών.

Για τον προγραμματισμό, τα παράλληλα συστήματα κοινής μνήμης ακολουθούν το μοντέλο κοινού χώρου διευθύνσεων. Γίνεται χρήση είτε διεργασιών, είτε νημάτων που δημιουργούνται από τη διεργασία εκτέλεσης του προγράμματος και διαθέτουν κοινόχρηστες μεταβλητές. Βέβαια σε αυτό το μοντέλο χρησιμοποιούνται περισσότερο τα νήματα καθώς οι διεργασίες έχουν δικό τους ξεχωριστό χώρο διευθύνσεων η κάθε μια και θα πρέπει η ύπαρξη κοινής μνήμης να διασφαλιστεί με κλήσεις συστήματος. Αντίθετα τα νήματα που έχουν δημιουργηθεί από την ίδια διεργασία, “βλέπουν” το χώρο διευθύνσεων της διεργασίας που τα δημιούργησε αλλά έχουν δικό τους μετρητή προγράμματος.

Ακόμα όμως και με την ύπαρξη των διεργασιών και των νημάτων, ο προγραμματιστής για να προγραμματίσει σε μοντέλο κοινού χώρου διευθύνσεων θα πρέπει πάλι να λάβει υπόψιν του τον τρόπο λειτουργίας αυτών. Για αυτό το λόγο, έχει αναπτυχθεί ένα πρότυπο προγραμματισμού για παράλληλα συστήματα κοινής μνήμης, το OpenMP. Για το OpenMP γίνεται εκτενής αναφορά στο Κεφάλαιο 2, μιας και αποτελεί μέρος αυτής της εργασίας.

1.2.2 Σύστημα Κατανεμημένης Μνήμης

Στο σύστημα κατανεμημένης μνήμης, οι επεξεργαστές έχουν ο καθένας τη δικιά του ιδιωτική μνήμη και συνδέονται μεταξύ τους μέσω ενός δικτύου με το οποίο επικοινωνούν. Κάθε επεξεργαστής μπορεί άμεσα να προσπελάσει και να επεξεργαστεί μόνο τα δεδομένα που βρίσκονται στην ιδιωτική του μνήμη. Αν κάποιος επεξεργαστής

επιθυμεί να προσπελάσει κάποιο δεδομένο (μεταβλητή) που δεν έχει στη μνήμη του, τότε στέλνει ένα μήνυμα στο δίκτυο το οποίο θα παραλάβει ο επεξεργαστής που έχει το δεδομένο. Στη συνέχεια ο τελευταίος θα του το στείλει με "μήνυμα" στο δίκτυο. Ένα σύστημα κατανεμημένης μνήμης είναι το cluster. Πρόκειται ουσιαστικά για μια συστάδα υπολογιστών συνδεδεμένων μεταξύ τους με ένα δίκτυο οι οποίοι συνεργάζονται για την επίλυση ενός προβλήματος.

Η αλήθεια είναι οτι σε αυτό το μοντέλο δεν έχουμε το πρόβλημα μεγέθους που είχαμε στο μοντέλο κοινής μνήμης. Εδώ μπορούμε να έχουμε οσοδήποτε αριθμό κόμβων και άρα επεξεργαστών και οποιαδήποτε τοπολογία δικτύου με την οποία αυτοί θα συνδέονται. Βέβαια αν το πρόβλημα μας απαιτεί οτι πολλά δεδομένα θα πρέπει να είναι προσπελάσιμα από πολλές διεργασίες, τότε ο φόρτος επικοινωνιών γίνεται μεγάλος και ως επόμενο έχουμε επιβάρυνση στο χρόνο εκτέλεσης του προγράμματος που φτάνει ακόμα και σε χειρότερες επιδόσεις από την περίπτωση του σειριακού.

Για τον προγραμματισμό ενός συστήματος κατανεμημένης μνήμης ακολουθείται το μοντέλο μεταβίβασης μηνυμάτων. Χρησιμοποιούνται συνήθως διεργασίες σε αντιστοιχία 1 προς 1 με κάθε επεξεργαστή. Όπως είπαμε η επικοινωνία σε αυτό το μοντέλο γίνεται με τη βοήθεια μηνυμάτων μέσω του δικτύου διασύνδεσης των επεξεργαστών. Συνήθως για τον προγραμματισμό όλης της διαδικασίας ανταλλαγής μηνυμάτων χρησιμοποιείται το πακέτο MPI [1]. Το MPI προσφέρει ρουτίνες για την αποστολή και παραλαβή μηνυμάτων, καθώς και ρουτίνες που προσφέρουν μια ποικιλία από λειτουργίες χωρίς να χρειάζεται ο προγραμματιστής να ασχοληθεί ή ακόμα και να γνωρίζει τις λεπτομέρειες του δικτύου που υπάρχει από κάτω.

Η χρήση της επικοινωνίας μέσω μηνυμάτων μεταξύ των κόμβων, προκύπτει από το γεγονός, οτι στο μοντέλο αυτό δεν έχουμε κοινόχρηστη μνήμη. Σιγά σιγά αναπτύχθηκε η λογική της δημιουργίας εικονικής κοινής μνήμης επάνω από τη φυσικά κατανεμημένη μνήμη, από το TreadMarks [2]. Πρόκειται ουσιαστικά για κοινή μνήμη που υλοποιείτε από λογισμικό, το λεγόμενο sDSM, δίνοντας την ψευδαίσθηση ύπαρξής της στους επεξεργαστές του μοντέλου. Τη λογική αυτή ακολούθησαν και άλλα sDSM συστήματα παρουσιάζοντας ίδια λειτουργικότητα αλλά διαφορές στον τρόπο διατήρησης της συνέπειας μνήμης.

Άξιο επισήμανσης λόγω της σχετικότητας με το αντικείμενο της παρούσας πτυχιακής εργασίας, είναι και η ύπαρξη clusters που αποτελούνται από μονάδες συστημάτων κοινής μνήμης. Με λίγα λόγια συνδυάζει τις δύο προαναφερθείσες αρχιτεκτονικές όπου πολλοί υπολογιστές που αποτελούνται από πολλές επεξεργαστικές μονάδες συνδυάζονται και επικοινωνούν για την επίλυση ενός προβλήματος.

1.3 Αντικείμενο Πτυχιακής Εργασίας

Το αντικείμενο της εργασίας αφορά τη βελτιστοποίηση του OMPi στην εκτέλεση προγραμμάτων σε συστάδες υπολογιστών. Ο OMPi (Κεφάλαιο 3) είναι ένας μεταγλωττιστής source-to-source προγραμμάτων OpenMP (Κεφάλαιο 2) σε γλώσσα προγραμματισμού C. O OMPi ήδη υποστηρίζει εκτέλεση προγραμμάτων OpenMP σε συστάδες υπολογιστών χωρίς όμως να εκμεταλλεύεται την κοινή μνήμη που μπορεί να υπάρχει σε κάποιο κόμβο, στην περίπτωση που αυτός είναι αρχιτεκτονικής κοινής μνήμης.

Το OpenMP όπως αναφέρθηκε προηγουμένως είναι ένα προγραμματιστικό πρότυπο για προγραμματισμό σε μοντέλο κοινού χώρου διευθύνσεων. Σε συστάδες υπολογιστών (clusters), δεν υπάρχει κοινόχρηστη μνήμη και πρέπει οι επικοινωνίες να γίνουν μέσω μηνυμάτων. Για να εκτελέσουμε προγράμματα OpenMP σε συστάδες υπολογιστών χρησιμοποιούμε ειδικό λογισμικό που μας παρέχει εικονική κοινή μνήμη, το SVM (Shared Virtual Memory). Ο OMPi σε αντίθεση με άλλους μεταφραστές που χρησιμοποιούν και είναι στενά συνδεδεμένοι με ένα SVM, μπορεί να λειτουργήσει με πολλά SVM όπως το Mocha, το Treadmarks κ.α. Επίσης χρησιμοποιεί ένα υβριδικό μοντέλο επικοινωνιών, καθώς για τις κοινές μεταβλητές χρησιμοποιεί την κοινόχρηστη μνήμη ενώ για τις επικοινωνίες μεταξύ των διεργασιών χρησιμοποιεί MPI.

Ο OMPi αποτελείται από πολλές βιβλιοθήκες χρόνου-εκτέλεσης οι οποίες διαφέρουν ως προς τις υπολογιστικές οντότητες που παρέχουν. Υπάρχουν βιβλιοθήκες νημάτων πυρήνα (POSIX, Solaris), νημάτων χρήστη (POSIX, Marcell) και διεργασιών. Η τελευταία είναι απαραίτητη στον OMPi, για την εκτέλεση προγραμμάτων OpenMP σε συστάδες υπολογιστών καθώς δημιουργεί διεργασίες ως υπολογιστικές οντότητες για την εκτέλεση του προγράμματος, οι οποίες επικοινωνούν μέσω μηνυμάτων. Στην περίπτωση που οι κόμβοι είναι αρχιτεκτονικής κοινής μνήμης μπορεί να δημιουργήσει περισσότερες από μια διεργασίες για να αξιοποιήσει την επεξεργαστική ισχύ των πυρήνων του κόμβου, αυτές όμως θα επικοινωνούν μέσω MPI και άρα δεν μπορούν να εκμεταλλευτούν την ύπαρξη κοινής μνήμης. Στόχος λοιπόν είναι η εκμετάλλευση αυτής της κοινής μνήμης καθώς και της επεξεργαστικής ισχύος αυτών των κόμβων με την δημιουργία και προσάρτηση μιας βιβλιοθήκης χρόνου-εκτέλεσης (runtime library) στις ήδη υπάρχουσες βιβλιοθήκες του OMPi, η οποία:

- Παρέχει τοπικά νήματα επιπέδου-πυρήνα σε κάθε κόμβο αντί για διεργασίες. Τα νήματα χαρακτηρίζονται ως “ελαφριές διεργασίες” καθώς έχουν μικρότερο κόστος δημιουργίας και διαχείρισης από τις διεργασίες.
- Χρησιμοποιεί την φυσική τοπική κοινόχρηστη μνήμη του κόμβου για την επικοινωνία των νημάτων και περιορίζει όσο το δυνατόν τις επικοινωνίες MPI μεταξύ των οντοτήτων υπολογισμού.

1.4 Δομή της Εργασίας

Ακολουθεί μια περιληπτική περιγραφή της δομής των κεφαλαίων της παρούσας εργασίας.

- Κεφάλαιο 2: Παρουσίαση και περιγραφή του προτύπου OpenMP για παράλληλο προγραμματισμό πάνω σε μοντέλο κοινού χώρου διευθύνσεων. Γίνεται επίσης και ανάλυση των εντολών που αποτελούν το πρότυπο.
- Κεφάλαιο 3: Παρουσίαση και περιγραφή του μετασχηματιστή OMPi. Αναλύεται ο τρόπος λειτουργίας καθώς και τα μέρη που τον αποτελούν.
- Κεφάλαιο 4: Ο OMPi χρησιμοποιώντας την νέα βιβλιοθήκη χρόνου-εκτέλεσης που προστέθηκε για καλύτερη συμπεριφορά σε εκτέλεση προγραμμάτων OpenMP σε συστάδες υπολογιστών.
- Κεφάλαιο 5: Περιγράφονται και αναλύονται τα αποτελέσματα από την εκτέλεση διάφορων εφαρμογών για την συγκριτική μελέτη του OMPi με τη νέα βιβλιοθήκη.

- Κεφάλαιο 6: Γίνεται ένας επίλογος της όλης εργασίας και δίνονται κάποιες συμβουλές για περαιτέρω μελλοντική εργασία πάνω στον OMPi.

Κεφάλαιο 2

Το πρότυπο OpenMP

2.1 Εισαγωγή στο OpenMP

Η όλο και συνεχόμενη ανάπτυξη των παράλληλων συστημάτων, ειδικότερα κοινής μνήμης, έφερε και νέες προγραμματιστικές απαιτήσεις στους προγραμματιστές αυτών των συστημάτων. Ο προγραμματιστής θα πρέπει να διαχειριστεί με πιό εξειδικευμένο τρόπο τα συστατικά του προγράμματος (συναρτήσεις, μεταβλητές κ.α.) καθώς αυτά προσπελάζονται από περισσότερες οντότητες και πρέπει να προστατευθούν για να είναι συνεπείς. Η δημιουργία ενός παράλληλου προγράμματος απέχει πολύ σε όγκο εντολών από οτι ένα αντίστοιχο σειριακό πρόγραμμα που κάνει την ίδια δουλειά. Λόγω αυτών των δυσκολιών, δημιουργήθηκε η ανάγκη για τη δημιουργία ενός προτύπου για την διευκόλυνση της συγγραφής παράλληλων προγραμμάτων. Αυτό το πρότυπο είναι το OpenMP (Open Multi-Processing).

Το OpenMP [3] είναι μια διεπαφή δημιουργίας παράλληλων εφαρμογών σε συστήματα κοινής μνήμης. Υποστηρίζει τις γλώσσες προγραμματισμού Fortran, C και C++ και υποστηρίζεται από πολλές αρχιτεκτονικές. Αποτελείται από ένα σύνολο:

- ***Οδηγιών προς τον μεταγλωττιστή.*** Τα λεγόμενα pragmas ή directives, τα οποία εισάγει ο προγραμματιστής στο πρόγραμμα του. Ουσιαστικά πρόκειται για οδηγίες που δίνει ο προγραμματιστής στο μεταγλωττιστή για το τι θέλει να παραλληλοποιήσει και πως.
- ***Συναρτήσεις βιβλιοθήκης.*** Είναι συναρτήσεις τις οποίες καλεί ο προγραμματιστής σε οποιοδήποτε σημείο του προγράμματος για να αλλάξει των αριθμό των νημάτων που θα εκτελέσουν μια παράλληλη περιοχή, να χρησιμοποιήσει τις κλειδαριές του OpenMP κ.α.
- ***Μεταβλητές περιβάλλοντος.*** Είναι μεταβλητές που ορίζουν εκ των προτέρων τον τρόπο εκτέλεσης των παράλληλων περιοχών του προγράμματος.

Η συγγραφή ενός προγράμματος OpenMP θα μπορούσαμε να πούμε οτι είναι κλιμακωτή καθώς μπορεί ο προγραμματιστής να δημιουργήσει το σειριακό πρόγραμμα και πολύ εύκολα εισάγοντας τις οδηγίες OpenMP στο πρόγραμμα του, σταδιακά να το κάνει παράλληλο. Βέβαια το να αποκρύπτονται οι λεπτομέρειες της παραλληλίας και της διαχείρισης των νημάτων από τον προγραμματιστή δεν είναι και πάντα θετικό. Το OpenMP δεν εξασφαλίζει την μέγιστη αποδοτικότητα της παραλληλίας σε κοινή μνήμη και επίσης δεν είναι εφικτή η διαχείριση ενός υποσυνόλου νημάτων από τα νήματα που θα εκτελέσουν ένα μέρος του κώδικα. Παρόλα στα θετικά του OpenMP θα μπορούσαμε να αναφέρουμε τη δυνατότητα υποστήριξης δυναμικού ορισμού του αριθμού των νημάτων που θα εκτελέσουν μια παράλληλη περιοχή OpenMP κατα τη διάρκεια του προγράμματος καθώς και την υποστήριξη λεπτού και αδρού καταμερισμού του παραλληλισμού.

2.2 Προγραμματιστικό Μοντέλο

Το OpenMP όπως αναφέρθηκε προηγουμένως είναι ένα πρότυπο παράλληλου

προγραμματισμού σε συστήματα κοινής μνήμης. Επομένως το προγραμματιστικό του μοντέλο, αφορά τη δημιουργία και τη συνύπαρξη νημάτων για την εκτέλεση του προγράμματος. Τα νήματα αυτά επικοινωνούν με κοινές μεταβλητές. Η δημιουργία τους είναι πολύ κοντά στην λογική fork-join που έχουμε στις διεργασίες.

```
int main () {
      int id;

      omp_set_num_threads(4);
      #pragma omp parallel
      {
            int id = omp_get_thread_num();
            printf("Hello, i am thread #%d\n", id);
      }
      return 0;
}
```

<u>Έξοδος:</u>

```
Hello, i am thread #0
Hello, i am thread #1
Hello, i am thread #2
Hello, i am thread #3
```

Το αρχικό νήμα του προγράμματος, το νήμα δηλαδή που ξεκινάει την εκτέλεση του προγράμματος, όταν συναντήσει μια περιοχή που πρέπει να παραλληλοποιηθεί (`#pragma omp parallel`), δημιουργεί μια ομάδα νημάτων των οποίων ο αριθμός ορίζεται είτε δυναμικά, είτε στατικά. Αυτά με την σειρά τους θα εκτελέσουν το τμήμα παραλληλίας που τους αντιστοιχεί. Για το παραπάνω παράδειγμα, το αρχικό νήμα θα δημιουργήσει μια ομάδα 4 νημάτων (`omp_set_num_threads(4)`) που θα εκτελέσουν τον κώδικα που βρίσκεται μέσα στο block του parallel. Το αρχικό νήμα περιμένει στο τέλος της παράλληλης περιοχής τα νήματα που δημιούργησε να τελειώσουν τη δουλειά τους, οπότε και καταστρέφεται και η ομάδα. Στη συνέχεια το αρχικό νήμα συνεχίζει την εκτέλεση του κώδικα σειριακά. Τα νήματα δηλαδή συγχρονίζονται και επικοινωνούν σε ένα πρόγραμμα OpenMP αυτόματα. Βέβαια η κοινοχρησία των πόρων απαιτεί και την προστασία αυτών, αλλά το OpenMP προσφέρει πληθώρα οδηγιών και παραμέτρων για αυτή τη δουλειά.

Λόγω της παραπάνω λογικής που ακολουθείται από το αρχικό νήμα για τη δημιουργία νημάτων, η συγγραφή ενός OpenMP προγράμματος θα μπορούσαμε να πούμε οτι είναι κλιμακωτή. Γράφουμε το παράλληλο πρόγραμμα αρχικά όπως θα γράφαμε ένα αντίστοιχο σειριακό και στη συνέχεια προσθέτουμε τις οδηγίες στις περιοχές που θέλουμε να παραλληλοποιήσουμε. Έτσι βαθμιαία το πρόγραμμα από σειριακό, μετατρέπετε σε παράλληλο με νήματα.

2.3 Οδηγίες OpenMP

Σε αυτό την την ενότητα θα ασχοληθούμε με τις οδηγίες του OpenMP πάνω στις γλώσσες C και C++, καθώς η παρούσα πτυχιακή εργασία και γενικότερα ο OMPi, αφορά των μετασχηματισμό των οδηγιών του OpenMP πάνω σε αυτές τις γλώσσες.

Στις ενότητες που ακολουθούν, αναλύεται η σύνταξη καθώς και οι παράμετροι των οδηγιών αυτών. Αναλύεται επίσης η δομή της παράλληλης περιοχής και γίνεται μια γενικότερη περιγραφή των οδηγιών αυτών χωρίζοντάς τες σε οδηγίες διαμοιρασμού εργασίας (workshare regions) και οδηγίες συγχρονισμού.

2.3.1 Σύνταξη Οδηγιών

Οι οδηγίες OpenMP ακολουθούν μια καθορισμένη σύνταξη που χωρίζεται σε τρία μέρη. Πρώτα εμφανίζεται η έκφραση ***#pragma omp***. Όλες οι οδηγίες στο πρότυπο ξεκινούν με αυτόν τον τρόπο. Δεν ορίζει κάτι συγκεκριμένο και ουσιαστικό ως προς τον τρόπο που θα μετασχηματιστεί το τμήμα που ακολουθεί. Στη συνέχεια τοποθετείται το όνομα της οδηγίας. Υπάρχουν οδηγίες συγχρονισμού και οδηγίες παραλληλίας. Τελευταίες μετά το όνομα της οδηγίας τοποθετούνται οι συνθήκες της περιοχής που ακολουθεί γενικότερα. Εδώ δεν έχουμε κάποια ειδική σειρά στον τρόπο με τον οποίο θα τοποθετήσουμε τις συνθήκες των οδηγιών. Οι οδηγίες θα πρέπει να εφαρμόζονται σε δομημένα τμήματα κώδικα, με ένα σημείο εισόδου και ένα σημείο εξόδου.

Από τη στιγμή που οι οδηγίες OpenMP είναι pragmas και directives πρόκειται ουσιαστικά για οδηγίες προς τον μεταγλωττιστή. Είναι οδηγίες που "λένε" στον μεταγλωττιστή με ποιό τρόπο θα μετασχηματίσει το block εντολών που βρίσκεται μέσα στις αγκύλες του pragma. Για αυτό ακριβώς το λόγο, αν ο μεταγλωττιστής δεν υποστηρίζει OpenMP απλά αγνοεί τα pragmas και τα directives και μεταφράζει το πρόγραμμα σαν να μην υπήρχαν. Ουσιαστικά το πρόγραμμα γίνεται σειριακό.

2.3.2 Παράλληλα Τμήματα

Για να ορίσουμε ένα τμήμα κώδικα ως παράλληλο τμήμα θα πρέπει να τοποθετήσουμε στην αρχή του την έκφραση ***#pragma omp parallel…*** και ο κώδικας που θα βρίσκεται μέσα σε αγκύλες θα εκτελεστεί παράλληλα. Όταν το αρχικό νήμα του προγράμματος, το νήμα master δηλαδή, συναντήσει αυτήν την έκφραση, θα δημιουργήσει μια ομάδα με τον απαιτούμενο αριθμό νημάτων και κάθε ένα από αυτά θα εκτελέσει τον κώδικα που βρίσκεται μέσα στις αγκύλες. Στην ομάδα των νημάτων που θα εκτελέσουν τον κώδικα συμπεριλαμβάνεται και το νήμα master και μάλιστα έχει αριθμό αναγνωριστικού το 0. Στο τέλος της περιοχής υπονοείται πως υπάρχει συγχρονισμός μεταξύ των νημάτων, δηλαδή θα πρέπει να φτάσουν όλα εκεί για να τελειώσει η παράλληλη περιοχή και να συνεχίσει το αρχικό νήμα, την εκτέλεση του υπόλοιπου προγράμματος.

Μπορούμε μέσα σε ένα παράλληλο τμήμα να τοποθετήσουμε ένα ή και περισσότερα παράλληλα τμήματα. Τα τελευταία ονομάζονται εμφωλευμένα (nested). Το OpenMP υποστηρίζει εμφωλευμένη παραλληλία αλλά εξαρτάται από τον μεταγλωττιστή αν θα την υλοποιήσει και πως. Κάθε νήμα που βρίσκει μια εμφωλευμένη παράλληλη περιοχή, παίρνει το ρόλο του νήματος master για αυτή την παράλληλη περιοχή. Δημιουργεί δηλαδή την ομάδα με τον απαιτούμενο αριθμό νημάτων που θα εκτελέσουν τον κώδικα της εμφωλευμένης περιοχής και αυτά εκτελούν για λογαριασμό τους τον κώδικα που τους αντιστοιχεί. Και σε αυτήν την περίπτωση το νήμα master που δημιούργησε την ομάδα συμμετέχει σε αυτήν και εκτελεί και αυτό τον κώδικα που του αντιστοιχεί. Μπορούμε να έχουμε πολλά επίπεδα παραλληλίας στα οποία ακολουθείτε η παραπάνω λογική.

