Τοπολογία και Συγχρονισμός στο OpenMP για συστήματα NUMA πάρα πολλών πυρήνων

Γεώργιος Ζ. Ζάχος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Πολυτεχνική Σχολή

Πανεπιστήμιο Ιωαννίνων

Ιούλιος 2021

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΑΛΓΟΡΙΘΜΩΝ

ΠΕΡΙΛΗΨΗ

Γεώργιος Ζ. Ζάχος, Δίπλωμα, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, Ιούλιος 2021.
Τοπολογία και Συγχρονισμός στο OpenMP για συστήματα NUMA πάρα πολλών πυρήνων.
Επιβλέπων: Βασίλειος Β. Δημακόπουλος, Αναπληρωτής Καθηγητής.

Περίληψη της εργασίας στην ίδια γλώσσα με το κείμενο. Αν το κείμενο είναι στα Ελληνικά τότε και αυτή η σελίδα πρέπει να είναι στα Ελληνικά. Αν το κείμενο είναι στα Αγγλικά τότε και αυτή η σελίδα πρέπει να είναι στα Αγγλικά.

Προτεινόμενο: 1 σελίδα.
Μέγιστο: 2 σελίδες.

ABSTRACT

Georgios Z. Zachos, Diploma, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, July 2021.
Topology and Synchronization in OpenMP for NUMA manycore systems.
Advisor: Vassilios V. Dimakopoulos, Associate Professor.

Εκτεταμένη περίληψη της εργασίας στην αντίθετη γλώσσα από αυτήν του κειμένου.
Αν το κείμενο είναι στα Ελληνικά τότε αυτή η σελίδα πρέπει να είναι στα Αγγλικά.
Αν το κείμενο είναι στα Αγγλικά τότε αυτή η σελίδα πρέπει να είναι στα Ελληνικά.

Προτεινόμενο: 2 σελίδες.
Μέγιστο: 4 σελίδες.

ΚΕΦΑΛΑΙΟ 1

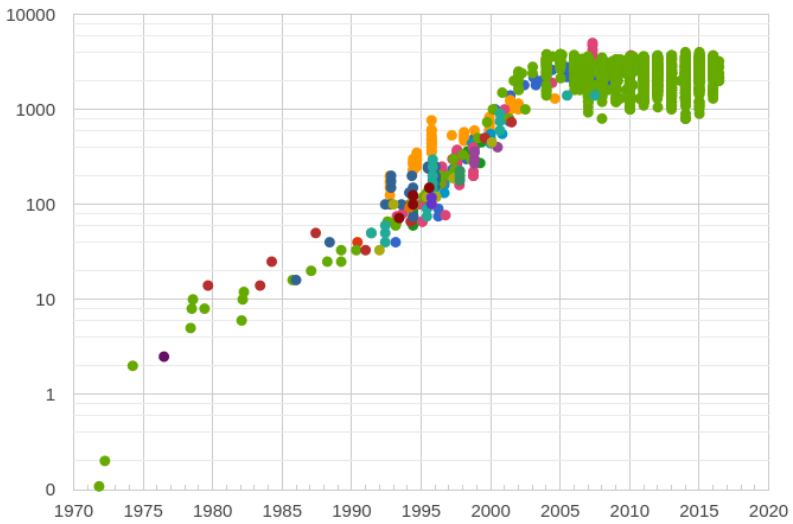
Εισαγωγή

1.1 Η ανάγκη για παράλληλα συστήματα

Η βασική ιδέα της οργάνωσης των ηλεκτρονικών υπολογιστών με τη μορφή που αυτοί είναι γνωστοί έως και σήμερα, βασίζεται στην αρχιτεκτονική von Neumann όπως αυτή περιγράφηκε το 1945 από τον ουγγρικής και αμερικανικής καταγωγής μαθηματικό John von Neumann. Βάσει αυτής της περιγραφής, υπάρχει μία μονάδα επεξεργασίας η οποία επικοινωνεί με το τμήμα μνήμης, εκτελώντας εντολές και τροποποιώντας δεδομένα. Ο επεξεργαστής λαμβάνει μία-μία τις εντολές από τη μνήμη και τις εκτελεί προσπελαύνοντας ή/και τροποποιώντας δεδομένα που βρίσκονται στη μνήμη όταν αυτό καθορίζεται από την εντολή, με αυτό τον κύκλο να επαναλαμβάνεται συνεχώς. Το μοντέλο προγραμματισμού που χρησιμοποιείται στη συγκεκριμένη αρχιτεκτονική είναι γνωστό και ως σειριακό μοντέλο και φαίνεται στο Σχήμα Χ.

Η εποχή της πληροφορίας που ξεκίνησε στα μέσα του $20^{ού}$ αιώνα και οδήγησε σε μία οικονομία βασισμένη στην τεχνολογία της πληροφορίας, καθώς και η ολοένα και μεγαλύτερη χρήση των Η/Υ σε κάθε πτυχή της ζωής του ανθρώπου είχαν ως αποτέλεσμα την έναρξη ενός αγώνα ταχύτητας με σκοπό την κατασκευή όλο και πιο γρήγορων υπολογιστών.

Η κλιμάκωση της συχνότητας (frequency scaling) που αφορά την αύξηση της συχνότητας ενός επεξεργαστή με στόχο την επίτευξη μεγαλύτερης επίδοσης στα συστήματα που τον χρησιμοποιούν, αποτέλεσε τον κύριο λόγο αύξησης των επιδόσεων των εμπορικών Η/Υ από τα μέσα του 1980 έως και περίπου τα τέλη του 2004, όταν αυτή η προσπάθεια προσέκρουσε στο τείχος της ισχύος (power wall) όπως φαί-



(Η απεικόνιση προήλθε από την ιστοσελίδα του CPUDB [1])

Σχήμα 1.1: Η εξέλιξη της συχνότητας λειτουργίας (σε MHz) των μικροεπεξεργαστών.

νεται στο Σχήμα 1.1. Αυτό συνέβει καθώς η αύξηση της συχνότητας οδήγησε στην αύξηση της καταναλισκόμενης ισχύος η οποία με τη σειρά της είχε ως αποτέλεσμα την αύξηση του κόστους λειτουργίας αλλά και την οδήγηση του υλικού στα όριά του με χαρακτηριστικό παράδειγμα την υπερθέρμανσή του.

Για να ξεπεραστεί το τείχος της ισχύος, οι κατασκευαστές των επεξεργαστών άρχισαν να ενσωματώνουν παραπάνω του ενός επεξεργαστικούς πυρήνες στο ίδιο ολοκληρωμένο κύκλωμα επεξεργαστή. Η αρχιτεκτονική αυτή βελτίωση είχε ως αποτέλεσμα τη γέννηση των πρώτων πολυπύρηνων (multicore) επεξεργαστών οι οποίοι έκαναν δυνατή την ταυτόχρονη εκτέλεση εντολών. Τα συστήματα που περιείχαν πολυπύρηνους επεξεργαστές αποτέλεσαν τα πρώτα εμπορικά μικρού μεγέθους παράλληλα συστήματα.

Στη σημερινή εποχή, τα πολυπύρηνα συστήματα είναι ευρέως διαδεδομένα καθώς απαντώνται από κινητά τηλέφωνα και ενσωματωμένους υπολογιστές χαμηλού κόστους και μεγέθους όσο μια πιστωτική κάρτα τραπέζης, μέχρι και υπολογιστικά συστήματα πολύ υψηλών επιδόσεων που διεξάγουν επιστημονικούς υπολογισμούς.

1.2 Κατηγορίες Παράλληλων Συστημάτων

Όταν μιλάμε για παράλληλα συστήματα ουσιαστικά αναφερόμαστε σε συστήματα που διαθέτουν μία συλλογή από επεξεργαστικές μονάδες που επικοινωνούν και συνεργάζονται για τη λύση ενός προβλήματος. Το πρόβλημα χωρίζεται σε επιμέρους εργασίες οι οποίες με τη σειρά τους ανατίθενται στις επεξεργαστικές μονάδες για εκτέλεση. Παρόλο που η ιδέα των πολλαπλών επεξεργαστικών μονάδων φαίνεται απλή, προκύπτουν ζητήματα τα οποία σχετίζονται πλην άλλων με τον τρόπο:

- επικοινωνίας, συγχρονισμού και συντονισμού των επεξεργαστικών μονάδων
- διαμοιρασμού των εργασιών και
- προγραμματισμού ανάλογα με την οργάνωση του εκάστοτε συστήματος.