Υπάρχουν πολλοί τρόποι για να ορίσουμε το πόσα νήματα θα εκτελέσουν μια παράλληλη περιοχή. Το OpenMP προσφέρει συναρτήσεις βιβλιοθήκης, μεταβλητές περιβάλλοντος καθώς και συνθήκες που τοποθετούνται στις εκφράσεις των οδηγιών με τις οποίες μπορούμε να διαχειριστούμε των αριθμό των νημάτων που θα εκτελέσουν μια παράλληλη περιοχή. Εκ των προτέρων, πριν την εκτέλεση δηλαδή του προγράμματος μπορούμε με την μεταβλητή περιβάλλοντος `OMP_NUM_THREADS`, με την κλήση της συνάρτησης `omp_set_num_threads()` μέσα στο πρόγραμμα και τοποθετώντας τη συνθήκη num_threads() στην έκφραση του pragma είναι οι τρεις τρόποι που μας παρέχει το OpenMP για να ορίσουμε τον αριθμό των νημάτων. Το OpenMP προσφέρει και τη δυνατότητα ο αριθμός των νημάτων να προσαρμόζεται δυναμικά για συγκεκριμένη παράλληλη περιοχή. Αυτό επιτυγχάνεται με δύο τρόπους, είτε εκ των προτέρων πριν την εκτέλεση του προγράμματος με την μεταβλητή περιβάλλοντος `OMP_DYNAMIC`, είτε με την συνάρτηση `omp_set_dynamic()`.

2.3.3 Οδηγίες Διαμοίρασης Έργου

Το OpenMP μας παρέχει τρεις δομές οδηγιών διαμοίρασης έργου. Την οδηγία "for", την οδηγία "sections" και την οδηγία "single". Οι δομές αυτές χρησιμεύουν στο διαχωρισμό και τη διαμοίραση της εργασίας που πρόκειται να εκτελεστεί μέσω ενός τμήματος κώδικα, στα νήματα. Για να χρησιμοποιήσουμε μια οδηγία διαμοίρασης έργου θα πρέπει να την τοποθετήσουμε μέσα σε μια παράλληλη περιοχή. Αυτό συνεπάγεται οτι με την ύπαρξη μιας τέτοιας οδηγίας δεν θα δημιουργηθεί μια καινούρια ομάδα νημάτων και οτι η διαμοίραση θα γίνει μεταξύ των νημάτων που συμμετέχουν στην ομάδα. Κατ' επέκταση δεν έχουμε συγχρονισμό των νημάτων κατά την είσοδο σε μια δομή διαμοίρασης έργου, υπονοείται όμως συγχρονισμός στο τέλος. Βέβαια προσθέτοντας σε οποιαδήποτε από αυτές τις οδηγίες τη συνθήκη ***nowait***, ο συγχρονισμός στο τέλος παραλείπεται.

Ας δούμε όμως πιο αναλυτικά αυτές τις τρεις δομές διαμοίρασης έργου περιγράφοντας τη χρησιμότητά τους καθώς και τον τρόπο που διαμοιράζουν το έργο μεταξύ των νημάτων.

- ***Οδηγία for***. Αμέσως μετά την ύπαρξη αυτής της οδηγίας θα πρέπει αυστηρά να εμφανίζεται μια δομή επανάληψης "for" την οποία και θα παραλληλοποιήσει. Δε χρειάζεται να τοποθετήσουμε αγκύλες για να ορίσουμε τον κώδικα της οδηγίας καθώς αυτός ορίζεται από τις αγκύλες της for. Με την οδηγία αυτή ουσιαστικά διαμοιράζονται οι επαναλήψεις της for που ακολουθεί, μεταξύ των νημάτων. Το πως θα διαμοιραστούν αυτές οι επαναλήψεις θα το δούμε πιο αναλυτικά αργότερα όταν θα μιλήσουμε για τις συνθήκες των οδηγιών. Το OpenMP προς χάριν συντόμευσης και λόγω της συχνής παραλληλοποίησης των δομών for προσφέρει τη δυνατότητα του ορισμού αυτής της οδηγίας μαζί με τον ορισμό της παράλληλης περιοχής με την συγγραφή:

```
#pragma omp parallel for
   for(;;)
   //code
```

- ***Οδηγία sections***. Με την οδηγία sections ορίζουμε τμήματα εργασιών κάθε ένα από τα οποία θα εκτελεστεί από ένα μόνο νήμα. Ουσιαστικά αναθέτουμε σε κάθε νήμα δομημένα block κώδικα που μπορούν να μην έχουν καμία σχέση όσον αφορά τη λειτουργικότητα αυτών. Άρα τα νήματα παράλληλα μπορούν να

εκτελούν διαφορετικές εργασίες. Για να ορίσουμε οτι ένα μέρος κώδικά μέσα στην οδηγία αποτελεί ξεχωριστή περιοχή που θα εκτελέσει ένα μόνο νήμα της ομάδας θα πρέπει να το τοποθετήσουμε μέσα σε αγκύλες στην αρχή των οποίων θα υπάρχει η οδηγία #pragma omp section.

- ***Οδηγία single.*** Η ύπαρξη αυτής της οδηγίας συνεπάγεται οτι το πρώτο νήμα της ομάδας που θα τη συναντήσει θα είναι αυτό και μόνο που θα εκτελέσει τον κώδικα που υπάρχει μέσα στις αγκύλες της. Όλα τα άλλα νήματα θα προσπεράσουν την οδηγία και θα περιμένουν μέχρι να τελειώσει τον κώδικα το νήμα που βρίσκεται μέσα σε αυτήν, καθώς όπως προαναφέρθηκε, υπονοείται συγχρονισμός μετά από κάθε δομή διαμοίρασης έργου.

Τέλος, είναι καλό να αναφερθεί και η ύπαρξη της οδηγίας “master”. Η συγκεκριμένη οδηγία μπορεί να μη θεωρείται οδηγία διαμοίρασης έργου, αλλά έχει παρόμοια συμπεριφορά με αυτήν της “single”. Και σε αυτήν την οδηγία, τον κώδικα που περικλείεται από αυτήν την εκτελεί ένα μόνο νήμα της ομάδας με τη διαφορά όμως οτι εδώ το νήμα θα είναι αυστηρά το master. Σε αντίθεση με τις οδηγίες διαμοίρασης εργασίας εδώ δεν υπονοείται συγχρονισμός στο τέλος.

2.3.4 Οδηγίες Συγχρονισμού

Το OpenMP παρέχει και οδηγίες για τον συγχρονισμό μεταξύ των νημάτων όσον αφορά την εκτέλεση κώδικα καθώς και τη συνέπεια των κοινόχρηστων μεταβλητών που υπάρχουν μεταξύ τους. Ακολουθεί περιγραφή των οδηγιών αυτών.

- ***Οδηγία critical.*** Με αυτή την οδηγία ορίζουμε μια κρίσιμη περιοχή κώδικα. Στην κρίσιμη περιοχή μπορεί να βρίσκεται μόνο ένα νήμα, ενώ όλα τα άλλα περιμένουν πριν από αυτή μέχρι να τελειώσει τη δουλειά του και να μπορέσουν και αυτά στη συνέχεια να αποκτήσουν το δικαίωμα να μπουν σε αυτήν. Η κρίσιμη περιοχή είναι απαραίτητη σε τμήματα κώδικα που πρόκειται να εκτελεστούν παράλληλα και αλλάζουν κοινόχρηστες μεταβλητές. Με την ύπαρξη ενός μόνο νήματος τη φορά μέσα στην κρίσιμη περιοχή διασφαλίζεται η ακεραιότητα των τιμών των μεταβλητών.
- ***Οδηγία atomic.*** Η atomic κάνει ουσιαστικά την ίδια δουλειά με την critical με τη διαφορά οτι χρησιμοποιείται για απλές εντολές που αλλάζουν την τιμή μιας κοινόχρηστης μεταβλητής και ανανεώνει την θέση μνήμης αυτής με τη νέα τιμή.
- ***Οδηγία barrier.*** Με τη χρησιμοποίηση αυτής της οδηγίας δίνουμε εντολή στα νήματα της ομάδας να συγχρονιστούν σε εκείνο το σημείο. Μόλις ένα νήμα συναντήσει αυτήν την οδηγία σταματάει και περιμένει όλα τα νήματα της ομάδας να φτάσουν σε εκείνο το σημείο. Όταν τελικά όλα τα νήματα συναντήσουν την οδηγία μπορούν να συνεχίσουν όλα μαζί να εκτελούν τον κώδικα που ακολουθεί. Ένα από τα ελαττώματα του OpenMP είναι οτι δεν μπορούμε να περιορίσουμε την οδηγία barrier σε ένα υποσύνολο νημάτων της ομάδας και πρέπει να την εφαρμόσουμε σε όλα.
- ***Οδηγία flush.*** Η οδηγία αυτή προσφέρει συνέπεια μνήμης. Ουσιαστικά επιβάλει την άμεση εγγραφή των κοινόχρηστων μεταβλητών που έχει ως ορίσματα η οδηγία ή αν σε περίπτωση που δεν έχει ορίσματα, όλων των κοινόχρηστων μεταβλητών στη μνήμη. Πρόκειται δηλαδή για ένα είδος συγχρονισμού μνήμης.
- ***Οδηγία ordered.*** Η οδηγία ordered επιβάλει την εκτέλεση ενός τμήματος κώδικα

με την ακολουθιακή του σειρά. Είναι χρήσιμη για τμήματα κώδικα που υπάρχουν μέσα σε παράλληλες περιοχές και θέλουμε να εκτελεστούν ακολουθιακά όπως θα εκτελούνταν στο σειριακό πρόγραμμα.

2.3.5 Συνθήκες Οδηγιών

Οι οδηγίες OpenMP επιδέχονται και κάποιες συνθήκες που μπορούμε να εισάγουμε ώστε να ρυθμίσουμε τον τρόπο που θα παραλληλοποιηθεί ο κώδικας που ακολουθεί καθώς και την πολιτική κοινοχρησίας ή μη, των μεταβλητών του κώδικα που ακολουθεί. Οι συνθήκες αυτές, όπως είδαμε και στην ενότητα 2.3.1 τοποθετούνται μετά το όνομα της οδηγίας.

Ακολουθεί μια περιγραφή και αναφορά των συνθηκών που χρησιμοποιούνται στο OpenMP.

- ***if(συνθήκη).*** Μόνο αν η συνθήκη μέσα στο if είναι αληθής, θα εκτελεστεί ο κώδικας που ακολουθεί παράλληλα. Αν όχι, τότε ο κώδικας θα εκτελεστεί σειριακά σαν να μην υπήρχε η οδηγία.
- ***shared(λίστα μεταβλητών).*** Οι μεταβλητές που βρίσκονται στη λίστα μεταβλητών, θα είναι κοινόχρηστες για όλα τα νήματα που θα εκτελέσουν τον κώδικα της παράλληλης περιοχής.
- ***private(λίστα μεταβλητών).*** Οι μεταβλητές που βρίσκονται στη λίστα μεταβλητών, θα είναι ιδιωτικές για κάθε νήμα που θα εκτελέσει τον κώδικα της παράλληλης περιοχής. Οι μεταβλητές αυτές δεν είναι αρχικοποιημένες.
- ***fisrtprivate(λίστα μεταβλητών).*** Οι μεταβλητές της λίστας μεταβλητών αποτελούν ιδιωτικά αντίγραφα από τις αντίστοιχες μεταβλητές του νήματος master για το κάθε νήμα. Είναι αρχικοποιημένες με τις τιμές του νήματος master.
- ***lastprivate(λίστα μεταβλητών).*** Οι ιδιωτικές μεταβλητές της λίστας μεταβλητών θα περάσουν τις τιμές που έχουν από την τελευταία επανάληψη ή τμήμα που εκτελέστηκε της παράλληλης περιοχής στις αντίστοιχες καθολικές μεταβλητές.
- ***nowait.*** Όπως αναφέρθηκε και προηγουμένως, η συγκεκριμένη συνθήκη χρησιμοποιείται για να ακυρωθεί ο συγχρονισμός στο τέλος της οδηγίας, αν αυτός υπάρχει.
- ***num_threads(αριθμός).*** Με αυτόν τον τρόπο μπορούμε να ορίσουμε τον αριθμό των νημάτων που θα συμμετέχουν στην ομάδα και θα εκτελέσουν την παράλληλη περιοχή που ακολουθεί.
- ***schedule(τύπος [,μέγεθος κόκκου]).*** Χρησιμοποιώντας αυτήν τη συνθήκη ρυθμίζουμε το μέγεθος της εργασίας καθώς και τον τρόπο με τον οποίο θα διαμοιραστούν οι επαναλήψεις του βρόγχου μεταξύ των νημάτων. Υπάρχουν τέσσερις τύποι για δρομολόγησης.
    - ***static:*** οι επαναλήψεις χωρίζονται στατικά σε κομμάτια ίσα με το μέγεθος που έχει οριστεί και διαμοιράζονται κυκλικά στα νήματα της ομάδας. Αν δεν έχει οριστεί κάποιο μέγεθος, τότε η εξ ορισμού συμπεριφορά του είναι τμήματα εργασίας μεγέθους n/p, όπου n ο αριθμός των επαναλήψεων και p ο αριθμός των νημάτων της ομάδας .
    - ***dynamic:*** οι επαναλήψεις χωρίζονται σε κομμάτια ίσα με το μέγεθος που έχει

οριστεί στη συνθήκη και διαμοιράζονται στα νήματα. Όταν ένα νήμα τελειώσει το δικό του κομμάτι, παίρνει δυναμικά το επόμενο. Αν δεν οριστεί μέγεθος συνήθως η εξ ορισμού συμπεριφορά, είναι τμήματα μεγέθους 1.

  - ***guided:*** οι επαναλήψεις χωρίζονται σε κομμάτια ίσα με το μέγεθος που έχει οριστεί και διαμοιράζονται στα νήματα. Όταν ένα νήμα τελειώσει το δικό του κομμάτι, παίρνει δυναμικά το επόμενο του οποίου το μέγεθος όλο και μικραίνει εκθετικά.
  - ***runtime:*** όταν έχουμε αυτόν τον τύπο τότε ο αλγόριθμος δρομολόγησης θα καθοριστεί από την αντίστοιχη μεταβλητή περιβάλλοντος.

- ***reduction(πράξη: λίστα μεταβλητών).*** Με την εν λόγω συνθήκη δημιουργείτε για κάθε μια μεταβλητή στη λίστα μεταβλητών ένα τοπικό αντίγραφο για κάθε νήμα. Τα αντίστοιχα αυτά τοπικά αντίγραφα συνδυάζονται στο τέλος με κριτήριο την πράξη στον ορισμό της συνθήκης για να συμπληρώσουν με ασφάλεια την τιμή της καθολικής μεταβλητής.

2.4 Συναρτήσεις Βιβλιοθήκης

Το OpenMP παρέχει και μια σειρά συναρτήσεων. Για να χρησιμοποιήσουμε τις συναρτήσεις αυτές θα πρέπει να κάνουμε include στο πρόγραμμα μας τη βιβλιοθήκη “omp.h”. Τις συναρτήσεις του OpenMP θα μπορούσαμε να τις διαχωρίσουμε σε δύο κατηγορίες. Τις συναρτήσεις περιβάλλοντος χρόνου εκτέλεσης και τις συναρτήσεις κλειδαριών. Η πρώτη κατηγορία αφορά συναρτήσεις που έχουν να κάνουν με τον έλεγχο αλλά και τη λήψη πληροφοριών σχετικά με διάφορες μεταβλητές του OpenMP που αφορούν το πρόγραμμα ενώ η δεύτερη κατηγορία αφορά τις κλειδαριές για την προστασία κώδικα που αφορά κοινόχρηστες μεταβλητές.

2.4.1 Συναρτήσεις Περιβάλλοντος Χρόνου Εκτέλεσης

Με τις συναρτήσεις αυτής της κατηγορίας μπορούμε να πάρουμε πληροφορίες αλλά και να ρυθμίσουμε κάποιες παραμέτρους του προγράμματος. Ακολουθεί αναφορά και περιγραφή των συναρτήσεων αυτών.

- ***void omp_set_num_threads(int).*** Με την κλήση αυτής της συνάρτησης ορίζεται ο αριθμός των νημάτων που θα εκτελέσουν το επόμενο παράλληλο τμήμα. Η κλήση θα πρέπει να γίνει σε σειριακό μέρος του προγράμματος και μπορεί να την καλέσει μόνο το νήμα master. Τα αποτελέσματα της κλήσης της προκύπτουν συναρτήσει της ύπαρξης δυναμικού ορισμού του αριθμού των νημάτων ή όχι. Αν επιτρέπεται ο δυναμικός ορισμός των νημάτων τότε η συνάρτηση ορίζει τον μέγιστο αριθμό των νημάτων που θα εκτελέσουν το επόμενο παράλληλο τμήμα. Σε αντίθετη περίπτωση, ορίζει τον ακριβή αριθμό των νημάτων που θα εκτελέσουν την παράλληλη περιοχή.
- ***int omp_get_num_threads(void).*** Η συνάρτηση αυτή μας επιστρέφει τον αριθμό των νημάτων που εκτελούν την παράλληλη περιοχή στην οποία βρισκόμαστε. Αν η συνάρτηση κληθεί έξω από παράλληλη περιοχή, δηλαδή σε σειριακό μέρος κώδικα, θα επιστρέψει την τιμή 1 εννοώντας την ύπαρξη του νήματος master.
- ***int omp_get_thread_num(void).*** Η συνάρτηση μας επιστρέφει τον αριθμό του αναγνωριστικού του νήματος που την κάλεσε. Όπως προαναφέρθηκε, το νήμα

master έχει το αναγνωριστικό 0 και όλα τα άλλα νήματα παίρνουν τιμές στα αναγνωριστικά τους από το 1 μέχρι και το συνολικό αριθμό τους μείον ένα.

- ***int omp_get_max_threads(void).*** Επιστρέφει το μέγιστο αριθμό νημάτων που θα εκτελέσουν την επόμενη παράλληλη περιοχή. Η τιμή που θα πάρει είναι ουσιαστικά η μέγιστη εκ των τιμών της μεταβλητής περιβάλλοντος `OMP_NUM_THREADS` και της τιμής που έχει τεθεί με την κλήση της συνάρτησης `omp_set_num_threads()`.
- ***void omp_set_nested(int).*** Με τη συνάρτηση αυτή ενεργοποιείτε ή απενεργοποιείτε η δυνατότητα εμφωλευμένων παράλληλων τμημάτων. Αν δοθεί η τιμή 0 απενεργοποιείτε αυτή η δυνατότητα, ενώ για άλλες τιμές έχουμε τα αντίθετα αποτελέσματα.
- ***int omp_get_nested(void).*** Επιστρέφει 0 αν η δυνατότητα εμφωλευμένης παραλληλίας είναι απενεργοποιημένη και 1 σε αντίθετη περίπτωση.
- ***void omp_set_dynamic(int).*** Με τη συνάρτηση αυτή ενεργοποιείτε ή απενεργοποιείτε η δυνατότητα δυναμικού ορισμού του αριθμού των νημάτων. Αν δοθεί η τιμή 0 απενεργοποιείτε αυτή η δυνατότητα, ενώ για άλλες τιμές έχουμε τα αντίθετα αποτελέσματα.
- ***int omp_get_dynamic(void).*** Επιστρέφει 0 αν η δυνατότητα δυναμικού ορισμού του αριθμού των νημάτων είναι απενεργοποιημένη και 1 σε αντίθετη περίπτωση.
- ***int omp_in_parallel(void).*** Επιστρέφει 1 αν βρισκόμαστε σε περιοχή που εκτελείται παράλληλα από πολλά νήματα και 0 σε αντίθετη περίπτωση.
- ***int omp_num_procs(void).*** Επιστρέφει τον αριθμό των επεξεργαστών που υπάρχουν στο σύστημα.

2.4.2 Συναρτήσεις Κλειδαριών

Οι κλειδαριές στον παράλληλο προγραμματισμό είναι απαραίτητες για τη συνέπεια και την ορθότητα των αποτελεσμάτων από τη στιγμή της ύπαρξης κοινόχρηστων μεταβλητών. Ουσιαστικά με τις κλειδαριές μπορούμε να ορίσουμε κρίσιμες περιοχές, περιοχές δηλαδή στις οποίες δεν μπορούν να βρίσκονται ταυτόχρονα παραπάνω από ένα νήματα. Το OpenMP παρέχει το δικό του τύπο κλειδαριάς omp_lock_t και omp_nest_lock_t. Ο δεύτερος τύπος αναφέρεται σε περιπτώσεις εμφωλευμένης παραλληλίας. Ακολουθεί αναφορά και περιγραφή των συναρτήσεων αυτών.

- ***void omp_init_lock(omp_lock_t*).*** Με την κλήση αυτής της συνάρτησης, η οποία είναι απαραίτητη αν στο πρόγραμμα μας έχουμε κλειδαριές, αρχικοποιείται ουσιαστικά η κλειδαριά που της έχουμε περάσει ως όρισμα.
- ***void omp_set_lock(omp_lock_t*).*** Όταν ένα νήμα φτάσει στην κλήση αυτής της συνάρτησης υπάρχουν δύο περιπτώσεις. Είτε να είναι κλειδωμένη η κλειδαριά από κάποιο άλλο νήμα και έτσι να περιμένει σε εκείνο το σημείο μέχρι να ξεκλειδωθεί η κλειδαριά και να μπορέσει εκείνο με τη σειρά του να την κλειδώσει, είτε να είναι ελεύθερη η κλειδαριά όπου στην περίπτωση αυτή κλειδώνει αυτό με τη σειρά του την κλειδαριά και εκτελεί τον κώδικα μέσα σε αυτή.
- ***void omp_unset_lock(omp_lock_t*).*** Με την κλήση αυτή, το νήμα που είχε την

κλειδαριά, την απελευθερώνει για να μπορέσει το επόμενο νήμα να την κλειδώσει με τη σειρά του και να αποκτήσει πρόσβαση στο τμήμα κώδικα.

- ***int omp_test_lock(omp_lock_t*).*** Επιστρέφει 1 σε περίπτωση που η κλειδαριά έχει αρχικοποιηθεί επιτυχώς και 0 σε αντίθετη περίπτωση.

2.5 Μεταβλητές Περιβάλλοντος

Εκτός από τις οδηγίες και τις συναρτήσεις βιβλιοθήκης, το OpenMP προσφέρει και τη δυνατότητα χρήσης μεταβλητών περιβάλλοντος για τη ρύθμιση της εκτέλεσης των παράλληλων προγραμμάτων. Οι μεταβλητές περιβάλλοντος παίρνουν τιμή πριν την εκτέλεση του προγράμματος και γράφονται με κεφαλαία γράμματα. Ακολουθεί αναφορά και περιγραφή των μεταβλητών περιβάλλοντος που προσφέρει το OpenMP.