1.2.1 Ταξινομία του Flynn

Μία αρχική κατηγοριοποίηση των παράλληλων συστημάτων μπορεί να γίνει βάσει της ταξινομίας του Flynn και η οποία περιλαμβάνει τις εξής κατηγορίες:

- SISD (single-instruction single-data) Σχήμα X
- SIMD (single-instruction multiple-data) Σχήμα X και
- MIMD (mutiple-instruction multiple-data) Σχήμα X

Στην κατηγορία SISD ανήκουν οι κλασικοί σειριακοί υπολογιστές οι οποίοι εκτελούν μία εντολή τη φορά (single-instruction) επάνω σε ένα δεδομένο (single-data). Στην κατηγορία SIMD συναντάμε επίσης υπολογιστές οι οποίοι εκτελούν μία εντολή τη φορά, αλλά μπορούν να την εφαρμόσουν ταυτόχρονα σε πολλαπλά δεδομένα (multiple-data) ώστε να εκμεταλλευτούν την παραλληλία επιπέδου δεδομένων (data-level parallelism). Τέλος, η κατηγορία MIMD θεωρείται ως η κατηγορία των καθαρά παράλληλων υπολογιστών οι οποίοι μπορούν και εκτελούν ταυτόχρονα πολλαπλές εντολές, με κάθε μία να ασχολείται με διαφορετικό δεδομένο.

Όπως θα δούμε στη συνέχεια, οι υπολογιστές MIMD, μπορούν να κατηγοριοποιηθούν περαιτέρω βάσει του πώς είναι οργανωμένη η μνήμη τους.

1.2.2 Ταξινόμηση βάσει της οργάνωσης της μνήμης

Η ταξινόμηση των παράλληλων υπολογιστών σχετικά με το πώς είναι οργανωμένη η μνήμη μπορεί να γίνει είτε βάσει της φυσικής οργάνωσης της μνήμης είτε βάσει της εικόνας/άποψης που έχει ο προγραμματιστής για αυτή.

Σχετικά με την πρώτη κατηγοριοποίηση, οι παράλληλοι υπολογιστές διακρίνονται σε υπολογιστές με (φυσικά) κοινόχρηστη μνήμη (γνωστοί και ως πολυεπεξεργαστές) και σε υπολογιστές με (φυσικά) κατανεμημένη μνήμη.

Όσον αφορά την εικόνα που έχει ο προγραμματιστής για την οργάνωση της μνήμης, οι υπολογιστές μπορούν να διαχωριστούν σε υπολογιστές με κοινόχρηστο και σε υπολογιστές με κατανεμημένο χώρο διευθύνσεων. Αξίζει να σημειωθεί ότι η εικόνα από προγραμματιστική άποψη δεν ταυτίζεται απαραίτητα με την φυσική οργάνωση της μνήμης καθώς με χρήση ειδικού λογισμικού ένα σύστημα με φυσικά κατανεμημένη μνήμη μπορεί να παρέχει κοινόχρηστο χώρο διευθύνσεων.

Συστήματα κοινόχρηστης μνήμης

Τα συστήματα κοινόχρηστης μνήμης (SMM - Shared Memory Machines [2]) αποτελούνται από επεξεργαστές, κοινόχρηστη μνήμη (γνωστή και ως καθολική) η οποία είναι καθολικά προσβάσιμη από όλους τους επεξεργαστές και ένα δίκτυο διασύνδεσης για την επικοινωνία των επεξεργαστών με τη μνήμη. Από άποψη φυσικής οργάνωσης, η μνήμη μπορεί να αποτελείται από περισσότερα του ενός τμήματα/μέρη (modules) τα οποία όμως παρέχουν έναν κοινόχρηστο χώρο διευθύνσεων που είναι προσβάσιμος από όλους τους επεξεργαστές. Οι επεξεργαστές επικοινωνούν, συνεργάζονται και ανταλλάσσουν δεδομένα διαβάζοντας ή γράφοντας σε κοινόχρηστες μεταβλητές που βρίσκονται αποθηκευμένες στη μνήμη.

Το δίκτυο διασύνδεσης επεξεργαστών-μνήμης μπορεί να είναι ένας απλός δίαυλος (bus), ένα διακοπτικό δίκτυο (π.χ. crossbar) ή κάποιο δίκτυο πολλαπλών επιπέδων (π.χ. Δίκτυο Δέλτα). Στην περίπτωση που το δίκτυο διασύνδεσης είναι δίαυλος, τότε αναφερόμαστε σε αυτά τα συστήματα ως Συμμετρικοί Πολυεπεξεργαστές (SMPs - Symmetric Multiprocessors). Οι Συμμετρικοί Πολεπεξεργαστές διαθέτουν μία κοινόχρηστη μνήμη η οποία απέχει εξίσου από όλους τους επεξεργαστές και συνεπώς χαρακτηρίζονται ως συστήματα ομοιόμορφης προσπέλασης μνήμης (UMA - Uniform Memory Access). Όπως είναι φυσικό, αν οι επεξεργαστές είναι πολυπύρηνοι και διαθέτουν ιεραρχία από πολύ μικρές αλλά ταυτόχρονα πολύ γρήγορες

μνήμες γνωστές ως κρυφές μνήμες (caches), τότε ο κάθε πολυπύρηνος επεξεργαστής αποτελεί ένα σύστημα SMP καθώς η πρόσβαση στην κρυφή μνήμη είναι πιο γρήγορη από την πρόσβαση στην κύρια μνήμη μέσω του διαύλου.

Επειδή το εύρος ζώνης (bandwidth) του διαύλου είναι σταθερό ανεξάρτητα από το πλήθος των επεξεργαστών που είναι συνδεδεμένοι σε αυτόν, όσο πιο πολλοί επεξεργαστές είναι συνδεδεμένοι, τόσο πιο πολύ υποβαθμίζεται η αποδοτικότητα του δικτύου λόγω των συγγρούσεων πρόσβασης στο κοινό μέσο και συνεπώς τόσο περισσότερο καθυστερεί η εξυπηρέτηση προσπελάσεων στη μνήμη, αυξάνοντας έτσι τον σχετικό (effective) χρόνο προσπέλασης. Για την αποδοτική υποστήριξη μεγαλύτερου αριθμού επεξεργαστών καταφεύγουμε στη χρήση κρυφής μνήμης ή άλλου τύπου δίκτυου διασύνδεσης.

Παράλληλα συστήματα μεγαλύτερου μεγέθους μπορούν να υλοποιηθούν χρησιμοποιώντας Συμμετρικούς Πολυεπεξεργαστές ως κόμβους ενός δικτύου διασύνδεσης (Σχήμα Χ). Σε αυτά τα συστήματα καθίσταται δυνατή η παροχή ενός κοινόχρηστου χώρου διευθύνσεων με χρήση κατάλληλων πρωτοκόλλων συνοχής τα οποία αποκρύπτουν από τον χρήστη του συστήματος την κατανεμημένη οργάνωση της φυσικής μνήμης. Η αρχιτεκτονική που μόλις περιγράφηκε είναι γνωστή και ως κατανεμημένη κοινόχρηστη μνήμη (DSM - Distrubuted Shared Memory), ενώ τα συστήματα αυτά ονομάζονται συστήματα ανομοιόμορφης προσπέλασης μνήμης (NUMA - Non-Uniform Memory Access). Σε περίπτωση ύπαρξης ιεραρχίας κρυφών μνημών στους κόμβους (Σχήμα Χ), χρειάζεται η χρήση ενός πρωτοκόλλου συνοχής κρυφής μνήμης (cache coherence protocol) το οποίο θα εξασφαλίζει ανά πάσα στιγμή ότι οποιαδήποτε προσπέλαση μνήμης θα επιστρέφει την πιο πρόσφατη τιμή. Τέτοια συστήματα είναι γνωστά ως cc-NUMA (Cache coherent NUMA). Τη σημερινή εποχή, οι όροι NUMA και cc-NUMA είναι συνήθως συνώνυμοι.

Συστήματα κατανεμημένης μνήμης

Τα συστήματα κατανεμημένης μνήμης (DMM - Distributed Memory Machines) αποτελούνται από επεξεργαστικά στοιχεία (γνωστά ως κόμβοι) και από ένα δίκτυο διασύνδεσης το οποίο επιτρέπει την επικοινωνία μεταξύ των κόμβων (Σχήμα Χ). Κάθε κόμβος περιέχει επεξεργαστή, τοπική μνήμη και ίσως περιφερειακές συσκευές. Η τοπική μνήμη κάθε κόμβου είναι απευθείας προσβάσιμη μόνο από τον επεξεργαστή του ίδιου κόμβου, ενώ όταν κάποιος επεξεργαστής χρειάζεται να προσπελάσει την τοπική μνήμη που βρίσκεται σε κάποιον άλλο κόμβο, αυτό επιτυγχάνεται με

μεταβίβαση μηνυμάτων μέσω του δικτύου διασύνδεσης.

Συλλογές υπολογιστών που είναι συνδεδεμένοι μέσω δικτύου διασύνδεσης αφιερωμένου αποκλειστικά στη μεταξύ τους επικοινωνία ονομάζονται συστάδες (clusters). Οι συστάδες υπολογιστών χρησιμοποιούνται ευρέως λόγω της διαθεσιμότητας δικτύων διασύνδεσης υψηλών επιδόσεων όπως το Switched Gigabit Ethernet, το Infiniband, το Myriret κ.ά. Πολλαπλές συστάδες υπολογιστών μπορούν να διασυνδεθούν μεταξύ τους και να δημιουργήσουν συστήματα πλέγματος (grids).

Το πλήθος των κόμβων και το πλήθος των επεξεργαστών στα συστήματα κατανεμημένης μνήμης μπορεί να φτάσει τις εκατοντάδες χιλιάδες και κάποια εκατομμύρια αντίστοιχα, σε αντίθεση με τα συστήματα κοινόχρησης μνήμης όπου το πλήθος των επεξεργαστών περιορίζεται σε μερικές δεκάδες. Προφανώς, η ικανότητα κλιμάκωσης των κατανεμημένων συστημάτων εξαρτάται από την σωστή επιλογής της τοπολογίας του δικτύου διασύνδεσης.

1.3 Προγραμματισμός Παράλληλων Συστημάτων

1.3.1 Συστήματα κοινόχρηστης μνήμης

Ο προγραμματισμός των συστημάτων κοινόχρηστης μνήμης συνήθως βασίζεται στη χρήση νημάτων, δηλαδή σε ξεχωριστές ακολουθίες ελέγχου (στοίβα + μετρητής προγράμματος) που μοιράζονται δεδομένα με τα υπόλοιπα νήματα μέσω κοινόχρηστου χώρου διευθύνσεων. Η ύπαρξη κοινόχρηστων δεδομένων εγείρει ζητήματα όπως αυτό της ταυτόχρονης προσπέλασής τους από διαφορετικά νήματα, καθώς κάτι τέτοιο θα μπορούσε να οδηγήσει σε συνθήκες ανταγωνισμού (race conditions). Όταν υπάρχουν συνθήκες ανταγωνισμού, το τελικό αποτέλεσμα των ταυτόχρονων προσπελάσεων μνήμης εξαρτάται από τις σχετικές ταχύτητες εκτέλεσεις των νημάτων. Για την αποφυγή συνθηκών ανταγωνισμού και την εξασφάλιση της ακεραιότητας των δεδομένων χρησιμοποιείται ο αμοιβαίος αποκλεισμός (mutual exclusion), βάσει του οποίου μόνο ένα νήμα κάθε φορά μπορεί να εκτελεί εντολές που τροποποιούν κοινόχρηστες μεταβλητές.

Η πιο διαδεδομένη βιβλιοθήκη νημάτων και μοντέλο εκτέλεσης είναι αυτό των POSIX threads (pthreads) το οποίο παρέχει κλήσεις διαχείρισης (π.χ. δημιουργία, τερματισμό) νημάτων, κλειδαριές (mutexes), μεταβλητές συνθήκης (condition vari-

ables) και κλήσεις συγχρονισμού νημάτων όπως για παράδειγμα κλήσεις φραγής (barriers). Ο προγραμματιστής έχει την πλήρη ευθύνη για τη δημιουργία των νημάτων, την ανάθεση εργασιών σε αυτά, προεραιτικά την αντιστοίχηση των νημάτων σε επεξεργαστές, πιθανώς την συλλογή των επιμέρους αποτελεσμάτων και την σύνθεση του τελικού αποτελέσματος από αυτά, καθώς και τον τερματισμό τους. Επιπλέον, είναι υπεύθυνος για τον συγχρονισμό των νημάτων και την αποφυγή συνθηκών ανταγωνισμού με χρήση κατάλληλων προγραμματιστικών δομών.

Για τη διευκόλυνση του προγραμματισμού, έχουν αναπτυχθεί εργαλεία πιο υψηλού επιπέδου όπως το OpenMP (Open Multi-Processing) [3]. Το OpenMP είναι μία διεπαφή προγραμματισμού εφαρμογών (API - Application Programming Interface) η οποία αποτελείται από οδηγίες (directives) προς τον μεταφραστή, ρουτίνες βιβλιοθήκης και μεταβλητές περιβάλλοντος (environment variables) και συντελεί στην συγγραφή πολυνηματικού κώδικα για συστήματα κοινόχρηστης μνήμης. Πολύ σημαντικό είναι το γεγονός ότι το OpenMP δίνει τη δυνατότητα παραλληλοποίησης του υπάρχοντα σειριακού κώδικα χωρίς την τροποποίησή του, παρά μόνο με την προσθήκη των ειδικών οδηγιών οι οποίες μπορούν να αγνοηθούν σε περίπτωση που δεν υποστηρίζονται από κάποιο μεταφραστή. Επίσης, η διαχείριση των νημάτων γίνεται αυτόματα, ενώ όλα τα υπόλοιπα ζητήματα που σχετίζονται με τον συγχρονισμό, ανάθεση εργασιών κλπ απλοποιούνται σε μεγάλο βαθμό. Η απλότητα του OpenMP αλλά και όλες οι διευκολύνσεις που προσφέρει, το κάνουν να βρίσκεται στην κορυφή των προτιμήσεων για προγραμματισμό συστημάτων κοινόχρηστης μνήμης, αφού μπορεί να χρησιμοποιηθεί ακόμα και από προγραμματιστές χωρίς ιδιαίτερη εμπειρία ή γνώσεις σχετικές με την παράλληλη επεξεργασία. Περισσότερα για το πρότυπο OpenMP θα ειπωθούν στο Κεφάλαιο 2.

1.3.2 Συστήματα κατανεμημένης μνήμης

Ο προγραμματισμός των συστημάτων κατανεμημένης μνήμης συνήθως βασίζεται στη μεταβίβαση μηνυμάτων μεταξύ αυτόνομων διεργασιών (προγράμματα υπό εκτέλεση) οι οποίες βρίσκονται σε διαφορετικούς κόμβους και δεν μοιράζονται κοινόχρηστες μεταβλητές, αλλά πραγματοποιούν μεταξύ τους αποστολή και λήψη μηνυμάτων. Η μη ύπαρξη κοινόχρηστων δεδομένων εξαλείφει την ανάγκη για αμοιβαίο αποκλεισμό αλλά ταυτόχρονα δυσκολεύει τον συγχρονισμό μεταξύ των διεργασιών. Ο προγραμματιστής είναι υπεύθυνος να καθορίσει πότε σταματούν οι υπολογισμοί

και πότε ξεκινούν οι επικοινωνίες, το περιεχόμενο, τον αποστολέα και τους παραλήπτες των μηνυμάτων, καθώς και να αποφασίσει τον τύπο της επικοινωνίας που θα χρησιμοποιήσει (σύγχρονη ή ασύγχρονη).

Στις σύγχρονες επικοινωνίες μία διεργασία που θέλει να στείλει (λάβει) δεδομένα μπλοκάρει στην αντίστοιχη κλήση αποστολής (λήψης) μέχρις ότου η διαδικασία παραλήπτης (αποστολέας) παραλάβει (αποστείλει) τα δεδομένα. Ο τρόπος λειτουργίας των σύγχρονων επικοινωνιών τις καθιστά ιδανικές για την επίτευξη συγχρονισμού μεταξύ των διεργασιών. Αντίθετα, στις ασύγχρονες επικοινωνίες, η διεργασία δεν μπλοκάρει αλλά συνεχίζει κανονικά την εκτέλεση της.