- ***OMP_NUM_THREADS***. Με την τιμή αυτής της μεταβλητής ορίζουμε τον αριθμό νημάτων που μπορούν να εκτελέσουν μια παράλληλη περιοχή στο πρόγραμμά μας. Ο αριθμός αυτός των νημάτων μπορεί να αλλάξει κατά τη διάρκεια εκτέλεσης είτε με τη συνθήκη οδηγίας num_threads(), είτε με τη κλήση της συνάρτησης βιβλιοθήκης `omp_set_num_threads()`. Βέβαια η αλλαγή αυτή του αριθμού μπορεί να γίνει μόνο προς τα κάτω γιατί η μεταβλητή περιβάλλοντος ορίζει ουσιαστικά το μέγιστο αριθμό των νημάτων που μπορούν να χρησιμοποιηθούν.
- ***OMP_NESTED***. Η μεταβλητή παίρνει τιμή 1 για ενεργοποίηση της δυνατότητας πολυεπίπεδου παράλληλου φωλιάσματος και 0 για απενεργοποίηση. Βέβαια όπως προαναφέρθηκε το αν και πως θα υλοποιηθεί τελικά το πολυεπίπεδο παράλληλο φώλιασμα εξαρτάται και από την υλοποίηση του μεταγλωττιστή.
- ***OMP_DYNAMIC***. Η μεταβλητή παίρνει την τιμή 1 αν θέλουμε η δυνατότητα του δυναμικού ορισμού του αριθμού των νημάτων να είναι ενεργοποιημένη και 0 για το αντίθετο αποτέλεσμα.
- ***OMP_SCHEDULE***. Αφορά τον τρόπο με τον οποίο θα παραλληλοποιηθούν οι επαναλήψεις του βρόγχου της “for” στην οποία έχει τεθεί η οδηγία διαμοιρασμού εργασίας #pragma omp for μαζί με την συνθήκη για δρομολόγηση schedule(runtime).

Κεφάλαιο 3

OMPi Compiler

3.1 Επισκόπηση του OMPi

Ο OMPi [4] είναι ένας μετασχηματιστής πηγαίου σε πηγαίο κώδικα προγραμμάτων OpenMP γραμμένων σε γλώσσα C. Αποτελείται ουσιαστικά από δύο τμήματα. Το τμήμα του μεταγλωττιστή και το τμήμα runtime. Η συνεργασία αυτών των τμημάτων είναι στενή καθώς ο μεταγλωττιστής αναλαμβάνει να μετατρέψει το πρόγραμμα C/OpenMP σε ένα πρόγραμμα C προσθέτοντας κάποιες κλήσεις στο σύστημα runtime του OMPi. Ο μεταφραστής θα συνδέσει αυτές τις κλήσεις με την αντίστοιχη βιβλιοθήκη runtime που έχει επιλεγεί στον OMPi, καθώς αυτός διαθέτει διάφορες βιβλιοθήκες οι οποίες θα αναλυθούν αργότερα.

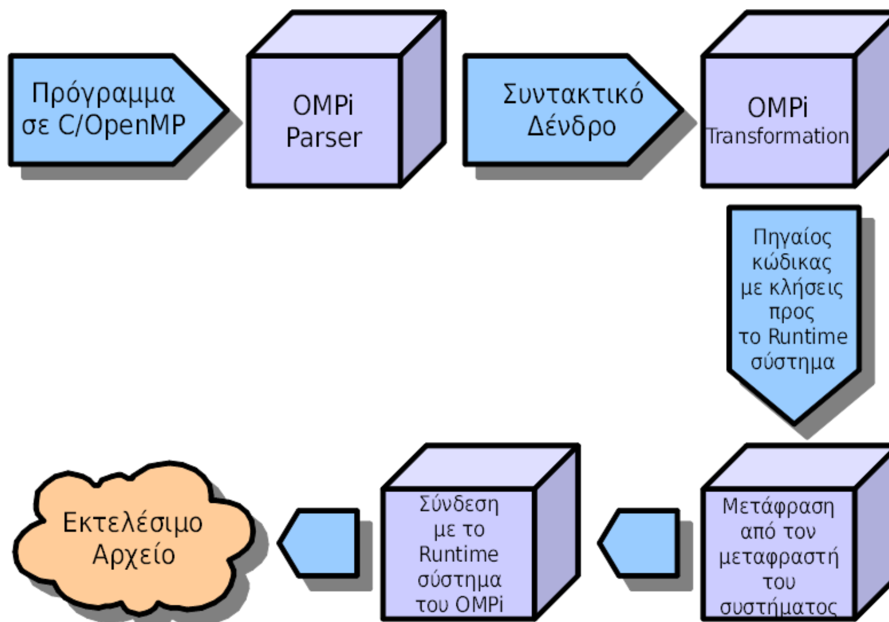
Το OpenMP ουσιαστικά δημιουργεί νήματα εκτέλεσης για τις περιοχές που θέλουμε να παραλληλοποιήσουμε. Ο OMPi δίνει πολλές επιλογές για το πως και τι θα είναι αυτά τα νήματα εκτέλεσης ή καλύτερα αυτές οι υπολογιστικές οντότητες. Είναι δομημένος έτσι ώστε τα νήματα να μετατρέπονται σε αόριστες υπολογιστικές οντότητες τις οποίες αναλαμβάνει το σύστημα runtime να τις υλοποιήσει και να τις χειριστεί. Ο OMPi περιέχει μια ποικιλία βιβλιοθηκών που παρέχουν διαφορετικές υπολογιστικές οντότητες που υλοποιούν τα νήματα εκτέλεσης του OpenMP. Αποτελείται από δύο βιβλιοθήκες νημάτων POSIX επιπέδου πυρήνα, των οποίων η διαφορά έγκειται στο σημείο οτι η μια δεν υποστηρίζει πολυεπίπεδο (nested) προγραμματισμό, και για η εκτέλεση παράλληλων περιοχών που βρίσκονται σε επίπεδο παραλληλισμού μεγαλύτερου του πρώτου γίνονται σειριακά από 1 νήμα και η άλλη υποστηρίζει περιορισμένο, και αυτό λόγω του οτι ο OMPi δημιουργεί εξ αρχής έναν ορισμένο αριθμό από υπολογιστικές οντότητες και χρησιμοποιεί αριθμητικά μόνο αυτές κατά την εκτέλεση. Υπάρχουν άλλες δύο βιβλιοθήκες νημάτων Solaris επιπέδου πυρήνα με την ίδια πολιτική για τον πολυεπίπεδο προγραμματισμό. Για την καλύτερη και πιό αποδοτική υποστήριξη του πολυεπίπεδου προγραμματισμού ο OMPi περιλαμβάνει και δύο βιβλιοθήκες νημάτων χρήστη (POSIX, Marcell).

Η τελευταία βιβλιοθήκη που προστέθηκε στον OMPi, παρέχει διεργασίες ως υπολογιστικές οντότητες καθώς και μια διεπαφή για τη χρήση διάφορων DSM, που υλοποιούν την εικονική κοινή μνήμη από "κάτω". Η προσθήκη της εν λόγω βιβλιοθήκης έγινε για την εκτέλεση προγραμμάτων OpenMP σε cluster όπου δεν υπάρχει κοινή μνήμη και όπως προαναφέρθηκε στο προηγούμενο κεφάλαιο, το OpenMP είναι πρότυπο προγραμματισμού για μοντέλο κοινού χώρου διευθύνσεων. Το πρόβλημα επιλύεται μέσω της εικονικής κοινής μνήμης που υλοποιεί το DSM. Υποστηρίζει έναν αριθμό από πυρήνες DSM συμπεριλαμβανομένου των, TreadMarks [2], JiaJia [6], Mocha [7], Mome [8] και Parade [9].

3.2 O Compiler του OMPi

Ο compiler του OMPi παίρνει ως είσοδο το πρόγραμμα σε C/OpenMP και παράγει ένα πηγαίο πρόγραμμα στο οποίο έχει ενσωματώσει κλήσεις προς το σύστημα runtime του. Το compiling περνάει από μια σειρά σταδίων (Σχήμα 3.1). Πρώτα το λεγόμενο

parsing κατά το οποίο ο parser διασχίζει το πρόγραμμα και δημιουργεί ένα αφηρημένο συντακτικό δέντρο που αντιπροσωπεύει το πρόγραμμα. Στη συνέχεια το δέντρο αυτό δίνεται ως είσοδο στο επόμενο στάδιο, αυτό του μετασχηματισμού. Κατά το στάδιο του μετασχηματισμού διασχίζεται κάθε κόμβος του δέντρου και όπου βρεθεί δήλωση OpenMP αυτός αντικαθίσταται από αντίστοιχο κώδικα C με τον αρχικό C/OpenMP, και με τη διαφορά επίσης, οτι περιέχει κάποιες κλήσεις στο σύστημα runtime.



Σχήμα 3.1: Διαδικασία μετάφρασης του OMPi.

Κατά τη διάρκεια του μετασχηματισμού επιτελείται και η πιό περίπλοκη εργασία. Μπορεί κάποιες οδηγίες OpenMP να έχουν σχετικά απλή λύση για το πως θα μετασχηματιστούν σε C κώδικα, αλλά υπάρχουν οδηγίες όπως η `parallel` που περιλαμβάνει και συνθήκες οι οποίες θέλουν ειδική μεταχείριση. Το βασικό πρόβλημα παρατηρείται στις μεταβλητές. Κάποιες θα πρέπει να είναι κοινές και στην περίπτωση που έχουμε νήματα αυτό φαντάζει εύκολο για τις καθολικές (global) μεταβλητές μιας και θεωρούνται κοινές εξ ορισμού. Υπάρχει όμως και η περίπτωση όπου έχουμε κοινές μεταβλητές μεταξύ των νημάτων αλλά δεν είναι καθολικές. Για παράδειγμα αν αυτές οι μεταβλητές δηλώνονται σε μια συνάρτηση στην οποία υπάρχει οδηγία παράλληλου τμήματος OpenMP. Σε αυτήν την περίπτωση ο OMPi αντικαθιστά αυτές τις μεταβλητές με δείκτες των οποίων τη διεύθυνση την περνά σε τμήμα κώδικα που θα πρέπει να εκτελέσουν τα νήματα. Εκτενέστερη αναφορά όμως θα δούμε στο επόμενο κεφάλαιο όπου περιγράφεται αναλυτικά η διαδικασία του μετασχηματισμού σε μια παρόμοια περίπτωση.

Στην τρίτη και τελευταία φάση του compilation, γίνεται μια διάσχιση του πλέον

μετασχηματισμένου δέντρου και παράγεται ο C κώδικας. Αυτός με τη σειρά του θα μεταφραστεί από τον C μεταφραστή του συστήματος ο οποίος θα αναλάβει να το συνδέσει και με τη βιβλιοθήκη runtime που έχει επιλεγεί για να υλοποιήσει τις υπολογιστικές οντότητες.

3.2.1 Διαδικασία Μετασχηματισμού για Νήματα

Σε αυτήν την ενότητα γίνεται μια ανάλυση των ιδιαιτεροτήτων που ακολουθεί ο μεταγλωττιστής του OMPi κατά το μετασχηματισμό ενός προγράμματος OpenMP στην περίπτωση που έχει επιλεγεί βιβλιοθήκη που παρέχει νήματα ως υπολογιστικές οντότητες. Επίσης γίνεται και μια περιγραφή του τρόπου με τον οποίο χειρίζεται τις μεταβλητές για τα νήματα ειδικά στην περίπτωση που αυτές πρέπει να είναι κοινές μεταξύ τους.

Ο μετασχηματισμός που απαιτεί τη μεγαλύτερη ανάλυση είναι αυτός της οδηγίας parallel. Οι άλλες οδηγίες έχουν απλούστερους μετασχηματισμούς. Όταν ο parser βρει μια οδηγία parallel, μεταφέρει τον κώδικα που βρίσκεται μέσα στην οδηγία σε μια συνάρτηση την οποία ονομάζει `_thrFunc#_`. Στην θέση της "#" τοποθετεί έναν αριθμό που ουσιαστικά είναι ο αύξοντας αριθμός της συνάρτησης που χρειάστηκε να δημιουργήσει, άρα στην πρώτη εμφάνιση κάποιας οδηγίας parallel θα βάλει τον αριθμό 0. Στην θέση της οδηγίας και του κώδικα αυτής, τοποθετεί μια κλήση προς το σύστημα runtime του OMPi με το όνομα `ort_execute_parallel()`. Η συνάρτηση αυτή δέχεται 3 ορίσματα. Πρώτα τον αριθμό των νημάτων που έχει οριστεί μέσω της συνθήκης της οδηγίας num_threads, αν δεν υπάρχει η συνθήκη τότε δίνεται η τιμή -1. Δεύτερο όρισμα είναι η συνάρτηση που θα εκτελέσουν τα νήματα (`_thrFunc#_`) και τρίτο μια δομή με τις μεταβλητές που θα πρέπει να είναι κοινές μεταξύ των νημάτων. Η `ort_execute_parallel()` θα αναλάβει να δώσει στα νήματα τη δουλειά που θα εκτελέσουν και η οποία βρίσκεται στον κώδικα της `_thrFunc#_` καθώς και να τους δώσει τις κοινές μεταβλητές. Τον κώδικα της συνάρτησης των νημάτων τον εκτελεί και το νήμα master, ενώ στο τέλος συγχρονίζονται όλα μεταξύ τους. Ας δούμε όμως καλύτερα ένα πιο συγκεκριμένο παράδειγμα.

Έστω οτι έχουμε τον παρακάτω κώδικα OpenMP και θέλουμε να τον μεταγλωττίσουμε με τον OMPi για να τον παραλληλοποιήσει:

```
int a;
void function(void) {
    int b, c, d;

    #pragma omp parallel private(d)
        a = b + c + d;
}
```

Παρατηρούμε πως κάποιες μεταβλητές θα πρέπει να γίνουν κοινές μεταξύ των νημάτων. Η μεταβλητή a δεν θέλει ιδιαίτερη μεταχείριση καθώς είναι καθολική και εξ ορισμού είναι κοινή στα νήματα. Στην οδηγία OpenMP υπάρχει η συνθήκη private(d) που έχει την έννοια οτι η μεταβλητή d θα πρέπει να είναι ιδιωτική για κάθε νήμα, πράγμα εύκολο καθώς μπορούμε απλά να τη δηλώσουμε στον ως μεταβλητή του κώδικα της συνάρτησης που θα εκτελέσουν τα νήματα. Το πρόβλημα παρουσιάζεται

στις μεταβλητές b, c οι οποίες θα πρέπει να γίνουν κοινές στα νήματα. Για την επίλυση αυτού του προβλήματος ο OMPi μετατρέπει τις εν λόγω μεταβλητές σε δείκτες και τις περνάει ως όρισμα στη συνάρτηση που θα εκτελέσουν τα νήματα. Πιο συγκεκριμένα, παράγει τον παρακάτω ισοδύναμο κώδικα:

```
static void *_thrFunc0_(void * _arg)
{
  struct{int(*b);int(*c);}*_shvars = ort_get_shared_vars(_arg);
  int (* b) = _shvars->b;
  int (* c) = _shvars->c;
  int d;

  (*a) = (*b) + (*c) + d;
  return (void *) 0;
}

void function(void)
{
    int b, c, d;

    struct { int (* b); int (* c); } _shvars = { &b, &c };
    ort_execute_parallel(-1, _thrFunc0_, &_shvars);
}
```

Στη συνάρτηση `function` έχει προστεθεί ο ορισμός μιας δομής, που ουσιαστικά περιλαμβάνει τους δείκτες προς τις κοινές μεταβλητές τις τιμές των οποίων αρχικοποιεί ώστε να δείχνουν στις σωστές διευθύνσεις. Στην ίδια συνάρτηση επίσης έχει αντικατασταθεί η οδηγία OpenMP, parallel με την κλήση της συνάρτησης `ort_execute_parallel()` στην οποία έχουν δοθεί 3 ορίσματα. Το πρώτο δηλώνει οτι την παράλληλη περιοχή θα την εκτελέσει ο εξ ορισμού αριθμός νημάτων, το δεύτερο είναι η διεύθυνση της συνάρτησης που θα εκτελέσουν τα νήματα και ουσιαστικά πρόκειται για τον κώδικα που υπήρχε μέσα στην οδηγία και το τρίτο είναι η δομή με τις διευθύνσεις των κοινών μεταβλητών. Η `ort_execute_parallel()` αναλαμβάνει να δώσει στον κατάλληλο αριθμό νημάτων, δουλειά. Τους δίνει να εκτελέσουν τη συνάρτηση `_thrFunc0_`. Στα νήματα που θα εκτελέσουν δουλειά συμπεριλαμβάνεται και το νήμα master. Στην `_thrFunc0_`, τα νήματα καλούν τη συνάρτηση runtime, `ort_get_shared_vars()` η οποία επιστρέφει τη διεύθυνση της δομής που περιλαμβάνει τις διευθύνσεις για τις κοινές μεταβλητές. Ο κώδικας που υπήρχε μέσα στην οδηγία parallel έχει μετασχηματιστεί πλέον και στη θέση των κοινών μεταβλητών έχουν τοποθετηθεί δείκτες.

3.2.2 Διαδικασία Μετασχηματισμού για Διεργασίες

Η βιβλιοθήκη των διεργασιών αναπτύχθηκε για την εκτέλεση προγραμμάτων OpenMP σε συστάδες υπολογιστών (clusters). Το OpenMP όπως έχει αναφερθεί είναι ένα πρότυπο για την ανάπτυξη προγραμμάτων πάνω σε κοινόχρηστη μνήμη, η οποία σε μια συστάδα υπολογιστών δεν υπάρχει. Παρόλα αυτά δεν μπορούμε να

περιορίσουμε ή να αλλάξουμε το πρότυπο ώστε να λαμβάνει υπόψιν του την ανυπαρξία κοινής μνήμης. Υπάρχουν εργαλεία πάνω σε προγραμματισμό συστάδων υπολογιστών που προσφέρουν την ψευδαίσθηση ύπαρξης κοινής μνήμης μεταξύ των κόμβων (DSM). Οι διεργασίες δε μοιράζονται τίποτα εξ ορισμού άρα θα πρέπει να τύχουν ειδικής μεταχείρισης όλες οι μεταβλητές που πρέπει να διαμοιραστούν είτε αυτές είναι καθολικές, είτε όχι.

Ας δούμε όμως καλύτερα τη διαδικασία του μετασχηματισμού και του χειρισμού των μεταβλητών μέσα από το παρακάτω παράδειγμα, το οποίο δίνουμε ως είσοδο στον OMPi να το παραλληλοποιήσει χρησιμοποιώντας ως υπολογιστικές οντότητες, διεργασίες:

```
int a = 1, b;
void function(void) {
    int c, d;

    #pragma omp parallel private(d)
        a = b + c + d;
}
```

Από το παράδειγμα γίνεται σαφές πως οι μεταβλητές a, b θα πρέπει να είναι κοινές μεταξύ των διεργασιών που θα συμμετέχουν στον υπολογισμό και λόγω του οτι οι διεργασίες δεν έχουν κάτι κοινό μεταξύ τους θα πρέπει να λάβουν και αυτές, ειδικής μεταχείρισης. Θα πρέπει να τοποθετήσουμε τις μεταβλητές αυτές στην κοινή μνήμη που παρέχει το σύστημα sDSM. Για να συμβεί αυτό θα πρέπει να γίνουν κάποιοι μετασχηματισμοί στο πρόγραμμα. Το πρόγραμμα που παράγεται με τους μετασχηματισμούς είναι το παρακάτω:

```
int _sglini_a = 1, (*a), (*b);

static void *_thrFunc0_(void * _arg)
{
  struct{int(*c);}*_shvars = ort_get_shared_vars(_arg);
  int (* c) = _shvars->c;
  int d;

  (*a) = (*b) + (*c) + d;
  return (void *) 0;
}

void function(void)
{
    int c;
   struct __shvt__ {
       int (* c);
   }_shvars = {
       &c
```

```
    };
    ort_execute_parallel(-1, _thrFunc0_, (void *) &_shvars);
}

 static void __attribute__ ((constructor)) _init_shvars_(void);
 static void _init_shvars_(void)
 {
 ort_sglvar_allocate((void **)&b, sizeof(int), (void *) 0);
 ort_sglvar_allocate((void**)&a,sizeof(int),(void *)_sglini_a);
 }
```

Ο μετασχηματιστής για τις καθολικές μεταβλητές ακολουθεί κάποια βήματα. Πρώτα μετατρέπει τις μεταβλητές σε δείκτες και προσθέτει και μια αντίστοιχη μεταβλητή (_sglini_a) για να κρατήσει την αρχική τιμή της μεταβλητής. Στη συνέχεια προσθέτει τον ορισμό μιας συνάρτησης constructor στην οποία υπάρχουν κλήσεις της `ort_sglvar_allocate()` προς το σύστημα runtime. Τοποθετείται μια κλήση για κάθε μια καθολική μεταβλητή. Η συνάρτηση αυτή δέχεται 3 ορίσματα εκ των οποίων το πρώτο είναι η διεύθυνση του δείκτη της καθολικής μεταβλητής, το δεύτερο είναι το μέγεθος της και το τελευταίο είναι η αρχική τιμή της μεταβλητής. Η `ort_sglvar_allocate()` αναλαμβάνει να δεσμεύσει χώρο στην κοινή μνήμη και να τοποθετήσει τις τιμές των δεικτών των καθολικών μεταβλητών σε αυτή στο κατάλληλο offset. Αξίζει να σημειωθεί βέβαια και ο λόγος για τον οποίο γίνεται η χρήση της συνάρτησης constructor. Αν έχουμε πολλά ανεξάρτητα προγράμματα που θέλουμε να συνδέσουμε, θα είναι δύσκολο να ξέρουμε πόσες είναι οι καθολικές μεταβλητές. Με το να ορίσουμε μια συνάρτηση constructor που θα εκτελεστεί πριν την main και θα μας εξασφαλίσει οτι όλες αυτές οι μεταβλητές θα μπουν στην κοινή μνήμη, λύνουμε το πρόβλημά μας.

Ας μην ξεχνάμε όμως και την ύπαρξη των μη καθολικών μεταβλητών που θα πρέπει να είναι κοινές μεταξύ των διεργασιών. Αν και στα νήματα αυτό το πρόβλημα λύθηκε με τη χρήση δεικτών και το πέρασμα των τιμών τους στη συνάρτηση που εκτελούν τα νήματα, αυτό δε θα ίσχυε στην περίπτωση των διεργασιών καθώς οι διευθύνσεις αυτές για την κάθε διεργασία δε θα ήταν έγκυρες. Ο λόγος είναι οτι οι διεργασίες δεν μπορούν να βλέπουν η μια τη στοίβα της άλλης και άρα οι μη-καθολικές μεταβλητές μιας διεργασίας είναι απροσπέλαστες από άλλη. Το πρόβλημα αυτό στον OMPi λύνεται τοποθετώντας τη στοίβα της αρχικής διεργασίας 0, στην κοινή μνήμη. Με αυτόν τον τρόπο, οι δείκτες θα δείχνουν σε χώρο της κοινής μνήμης και οι διευθύνσεις τους θα είναι έγκυρες. Αυτή η διαδικασία επιτυγχάνεται από τον compiler με μια σειρά από μετασχηματισμούς που θα συζητηθούν στην επόμενη ενότητα.