Λόγω της φύσης του μοντέλου μεταβίβασης μηνυμάτων, ο προγραμματιστής θα πρέπει να δώσει προσοχή στο πώς θα ελαχιστοποιήσει την επικοινωνία μεταξύ διαφορετικών κόμβων, καθώς η προσπέλαση της τοπικής μνήμης κάθε κόμβου είναι πολύ πιο γρήγορη απ' ότι η προσπέλαση της μνήμης άλλων κόμβων. Γι' αυτό το λόγο πρέπει να σχεδιαστεί με σύνεση ο διαμοιρασμός των δεδομένων ανάμεσα στους κόμβους, καθώς και η ανάθεση φόρτου σε κάθε διεργασία ώστε εκτός από την αποφυγή καθυστερήσεων σε απομακρυσμένες προσπελάσεις, να αποφευχθεί και συμφόρηση του δίκτυου.

Δημοφιλές πρότυπο μεταβίβασης μηνυμάτων αποτελεί το MPI (Message Passing Interface) με τις δύο πιο γνωστές υλοποιήσεις του να είναι το Open MPI και το MPICH. Το MPI υποστηρίζει τη μεταβίβαση μηνυμάτων στις γλώσσες C, C++ και Fortran με στόχο την ανάπτυξη μεταφέρσιμων παράλληλων εφαρμογών μεγάλης κλίμακας. Μεγάλο προτέρημα αποτελεί η απόκρυψη των πληροφοριών χαμηλού επιπέδου όπως ο τύπος του υποκείμενου δικτύου, το λειτουργικό σύστημα του κάθε κόμβου, αλλά και η ύπαρξη βοηθητικών προγραμματιστικών δομών, όπως οι συλλογικές και μη επικοινωνίες που μπορούν να πραγματοποιηθούν χωρίς τη γνώση δικτυακού προγραμματισμού (sockets).

1.4 Αντικείμενο της Διπλωματικής Εργασίας

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι η επίτευξη καλύτερων επιδόσεων στον ερευνητικό μεταφραστή OMPi, μέσω της εκμετάλλευσης πληροφοριών που σχετίζονται με την τοπολογία του συστήματος. Ο OMPi είναι ένας μεταφραστής πηγαίου σε πηγαίο κώδικα (source-to-source) που υποστηρίζει τη διεπαφή

προγραμματισμού εφαρμογών OpenMP και αναπτύσσεται από την Ομάδα Παράλληλης Επεξεργασίας του Πανεπιστημίου Ιωαννίνων.

Στην έκδοση 4.0 του OpenMP προστέθηκε η δυνατότητα ανάθεσης (binding) των νημάτων σε σύνολα από συγκεκριμένους επεξεργαστές (τα λεγόμενα OpenMP places) ώστε να εκτελεστούν σε αυτά. Η ανάθεση των νημάτων σε OpenMP places γίνεται βάσει των διάφορων διαθέσιμων πολιτικών (γνωστές ως processor binding policies). Ο χρήστης έχει τη δυνατότητα προσδιορισμού της λίστας των διαθέσιμων OpenMP places αλλά και της πολιτικής ανάθεσης των νημάτων σε αυτά. Στο OpenMP 5.1 (Νοέμβριος 2020) επεκτάθηκαν περαιτέρω οι τρόποι προσδιορισμού των OpenMP places οι οποίοι περιγράφονται με λεπτομέρεια στην Ενότητα 3.4. Στα πλαίσια της διπλωματικής εργασίας υλοποιήθηκε πλήρης υποστήριξη των προαναφερθέντων λειτουργιών και οι οποίες συμμορφώνονται με τις προδιαγραφές του OpenMP 5.1.

Επιπρόσθετα, ανεξάρτητα από το πρότυπο OpenMP, έγινε εκμετάλλευση της τοπολογίας για τη βελτίωση των κλήσεων φραγής που υλοποιούνται στο μεταφραστή OMPi. Συγκεκριμένα, σε περίπτωση που αναγνωρίζεται το υποκείμενο σύστημα ως σύστημα ανομοιόμορφης προσπέλασης μνήμης (NUMA), τότε εφαρμόζεται ένας νέος αλγόριθμος για τις κλήσεις φραγής που λαμβάνει υπόψην του την τοπολογία για την επίτευξη ταχύτερου συγχρονισμού των νημάτων.

1.5 Διάρθρωση της Διπλωματικής Εργασίας

Η διπλωματική εργασία είναι οργανωμένη με τον ακόλουθο τρόπο:

- Κεφάλαιο 2: Παρουσίαση της διεπαφής προγραμματισμού εφαρμογών OpenMP και περιγραφή της σύνταξης των πιο διαδεδομένων οδηγιών. Αναφορά στους διάφορους μεταφραστές που υποστηρίζουν το OpenMP και περιγραφή του μεταφραστή OMPi.

- Κεφάλαιο 3: Εισαγωγή στην τοπολογία, περιγραφή των συστημάτων NUMA, εργαλεία για την αναγνώριση και εκμετάλλευση της τοπολογίας και η χρήση της τελευταίας στα πλαίσια του OpenMP. Παρουσίαση των λειτουργιών του OpenMP που σχετίζονται με την τοπολογία και υλοποιήθηκαν στον μεταφραστή OMPi.

- Κεφάλαιο 4: Περιγραφή του τρόπου επίτευξης συγχρονισμού με χρήση κλήσεων φραγής (barriers) και παρουσίαση των ιδιαίτερων χαρακτηριστικών τους στα πλαίσια του OpenMP. Επεξήγηση της υλοποίησης των κλήσεων φραγής στον μεταφραστή OMPi και των βελτιώσεων που έγιναν για την επίτευξη καλύτερων επιδόσεων σε πολυπύρηνα συστήματα NUMA.

- Κεφάλαιο 5: Παρουσίαση και ανάλυση των αποτελεσμάτων από την εκτέλεση διάφορων πειραμάτων για τη μέτρηση της απόδοσης του μεταφραστή OMPi μετά την υλοποίηση των λειτουργιών που σχετίζονται με την τοπολογία και των βελτιώσεων στις κλήσεις φραγής.

- Κεφάλαιο 6: Σύνοψη της διπλωματικής εργασίας και αναφορά σε μελλοντικές εργασίες.

ΚΕΦΑΛΑΙΟ 2

Η διεπαφή προγραμματισμού OpenMP

2.1 Εισαγωγή στο OpenMP

Όπως αναφέρθηκε ήδη στην Υποενότητα 1.3.1, η διεπαφή προγραμματισμού εφαρμογών OpenMP (Open Multi-Processing) αναπτύχθηκε για τη διευκόλυνση της ανάπτυξης πολυνηματικών εφαρμογών για συστήματα κοινόχρηστης μνήμης. Οι γλώσσες οι οποίες υποστηρίζονται είναι οι C, C++ και Fortran. Το OpenMP αποτελείται από:

- Οδηγίες (directives): Συνιστούν οδηγίες προς τον μεταφραστή για το πώς και τι να εκτελέσει πολυνηματικά. Στις γλώσσες C/C++ χρησιμοποιείται ο μηχανισμός που είναι γνωστός ως pragmas και απευθύνεται στον προεπεξεργαστή. Αυτές οι οδηγίες προστίθενται στο υπάρχον σειριακό πρόγραμμα και μπορούν να αγνοηθούν από έναν μεταφραστή που δεν τις υποστηρίζει. Αυτό είναι μεγάλο προσόν καθώς το ίδιο πρόγραμμα μπορεί να εκτελεστεί σειριακά ή παράλληλα.

- Ρουτίνες βιβλιοθήκης: σύνολο συναρτήσεων οι οποίες βοηθούν στη διαχείριση των χαρακτηριστικών των νημάτων και του περιβάλλοντος εκτέλεσης. Για παράδειγμα, η συνάρτηση `omp_set_num_threads(int)` καθορίζει το πλήθος των νημάτων που θα συμμετάσχουν σε επερχόμενη πολυνηματική εκτέλεση (παράλληλη περιοχή).

- Μεταβλητές περιβάλλοντος: χρησιμοποιούνται για τον καθορισμό διάφορων χαρακτηριστικών των νημάτων και του περιβάλλοντος εκτέλεσης. Οι τιμές των μεταβλητών περιβάλλοντος οριστικοποιούνται στην αρχή της εκτέλεσης

και χρησιμοποιούνται ως προκαθορισμένες τιμές. Κάποιες από αυτές τις προκαθορισμένες αυτές τιμές μπορούν να τροποποιηθούν σε χρόνο εκτέλεσης με χρήση των διαθέσιμων ρουτινών βιβλιοθήκης.

Από τη στιγμή που η παραλληλοποίηση ενός σειριακού προγράμματος μπορεί να γίνει με την απλή προσθήκη οδηγιών στο υπάρχοντα κώδικα, η διαδικασία της παραλληλοποίησης απλοποιείται σε μεγάλο βαθμό και μπορεί να γίνει σταδιακά (π.χ. παραλληλοποίηση ενός βρόγχου for τη φορά) και χωρίς τη χρήση διαφορετικής λογικής, όπως για παράδειγμα θα γινόταν με χρήση των POSIX Threads.

Ένα από τα σχετικά καινούρια χαρακτηριστικά του OpenMP είναι η δυνατότητα αποστολής κώδικα για εκτέλεση σε συσκευές όπως κάρτες γραφικών γενικού σκοπού (GPGPUs), συνεπεξεργαστές (coprocessors) ή διάφορους άλλους επιταχυντές. Το πλεονέκτημα είναι ότι ο προγραμματιστής δεν χρειάζεται να μάθει να προγραμματίζει σε γλώσσες προγραμματισμού χαμηλού επιπέδου όπως OpenCL, CUDA κλπ για να αξιοποιήσει την επεξεργαστική ισχύ των διαθέσιμων συσκευών. Αυτό το χαρακτηριστικό είναι ιδιαίτερα βοηθητικό σε συστήματα υπολογισμών υψηλών επιδόσεων (High Performance Computing - HPC) όπου υπάρχει η τάση να εξοπλίζεται ένα σύνολο των υπολογιστικών κόμβων με ισχυρούς επιταχυντές για την επίτευξη μεγαλύτερων επιδόσεων. Για παράδειγμα, ο υπερυπολογιστής Aris του Εθνικού Δικτύου Υποδομών και Έρευνας (ΕΔΥΤΕ - GRNET) διαθέτει πλην άλλων:

- 18 κόμβους με 2 επεξεργαστές Intel Xeon E5-2660v3 και 2 συνεπεξεργαστές Intel Xeon Phi 7120P.
- 44 κόμβους με 2 επεξεργαστές Intel Xeon E5-2660v3 και 2 κάρτες γραφικών NVIDIA K40.
- 1 κόμβο με 2 επεξεργαστές Intel E5-2698v4 και 8 κάρτες γραφικών NVIDIA V100 για εκτέλεση προγραμμάτων μηχανικής μάθησης.

Λαμβάνοντας υπόψην ότι η διαχείριση των νημάτων μετατίθεται από τον προγραμματιστή στο μεταφραστή, ότι απλοποιούνται ουσιώδη ζητήματα ενός παράλληλου προγράμματος όπως η επίτευξη συγχρονισμού και αμοιβαίου αποκλεισμού, καθώς επίσης ότι μπορούν να αξιοποιηθούν επιταχυντές χωρίς γνώση του πως προγραμματίζονται, γίνεται εύκολα αντιληπτό ότι το OpenMP είναι ένα προσιτό εργαλείο ακόμα και για άτομα χωρίς μεγάλη εμπειρία στον παράλληλο προγραμματισμό. Αυτό το χαρακτηριστικό του OpenMP το κάνει ιδιαίτερα διαδεδομένο σε

χρήστες που το υπόβαθρό τους διαφέρει από αυτό της επιστήμης της πληροφορικής, όπως για παράδειγμα φυσικοί, χημικοί, αστρονόμοι κόκ. Ταυτόχρονα όμως, η απόκρυψη των λεπτομερειών χαμηλού επιπέδου είναι πιθανό να καταστήσει σε ορισμένες περιπτώσεις μη εφικτή την εξασφάλιση της μέγιστης αποδοτικότητας του παραλληλισμού.

2.2 Το προγραμματιστικό μοντέλο του OpenMP

Το OpenMP βασίζεται στη χρήση πολλαπλών νημάτων όπως άλλωστε συνηθίζεται στον προγραμματισμό συτημάτων κοινόχρηστης μνήμης καθώς και στο προγραμματιστικό μοντέλο fork-join που συναντάται στις διεργασίες.

Στο μοντέλο fork-join (Σχήμα X), η εκτέλεση ξεκινάει σειριακά από ένα νήμα (γνωστό ως αρχηγός - master) και σε προκαθορισμένα σημεία όπου απαιτείται παράλληλη εκτέλεση, δημιουργούνται επιπλέον νήματα τα οποία μαζί με το νήμα-αρχηγό συμμετέχουν στον παράλληλο υπολογισμό. Τα σημεία στα οποία πραγματοποιείται παράλληλη εκτέλεση είναι γνωστά ως παράλληλες περιοχές (parallel sections/regions). Μόλις ο παράλληλος υπολογισμός τελειώσει τα νήματα που δημιουργήθηκαν τερματίζουν και η εκτέλεση συνεχίζεται σειριακά από το νήμα-αρχηγό.

Ενδιαφέρον είναι το γεγονός ότι υποστηρίζονται αυθαίρετα πολλές εμφωλευμένες παράλληλες περιοχές, δηλαδή ένα οποιοδήποτε νήμα το οποίο συμμετέχει στην εκτέλεση μιας παράλληλης περιοχής μπορεί να αποτελέσει με τη σειρά του νήμα-αρχηγός και να δημιουργήσει μία νέα παράλληλη περιοχή. Οι εμφωλευμένες παράλληλες περιοχές μπορούν να χρησιμοποιηθούν για την ανάθεση εργασιών με το επιθυμητό μέγεθος κόκκου παραλληλίας (granularity) σε κάθε νήμα.

Είναι χρήσιμο να αναφερθεί πως όταν ξεκινάει να εκτελείται μία διεργασία, χρησιμοποιείται ένα νήμα για την εκτέλεση των εντολών σειριακά και το οποίο νήμα στα πλαίσια του OpenMP ονομάζεται αρχικό νήμα (initial thread).

2.3 Εισαγωγή στη διεπαφή προγραμματισμού OpenMP

Το πλήθος των σελίδων των προδιαγραφών του OpenMP τείνει να αυξάνεται εκθετικά στις τελευταίες εκδόσεις με αποτέλεσμα να είναι αδύνατο να περιγραφούν όλες

οι δυνατότητές του στα πλαίσια μίας διπλωματικής εργασίας. Για το λόγο αυτό, θα γίνει περιγραφή ενός υποσυνόλου των διαθέσιμων λειτουργιών, πολλές από τις οποίες αποτελούν τις πιο συνηθισμένες και καλύπτουν τις ανάγκες της πλειοψηφίας των διαθέσιμων εφαρμογών OpenMP. Αυτές οι πιο διαδεδομένες λειτουργίες είναι εικοσιμία (21) στο πλήθος και αποτελούν το λεγόμενο OpenMP Common Core @Ref.

2.3.1 Σύνταξη οδηγιών (directives)

Η γενική μορφή μίας οδηγίας στο OpenMP είναι της μορφής:

```
#pragma omp directive-name [[,] clause [[,] clause] ... ] <new-line>
```

Κάθε οδηγία ξεκινάει υποχρεωτικά με το `#pragma omp` και ακολουθεί το όνομά της που καθορίζει ποιά λειτουργία θα εκτελεστεί στην περιοχή του κώδικα που ακολουθεί. Υπάρχουν διάφορες διαθέσιμες οδηγίες οι οποίες μπορούν να χρησιμοποιηθούν, πλην άλλων, για τη δημιουργία παράλληλης ομάδας, το συγχρονισμό των νημάτων και τον ορισμό της πολιτικής με την οποία τα νήματα θα ανατεθούν στους διαθέσιμους επεξεργαστές.

Στη συνέχεια τοποθετούνται προαιρετικά φράσεις (clauses) οι οποίες παραμετροποιούν τις συνθήκες υπό τις οποίες θα εκτελεστεί η λειτουργία που ορίζει η οδηγία. Για παράδειγμα, η φράση `num_threads` μπορεί να χρησιμοποιηθεί σε συνδιασμό με την οδηγία δημιουργίας παράλληλης ομάδας για να καθορίσει το επιθυμητό πλήθος των νημάτων που θα συμμετάσχουν στην παράλληλη εκτέλεση. Η σειρά με την οποία αναγράφονται οι φράσεις δεν έχει σημασία.