3.3 Το Σύστημα Runtime του OMPi

Το σύστημα runtime του OMPi [5] ασχολείται με τις υπολογιστικές οντότητες που θα αναλάβουν την εκτέλεση των τμημάτων κώδικα που θα πρέπει να παραλληλοποιηθούν. Αποτελείται από δύο τμήματα τα οποία είναι ανεξάρτητα μεταξύ τους αλλά διέπονται από στενή συνεργασία. Το πρώτο τμήμσ είναι το ORT, το οποίο συνδυάζει και χειρίζεται τις υπολογιστικές οντότητες του OMPi. Παρόλα αυτά δε γνωρίζει το είδος τους γιατί δεν τις υλοποιεί. Το δεύτερο τμήμα, το EELIB, είναι αυτό

που θα υλοποιήσει και θα παρέχει τις υπολογιστικές οντότητες στο ORT. Όπως έχει αναφερθεί, στον OMPi υπάρχουν πολλές βιβλιοθήκες που παρέχουν πληθώρα υπολογιστικών οντοτήτων και κάθε μια τηρεί την ανεξαρτησία των δύο αυτών τμημάτων.

Κατά την εκκίνηση του προγράμματος, εισάγεται από το μετασχηματιστή η κλήση προς το σύστημα runtime, `ort_initialize()`. Σε αυτήν τη συνάρτηση το νήμα master, θα αναλάβει να αρχικοποιήσει κάποιες σημαντικές για τη συνέχεια της εκτέλεσης, παραμέτρους. Αρχικά, λαμβάνει τις τιμές των μεταβλητών περιβάλλοντος και σύμφωνα με τις τιμές τους αρχίζει μια διαδικασία αρχικοποιήσεων. Στη συνέχεια καλείται η αντίστοιχη συνάρτηση αρχικοποίησης του έτερου τμήματος (`ee_initialize`), στην οποία δηλώνεται ο αριθμός των υπολογιστικών οντοτήτων που θα χρειαστούν. Η ιδιαιτερότητα του OMPi είναι οτι δημιουργεί εκ των προτέρων τις υπολογιστικές οντότητες και τις αφήνει να περιμένουν. Όταν χρειαστεί έναν αριθμό από αυτές, τις ξυπνάει δίνοντάς τους δουλειά. Η τελευταία και πολύ σημαντική αρμοδιότητα της εν λόγω συνάρτησης αρχικοποίησης, είναι η δημιουργία της δομής ελέγχου (eecb) του νήματος master. Σε αυτήν τη δομή υπάρχουν όλες οι πληροφορίες που χρειάζεται το τμήμα ORT για να δρομολογήσει τις οντότητες που του παρέχει το τμήμα EELIB.

3.3.1 Είσοδος σε Παράλληλη Περιοχή

Όταν ο compiler συναντήσει την οδηγία parallel στο πρόγραμμα την αντικαθιστά με την κλήση συστήματος runtime, `ort_execute_parallel()`. Η συνάρτηση αυτή σύμφωνα και με τις τιμές των παραμέτρων της, που αφορούν τον αριθμό των οντοτήτων που θα εκτελέσουν την παράλληλη περιοχή, τη συνάρτηση που θα εκτελέσουν καθώς και τις παραμέτρους της συνάρτησης, είναι υπεύθυνη μέσα από κλήσεις προς το EELIB να υλοποιήσει τη διαδικασία παραλληλοποίησης. Επικοινωνεί αρχικά με το EELIB για να ζητήσει έναν αριθμό από οντότητες που θέλει να του παρασχεθούν, ανάλογα πάντα και με το τι ισχύει για τον πολυεπίπεδο παραλληλισμό και το δυναμικό ορισμό αριθμού οντοτήτων σε εκείνο το σημείο. Αν η επιθυμία της γίνει αποδεκτή τότε ζητάει από το EELIB να ξυπνήσει αυτόν τον αριθμό οντοτήτων για να αποτελέσουν την ομάδα εκτέλεσης, χωρίς όμως να γνωρίζει το είδος τους. Με τη σειρά τους οι οντότητες αυτές θα πρέπει να πάρουν δουλειά από το ORT για αυτόν το λόγο καλούν τη συνάρτηση `ort_get_thread_work()`. Με την κλήση της συνάρτησης αυτής αναλαμβάνει η ORT να δώσει όλες τις απαραίτητες πληροφορίες για τη δουλειά που έχουν να επιτελέσουν οι οντότητες μέσω μιας δομής.

Οι πληροφορίες δίνονται μέσω του block ελέγχου του πατέρα-νήματος της ομάδας (αρχικό νήμα) που έχει δημιουργηθεί. Αυτές αφορούν το αναγνωριστικό του νήματος μέσα στην ομάδα, το επίπεδο του παραλληλισμού, το δείκτη προς το block του πατέρα, το μέγεθος της ομάδας καθώς και το δείκτη προς τη συνάρτηση που θα πρέπει να εκτελέσουν. Όλες αυτές οι πληροφορίες είναι απαραίτητες για τις οντότητες και μόλις τις παραλάβουν ξεκινούν να εκτελούν τον κώδικα που τους αναλογεί. Εκτός από τις οντότητες που έχουν ξυπνήσει, στη δουλειά συμμετέχει και ο πατέρας, παίρνοντας το αναγνωριστικό 0 για την ομάδα. Αν υπάρχει υποστήριξη για πολυεπίπεδο παραλληλισμό και κάποιο από τα μέλη της ομάδας συναντήσει μια παράλληλη περιοχή, γίνεται αυτό πατέρας για τη νέα ομάδα οντοτήτων. Κατά συνέπεια δημιουργείτε ένα δέντρο που αυξάνεται και μειώνεται ανάλογα αν αρχίζει ή

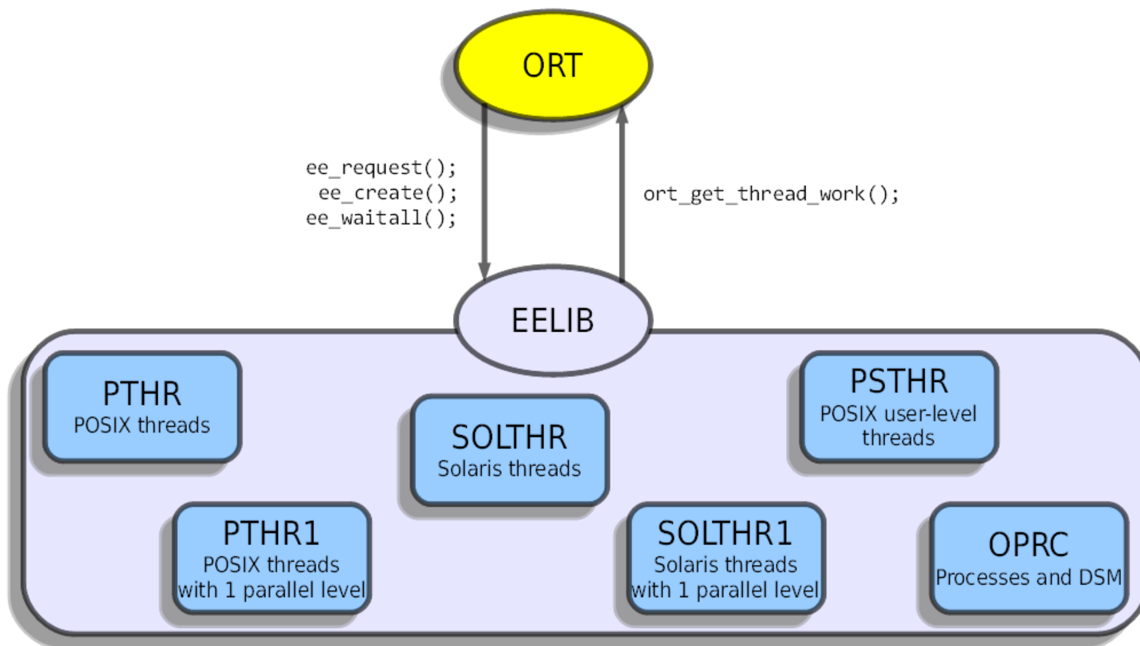
τελειώνει μια παράλληλη περιοχή αντίστοιχα.

3.3.2 Χειρισμός Δομών Διαμοίρασης Έργου

Στο OpenMP, υπάρχουν τρεις δομές διαμοίρασης έργου, το for, το section και το single. Η εξ ορισμού συμπεριφορά των δομών αυτών είναι εμποδιστική, με την έννοια ότι στο τέλος τους υπονοείται ένας συγχρονισμός μεταξύ των οντοτήτων. Με την προσθήκη της συνθήκης nowait στην οδηγία της δομής, παραλείπεται αυτός ο συγχρονισμός και οι οντότητες συνεχίζουν αμέσως την εκτέλεσή τους. Η υλοποίηση σε κώδικα αυτών των δομών συνεπάγεται τη χρήση μεταβλητών-μετρητών για την καταμέτρηση των οντοτήτων που έχουν ολοκληρώσει τον κώδικα. Για παράδειγμα, αν μια δομή section έχει τρία sections, θα πρέπει να δώσει τη δουλειά αυτών στις τρεις πρώτες υπολογιστικές οντότητες που θα φτάσουν σε αυτό. Οι υπόλοιπες ανάλογα με τις συνθήκες που έχει οριστεί η οδηγία (nowait) είτε θα περιμένουν τις οντότητες που έχουν δουλειά να την ολοκληρώσουν, είτε όχι. Παρόλα αυτά, δεν αρκεί ένας μετρητής καθώς μπορούν περισσότερες από μια δομές διαμοίρασης να είναι ενεργές, λόγω της ύπαρξης της συνθήκης nowait. Ο OMPi επιλύει αυτό το πρόβλημα επιτρέποντας ένα προκαθορισμένο αριθμό ενεργών δομών ταυτόχρονα. Ενεργή χαρακτηρίζεται μια δομή αν έχει εισέλθει σε αυτήν τουλάχιστον ένα νήμα και δεν την έχουν ολοκληρώσει όλα τα νήματα της ομάδας. Μετά από αυτόν τον αριθμό, κάθε δομή διαμοίρασης έργου που συναντάται, "περιμένει" μέχρι να ολοκληρωθεί η διαδικασία κάποιας προηγούμενης. Για την εκτέλεση σε clusters, ακολουθεί μια διαφορετική λογική. Σε αυτήν την περίπτωση ο OMPi δεν επιτρέπει πολλές ενεργές δομές διαμοίρασης έργου και κατά συνέπεια, ο κώδικας όλων των δομών διαμοίρασης έργου καταλήγει σε συγχρονισμό των οντοτήτων. Με λίγα λόγια αν έχει τοποθετηθεί η συνθήκη nowait, αγνοείται.

3.3.3 Η Διεπαφή με το EELIB

Τα δύο τμήματα του συστήματος runtime του OMPi, μπορεί να είναι ανεξάρτητα μεταξύ τους αλλά διέπονται από στενή επικοινωνία και ανταλλαγή πληροφοριών. Το ORT δε γνωρίζει τον τύπο και το είδος των υπολογιστικών οντοτήτων αλλά μπορεί να τις δρομολογεί και να τις χειρίζεται μέσω των συναρτήσεων που προσφέρει το EELIB. Οι τελευταίες αποτελούν την διεπαφή του ORT προς τα διάφορα EELIB που είναι υπεύθυνα να του παρέχουν τον αριθμό των οντοτήτων που θα ζητήσει, εκτός βέβαια από το νήμα master παρόλο που και αυτό συμμετέχει στην υπολογιστική διαδικασία. Εκτός αυτού, το EELIB παρέχει στο ORT και τρεις κλειδαριές, την κανονική, την κλειδαριά πολυεπίπεδου παραλληλισμού και την κλειδαριά spin. Οι δύο πρώτες αποτελούν μέρος του προτύπου του OpenMP, ενώ η τρίτη έχει προστεθεί για ανάγκες του ORT.



Σχήμα 3.2: Διεπαφή τμήματος ORT
με τα διάφορα EELIBS.

Η πρώτη επαφή με το τμήμα EELIB, εμφανίζεται με την έναρξη της εκτέλεσης του προγράμματος καθώς μέσω της συνάρτησης αρχικοποίησης του, το ORT καλεί την συνάρτηση αρχικοποίησης του EELIB. Με αυτό τον τρόπο το EELIB ενημερώνει το ORT για τις δυνατότητες του, που αφορούν τον πολυεπίπεδο προγραμματισμό, την δυνατότητα δυναμικού ορισμού αριθμού οντοτήτων καθώς και πόσες οντότητες μπορεί να δημιουργήσει. Το EELIB παρέχει και άλλες τρεις συναρτήσεις για τη διεπαφή του με το ORT.

- **`ee_request()`**, με την οποία το ORT ζητάει συγκεκριμένο αριθμό οντοτήτων από το EELIB. Το EELIB με τη σειρά του απαντάει με τον αριθμό που μπορεί εκείνο να παρέχει στο ORT. Αυτός ο αριθμός εξαρτάται από μια σειρά παραμέτρων (βλ. 3.3.1).
- **`ee_create()`**, με την οποία το ORT ζητάει από το EELIB να δημιουργήσει τον αριθμό οντοτήτων που τελικά μπορεί να υποστηρίξει. Στις παραμέτρους της συνάρτησης προωθεί στο EELIB και κατά επέκταση στις οντότητες που θα συμμετέχουν στους υπολογισμούς, τις απαραίτητες πληροφορίες για τη δουλειά τους όπως, τη συνάρτηση που θα εκτελέσουν, το επίπεδο παραλληλίας στο οποίο θα βρίσκονται καθώς και πληροφορίες που έχουν να κάνουν με τον πατέρα της ομάδας.
- **`ee_waitall()`**, την οποία καλεί το αρχικό νήμα όταν ολοκληρώσει τον κώδικα

του και ουσιαστικά περιμένει (συγχρονίζεται με) όλες τις οντότητες της ομάδας να τελειώσουν τη δουλειά τους.

3.4 Βιβλιοθήκες Νημάτων

Το τμήμα EELIB του OMPi αποτελείται από έναν αριθμό διαφορετικών βιβλιοθηκών που προσφέρουν διάφορες υπολογιστικές οντότητες. Σε αυτήν την ενότητα θα ασχοληθούμε με το τι προσφέρουν οι βιβλιοθήκες που παρέχουν νήματα. Ο OMPi δημιουργεί εκ των προτέρων Ν νήματα (το Ν καθορίζεται από τη μεταβλητή περιβάλλοντος), τα οποία περιμένουν μέχρι να τους ανατεθεί δουλειά. Όταν το ORT ζητήσει έναν αριθμό νημάτων, το EELIB θα κοιτάξει τον αριθμό των νημάτων που περιμένουν για να εξετάσει αν μπορεί να παρέχει τον ζητούμενο αριθμό από αυτά στο ORT. Αν βρισκόμαστε σε βαθύτερο του πρώτου επιπέδου παραλληλισμού, το αν θα δοθεί ο απαιτούμενος αριθμός νημάτων εξαρτάται και από τις αντίστοιχες μεταβλητές περιβάλλοντος αλλά και από την πολιτική με την οποία έχει δομηθεί η βιβλιοθήκη.

Πιο συγκεκριμένα ο OMPi προσφέρει 4 βιβλιοθήκες νημάτων πυρήνα και 1 νημάτων χρήστη (Σχήμα 3.2). Αξίζει να σημειωθεί πως υπάρχει ακόμα μια βιβλιοθήκη νημάτων επιπέδου χρήστη (Marcell) στην οποία όμως δε θα αναφερθούμε περισσότερο. Υπάρχουν δύο βιβλιοθήκες με νήματα POSIX [10] που καλούνται PTHREADS. Η διαφορά τους έγκειται στο σημείο της υποστήριξης πολυεπίπεδου παραλληλισμού καθώς η μια, ονόματι PTHREADS η οποία είναι και η βασική βιβλιοθήκη του OMPi, παρέχει περιορισμένη υποστήριξη ενώ η έτερη βιβλιοθήκη ονόματι PTHREADS1, σε επίπεδα μεγαλύτερα του πρώτου, δεν παρέχει νήματα και τον κώδικα τον εκτελεί το σειριακά το νήμα που θα έπαιρνε το ρόλο του πατέρα της ομάδας. Εκτός από τα νήματα POSIX, ο OMPi παρέχει και νήματα Solaris. Και εδώ υπάρχουν δύο βιβλιοθήκες με τις ίδιες ιδιότητες όπως και στα POSIX. Η SOLTHR παρέχει περιορισμένο πολυεπίπεδο παραλληλισμό, ενώ η SOLTHR1 δεν υποστηρίζει παραλληλία σε επίπεδα μεγαλύτερα του πρώτου και εκτελεί τον κώδικα σειριακά για αυτά τα επίπεδα. Για τις περιπτώσεις που απαιτούν έντονα, πολυεπίπεδο παραλληλισμό ο OMPi παρέχει μια βιβλιοθήκη με νήματα POSIX επιπέδου χρήστη (PSTHREADS) η οποία μάλιστα επιτυγχάνει ιδιαίτερα καλές επιδόσεις. Τα νήματα χρήστη τρέχουν στους δικούς τους εικονικούς επεξεργαστές και η δρομολόγησή τους γίνεται από τη βιβλιοθήκη πάνω στους φυσικούς επεξεργαστές.

Λόγω της διεπαφής που υπάρχει ανάμεσα στα δύο αυτά τμήματα του συστήματος runtime του OMPi αλλά και της ανεξαρτησίας μεταξύ τους, είναι εύκολο από κάποιον προγραμματιστή να δημιουργήσει και να προσαρτήσει στον OMPi τη δικιά του βιβλιοθήκη οντοτήτων. Αυτό που χρειάζεται είναι να δομήσει σωστά τις συναρτήσεις που συμμετέχουν στη διεπαφή μεταξύ των δύο τμημάτων.

3.5 OMPi και Clusters

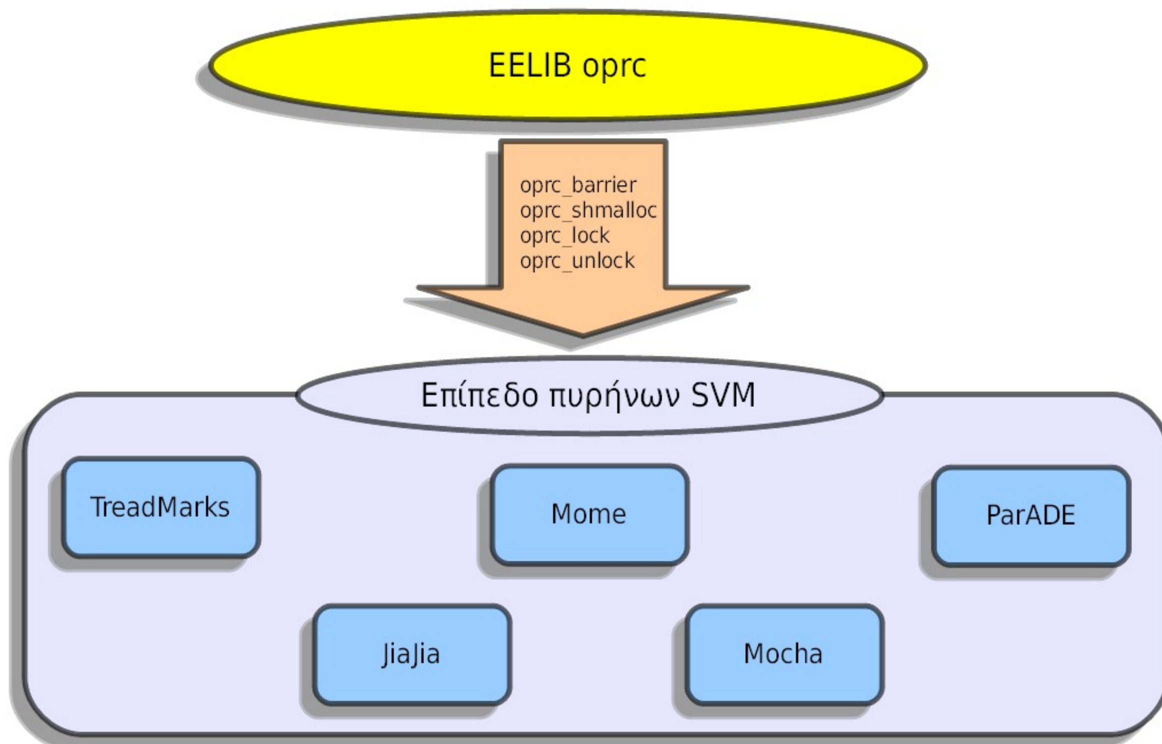
Για την υποστήριξη της εκτέλεσης προγραμμάτων OpenMP σε cluster, έχει αναπτυχθεί για τον OMPi μια βιβλιοθήκη που προσφέρει διεργασίες ως υπολογιστικές οντότητες. Το OpenMP όμως είναι ένα πρότυπο για τη συγγραφή παράλληλων προγραμμάτων πάνω σε συστήματα με κοινόχρηστη μνήμη. Στην περίπτωση ενός cluster δεν υπάρχει κοινόχρηστη μνήμη. Παρόλα αυτά όπως αναφέρθηκε και σε προηγούμενες ενότητες, με το λογισμικό sDSM (software Distributed Memory),

μπορούμε να υλοποιήσουμε εικονική κοινή μνήμη πάνω στους κόμβους του cluster. Στον OMPi έχει προσαρτηθεί ένας αριθμός από πυρήνες sDSM μέσω μιας διεπαφής μεταξύ αυτών και του τμήματος EELIB του OMPi.