Το τέλος της οδηγίας σηματοδοτείται από αλλαγή γραμμής (newline).

2.3.2 Η οδηγία parallel

Η οδηγία `parallel` συντάσσεται ως εξής:

```
#pragma omp parallel [clause [[,] clause] ... ] <new-line>
structured-block
```

Όταν ένα οποιοδήποτε νήμα συναντήσει μία οδηγία `parallel`, δημιουργείται μία ομάδα νημάτων η οποία εκτελεί την παράλληλη περιοχή. Μια παράλληλη περιοχή υποδηλώνει ένα τμήμα κώδικα το οποίο προορίζεται για πολυνηματική εκτέλεση.

Στη συγκεκριμένη περίπτωση, ο κώδικας που περιέχεται στο δομημένο τμήμα κώδικα (structured block) που ακολουθεί την οδηγία `parallel` είναι αυτός που εν τέλει θα εκτελεστεί παράλληλα. Ώς δομημένο τμήμα κώδικα ορίζεται μία εντολή ή μία ακολουθία εντολών που περικλείονται από άγκιστρα.

Το πλήθος των νημάτων (N) που συμμετέχουν σε μια παράλληλη ομάδα είναι σταθερό καθόλη τη διάρκειά της. Το νήμα το οποίο συνάντησε την οδηγία `parallel` λαμβάνει το ρόλο του αρχηγού (master[1]) της ομάδας, ενώ συμμετέχει και αυτό στον παράλληλο υπολογισμό μαζί με τα υπόλοιπα $N-1$ νήματα που δημιουργήθηκαν.

Μέσα σε μία παράλληλη περιοχή μπορούν να χρησιμοποιηθούν τα αναγνωριστικά των νημάτων για την αναγνώριση του κάθε νήματος. Τα αναγνωριστικά είναι ακέραιοι αριθμοί και συγκεκριμένα για μία ομάδα N νημάτων, η τιμή τους κυμαίνεται από μηδέν (για τον αρχηγό της ομάδας) έως και ένα λιγότερο από το μέγεθος της ομάδας, δηλαδή $N-1$.

Καθώς ο χρόνος που χρειάζεται ένα νήμα για να εκτελέσει την παράλληλη περιοχή εξαρτάται από πολλούς παράγοντες, όπως για παράδειγμα το πόσο δίκαιη είναι η κατανομή του φόρτου μεταξύ των νημάτων της ομάδας, στο τέλος της παράλληλης περιοχής υπονοείται μία κλήση φραγής (barrier). Η κλήση αυτή εξασφαλίζει ότι τα νήματα θα περιμένουν στην κλήση φραγής μέχρις ότου όλα τα νήματα να φτάσουν σε αυτό το σημείο πριν τους επιτραπεί να τερματίσουν και το νήμα-αρχηγός συνεχίσει την εκτέλεση του υπόλοιπου προγράμματος που ακολουθεί της παράλληλης περιοχής.

Την οδηγία `parallel` μπορεί προεραιτικά να ακολουθούν φράσεις διαμοιρασμού δεδομένων που θα δούμε στην Υποενότητα 2.3.3 καθώς και οι φράσεις `num_threads`, `redution` και `proc_bind` που περιγράφονται αμέσως μετά.

Η φράση num_threads

Η φράση `num_threads` δέχεται ως παράμετρο έναν ακέραιο αριθμό και καθορίζει το επιθυμητό πλήθος των νημάτων που θα εκτελέσουν την παράλληλη περιοχή.

[1]Στο πρότυπο του OpenMP 5.1 (Νοέμβριος 2020) καθορίζεται αλλαγή της ονομασίας master σε primary. Στο παρόν κείμενο θα ακολουθηθεί η ορολογία master (αρχηγός) για λόγους συμβατότητας με τους υπάρχοντες πόρους της Ομάδας Παράλληλης Επεξεργασίας του Πανεπιστημίου Ιωαννίνων.

Η φράση reduction

Η φράση `reduction` χρησιμοποιείται για τη διεκπεραίωση μιας αριθμητικής πράξης από τα νήματα της ομάδας πάνω σε μια κοινόχρηστη μεταβλητή, εξασφαλίζοντας την αποφυγή συνθηκών ανταγωνισμού. Η απλουστευμένη σύνταξη της φράσης είναι η εξής:

reduction(*reduction-identifier : list***)**

Ο `reduction-identifier` είναι η πράξη η οποία θέλουμε να εφαρμοστεί πάνω στις μία ή περισσότερες μεταβλητές που αναγράφονται στη λίστα `list` και είναι διαχωρισμένες μεταξύ τους με κόμμα. Οι διαθέσιμες πράξεις είναι οι `+`, `-`, `*`, `&`, `|`, `^`, `&&` και `||`.

Η φράση proc_bind

Η φράση `proc_bind` χρησιμοποιείται για τον ορισμό της πολιτικής με την οποία τα νήματα θα ανατεθούν στους διαθέσιμους επεξεργαστές. Οι διαθέσιμες επιλογές είναι οι `master/primary`, `close` και `spread`. Περισσότερες λεπτομέρειες θα δούμε στην Υποενότητα 3.4 όπου θα ασχοληθούμε λεπτομερώς με την τοπολογία του υποκείμενου συστήματος και τον τρόπο ανάθεσης των νημάτων OpenMP στους επεξεργαστές.

2.3.3 Φράσεις διαμοιρασμού δεδομένων

Οι φράσεις διαμοιρασμού δεδομένων χρησιμοποιούνται για να δηλώσουν τον τρόπο διαμοιρασμού μίας ή περισσότερων μεταβλητών μεταξύ των νημάτων ως εξής:

- `shared`: Οι μεταβλητές είναι κοινόχρηστες.
- `private`: Οι μεταβλητές είναι ιδιωτικές καθώς δημιουργείται από ένα αντίγραφο για κάθε νήμα.
- `firstprivate`: Οι μεταβλητές είναι ιδιωτικές και καθεμιά αρχικοποιείται στην τιμή της αντίστοιχης αρχικής μεταβλητής.
- `lastprivate`: Οι μεταβλητές είναι ιδιωτικές και μετά το πέρας της εκτέλεσης η τιμή της καθεμιάς τους θα χρησιμοποιηθεί για την ενημέρωση της τιμής της αντίστοιχης αρχικής μεταβλητής.

- `default`: Καθορίζει την προκαθορισμένη πολιτική διαμοιρασμού με διαθέσιμες επιλογές να είναι οι `shared`, `firstprivate`, `private`, `none`.

2.3.4 Οδηγίες Διαμοιρασμού Εργασίας

Το OpenMP παρέχει τη δυνατότητα κατανομής του φόρτου εργασίας ανάμεσα στα νήματα της ομάδας μέσω των οδηγιών που καθορίζουν περιοχές διαμοιρασμού εργασίας (worksharing regions).

Η βασική διαφορά μίας περιοχής διαμοιρασμού εργασίας με μία παράλληλη περιοχή είναι ότι στην πρώτη σε αντίθεση με τη δεύτερη, δεν δημιουργούνται νέα νήματα αλλά χρησιμοποιούνται τα υπάρχοντα. Βάσει αυτής της παρατήρησης συμπεραίνουμε ότι οι περιοχές διαμοιρασμού έχουν νόημα όταν εντοπίζονται εντός παράλληλων περιοχών. Είναι πιθανό μία παράλληλη ομάδα που αποτελείται από ένα μόνο νήμα να συναντήσει μία οδηγία περιοχής διαμοιρασμού εργασίας. Όπως είναι προφανές, σε αυτή την περίπτωση, η εκτέλεση είναι σειριακή και όχι παράλληλη.

Στο τέλος των περιοχών διαμοιρασμού εργασίας, όπως και στο τέλος των παράλληλων περιοχών, υπονοείται μία κλήση φραγής για την επίτευξη συγχρονισμού μεταξύ των νημάτων, με τη διαφορά ότι στις πρώτες η κλήση φραγής μπορεί να παραληφθεί με τη χρήση της φράσης `nowait`.

Οδηγία sections

Χρησιμοποιείται για την κατανομή μη επαναληπτικών (non-iterative) εργασιών και συντάσσεται ως εξής:

```
#pragma omp sections [clause [[,] clause] ... ] <new-line>
{
[ #pragma omp section <new-line>
<structured-block> ]
[ #pragma omp section <new-line>
<structured-block> ]
...
}
```

Κάθε δομημένο τμήμα κώδικα που ακολουθεί μία οδηγία `#pragma omp section` που περιέχεται μέσα στην οδηγία `sections` θα ανατεθεί σε ένα νήμα και θα εκτε-

λεστεί ακριβώς μία φορά. Συνήθεις φράσεις αποτελούν οι `private`, `firstprivate`, `lastprivate` και `reduction`.

Οδηγία single

Η οδηγία αυτή καθορίζει ότι το δομημένο τμήμα κώδικα που την ακολουθεί θα εκτελεστεί μόνο από ένα νήμα (όχι απαραίτητα το νήμα-αρχηγό) και η σύνταξή της είναι η εξής:

```
#pragma omp single [clause [[,] clause] ... ] <new-line>
<structured-block>
```

Συνήθεις φράσεις που ακολουθούν είναι οι `private` και `firstprivate`.

2.3.5 Οδηγίες Διαμοιρασμού Εργασίας Βρόγχου

Οι οδηγίες διαμοιαρμού εργασίας βρόγχου είναι αντίστοιχες με τις οδηγίες διαμοιρασμού εργασίας που είδαμε στην Υποενότητα 2.3.4, με τη διαφορά ότι αφορούν την κατανομή των επαναλήψεων ενός βρόγχου στα νήματα.

Οδηγία for

Χρησιμοποιείται για την κατανομή των επαναλήψεων ενός βρόγχου[2] `for` και συντάσσεται ως εξής:

```
#pragma omp for [clause [[,] clause] ... ] <new-line>
<loop-nest>
```

Την οδηγία αυτή ακολουθεί υποχρεωτικά βρόγχος `for` ενώ συνήθεις φράσεις αποτελούν οι `private`, `firstprivate`, `lastprivate`, `reduction` και `schedule`.

Η φράση `schedule` χρησιμοποιείται για τον καθορισμό της πολιτικής με την οποία θα διαμοιραστούν οι επαναλήψεις ενός βρόγχου `for` στα νήματα και η απλουστευμένη σύνταξη της είναι η ακόλουθη:

```
schedule(kind[, chunk_size])
```

[2]Η σύνταξη του βρόγχου δεν μπορεί να είναι τόσο αυθαίρετη όσο επιτρέπει το συντακτικό των γλωσσών C/C++, αλλά στα πλαίσια αυτής της εργασίας δεν θα μας απασχολήσει αυτό το ζήτημα.

Η τιμή `kind` καθορίζει τον τρόπο διαμοιρασμού των επαναλήψεων ενώ η τιμή `chunk_size` καθορίζει το μέγεθος του κόκκου παραλληλίας (granularity) που αναλαμβάνει το κάθε νήμα. Η χρονοδρομολόγηση των επαναλήψεων γίνεται όπως περιγράφεται ακολούθως:

- `static`: Συνεχόμενες επαναλήψεις διασπώνται σε τμήματα μεγέθους όσο η τιμή `chunk_size` και ανατίθενται κυκλικά (round-robin) στα νήματα βάσει του αναγνωριστικού τους. Σε περίπτωση που δεν έχει καθοριστεί συγκεκριμένη τιμή `chunk_size`, το σύνολο των επαναλήψεων χωρίζεται σε τόσα ισομεγέθη τμήματα όσα και το πλήθος των νημάτων.

- `dynamic`: Συνεχόμενες επαναλήψεις διασπώνται σε τμήματα μεγέθους όσο η τιμή `chunk_size` και ανατίθενται στα νήματα όταν αυτά ζητήσουν το επόμενο τμήμα προς εκτέλεση. Η δυναμική ανάθεση συνεχίζεται μέχρι να ανατεθούν όλα τα τμήματα. Σε περίπτωση που δεν έχει καθοριστεί συγκεκριμένη τιμή `chunk_size`, τότε τα τμήματα έχουν μέγεθος ίσο με 1.

- `guided`: Η πολιτική αυτή μοιάζει στην πολιτική `dynamic` με τη διαφορά ότι το μέγεθος των τμημάτων δεν είναι σταθερό. Συγκεκριμένα, για `chunk_size` ίσο με 1 (k), το μέγεθος του πρώτου τμήματος ισούται με το πλήθος όλων των επαναλήψεων διαιρεμένο με το πλήθος των νημάτων, με το μέγεθος των επόμενων τμημάτων να μειώνεται εκθετικά μέχρι να ισούται με 1 (k).

- `auto`: Η επιλογή πολιτικής και μεγέθους κόκκου παραλληλίας μεταφέρεται από τον προγραμματιστή στο μεταφραστή ή στο σύστημα χρόνου εκτέλεσης (runtime).

- `runtime`: Η πολιτική και το μέγεθος κόκκου παραλληλίας καθορίζονται σε χρόνο εκτέλεσης βάσει της τιμής της μεταβλητής περιβάλλοντος `OMP_SCHEDULE`.

Στην περίπτωση των πολιτικών `auto` και `runtime` απαγορεύεται ο προσδιορισμός τιμής `chunk_size`.

2.3.6 Η οδηγία task

Η οδηγία αυτή ορίζει μία εργασία προς εκτέλεση και συντάσσεται ως εξής:

```
#pragma omp task [clause [[,] clause] ... ] <new-line>
<structured-block>
```

Όταν ένα νήμα συναντήσει την οδηγία `task`, δημιουργεί μία νέα εργασία που αποτελείται από τον κώδικα του δομημένου τμήματος κώδικα και το περιβάλλον δεδομένων (data environment) που χρειάζεται η εργασία για να διεκπεραιωθεί. Η εκτέλεση της εργασίας μπορεί να πραγματοποιηθεί από οποιοδήποτε νήμα και δεν είναι γνωστό πότε θα ξεκινήσει. Να σημειωθεί ότι υποστηρίζονται εμφωλευμένες οδηγίες `task`.

Συνήθεις φράσεις αποτελούν οι `default`, `private`, `firstprivate` και `shared`.

2.3.7 Οδηγίες συγχρονισμού

Στα προγράμματα OpenMP ως προγράμματα για συστήματα κοινόχρηστης μνήμης, σημαντικός είναι ο ρόλος του συγχρονισμού για την εξασφάλιση της συνέπειας των δεδομένων και της ορθότητας του προγράμματος. Για το λόγο αυτό, το OpenMP παρέχει έτοιμες λειτουργίες συγχρονισμού ως οδηγίες.

Οδηγία atomic

Απλουστευμένη σύνταξη:

```
#pragma omp atomic <new-line>
<statement>
```

Εξασφαλίζει ότι μια θέση μνήμης προσβαίνεται ατομικά εξαλείφοντας την πιθανότητα πολλαπλών ταυτόχρονων προσβάσεων από διαφορετικά νήματα.

Οδηγία barrier

Σύνταξη:

```
#pragma omp barrier <new-line>
```

Η γνωστή κλήση φραγής η οποία εξασφαλίζει ότι όλα τα νήματα της ομάδας θα περιμένουν σε αυτό το σημείο πριν μπορέσουν να συνεχίσουν την εκτέλεση με τις εντολές που ακολουθούν.

Οδηγία critical

Απλουστευμένη σύνταξη:

```
#pragma omp critical <new-line>
<structured-block>
```

Εξασφαλίζει ότι το δομημένο τμήμα κώδικα που ακολουθεί θα εκτελείται από ένα νήμα μόνο τη φορά.

Οδηγία taskwait

Απλουστευμένη σύνταξη:

```
#pragma omp taskwait <new-line>
```

Εξασφαλίζει ότι οι εργασίες-παιδιά της τρέχουσας εργασίας (task) θα έχουν ολοκληρωθεί πριν συνεχιστεί η εκτέλεση μετά από αυτό το σημείο.

2.3.8 Ρουτίνες βιβλιοθήκης χρόνου εκτέλεσης

- `void omp_set_num_threads(int num_threads)`: Θέτει την τιμή της παραμέτρου `num_threads` ως το πλήθος των νημάτων που θα χρησιμοποιηθούν σε επερχόμενες παράλληλες περιοχές. Εξαίρεση αποτελούν οι παράλληλες περιοχές που χρησιμοποιούν τη φράση `num_threads` η οποία υπερισχύει.
- `int omp_get_num_threads(void)`: Επιστρέφει το πλήθος των νημάτων που συνιστούν την τρέχουσα ομάδα.
- `int omp_get_thread_num(void)`: Επιστρέφει το αριθμητικό αναγνωριστικό του καλούντος νήματος μέσα στο πλαίσιο της τρέχουσας ομάδας.
- `double omp_get_wtime(void)`: Επιστρέφει το χρόνο (wall clock) σε δευτερόλεπτα που παρήλθε μετά από μία δεδομένη αλλά ταυτόχρονα αυθαίρετη στιγμή στο παρελθόν.