Ο OMPi για το χειρισμό των κοινών μεταβλητών σε εκτέλεση πάνω σε cluster υλοποιεί μια υβριδική προσέγγιση, χρησιμοποιώντας και το DSM αλλά και το πρότυπο αποστολής μηνυμάτων MPI. Όλες οι καθολικές μεταβλητές που είναι εξ ορισμού κοινές τοποθετούνται στον κοινόχρηστο χώρο του sDSM. Για την επίλυση του προβλήματος της κοινοχρησίας και των μη καθολικών κοινών μεταβλητών, όπως αναφέρθηκε στην ενότητα 3.2.2, η αρχική διεργασία τοποθετεί τη στοίβα της στον κοινόχρηστο χώρο του DSM. Αυτό επιτυγχάνεται και με τη βοήθεια του μετασχηματιστή ο οποίος αλλάζει το όνομα της main του προγράμματός μας, μετονομάζοντας την σε `_original_main`. Η εκτέλεση ξεκινάει από την main που υπάρχει στο EELIB την οποία εκτελεί η αρχική διεργασία (οι έννοιες αρχική διεργασία και αρχικό νήμα έχουν την ίδια σημασία στην παρούσα αναφορά και αναφέρονται στην ίδια οντότητα) και αμέσως η εκτέλεση περνάει σε ένα νήμα χρήστη το οποίο αναλαμβάνει να τοποθετήσει τη στοίβα της διεργασίας στην κοινή μνήμη. Έτσι έχουν πρόσβαση σε όλες τις κοινές μεταβλητές όλες οι διεργασίες μέσω της κοινής μνήμης. Παρόλα αυτά, υπάρχουν και μεταβλητές του ORT οι οποίες τοποθετούνται στο block ελέγχου του αρχικού νήματος και αφορούν πληροφορίες δρομολόγησης, παραμέτρους εκτέλεσης κ.α. Σε αυτές τις μεταβλητές θα πρέπει να έχουν πρόσβαση όλες οι διεργασίες. Για αυτές τις μεταβλητές, ο OMPi χρησιμοποιεί το πρότυπο αποστολής μηνυμάτων MPI, αποφεύγοντας έτσι το sDSM για λόγους ταχύτητας. Αν μια διεργασία χρειαστεί μια πληροφορία την οποία κατέχει το αρχικό νήμα, το οποίο είναι και ο πατέρας της ομάδας εκτέλεσης, θα του στείλει ένα μήνυμα MPI με το οποίο θα ζητάει τη συγκεκριμένη πληροφορία. Το αρχικό νήμα με τη σειρά του θα της απαντήσει στέλνοντας την πληροφορία και αυτό με μήνυμα MPI.

Το μοντέλο που ακολουθεί ο OMPi για το χειρισμό των μηνυμάτων μεταξύ των διεργασιών, περιλαμβάνει ένα νήμα server και ένα νήμα υπολογισμού σε κάθε κόμβο (αρχική διεργασία εκτέλεσης του κόμβου). Όταν κάποια διεργασία χρειαστεί κάποια πληροφορία από το block ελέγχου του αρχικού νήματος, στέλνει την αίτηση του αρχικά στο νήμα server που του αντιστοιχεί. Αυτό στη συνέχεια αναλαμβάνει να προωθήσει το αίτημα αυτό στο νήμα server του αρχικού νήματος. Με τη σειρά του το τελευταίο, στέλνει ένα αντίγραφο του block ελέγχου του αρχικού νήματος, στην αιτούμενη διεργασία. Όλα τα νήματα server γνωρίζουν ποιο είναι ο το αρχικό νήμα (κόμβος 0) καθώς η πληροφορία αυτή τους δίνεται κατά τη διάρκεια της αρχικοποίησης. Όλη η παραπάνω διαδικασία ανταλλαγής μηνυμάτων επιτυγχάνεται μέσω MPI.

Με το EELIB των διεργασιών έχει συνδεθεί ένας αριθμός διαφορετικών πυρήνων DSM μέσω μιας διεπαφής. Για κάθε πυρήνα υπάρχει μια ενότητα κώδικα C που υλοποιεί τις ρουτίνες που χρειάζεται το EELIB για να χειριστεί τις συναρτήσεις του κατά περίπτωση πυρήνα (Σχήμα 5.3). Ο OMPi υποστηρίζει τα DSM: TreadMarks, JiaJia, Mocha, Parade και Mome. Αν και μεταξύ τους παρατηρείται μια κοινή λειτουργικότητα, υπάρχουν κάποια ζητήματα που χρειάζονται ειδική μεταχείριση και αφορούν το συγχρονισμό και τον τρόπο δέσμευσης μνήμης.



Σχήμα 3.3: Διεπαφή βιβλιοθήκης διεργασιών
με τους διάφορους πυρήνες DSM.

Στο επόμενο κεφάλαιο περιγράφεται η νέα βιβλιοθήκη του OMPi που χρησιμοποιεί πολυνηματική λογική στους κόμβους του cluster, αν αυτοί το υποστηρίζουν.

Κεφάλαιο 4

Υποστήριξη Πολλαπλών Νημάτων Ανά Κόμβο

4.1 Εισαγωγή

Με την όλο και αυξανόμενη διάδοση του OpenMP άρχισαν να αναπτύσσονται ακόμα περισσότεροι μεταφραστές που υποστηρίζουν το πρότυπο. Ο γνωστός GCC compiler, άρχισε να υποστηρίζει προγράμματα OpenMP από την έκδοση 4.2 μέσω του περιβάλλοντος GOMP [11]. Στην ίδια λογική κινήθηκε και ο Intel Compiler (ICC). Οι περισσότερες υλοποιήσεις όμως, είναι συνδεδεμένες με μια καθορισμένη βιβλιοθήκη νημάτων επιπέδου πυρήνα των οποίων η απόδοση στην περίπτωση του πολυεπίπεδου προγραμματισμού είναι χαμηλή. Σε αυτές τις περιπτώσεις είναι προτιμότερη η χρήση νημάτων επιπέδου χρήστη. Μια τέτοια υλοποίηση είναι αυτή του OMNI/ST [12] που παρέχει μια υλοποίηση νημάτων επιπέδου χρήστη με καλή απόδοση αλλά χωρίς φορητότητα. Όσον αφορά την εκτέλεση προγραμμάτων OpenMP πάνω σε συστάδες υπολογιστών, όλες οι υλοποιήσεις χρησιμοποιούν sDSM, μια ιδέα που παρουσιάστηκε πρώτη φορά στα [13, 14] όπου χρησιμοποιείται μια τροποποίηση του TreadMarks που είναι στενά συνδεδεμένη με τον μεταφραστή. Την ίδια λογική ακολουθεί και ο Intel Compiler με μια δική του τροποποίηση του TreadMarks [15]. Άλλες υλοποιήσεις μπορούμε να βρούμε στο NANOS Compiler [16] καθώς και στον Omni [17]. Η υλοποίηση όμως που μπορούμε να πούμε οτι πλησιάζει περισσότερο στον τρόπο λειτουργίας του OMPi, είναι το Parade [9] που εκτός από την κοινή μνήμη που προσφέρουν τα sDSM χρησιμοποιεί και την επικοινωνία μέσω MPI για θέματα συγχρονισμού και δρομολόγησης. Εκτός από αυτό το στοιχείο όμως συμπεριλαμβάνει και το χαρακτηριστικό της πολυνηματικής λειτουργίας σε κόμβους του cluster, κάτι που αποτελεί το στόχο της παρούσας πτυχιακής εργασίας για τον OMPi.

Ο OMPi για την εκτέλεση προγραμμάτων OpenMP σε συστάδες υπολογιστών, χρησιμοποιεί μια βιβλιοθήκη που παρέχει διεργασίες ως υπολογιστικές οντότητες, την οποία από εδώ και πέρα στις ενότητες που ακολουθούν θα αναφερόμαστε σε αυτή με το όνομα fproc. Σε κάθε κόμβο τρέχει μια διεργασία, η οποία με τη σειρά της δημιουργεί ένα νήμα server και περιμένει μέχρι να της ανατεθεί δουλειά από το αρχικό νήμα, το οποίο είναι και αυτό που εκτελεί τον κώδικα του προγράμματος. Το νήμα server του κάθε κόμβου είναι αυτό που θα λάβει τη δουλειά και θα την προωθήσει στο μοναδικό υπολογιστικό νήμα του κόμβου. Η όλη διαδικασία ακολουθεί μια πολύ λογική σειρά. Το θέμα όμως είναι πως εκμεταλλεύεται ο OMPi την ύπαρξη πολλών επεξεργαστών σε έναν κόμβο, αν υπάρχουν. Επίσης αντίθετα με τη λογική του cluster όπου δεν υφίσταται κοινή μνήμη, τουλάχιστον σε επίπεδο υλικού, σε έναν κόμβο του cluster και μεταξύ των επεξεργαστών του, υπάρχει φυσική κοινή μνήμη εσωτερικά. Στην περίπτωση αυτή ο OMPi δεν έχει σχεδιαστεί έτσι ώστε να μπορεί να εκμεταλλευτεί αποδοτικά την υπολογιστική δυνατότητα των πολλών πυρήνων και την ύπαρξη κοινής μνήμης μέσα σε έναν κόμβο της συστάδας. Πιό συγκεκριμένα, ο μόνος τρόπος για να εκμεταλλευτεί πολυπύρηνους κόμβους είναι η δημιουργία πολλαπλών ανεξάρτητων διεργασιών, οι οποίες είναι διαχειριστικά ακριβότερες από τα νήματα και δεν έχουν τη δυνατότητα να "βλέπουν" την κοινόχρηστη μνήμη.

Η συγκεκριμένη πτυχιακή εργασία υλοποιεί μια βιβλιοθήκη για την εκτέλεση

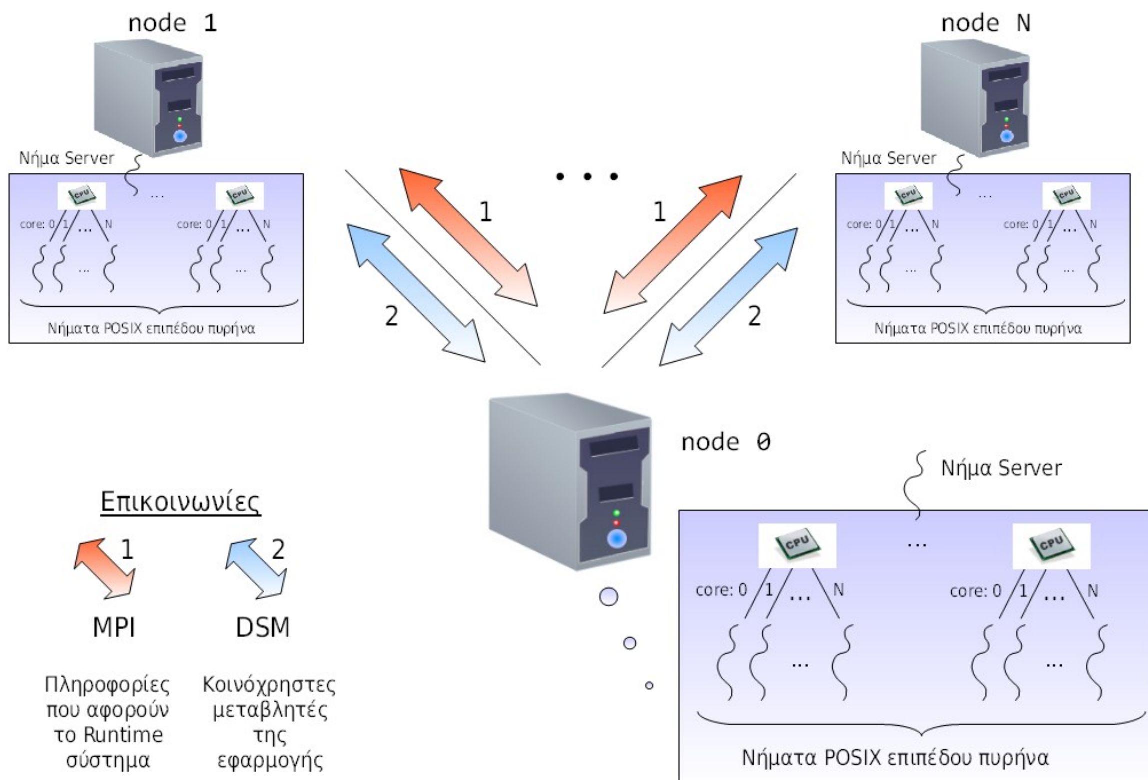
προγραμμάτων OpenMP πάνω σε συστάδες υπολογιστών, η οποία δημιουργεί νήματα σε κάθε πολυπύρηνο κόμβο που συμμετέχει στους υπολογισμούς του προβλήματος και εκμεταλλεύεται, όπου αυτό είναι εφικτό, την κοινόχρηστη μνήμη του. Η λογική του νήματος server συνεχίζει να υφίσταται και εδώ. Πλέον όμως, υπάρχει ένα νήμα server για όλα τα νήματα του κόμβου και αντί για τη δημιουργία πολλών νημάτων server που καθένα από αυτά θα αντιστοιχούσαν σε μια ανεξάρτητη διεργασία. Για τη ρύθμιση του αριθμού των νημάτων που θα δημιουργήσει κάθε κόμβος, έχει προστεθεί μια μεταβλητή περιβάλλοντος (`OMPI_THREADS_PER_NODE`). Περισσότερες λεπτομέρειες για την υλοποίηση της βιβλιοθήκης θα ακολουθήσουν στις επόμενες ενότητες.

4.2 Η Βιβλιοθήκη OPRC

Η βιβλιοθήκη OPRC που παρουσιάζεται σε αυτήν την ενότητα θα μπορούσε να χαρακτηριστεί ως μια υβριδική βιβλιοθήκη, καθώς προσφέρει ένα συνδυασμό διεργασιών και νημάτων (Σχήμα 4.1). Από εδώ και πέρα στις ενότητες που ακολουθούν, θα αναφερόμαστε στη βιβλιοθήκη αυτή με το χαρακτηρισμό υβριδική. Όπως και στην fproc έτσι και εδώ, μια διεργασία σε κάθε κόμβο ξεκινάει να τρέχει και δημιουργεί ένα νήμα server για να ακούει και να μεταφέρει τις αιτήσεις προς τον κόμβο του αρχικού νήματος. Η διαφορά στις δύο λογικές έγκειται στο οτι μια από τις αρμοδιότητες των διεργασιών αυτής της υβριδικής βιβλιοθήκης, είναι να δημιουργήσει και νήματα στον κόμβο της, που θα συμμετέχουν στους υπολογισμούς. Η εξ ορισμού συμπεριφορά της είναι να δημιουργήσει τόσα νήματα όσοι και οι πυρήνες του κόμβου. Τα νήματα αυτά είναι νήματα POSIX, επιπέδου πυρήνα. Επίσης το νήμα server του κόμβου πλέον δε θα εξυπηρετεί μόνο τις αιτήσεις της αρχικής διεργασίας του κόμβου, αλλά και τις αιτήσεις των νημάτων. Με αυτόν τον τρόπο εκμεταλλευόμαστε καλύτερα το χρόνο που περιμένει το server για να του έρθουν αιτήσεις. Η συμπεριφορά του νήματος server της βιβλιοθήκης περιγράφεται αναλυτικότερα σε επόμενη ενότητα (4.2.2) του κεφαλαίου.

Για τις επικοινωνίες των οντοτήτων της βιβλιοθήκης, χρησιμοποιείται η ίδια υβριδική λογική με την fproc. Από τη μια υπάρχει το sDSM στο οποίο αποθηκεύονται οι πληροφορίες που έχουν να κάνουν με την εφαρμογή (καθολικές μεταβλητές) και από την άλλη το πρότυπο αποστολής μηνυμάτων MPI για τις πληροφορίες που χρειάζονται να πάρουν οι οντότητες από το block ελέγχου του αρχικού νήματος (Σχήμα 4.1). Παρόλα αυτά σε αυτήν τη βιβλιοθήκη όπως προαναφέρθηκε, έχουν εισαχθεί νήματα που η χρήση τους έχει ως απώτερο σκοπό την εκμετάλλευση της κοινής μνήμης του κόμβου για να επικοινωνούν μεταξύ τους. Με αυτόν τον τρόπο "γλυτώνουμε" ένα μέρος των επικοινωνιών MPI. Τη λογική θα την εξετάσουμε σε επόμενες ενότητες του κεφαλαίου.

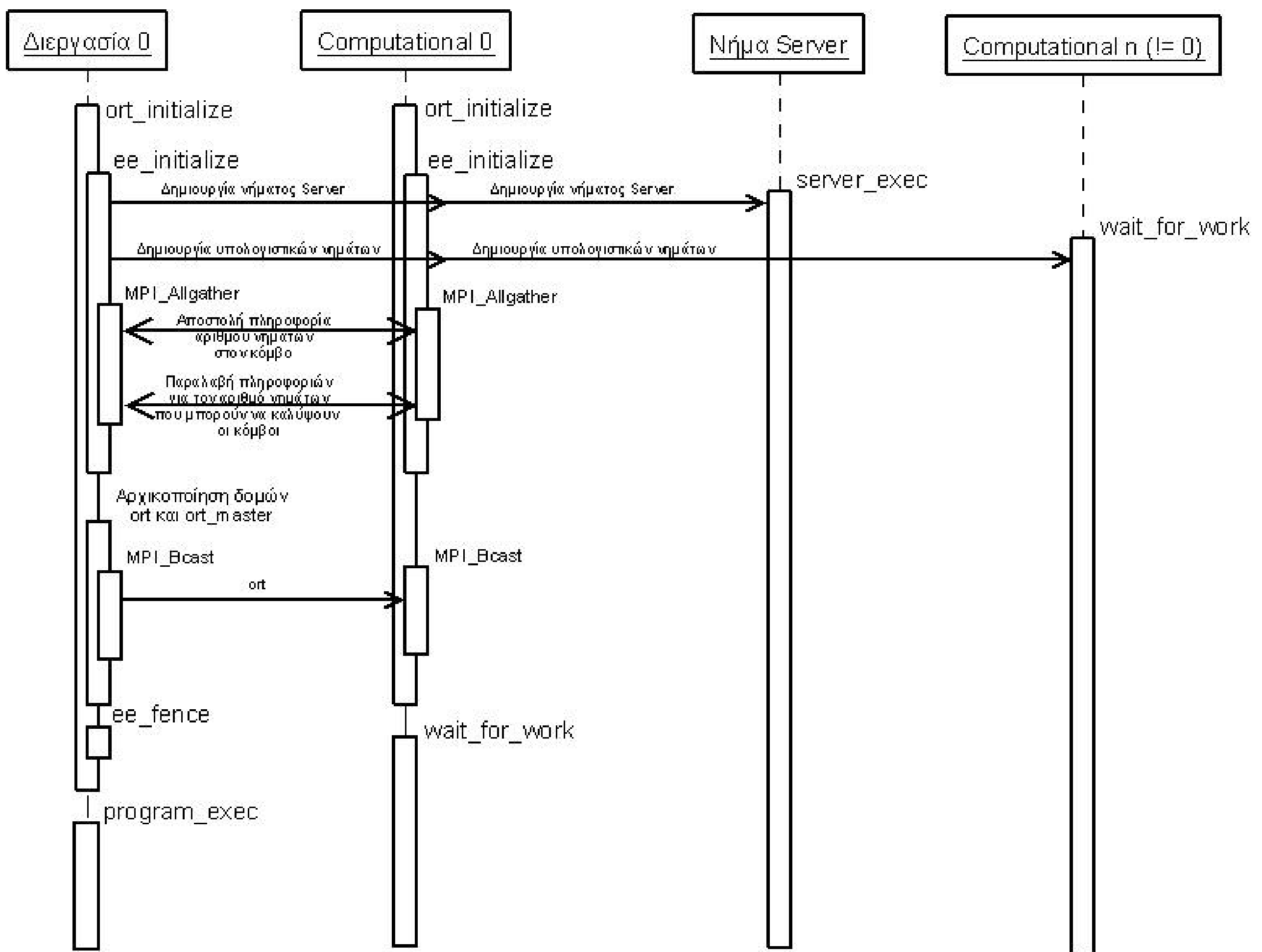
Η εκτέλεση της εφαρμογής ξεκινάει από την εν λόγω βιβλιοθήκη με τον ίδιο ακριβώς τρόπο όπως και στην fproc. Ο OMPi μετονομάζει την main της εφαρμογής σε `original_main` και η εκτέλεση ξεκινάει από την main της βιβλιοθήκης. Λόγω της κοινοχρησίας των μεταβλητών μεταξύ των διεργασιών, θα πρέπει η στοίβα της αρχικής διεργασίας να τοποθετηθεί στον κοινόχρηστο χώρο του DSM. Στην αρχή της main δημιουργείτε ένα νήμα επιπέδου χρήστη, του οποίου η στοίβα τοποθετείται στο DSM και στη συνέχεια η εκτέλεση περνάει σε αυτό. Με αυτόν τον τρόπο, η στοίβα της διεργασίας έχει τοποθετηθεί στην κοινή μνήμη και όλες οι μεταβλητές της αυτομάτως τοποθετούνται σε κοινόχρηστο χώρο διευθύνσεων.



Σχήμα 4.3: Αρχιτεκτονική και Επικοινωνίες της υβριδικής βιβλιοθήκης.

4.2.1 Διαδικασία Αρχικοποίησης

Η συνάρτηση αρχικοποίησης του συστήματος runtime (`ort_initialize()`), καλείται από όλες τις διεργασίες που συμμετέχουν στην εκτέλεση. Μια από τις πρώτες αρμοδιότητές της, είναι να καλέσει τη συνάρτηση αρχικοποίησης της βιβλιοθήκης (`oprc_initialize()`). Η τελευταία ουσιαστικά ανακοινώνει στην πρώτη ποιές είναι οι δυνατότητες με τις οποίες έχει σχεδιαστεί, όσον αφορά το δυναμικό ορισμό αριθμό διεργασιών και τον πολυεπίπεδο προγραμματισμό. Στην παρούσα βιβλιοθήκη δεν υποστηρίζεται πολυεπίπεδος προγραμματισμός. Σε κάθε επίπεδο μεγαλύτερο του πρώτου, οι παράλληλες περιοχές εκτελούνται σειριακά. Από την άλλη, υπάρχει η δυνατότητα δυναμικής προσαρμογής του αριθμού των διεργασιών. Ο μέγιστος αριθμός διεργασιών που θα συμμετέχουν στην εκτέλεση δίνεται μέσω της γραμμής εντολών τη στιγμή που δίνεται η εντολή εκτέλεσης του προγράμματος. Λόγω της χρήσης διεργασιών και της αντιστοιχίας ένα προς ένα με κάθε κόμβο του cluster, δεν είναι δυνατόν αυτός ο αριθμός να αλλάξει κατά τη διάρκεια της εκτέλεσης.



Σχήμα 4.2: Διάγραμμα ακολουθίας για τη διαδικασία αρχικοποίησης της βιβλιοθήκης.

Στη συνάρτηση αρχικοποίησης της βιβλιοθήκης, της οποίας τα βήματα φαίνονται στο Σχήμα 4.2, κάθε μια διεργασία αρχικοποιεί και το block ελέγχου της, το οποίο περιλαμβάνει πληροφορίες όπως το αναγνωριστικό της διεργασίας κατά την εκτέλεση, το αναγνωριστικό της αρχικής διεργασίας, τον αριθμό των διεργασιών που συμμετέχουν στον υπολογισμό μιας παράλληλης περιοχής, τον αριθμό των διεργασιών που συμμετέχουν στην εκτέλεση, το αναγνωριστικό του νήματος server του κόμβου. Πριν όμως από αυτή τη διαδικασία, οι διεργασίες μέσω της μεταβλητής περιβάλλοντος `OMPI_THREADS_PER_NODE`, της οποίας η τιμή έχει ληφθεί και έχει περάσει σε αυτές από την συνάρτηση αρχικοποίησης του συστήματος runtime, ενημερώνονται για το πόσα υπολογιστικά νήματα θα δημιουργήσουν. Αν η τιμή της είναι -1 (εξ ορισμού τιμή αν δεν έχει οριστεί), τότε δημιουργεί τόσα νήματα όσα και ο αριθμός των πυρήνων στον κόμβο. Σε αντίθετη περίπτωση, ο αριθμός των νημάτων που θα δημιουργήσει είναι ίσος με την τιμή της μεταβλητής, αν βέβαια αυτή δεν ξεπερνάει τον αριθμό των πυρήνων του κόμβου. Σε αυτήν την περίπτωση, ο αριθμός τους μειώνεται στον αριθμό των πυρήνων του κόμβου. Η βιβλιοθήκη έχει σχεδιαστεί έτσι ώστε να χειρίζεται και να δημιουργεί νήματα POSIX, επιπέδου πυρήνα και το πρώτο από τα νήματα που θα δημιουργηθούν, θα αναλάβει το ρόλο του νήματος server (Σχήμα 4.2). Τα υπόλοιπα θα λάβουν αναγνωριστικά όταν δημιουργηθούν και θα εκτελέσουν τη συνάρτηση αναμονής

εργασίας (`wait_for_work()`). Τα αναγνωριστικά των νημάτων ξεκινούν από τον αριθμό 1. Το 0, ανατίθεται στην αρχική διεργασία του κόμβου η οποία συμμετέχει και αυτή στους υπολογισμούς.

Για το αρχικό νήμα, δεν αρκεί να έχει επίγνωση μόνο του αριθμού των κόμβων που συμμετέχουν στο σύστημα και του αριθμού των τοπικών νημάτων, καθώς αργότερα θα καλεστεί να μοιράσει τη δουλειά στους κόμβους ανάλογα με τα νήματα που αυτοί προσφέρουν. Για αυτόν το λόγο, στο τελευταίο στάδιο της αρχικοποίησης της βιβλιοθήκης οι διεργασίες μέσω μιας ειδικής συνάρτησης MPI (`MPI_Allgather`) (Σχήμα 4.2), η οποία συγκεντρώνει και συνδυάζει σε έναν πίνακα με αριθμό θέσεων όσες και οι διεργασίες που συμμετέχουν σε αυτήν καλώντας την, έρχονται σε επικοινωνία, ενημερώνοντας η μια την άλλη για τον αριθμό των νημάτων που δημιούργησε. Αφού πλέον οι πληροφορίες αυτές είναι διαθέσιμες στο αρχικό νήμα, αρκεί με τη σειρά του να τις επεξεργαστεί για να πάρει τις επόμενες χρήσιμες πληροφορίες. Αυτές αφορούν αφενός τον τρόπο με τον οποίο θα απονέμει τα αναγνωριστικά των νημάτων που θα συμμετέχουν στους υπολογισμούς όταν θα χρειαστεί να μοιράσει τη δουλειά και αφετέρου το συνολικό αριθμό των νημάτων του συστήματος. Η πρώτη πληροφορία δίνεται με τη μέθοδο της προθεματικής πρόσθεσης πάνω στον πίνακα με τους αριθμούς των νημάτων κάθε κόμβου. Η πληροφορία αυτή χρειάζεται κατά τη στιγμή που το αρχικό νήμα θα χρειαστεί να μοιράσει τις δουλειές στους κόμβους και θα πρέπει μαζί με αυτές να στείλει και το αναγνωριστικό από το οποίο θα πρέπει να ξεκινήσει να δίνει τιμές στα αναγνωριστικά των νημάτων του κόμβου, το νήμα server του κόμβου. Η δεύτερη πληροφορία αφορά διαχειριστικές λειτουργίες του αρχικού νήματος που έχουν να κάνουν με διάφορα μέρη της εκτέλεσης. Αργότερα θα δούμε λεπτομερέστερα τη χρησιμότητα των εν λόγω πληροφοριών.

Στο τέλος της διαδικασίας αρχικοποίησης, το αρχικό νήμα ξεκινά την εκτέλεση του προγράμματος. Όλες οι άλλες διεργασίες των άλλων κόμβων αναλαμβάνουν το ρόλο υπολογιστικού νήματος, μαζί με τα νήματα που έχουν δημιουργήσει, εκτελώντας τη συνάρτηση `wait_for_work()`. Στη συγκεκριμένη συνάρτηση, τα νήματα αναμένουν για δουλειά.

4.2.2 Το Νήμα Server

Η λογική του νήματος server διατηρήθηκε και σε αυτήν τη βιβλιοθήκη παρόμοια με την fproc του OMPi. Κάθε κόμβος εκτός από τα υπολογιστικά νήματα, έχει ένα μοναδικό νήμα που επιτελεί το ρόλο του server. Η βιβλιοθήκη υλοποιεί το server ως νήμα POSIX επιπέδου πυρήνα. Το νήμα server τοποθετείται στο μοντέλο επικοινωνίας των οντοτήτων που συμμετέχουν στον OMPi, ανάμεσα στο αρχικό νήμα της εκτέλεσης και στα υπολογιστικά νήματα του κόμβου (Σχήμα 4.1). Οι αρμοδιότητές του είναι ουσιαστικά δύο.

- Να "ακούει" τις αιτήσεις των υπολογιστικών νημάτων του κόμβου και να τις προωθεί στο αρχικό νήμα. Οι αιτήσεις αυτές αφορούν την ανάγκη παραλαβής από πλευράς των νημάτων για πληροφορίες που κατέχει το αρχικό νήμα.
- Να παραλαμβάνει τις αιτήσεις από το αρχικό νήμα, προς τα υπολογιστικά νήματα του κόμβου. Οι αιτήσεις αυτές αφορούν την ανάθεση δουλειάς από το αρχικό νήμα προς τα υπολογιστικά νήματα του κόμβου, το συγχρονισμό μεταξύ των νημάτων του κόμβου, τη δέσμευση μνήμης κατά την αρχικοποίηση και τη

διαδικασία περάτωσης των νημάτων κατά το τέλος της εκτέλεσης του προγράμματος.

Πιο αναλυτικά οι αιτήσεις που μπορεί να διαχειριστεί το νήμα server έχουν ως εξής:

- ***PARALLEL.*** Το αίτημα αυτό αποστέλλεται από το αρχικό νήμα προς τα νήματα server όλων των κόμβων. Αφορά την ανάθεση δουλειάς στα υπολογιστικά νήματα του κόμβου. Το νήμα server αναλαμβάνει να λάβει από το μήνυμα τις πληροφορίες για τη δουλειά και να τις μοιράσει στον κατάλληλο αριθμό νημάτων του κόμβου.
- ***SYNCHRONIZE.*** Το αίτημα αυτό αποστέλλεται από το αρχικό νήμα προς τα νήματα server όλων των κόμβων. Η αποστολή του γίνεται κατά τη διάρκεια της δουλειάς όταν απαιτείται συγχρονισμός όλων των νημάτων του συστήματος που συμμετέχουν στους υπολογισμούς.
- ***FINALIZE.*** Αίτημα που επίσης αποστέλλεται από το αρχικό νήμα στα νήματα server των κόμβων. Είναι το αίτημα που σηματοδοτεί οτι έχει φτάσει το τέλος του προγράμματος και πρέπει να υλοποιηθούν οι απαραίτητες ενέργειες για την ομαλή περάτωση, όπως είναι ο τερματισμός των νημάτων και η απελευθέρωση της μνήμης των δομών που χρησιμοποιεί ο κόμβος.
- ***SHMALLOC.*** Αίτημα που ανάλογα με το DSM που έχει επιλεγεί, στέλνεται μόνο στον κόμβο 0 (αρχικό νήμα) ή σε όλους τους κόμβους κατά την αρχικοποίηση και αφορά τη δέσμευση μνήμης για το DSM.
- ***ORT.*** Αίτημα που αποστέλλεται από τα υπολογιστικά νήματα του κόμβου, προς το server. Το αίτημα αφορά την ανάγκη του υπολογιστικού νήματος για πληροφορίες που βρίσκονται στο block ελέγχου του αρχικού νήματος όπως για παράδειγμα οι επαναλήψεις ενός for που πρέπει να του ανατεθούν. Το server με τη σειρά του, αναλαμβάνει να προωθήσει το αίτημα στο αρχικό νήμα.

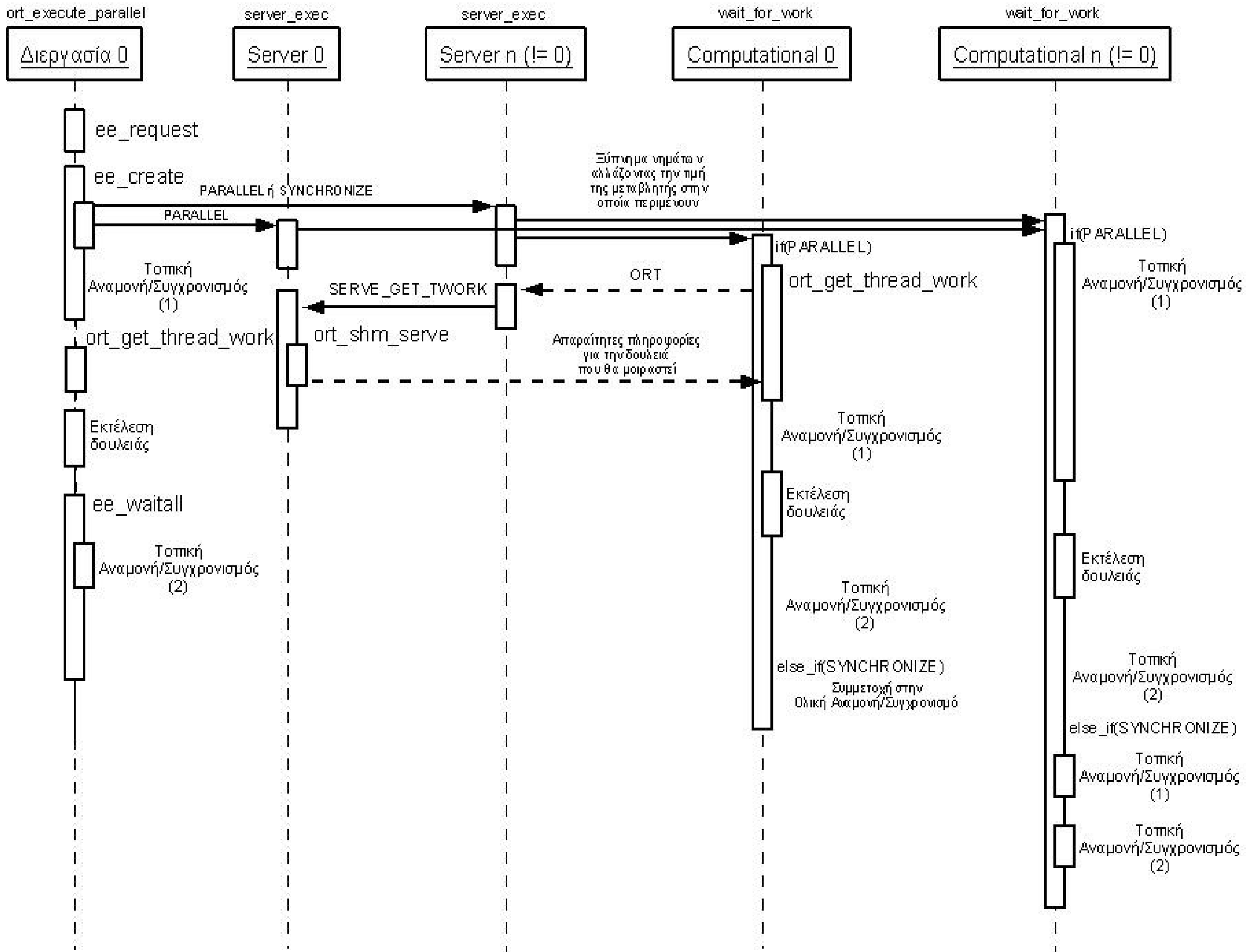
4.2.3 Εκτέλεση Παράλληλης Περιοχής

Με την εμφάνιση μιας παράλληλης περιοχής σε πρώτο επίπεδο παραλληλισμού, το αρχικό νήμα, το οποίο είναι και αυτό που διατρέχει το πρόγραμμα, θα κληθεί να μοιράσει τη δουλειά στους κόμβους και κατ' επέκταση στα υπολογιστικά νήματα αυτών. Η διαδικασία αυτή επιτελείται στην συνάρτηση `oprc_create()` της υβριδικής βιβλιοθήκης και ανάμεσα στα ορίσματα αυτής, είναι ο αριθμός των νημάτων που θα συμμετέχουν στους υπολογισμούς, το επίπεδο του παραλληλισμού καθώς και η συνάρτηση την οποία θα εκτελέσουν τα νήματα. Πιό συγκεκριμένα το αρχικό νήμα θα πρέπει να εκτελέσει τα παρακάτω βήματα (Σχήμα 4.3):

1. Να δημιουργήσει το μήνυμα που θα στείλει και να το δομήσει εισάγοντας σε αυτό όλες τις απαραίτητες πληροφορίες για τη δουλειά που πρέπει να εκτελέσουν τα υπολογιστικά νήματα. Το είδος του μηνύματος είναι PARALLEL.
2. Να μοιράσει τη δουλειά ανάμεσα στα νήματα των κόμβων. Η ανάθεση της δουλειάς στους κόμβους γίνεται σειριακά ξεκινώντας από τον κόμβο 0 και συνεχίζει μέχρι να τελειώσει ο αριθμός των νημάτων που συμμετέχουν στους υπολογισμούς. Αυτό συνεπάγεται πως κάποιοι κόμβοι ή κάποια νήματα κόμβων μπορεί να μην έχουν δουλειά. Παρόλα αυτά το αρχικό νήμα θα στείλει σε όλους του κόμβους αίτημα για δουλειά κυρίως για λόγους που έχουν να κάνουν με το

συγχρονισμό ανεξάρτητα από το αν μια οντότητα έχει δουλειά ή όχι.

3. Να αναλάβει και αυτό με τη σειρά του ένα μέρος της δουλειάς, να γίνει δηλαδή υπολογιστικό νήμα για τον κόμβο.



Σχήμα 4.3: Διάγραμμα ακολουθίας για τη διαδικασία ανάθεσης δουλειάς στους κόμβους.

Τα νήματα server σε όλους τους κόμβους, με το που λάβουν το μήνυμα (αίτημα PARALLEL) από το αρχικό νήμα (βήμα 2), θα "εξορύξουν" τις πληροφορίες που χρειάζεται από τη δομή του μηνύματος. Η πρώτη πληροφορία στην οποία θα ανατρέξει ένα νήμα server, είναι ο αριθμός των νημάτων που θα χρησιμοποιηθούν από τον κόμβο του. Αν ο αριθμός είναι 0, δηλαδή δε συμμετέχει κανένα νήμα του κόμβου στους υπολογισμούς, τότε το server δεν επιτελεί κάποια ιδιαίτερη λειτουργία. Αυτό δε σημαίνει οτι ο κόμβος θα μείνει ανενεργός για όλη τη διάρκεια των υπολογισμών. Θα χρειαστεί αργότερα να συμμετέχει στο συγχρονισμό. Σε αντίθετη περίπτωση, όπου το server πρέπει να ξυπνήσει έναν αριθμό νημάτων, καλείται να μοιράσει τη δουλειά στα νήματα (`AssignWork()`). Αναλαμβάνει να αντιγράψει τα δεδομένα του μηνύματος στη δομή δουλειάς του αντίστοιχου νήματος και στη συνέχεια να αλλάξει την τιμή αναγνωριστικού της δουλειάς για το νήμα. Τη διαδικασία αυτή την επιτελεί σειριακά για όλα τα νήματα του κόμβου που συμμετέχουν στους υπολογισμούς. Τα νήματα του

κόμβου που δεν θα έχουν συμμετοχή στους υπολογισμούς, θα εκτελέσουν ένα τοπικό POSIX `barrier` (βλ. 4.2.4).

Όσον αφορά τα υπολογιστικά νήματα του κόμβου, ξυπνάνε με το που τους αναθέσει δουλειά το server, αλλάζοντας την τιμή της μεταβλητής στην οποία περιμένουν. Εδώ ξεκινάει μια διαφοροποίηση στον τρόπο διαχείρισης των νημάτων, ανάλογα αν το νήμα είναι το master του κόμβου (αρχική διεργασία του κόμβου) ή ένα οποιοδήποτε άλλο υπολογιστικό νήμα του κόμβου που συμμετέχει στους υπολογισμούς. Στην πρώτη περίπτωση, το νήμα master του κόμβου, θα έρθει μέσω της συνάρτησης `ort_get_thread_work()` σε επικοινωνία (μηνύματα MPI) με το αρχικό νήμα για τη λήψη πληροφοριών που αφορούν την ομάδα των νημάτων που θα εκτελέσουν τους υπολογισμούς. Οι πληροφορίες αυτές αφορούν τον αριθμό των νημάτων που συμμετέχουν συνολικά στους υπολογισμούς, το επίπεδο παραλληλισμού καθώς και το δείκτη για την κοινή μνήμη. Οι πληροφορίες αυτές αποθηκεύονται σε μια καθολική μεταβλητή του κόμβου από το νήμα master. Με την περάτωση της διαδικασίας ακολουθεί περίπου η ίδια διαδικασία και για τα υπόλοιπα νήματα του κόμβου. Η διαφορά τώρα είναι οτι δεν θα χρειαστεί να έρθουν σε επικοινωνία με το αρχικό νήμα για να πάρουν τις απαραίτητες πληροφορίες. Λόγω της κοινής μνήμης (μοντέλο κοινού χώρου διευθύνσεων) του κόμβου και της κοινής προσπέλασης των καθολικών μεταβλητών του κόμβου από όλα τα νήματα, αρκεί να προσπελάσουν την καθολική μεταβλητή στην οποία νωρίτερα το νήμα master αποθήκευσε τις απαραίτητες πληροφορίες. Όταν ολοκληρωθεί η διαδικασία, όλα τα νήματα είναι έτοιμα να εκτελέσουν τη δουλειά που τους αντιστοιχεί. Αφού εκτελέσουν τη δουλειά, επιστρέφουν στην κατάσταση αναμονής δουλειάς. Αξίζει να σημειωθεί οτι πριν και μετά την εκτέλεση της δουλειάς, τα νήματα συγχρονίζονται μεταξύ τους. Τα θέματα συγχρονισμού αναλύονται στην αμέσως επόμενη ενότητα.

4.2.4 Συγχρονισμός

Το OpenMP κατά τις οδηγίες συγχρονισμού επιβάλει και ένα συγχρονισμό μνήμης για την καλύτερη συνοχή των μεταβλητών. Στους περισσότερους πυρήνες sDSM, χρησιμοποιούνται χαλαρά πρωτόκολλα συνοχής μνήμης. Κατά την εκτέλεση λοιπόν, μιας εντολής συγχρονισμού είτε αυτή είναι του OpenMP, είτε του sDSM, μεταξύ των οντοτήτων που συμμετέχουν στους υπολογισμούς επιτελείτε και στον OMPi, συγχρονισμός μνήμης.

Στη βιβλιοθήκη εμφανίζονται δύο τύποι συγχρονισμού. Ο ολικός, που αφορά τους κόμβους που συμμετέχουν στην εκτέλεση και ο τοπικός, που αφορά τα υπολογιστικά νήματα του κόμβου. Όσον αφορά τον ολικό συγχρονισμό υπάρχουν δύο μέθοδοι πραγμάτωσης του και τις περισσότερες φορές συντελείται μετά από αίτημα SYNCHRONIZE που αποστέλλει το αρχικό νήμα. Υπάρχουν επίσης και δύο τύποι δευτερευόντων αναγνωριστικών αιτημάτων συγχρονισμού, ο τύπος OPRC και ο τύπος MPI. Ο πρώτος αναφέρεται σε συγχρονισμό του MPI με τη χρήση της συνάρτησης `MPI_Barrier()` και ο δεύτερος αναφέρεται σε συγχρονισμό του sDSM με τη χρήση της `oprc_barrier()`. Όσον αφορά τον τοπικό συγχρονισμό μεταξύ των υπολογιστικών νημάτων του κόμβου, αυτός υλοποιείται με συστατικά που προσφέρει η βιβλιοθήκη των νημάτων POSIX. Τα νήματα του κόμβου που συμμετέχουν στους υπολογισμούς, εκτελούν αυτόν τον τοπικό συγχρονισμό καλώντας την κατάλληλη συνάρτηση μετά από τις απαραίτητες ενέργειες που πρέπει να κάνουν. Το πρόβλημα

παρουσιάζεται για τα νήματα του κόμβου που δεν έχουν δουλειά. Λόγω της μη ύπαρξης δυνατότητας δυναμικού ορισμού του αριθμού νημάτων που θα εκτελέσουν το συγκεκριμένο barrier, είναι αναγκαία η εκτέλεση του από όλα τα νήματα του κόμβου ανεξάρτητα από το αν έχουν δουλειά ή όχι. Αυτό επιτυγχάνεται με το να καλέσει το master νήμα του κόμβου τη συνάρτηση ανάθεσης δουλειάς (`AssignWork()`), πριν από την εκτέλεση του εν λόγω barrier για λογαριασμό της. Με αυτόν τον τρόπο προωθεί ένα αίτημα SYNCHRONIZE σε όλα τα νήματα του κόμβου που δεν συμμετέχουν στους υπολογισμούς, με δευτερεύον αναγνωριστικό αιτήματος συγχρονισμού το LOCAL. Με τη σειρά τους, ξυπνάνε και εκτελούν το τοπικό POSIX barrier. Το ίδιο πρόβλημα όμως εμφανίζεται και στον ολικό συγχρονισμό για τους κόμβους που δεν τους έχει ανατεθεί δουλειά. Στην περίπτωση αυτή το πρόβλημα λύνεται εύκολα, καθώς το αρχικό νήμα αποστέλλει αίτημα SYNCHRONIZE στους κόμβους αυτούς ώστε να συμμετέχουν στο barrier.

Στη βιβλιοθήκη για να υλοποιηθούν οι διαδικασίες συγχρονισμού, έχουν δημιουργηθεί δύο συναρτήσεις που καλούνται `oprc_barrier_wait()` και `oprc_barrierall()`. Η διαφορά τους έγκειται στους τύπους συγχρονισμού που καλούν. Στην πρώτη συνάρτηση, στην περίπτωση που την καλεί το αρχικό νήμα, δημιουργείται ένα αίτημα SYNCHRONIZE με δεύτερο αναγνωριστικό αιτήματος το OPRC και αποστέλλεται σε όλους τους κόμβους που δε συμμετέχουν στην εκτέλεση. Στη συνέχεια καλείται barrier του sDSM. Στην περίπτωση οποιουδήποτε άλλου νήματος, καλείται απλά το barrier του sDSM. Για τη δεύτερη συνάρτηση, υπάρχουν τρεις διαφορετικές περιπτώσεις εκτέλεσης της. Η πρώτη περίπτωση αφορά το αρχικό νήμα, το οποίο δημιουργεί ένα αίτημα SYNCHRONIZE με δεύτερο αναγνωριστικό αιτήματος MPI και αποστέλλει το αίτημα σε όλους τους κόμβους που δε συμμετέχουν στην εκτέλεση. Η δεύτερη περίπτωση αφορά τα νήματα με αναγνωριστικό 0 (νήμα master) σε όλους τους κόμβους. Σε αυτήν την περίπτωση, αυτά εκτελούν αρχικά ένα τοπικό barrier και στη συνέχεια εκτελούν συγχρονισμό MPI. Η τρίτη περίπτωση αναφέρεται στα υπόλοιπα νήματα που θα καλέσουν τη συνάρτηση και στην περίπτωση αυτή θα εκτελέσουν τοπικό barrier.

4.2.5 Διαδικασία Περάτωσης Εκτέλεσης

Όταν η εκτέλεση έχει φτάσει στο τέλος, η αρχική διεργασία καλεί τη συνάρτηση `oprc_finalize()`. Το αρχικό νήμα δημιουργεί ένα αίτημα FINALIZE και το στέλνει σε όλους τους κόμβους, συμπεριλαμβανομένου και του δικού του. Ας μην ξεχνάμε οτι τα αιτήματα τα παραλαμβάνει το νήμα server του κόμβου. Το νήμα server, μόλις παραλάβει το αίτημα το προωθεί σε όλα τα υπολογιστικά νήματα και στη συνέχεια τερματίζει την εκτέλεσή του. Όλα τα νήματα εκτός από το νήμα master του κόμβου, απλά ξυπνάνε από την αναμονή τους και τερματίζουν. Το νήμα master του κόμβου πριν τερματίσει θα πρέπει να επιτελέσει κάποιες απαραίτητες ενέργειες για την ομαλή περάτωση της εκτέλεσης για τον κόμβο, καλώντας τη συνάρτηση `oprc_finalize()` για λογαριασμό του. Όλες πλέον οι αρχικές διεργασίες των κόμβων ακολουθούν μια σειρά ενεργειών. Αρχικά ολοκληρώνουν τη διαδικασία για την περάτωση της ύπαρξης των νημάτων και εκτελούν ένα συγχρονισμό MPI. Στη συνέχεια ελευθερώνουν το χώρο μνήμης που είχε δεσμευτεί για δομές που χρειαζόταν ο κόμβος για την αποθήκευση πληροφοριών. Η τελευταία αρμοδιότητα των διεργασιών είναι να καλέσουν τη συνάρτηση τερματισμού του sDSM.

4.3 Διαχείριση της ORT

Στην ενότητα ORT του OMPi, ορίζονται μεταβλητές στις οποίες όλες οι οντότητες που συμμετέχουν στους υπολογισμούς, πρέπει να έχουν πρόσβαση. Οι μεταβλητές αυτές αφορούν πληροφορίες που υπάρχουν στο block ελέγχου του αρχικού νήματος καθώς και τιμές των μεταβλητών περιβάλλοντος. Για λόγους αποδοτικότητας αυτές οι μεταβλητές δεν τοποθετούνται στην κοινή μνήμη του sDSM. Ο υβριδικός σχεδιασμός (Σχήμα 4.1) του OMPi επιτρέπει τη χρήση του προτύπου αποστολής μηνυμάτων MPI, ώστε να μπορούν αυτές οι πληροφορίες να μεταφέρονται στα νήματα των κόμβων. Όταν κάποιο νήμα χρειαστεί να προσπελάσει κάποιο από αυτά τα δεδομένα, δημιουργεί ένα αίτημα το οποίο στέλνει στο νήμα server του κόμβου του. Το αίτημα αυτό έχει να κάνει με τις μεταβλητές δρομολόγησης εργασίας των τμημάτων διαμοιρασμού εργασίας και τις περισσότερες των περιπτώσεων αφορά μια απλή αλλαγή στις τιμές αυτών από το αρχικό νήμα, το οποίο στη συνέχεια στέλνει στο υπολογιστικό νήμα του κόμβου τη νέα τιμή.

Όπως κάθε κοινόχρηστη μεταβλητή έτσι και αυτές του ORT θα πρέπει να προστατευτούν από ταυτόχρονες προσπελάσεις με τη χρήση κλειδαριών. Λόγω της ύπαρξης και των νημάτων σε κάθε κόμβο, πρέπει η προστασία αυτή να επιτευχθεί σε δύο στάδια. Σε πρώτο στάδιο αφορά το κλείδωμα από τον κόμβο του cluster και σε δεύτερο στάδιο αφορά το κλείδωμα από ένα νήμα του κόμβου. Η κλειδαριά του πρώτου σταδίου υλοποιείται σε επίπεδο sDSM, ενώ αυτή του δεύτερου στο EELIB με συναρτήσεις που προσφέρει η βιβλιοθήκη νημάτων POSIX. Οι πιό συχνές περιπτώσεις που χρειάζεται αυτού του τύπου η διπλή προστασία, είναι στη διαδικασία συγκέντρωσης αποτελεσμάτων (`reduction`) καθώς και στις λειτουργίες `atomic`.

Τέλος, αξίζει να σημειωθεί πως όλες οι δομές διαμοίρασης έργου σε αυτή την υλοποίηση είναι εμποδιστικές (blocking), πράγμα που συνεπάγεται πως μόνο μια δομή μπορεί να είναι ενεργή. Για να ολοκληρωθεί θα πρέπει όλα τα νήματα που συμμετέχουν σε αυτή να ολοκληρώσουν τη δουλειά τους.

Κεφάλαιο 5

Πειραματικά Αποτελέσματα

5.1 Εισαγωγικές Πληροφορίες για τα Πειράματα

Σε αυτό το κεφάλαιο παρουσιάζονται τα αποτελέσματα του OMPi με τη χρήση της υβριδικής βιβλιοθήκης που παρουσιάστηκε στο Κεφάλαιο 4, η οποία αποτελεί και το αντικείμενο της πτυχιακής. Όλα τα πειράματα εκτελέστηκαν στο *SUN X4100* cluster του τμήματος *Πληροφορικής Ιωαννίνων*. Το cluster αποτελείται από 16 κόμβους, καθένας από τους οποίους έχει 2 CPUs και κάθε CPU έχει δύο πυρήνες *AMD Opteron 200 series*, με λειτουργικό σύστημά *Linux masterprivate 2.6.25-2-amd64 #1 SMP Mon Jul 14 11:05:23 UTC 2008 x86_64 GNU/Linux*. Οι κόμβοι είναι διασυνδεδεμένοι με Gigabit Ethernet. Το DSM που χρησιμοποιήθηκε από τη βιβλιοθήκη για την επίτευξη των πειραμάτων είναι το TreadMarks Version 1.0.3.3. Οι επικοινωνίες MPI έγιναν με την βιβλιοθήκη MPICH2-1.1.1p1, η οποία χρησιμοποιήθηκε με το κανάλι ch3: sock για το οποίο μετά από μετρήσεις και πειραματισμούς διαπιστώθηκε οτι είναι το πιο αποδοτικό για τη λογική που ακολουθεί ο OMPi. Η έκδοση του OMPi που χρησιμοποιήθηκε για τα πειράματα είναι η 1.0.0.

Στις επόμενες δύο ενότητες που ακολουθούν, παρουσιάζονται και αναλύονται τα αποτελέσματα των πειραμάτων. Η πρώτη, αναφέρεται στα EPCC Microbenchmarks και έχει ως απώτερο στόχο να συγκρίνει αφενός τη υβριδική βιβλιοθήκη με την fproc του OMPi και αφετέρου διάφορες διαμορφώσεις των νημάτων στους κόμβους του cluster. Η δεύτερη ενότητα, αναφέρεται στα αποτελέσματα εκτέλεσης παράλληλων εφαρμογών των οποίων ο κώδικας μεταγλωττίστηκε από την υβριδική βιβλιοθήκη. Τα πειράματα αυτά έχουν ως στόχο να διαπιστωθεί αν πράγματι βελτιώνεται η επίδοση της εκτέλεσης ενός προγράμματος όταν αυξάνουμε τους κόμβους άρα και τα νήματα που συμμετέχουν στους υπολογισμούς με αυτήν τη νέα βιβλιοθήκη.

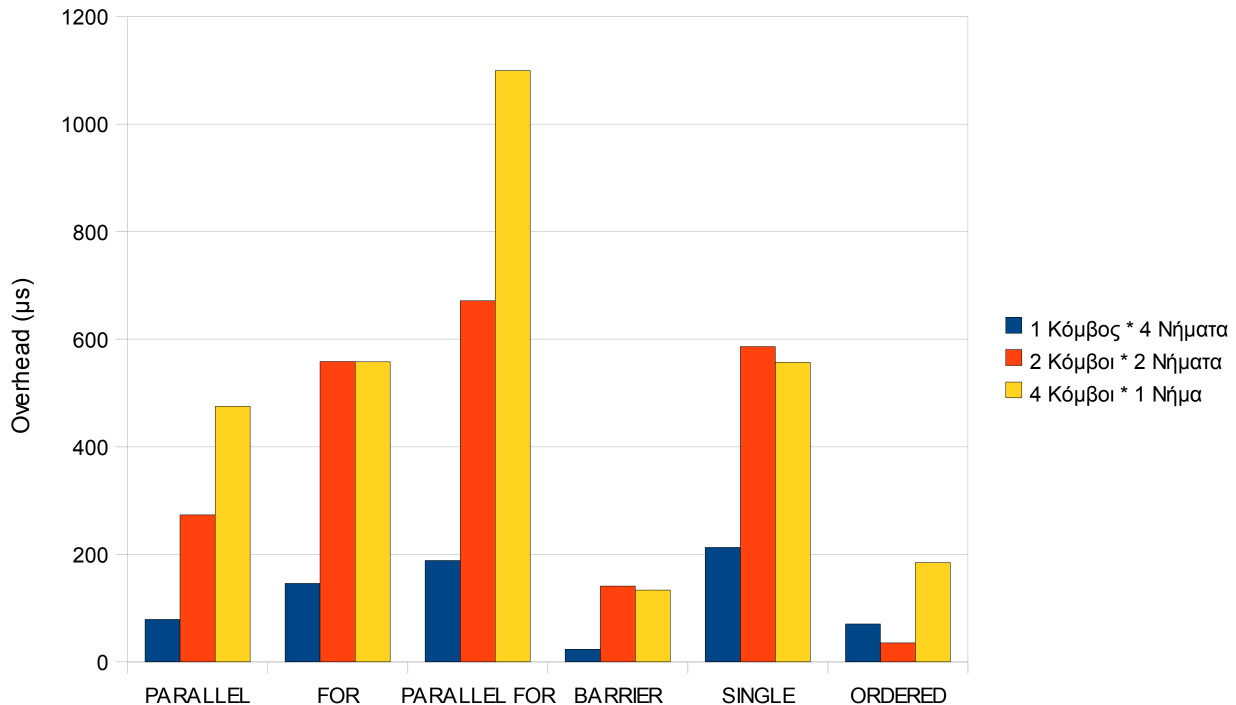
5.2 EPCC Microbenchmarks

Τα EPCC Microbenchmarks [18], είναι μικρά προγράμματα που σκοπό έχουν να μετρήσουν την επιβάρυνση των οδηγιών OpenMP, δηλαδή το χρόνο που απαιτούν για να ολοκληρωθούν ανάλογα και με τα νήματα που συμμετέχουν σε αυτές. Τα προγράμματα αυτά χωρίζονται σε δύο κατηγορίες, schedbench και syncbench. Κάθε κατηγορία εξετάζει διαφορετικές λειτουργίες. Η schedbench κατηγορία αποτελείται από διάφορες δομές επανάληψης “for” οι οποίες έχουν παραλληλοποιηθεί με OpenMP και οι επαναλήψεις τους έχουν δρομολογηθεί με διάφορες παραμέτρους. Η syncbench κατηγορία εξετάζει τις επιβαρύνσεις όλων των οδηγιών του OpenMP γενικότερα.

Για τα EPCC Microbenchmarks, πραγματοποιήθηκαν τρία διαφορετικά είδη πειραμάτων, καθένα για διαφορετικούς σκοπούς. Στις διάφορες εκτελέσεις μεταβαλλόταν ο αριθμός των νημάτων και των κόμβων του cluster που θα συμμετείχαν σε αυτές. Ο κώδικας των microbenchmarks εκτελέστηκε χωρίς να γίνουν αλλαγές. Στην παρούσα πτυχιακή θα αναλυθούν τα αποτελέσματα των εκτελέσεων της κατηγορίας syncbench. Για λόγους παρουσίασης θα παρατεθούν τα αποτελέσματα για τις οδηγίες `parallel`, `for`, `parallel for`, `barrier`, `single` και `ordered`.

5.2.1 Πείραμα 1: Σταθερός αριθμός 4 νημάτων

Στο πρώτο πείραμα εκτελέστηκαν τα Microbenchmarks στους κόμβους του cluster κρατώντας σταθερά τον αριθμό των υπολογιστικών νημάτων σε 4, μεταβάλλοντας όμως τον αριθμό των κόμβων που θα συμμετείχαν στους υπολογισμούς. Σκοπός του πειράματος ήταν να διαπιστωθούν τα πλεονεκτήματα της χρήσης νημάτων όσον αφορά τη μείωση των επικοινωνιών και την εκμετάλλευση κοινής μνήμης σε ένα κόμβο. Εμφανίζονται λοιπόν 3 διαφορετικές εκτελέσεις, όπου στην πρώτη συμμετέχει ένας κόμβος με 4 νήματα, στην δεύτερη 2 κόμβοι με 2 νήματα ο καθένας και στην τρίτη 4 κόμβοι με 1 νήμα ο καθένας.



Σχήμα 5.1: Overheads για διαφορετικές εκτελέσεις
με 4 υπολογιστικά νήματα.

Όπως ήταν αναμενόμενο, η καλύτερη απόδοση παρατηρείται στην εκτέλεση όπου τα 4 υπολογιστικά νήματα βρίσκονται στον ίδιο κόμβο και η χαμηλότερη όταν αυτά μοιράζονται σε 4 κόμβους. Το γεγονός οτι στην υλοποίηση των οδηγιών OpenMP χρησιμοποιούνται επικοινωνίες MPI και πιο συγκεκριμένα πρέπει να επικοινωνήσουν με τον κόμβο του αρχικού νήματος. Η καλύτερη απόδοση της πρώτης εκτέλεσης (χαμηλότερο overhead) οφείλεται στο γεγονός οτι ο κόμβος του αρχικού νήματος είναι ο μοναδικός που συμμετέχει στους υπολογισμούς και οι μόνες επικοινωνίες MPI που πρέπει να γίνουν, αφορούν τα υπολογιστικά νήματα του κόμβου με το νήμα server του κόμβου. Στη δεύτερη εκτέλεση, όπου τα 4 νήματα μοιράζονται ισομερώς σε 2 κόμβους, παρατηρείται μια σχετική μείωση της απόδοσης καθώς αυξάνονται αναλογικά με τους κόμβους και οι επικοινωνίες MPI. Αυτός είναι και ο λόγος που η τρίτη εκτέλεση έχει και τη χειρότερη απόδοση. Επίσης σε αυτήν την περίπτωση, στην οποία δημιουργείται 1 υπολογιστικό νήμα σε κάθε κόμβο, δεν αξιοποιείται η κοινόχρηστη μνήμη του κόμβου

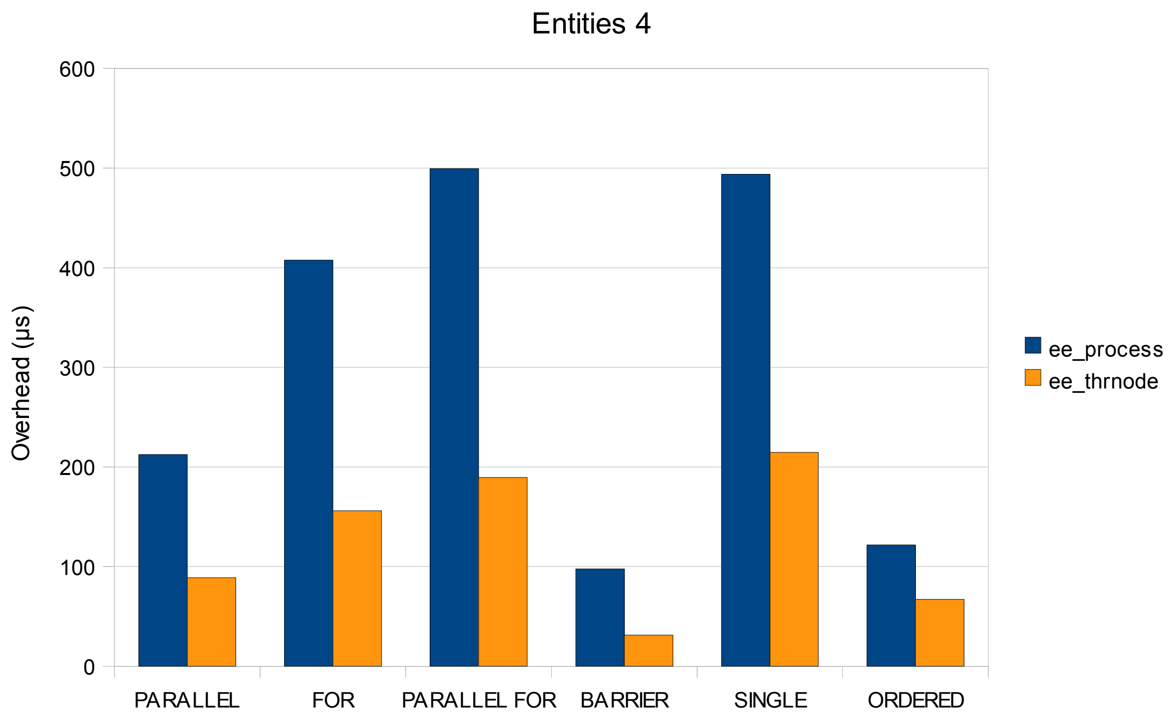
και για τις πληροφορίες που θα μπορούσε να παρείχε αυτή, το νήμα διαφεύγει στη λύση του MPI. Αξίζει να σημειωθεί οτι για την οδηγία `ordered` τα αποτελέσματα διέφεραν σε σχέση με τις άλλες οδηγίες. Η πρώτη εκτέλεση στην οποία έχουμε έναν κόμβο πλήρη, με την έννοια οτι υποστηρίζουμε σε αυτόν το μέγιστο αριθμό νημάτων σε αντιστοιχία, έχει χαμηλότερη απόδοση από οτι η δεύτερη εκτέλεση με 2 νήματα σε 2 κόμβους. Αυτό ερμηνεύεται από το γεγονός οτι σε ένα πλήρη κόμβο για το παράδειγμά μας (4 νήματα), εμφανίζεται εντονότερος ανταγωνισμός μεταξύ των νημάτων του για να πάρουν δουλειά παρά σε έναν κόμβο με λιγότερα νήματα. Η απόδοση της τρίτης εκτέλεσης είναι αναμενόμενη καθώς αυξάνονται οι επικοινωνίες MPI.

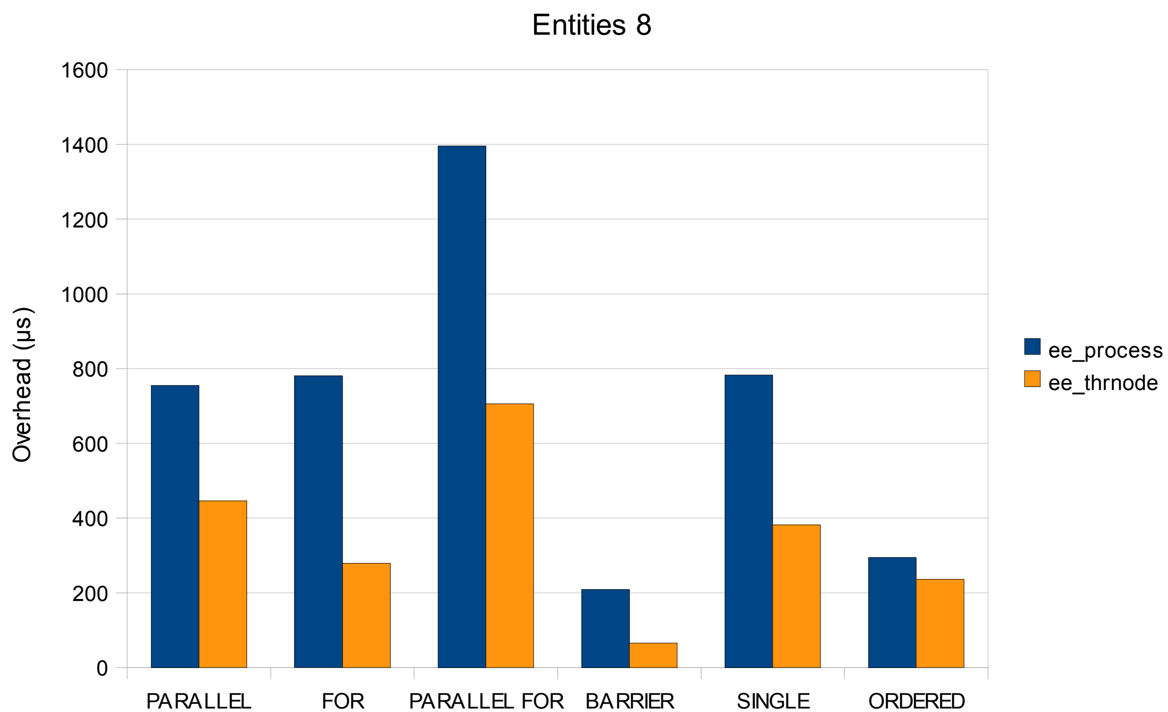
5.2.2 Πείραμα 2: Σύγκριση με τη βιβλιοθήκη διεργασιών του OMPi

Στο δεύτερο πείραμα, γίνεται μια σύγκριση των δύο βιβλιοθηκών, της fproc του OMPi (ee_process) και της υβριδικής βιβλιοθήκης που παρουσιάστηκε στην παρούσα πτυχιακή (ee_thrnode). Στόχος ήταν να αναδειχτούν τα πλεονεκτήματα της χρήσης της υβριδικής βιβλιοθήκης έναντι της fproc, τα οποία αφορούν την εκμετάλλευση της κοινόχρηστης μνήμης που υπάρχει στον κόμβο και τη μείωση των επικοινωνιών καθώς και τα πλεονεκτήματα των νημάτων έναντι των διεργασιών όσον αφορά διαχειριστικούς λόγους (νήμα = ελαφριά διεργασία). Το πείραμα αυτό έγινε σε δύο στάδια, που διαφέρουν ως προς τον αριθμό των υπολογιστικών οντοτήτων που συμμετέχουν στους υπολογισμούς για να βγουν ασφαλέστερα συμπεράσματα. Σε πρώτο στάδιο η εκτέλεση έγινε για 4 οντότητες και σε δεύτερο στάδιο έγινε για 8 οντότητες.

	Πείραμα 2α	**Πείραμα 2β**
Οντότητες (Κόμβοι)	4 (1)	8 (2)
ee_process	4 διεργασίες	8 διεργασίες
ee_thrnode	1 διεργασία -> 4 νήματα	2 διεργασίες -> 8 νήματα

Αξίζει να σημειωθεί οτι και στα δύο στάδια εκτελέσεων για τις δύο βιβλιοθήκες χρησιμοποιήθηκε ο ίδιος αριθμός κόμβων. Στο πρώτο στάδιο συμμετείχε 1 κόμβος και στο δεύτερο 2. Η διαφορά εμφανίζεται στο γεγονός οτι στη βιβλιοθήκη ee_process χρησιμοποιούνται μόνο διεργασίες και στη βιβλιοθήκη ee_thrnode, συνδυασμός νημάτων και διεργασιών.





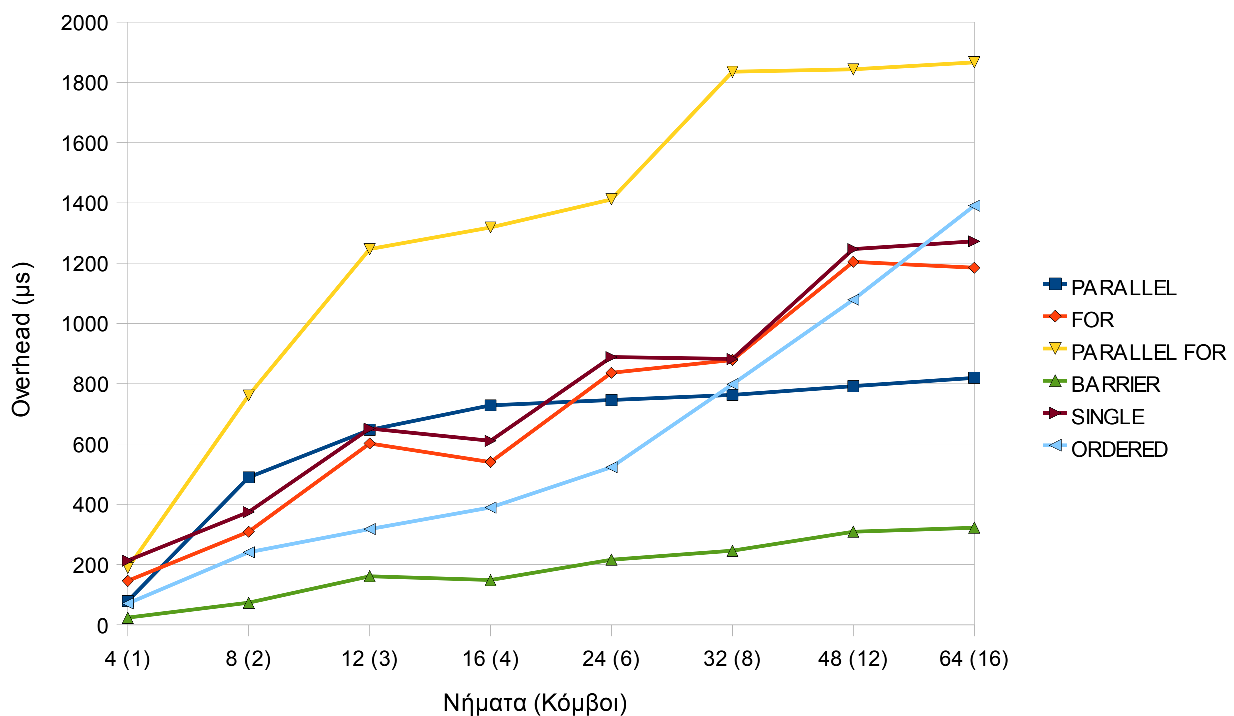
Σχήμα 5.2: Overheads για σύγκριση βιβλιοθηκών ee_process και ee_thrnode.

Είναι εμφανής η διαφορά στην απόδοση των δύο βιβλιοθηκών σε όλες τις οδηγίες OpenMP που παρουσιάζονται και στα δύο στάδια εκτελέσεων. Αυτή η διαφορά οφείλεται στους λόγους που αναφέρθηκαν παραπάνω, Με την ύπαρξη νημάτων στον κόμβο μειώνεται ο αριθμός των επικοινωνιών MPI λόγω της ύπαρξης κοινής μνήμης

στον κόμβο. Η συνύπαρξη πολλών διεργασιών στον ίδιο κόμβο (ee_process), δε συνεπάγεται και τη μείωση των επικοινωνιών καθώς σε κάθε μια από αυτές αντιστοιχεί και ένα νήμα server, το οποίο θα χρειαστεί να έρθει σε επικοινωνία με το νήμα server του αρχικού νήματος. Η απόδοση επίσης οφείλεται και στο μικρότερο κόστος δημιουργίας και διαχείρισης των νημάτων έναντι των διεργασιών.

5.2.3 Πείραμα 3: Κλιμακωτή Εκτέλεση

Στο τρίτο πείραμα, εξετάζεται μια κλιμακωτά αυξανόμενη ως προς τον αριθμό των υπολογιστικών νημάτων εκτέλεση των Microbenchmarks με την υβριδική βιβλιοθήκη. Στόχος του εν λόγω πειράματος ήταν η εξέταση της απόδοσης της βιβλιοθήκης για διαφορετικές διαμορφώσεις στον αριθμό των νημάτων και των κόμβων που συμμετέχουν στους υπολογισμούς.



Σχήμα 5.3: Overheads των οδηγιών OpenMP
για κλιμακωτά αυξανόμενη εκτέλεση.

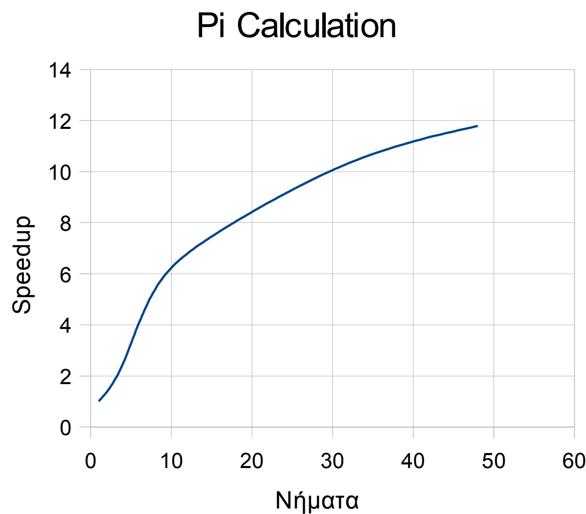
Είναι προφανές οτι η απόδοση για όλες τις οδηγίες OpenMP, πέφτει όσο αυξάνουμε τα υπολογιστικά νήματα και επομένως και των κόμβων που συμμετέχουν. Αξίζει να επισημανθεί οτι κάθε κόμβος του cluster, διαθέτει 4 πυρήνες σε κάθε ένα από τους οποίους έχει ανατεθεί 1 υπολογιστικό νήμα.

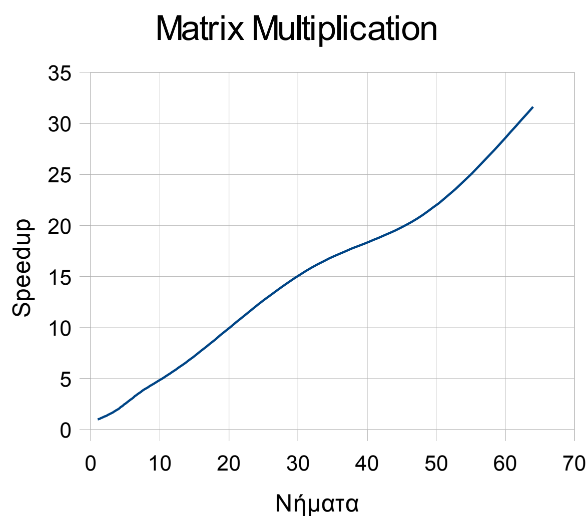
5.3 Παράλληλες Εφαρμογές

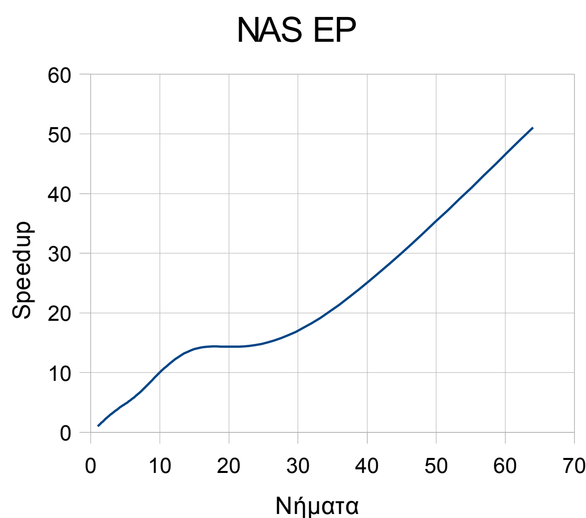
Σε αυτήν την ενότητα παρουσιάζονται τα αποτελέσματα εκτελέσεων του OMPi με την υβριδική βιβλιοθήκη για 3 εφαρμογές που έχουν παραλληλοποιηθεί με OpenMP. Οι εφαρμογές αυτές, είναι ο υπολογισμός του "π", ο πολλαπλασιασμός πινάκων και η εφαρμογή NAS EP. Στόχος των πειραμάτων με αυτές τις εφαρμογές ήταν να εξεταστεί η ορθότητα της βιβλιοθήκης καθώς και η συμπεριφορά της απόδοσής της σε ρεαλιστικά προβλήματα.

Για τον υπολογισμό του "π" ανατίθεται σε κάθε νήμα της ομάδας ένα μέρος των υπολογισμών που χρειάζονται για να προσεγγίσουμε το "π". Στο τέλος του προγράμματος, το αρχικό νήμα (πατέρας της ομάδας), συγκεντρώνει τα επιμέρους αποτελέσματα των νημάτων και τα συνδυάζει για να προκύψει το τελικό αποτέλεσμα. Παρόμοια είναι και η λογική του πολλαπλασιασμού πινάκων. Σε κάθε νήμα ανατίθεται ένα μέρος του πολλαπλασιασμού 2 τετραγωνικών πινάκων. Στο τέλος αυτά τα αποτελέσματα συγκεντρώνονται και συνδυάζονται από το αρχικό νήμα. Η εφαρμογή NAS EP αποτελεί μέρος των εφαρμογών NAS Parallel Benchmarks [19]. Η εφαρμογή δημιουργεί τυχαία ζευγάρια αριθμών με κατανομή Gaussian που αποκλίνουν σύμφωνα με ένα συγκεκριμένο σύστημα με στόχο να καθοριστεί το σημείο αναφοράς μιας κορυφής, μιας δοσμένης πλατφόρμας.

Οι εκτελέσεις των εφαρμογών έγιναν για τις διαμορφώσεις εκτέλεσης: 1) 4 νήματα – 1 κόμβος, 2) 8 νήματα - 2 κόμβοι, 3) 16 νήματα – 4 κόμβοι, 4) 32 νήματα – 8 κόμβοι, 5) 48 νήματα – 12 κόμβοι, 6) 64 νήματα – 16 κόμβοι.







Σχήμα 5.4: Speeups για τις 3 εφαρμογές

Και στις 3 εφαρμογές παρατηρείται μια ομαλή αύξηση στην απόδοση (Speedup), που θα μπορούσε να χαρακτηριστεί γραμμική, όσο αυξάνουμε τον αριθμό των υπολογιστικών νημάτων άρα και των κόμβων που συμμετέχουν στους υπολογισμούς. Εκτός από τους στόχους των πειραμάτων αυτών που προαναφέρθηκαν στην αρχή της ενότητας και πληρούντε από τα αποτελέσματα των πειραμάτων. Το σημαντικό στα συμπεράσματα που μπορούμε να εξάγουμε από τα διαγράμματα είναι πως η απόδοση δεν ξεφεύγει δραματικά για μεγάλο αριθμό υπολογιστικών νημάτων,.

Κεφάλαιο 6

Σύνοψη και Μελλοντική Εργασία

Ο OMPi είναι ένας source-to-source μεταφραστής προγραμμάτων OpenMP/C. Η ιδιαιτερότητα του έγκειται στο γεγονός οτι προσφέρει μια μεγάλη ανεξαρτησία μεταξύ των δύο τμημάτων του συστήματος Runtime. Μια ανεξαρτησία που προσφέρει τη δυνατότητα της εύκολης προσάρτησης μιας βιβλιοθήκης οντοτήτων οποιουδήποτε είδους, είτε αυτές είναι διεργασίες, είτε νήματα επιπέδου πυρήνα, είτε νήματα επιπέδου χρήστη, υλοποιώντας κάποιες απαραίτητες συναρτήσεις διεπαφής. Ο OMPi υποστηρίζει 4 βιβλιοθήκες νημάτων επιπέδου πυρήνα, 2 βιβλιοθήκες επιπέδου χρήστη και 1 που προσφέρει διεργασίες. Η τελευταία, καθώς έχει σχεδιαστεί για την εκτέλεση προγραμμάτων OpenMP πάνω στους κόμβους ενός clusters, με τη σειρά της προσφέρει τη δυνατότητα συνδυασμού της με μια σειρά από διάφορους πυρήνες sDSM, που υλοποιούν μια εικονική μνήμη για τους κόμβους. Η ανεξαρτησία του τμήματος της βιβλιοθήκης με το τμήμα του sDSM, καθιστά εύκολη την ενσωμάτωση και άλλων sDSM. Αυτή τη στιγμή ο OMPi υποστηρίζει 5 πυρήνες sDSM.

Στην παρούσα πτυχιακή παρουσιάστηκε μια επιπρόσθετη βιβλιοθήκη που συνδυάζει διεργασίες και νήματα σε ένα κόμβο του cluster. Ο μέχρι πρότινος σχεδιασμός του OMPi για εκτέλεση προγραμμάτων σε cluster, περιοριζόταν στη δημιουργία διεργασιών στους κόμβους, διεργασίες που στην περίπτωση ύπαρξης πολλών επεξεργαστών και πυρήνων στον κόμβο, δεν εκμεταλλευόντουσαν την πλήρη υπολογιστική δυνατότητα του κόμβου και την κοινή μνήμη σε αυτόν. Αναπτύχθηκε λοιπόν μια βιβλιοθήκη που ξεκινάει μια διεργασία σε κάθε κόμβο του cluster και αυτή με τη σειρά της δημιουργεί έναν αριθμό νημάτων τα οποία αναμένουν για να πάρουν δουλειά. Οι επικοινωνίες μεταξύ των οντοτήτων του OMPi γίνονται μέσω sDSM για τις κοινόχρηστες μεταβλητές του προγράμματος και μέσω MPI για τις μεταβλητές που έχουν να κάνουν με πληροφορίες της εργασίας. Η βελτίωση έγκειται στο ότι:

- Πλέον οι MPI επικοινωνίες περιορίζονται σε μεγάλο βαθμό αριθμητικά καθώς μόνο τα νήματα server κάθε κόμβου συμμετέχουν σε αυτές και όχι τα νήματα server κάθε διεργασίας.
- Τα νήματα του κόμβου επικοινωνούν μέσω κοινόχρηστων μεταβλητών.
- Τα νήματα παρουσιάζουν πλεονεκτήματα έναντι των διεργασιών. Αυτά έχουν να κάνουν με το μικρότερο κόστος δημιουργίας και διαχείρισης αλλά και στο γεγονός οτι είναι πιο ελαφριά από τις διεργασίες.

Η βιβλιοθήκη αυτή, δεν προσφέρει παραλληλισμό σε μεγαλύτερα επίπεδα του πρώτου (nested). Σε μια τέτοια περίπτωση, ο κώδικας εκτελείται από ένα μόνο νήμα, δηλαδή σειριακά. Γενικότερα οι επιδόσεις σε παραλληλισμό για επίπεδα μεγαλύτερα του πρώτου είναι χαμηλές, ειδικότερα στην περίπτωση που τα νήματα ξεπερνούν τον αριθμό των επεξεργαστών. Ο παραλληλισμός nested εμφανίζεται συχνά σε προβλήματα που έχουν παραλληλοποιηθεί και η καλύτερη υποστήριξη του από τον OMPi είναι ένα θέμα για μελλοντική εργασία, τόσο για τις βιβλιοθήκες των νημάτων που εμφανίζονται σε εκτέλεση συστημάτων κοινής μνήμης, όσο για τις βιβλιοθήκες διεργασιών που εμφανίζονται σε εκτέλεση συστημάτων κατανεμημένης μνήμης. Η έρευνα για καλύτερη υποστήριξη παραλληλισμού nested μπορεί να προκύψει από την

ανάπτυξη εφαρμογών-ελέγχου για την καλύτερη κατανόηση και μελέτη αυτού.

Όσον αφορά την εκτέλεση προγραμμάτων OpenMP πάνω σε συστάδες υπολογιστών, με τη δημιουργία κατάλληλων βιβλιοθηκών, όπως η βιβλιοθήκη που παρουσιάζεται εδώ, αλλά και την προσάρτηση στον OMPi μιας σειράς από διαφορετικούς πυρήνες sDSM, έχουν προκύψει θέματα για μελλοντική εργασία πάνω στον OMPi. Ένα από αυτά αποτελεί και η ανάπτυξη ενός sDSM που δεν θα καταστρατηγεί την ανεξαρτησία με τη βιβλιοθήκη οντοτήτων και θα λαμβάνει υπόψιν του κάποιες ιδιαιτερότητες του OMPi με σκοπό τη βελτίωση της απόδοσης των λειτουργιών που υφίστανται πάνω στην εικονική κοινή μνήμη. Ένα sDSM που θα λαμβάνει υπόψιν του την ύπαρξη και τον νημάτων στους κόμβους με σκοπό την βελτίωση της υλοποίησης κάποιων λειτουργιών που έχουν να κάνουν κυρίως με το συγχρονισμό, τη συνέπεια μνήμης καθώς και τις κλειδαριές.

Με την είσοδο της πολυνηματικής λογικής πάνω στους κόμβους, προκύπτουν και ζητήματα καλύτερης δρομολόγησης της δουλειάς σε αυτά από τον αρχικό κόμβο 0. Η λογική της παρούσας βιβλιοθήκης για τη δρομολόγηση της δουλειάς, είναι οτι το αρχικό νήμα μοιράζει τη δουλειά με τη σειρά στα νήματα του κάθε κόμβου. Η παραπάνω λογική παρουσιάζει το πρόβλημα οτι δεν λαμβάνει υπόψιν της, πιθανές καθυστερήσεις που μπορεί να παρουσιάσουν κάποιοι κόμβοι είτε λόγω τεχνικών ζητημάτων, είτε λόγω κίνησης στον κόμβο. Η λύση μιας δυναμικής δρομολόγησης που θα λαμβάνει υπόψιν της την on-the-fly συμπεριφορά των νημάτων και των κόμβων στο σύστημα θα ήταν χρήσιμη για την ανίχνευση των πιο "αργών" κόμβων και την αποφυγή ανάθεσης δουλειάς σε αυτούς.

Βιβλιογραφία

[1] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, Jack Dongarra, MPI: The Complete Reference, 1996 Massachusetts Institute of Technology.

[2] P. Keleher, A. L. Cox, S. Dwarkadas, W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In Proc. of the Winter 94 USENIX Conference, San Francisco, CA, USA, Jan. 1994.

[3] OpenMP Architecture Review Board: OpenMP C++ Application Program Interface, Version 3.0, May 2008.

[4] Vassilios V. Dimakopoulos, Elias Leontiadis, George Tzoumas, A portable C Compiler for OpenMP V.2.0, In Proc. of the 5th European Workshop on OpenMP (EWOMP ‘03), Aachen, Germany, October 2003.

[5] Giorgos Ch. Philos, Vassilios V. Dimakopoulos, Panagiotis E. Hadjidoukas, A runtime system architecture for ubiquitous support of OpenMP, In Proc. of the 7th International Symposium on Parallel and Distributed Computing (ISPDC ‘08), Krakow, Poland, July 2008.

[6] M. Rasit Eskicioglu, T. Anthony Marsland, Weiwu Hu, Weisong Shi, Evaluation of the JIAJIA Software DSM System on High Performance Computer Architectures, Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8, January 1999.

[7] Kenji Kise, Takahiro Katagiri, Hiroki Honda, Toshitsugu Yuba, Mocha Version 0.2: Yet Another Software-DSM System, Technical Report UEC-IS-2005-3, August 2005.

[8] Y. Jegou, Dynamic Memory Management on Mome DSM, Volume 2. Singapore, May 2006.

[9] Yang-Suk Kee, Jin-Soo Kim, Soonhoi Ha, ParADE: An OpenMP Programming Environment for SMP Cluster Systems, In Proc. of the 2003 ACM/IEEE conference on Supercomputing, 2003.

[10] Blaise Barney, Lawrence Livermore, POSIX Threads Programming

[11] D. Novillo, OpenMP and automatic parallelization in GCC, In Proc. of the 2006 GCC Summit, Ottawa, Canada, 2006.

[12] Y. Tanaka, K. Taura, M. Sauto, A. Yonezawa. Performance Evaluation of OpenMP Applications with Nested Parallelism. In Proc, of the 5th Workshop on Languages, Compilers and Run-Time Systems for Scalable Computers (LCR ‘00), Rochester, NY, USA, May 2000.

[13] Y. C. Hu, H. L. Amd, A. L. Cox, W. Zwaenepoel, OpenMP for Networks of SMPs, Journal of Parallel and Distributed Computing, 2000.

[14] H. Lu, Y. C. Hu, W. Zwaenepoel, OpenMP on Networks of Workstations. In Proc. ACM/IEEE Conf. on High Perf. Networking and Computing (SC ‘98), Orlando, FL, 1998

[15] J. P. Hoeflinger, Extending OpenMP to Clusters, 2006, White Paper, Intel Corporation.

[16] P. E. Hadjidoukas, E. D. Polychronopoulos, T. S. Papatheodorou, OpenMP Runtime Support for Clusters of Multiprocessors, In Proc. of the Int'l Workshop on OpenMP Applications and Tools (WOMPAT '03), Toronto, Canada, 2003.

[17] Mitsuhisa Sato, Yoshinori Ojima, Experiment with Cluster-enabled OpenMP: OpenMP for Software DSM System on PC Clusters, In Proc. of the 5th European Workshop on OpenMP (EWOMP '03), Aachen, Germany, September 2003.

[18] J. M. Bull, Measuring Synchronization and Scheduling Overheads in OpenMP, In Proc. EWOMP 1999, Europ. Worksh. OpenMP, Lund, Sweden, Sept. 1999.

[19] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, S. Weeratunga, The NAS Parallel Benchmarks, RNR Technical Report RNR-94-007, March 1994