2.3.9 Μεταβλητές περιβάλλοντος

Η μεταβλητή περιβάλλοντος `OMP_NUM_THREADS` μπορεί να χρησιμοποιηθεί για τον ορισμό ενός προκαθορισμένου πλήθους νημάτων που θα συμμετέχουν στις παράλληλες περιοχές του προγράμματος. Η τελική απόφαση για το πλήθος των νημάτων που θα χρησιμοποιηθούν σε μία παράλληλη περιοχή καθορίζεται σε φθίνουσα προτεραιότητα από:

1. Τη μεταβλητή περιβάλλοντος `OMP_NUM_THREADS`
2. Τη ρουτίνα βιβλιοθήκης χρόνου εκτέλεσης `omp_set_num_threads`
3. Τη φράση `num_threads` της οδηγίας `parallel`

2.4 Μεταφραστές OpenMP

Η διεπαφή που ορίζεται από το πρότυπο του OpenMP υλοποιείται από διάφορους εμπορικούς και ερευνητικούς μεταφραστές. Προμηθευτές μεταφραστών για C/C++ που υποστηρίζουν το OpenMP είναι οι εξής:

- AMD:
  - Ο AOMP είναι βασισμένος στον LLVM/Clang και υποστηρίζει την εκτέλεση κώδικα σε πολλαπλές κάρτες γραφικών/επιταχυντές.
  - Ο AOCC είναι επίσης βασισμένος στον clang/LLVM και υποστηρίζει πλήρως το OpenMP 4.5 και μερικώς το OpenMP 5.0.
- ARM: Ο μεταφραστής της ARM παρέχει πλήρη υποστήριξη για το OpenMP 3.1 και υποστηρίζει το OpenMP 4.0/4.5 χωρίς τη δυνατότητα εκτέλεσης κώδικα σε συσκευές η οποία βρίσκεται υπό ανάπτυξη.
- Barcelona Supercomputing Center: Ο Mercurium είναι ένας ερευνητικός μεταφραστής πηγαίου σε πηγαίο κώδικα (source-to-source) ο οποίος υποστηρίζει σχεδόν πλήρως το OpenMP 3.1 καθώς και χαρακτηριστικά νεότερων εκδόσεων που σχετίζονται με τον μηχανισμό tasking του OpenMP.
- Fujitsu: Οι μεταφραστές για τον υπερυπολογιστή PRIMEHPC FX100 της Fujitsu υποστηρίζουν το OpenMP 3.1.
- GNU: Ο GCC υποστηρίζει πλήρως το OpenMP 4.5 (έκδοση 6) και μερικώς το OpenMP 5.0. Επίσης, σε συστήματα Linux υποστηρίζει την εκτέλεση κώδικα σε κάρτες γραφικών NVIDIA (nvptx) και τις κάρτες Fiji και Vega της AMD Radeon (GCN).
- HPE: Το Cray Compiling Environment (CCE) παρέχει πλήρη υποστήριξη για το OpenMP 4.5 και μερική υποστήριξη για το OpenMP 5.0.

- IBM: Ο μεταφραστής XL C/C++ για Linux υποστηρίζει πλήρως το OpenMP 4.5.

- Intel: Οι μεταφραστές της Intel υποστηρίζουν πλήρως το OpenMP 4.5 και μερικώς το OpenMP 5.0.

- LLNL Rose Research Compiler: Ο ROSE είναι ένας ερευνητικός μετφραστής πηγαίου σε πηγαίο κώδικα που υποστηρίζει το OpenMP 3.0 και κάποια χαρακτηριστικά του OpenMP 4.0 που σχετίζονται με την εκτέλεση κώδικα σε κάρτες γραφικών/επιταχυντές της NVIDIA.

- LLVM: Ο Clang παρέχει υποστήριξη για το OpenMP 4.5 με περιορισμένη υποστήριξη για την εκτέλεση κώδικα σε συσκευές. Επίσης, υποστηρίζεται μεγάλο μέρος του OpenMP 5.0 και μικρό μέρος του OpenMP 5.1.

- Siemens: Ο Sourcery CodeBench (AMD GCN) Lite για συστήματα x86_64 GNU/Linux είναι βασισμένος στον GCC, παρέχει πλήρη υποστήριξη για το OpenMP 4.5, μερική υποστήριξη για το OpenMP 5.0. και επιτρέπει την εκτέλεση κώδικα σε κάρτες γραφικών AMD Radeon (GCN) όπως οι Fiji, gfx900 Vega 10 και gfx906 Vega 20.

- NVIDIA HPC Compiler: Οι μεταφραστές NVIDIA HPC παρέχουν πλήρη υποστήριξη του OpenMP 3.1 και μερική υποστήριξη του OpenMP 5.0 για συστήματα Linux/x86-64, Linux/OpenPOWER, Linux/Arm. Επίσης, σε κάρτες γραφικών της NVIDIA υποστηρίζεται μερικώς το OpenMP 5.0.

- OpenUH Research Compiler: Ο OpenUH είναι ερευνητικός μεταφραστής και υποστηρίζει πλήρως το OpenMP 2.5 και σχεδόν πλήρως το OpenMP 3.0 σε συστήματα Linux.

- Oracle: Οι μεταφραστές του Oracle Developer Studio υποστηρίζουν το OpenMP 4.0.

- PGI: Οι μεταφραστές NVidia HPC υποστηρίζουν μερικώς το OpenMP 5.0.

- Texas Instruments:

  – Ο μεταφραστής TI cl6x υποστηρίζει το OpenMP 3.0. (C66x).

– Το βασισμένο στον GCC Linaro toolchain υποστηρίζει το OpenMP 4.5 (Cortex-A15).

– Ο μεταφραστής TI clacc υποστηρίζει το OpenMP 3.0 και εκτέλεση κώδικα σε συσκευές βάσει του OpenMP 4.0 (Cortex-A15+C66x-DSP).

Να σημειωθεί ότι η παρεχόμενη υποτήριξη αφορά προϊόντα system on a chip (SoC) της Texas Instruments.

2.4.1 Ο μεταφραστής OMPi

Ο μεταφραστής OMPi αναπτύσσεται από την Ομάδα Παράλληλης Επεξεργασίας του Πανεπιστημίου Ιωαννίνων από το 2001 και είναι ένας μεταφραστής για τη γλώσσα C που υποστηρίζει τη διεπαφή προγραμματισμού εφαρμογών OpenMP. Ο OMPi είναι οργανωμένος σε δύο βασικά μέρη, τον μεταφραστή (compiler) και το σύστημα χρόνου εκτέλεσης (runtime).

Διαδικασία μετάφρασης προγραμμάτων χρήστη

Η πλήρης διαδικασία μετάφρασης που ακολουθεί ο OMPi φαίνεται στο Σχήμα Χ. Το πρόγραμμα εισόδου είναι πηγαίος κώδικας σε γλώσσα C που περιλαμβάνει οδηγίες OpenMP. Όπως συμβαίνει με όλα τα προγράμματα C, αρχικά περνάνε από το στάδιο της προεπεξεργασίας. Η έξοδος αυτού του πρώτου σταδίου τροφοδοτείται στο τμήμα του μεταφραστή (compiler) του OMPi, ο οποίος κάνει λεκτική και συντακτική ανάλυση. Από τη συντακτική ανάλυση προκύπτει το αφηρημένο συντακτικό δέντρο (AST - Abstract Syntax Tree) το οποίο χρησιμεύει στην αντικατάσταση των οδηγιών OpenMP με κλήσεις συναρτήσεων του συστήματος χρόνου εκτέλεσης, οι οποίες υλοποιούν τις λειτουργίες που καθορίζουν οι αντίστοιχες οδηγίες. Στη συνέχεια, ο μετασχηματισμένος κώδικας δίνεται ως είσοδος σε έναν απλό μεταφραστή για γλώσσα C (π.χ. GCC) ώστε να παραχθεί το αντικείμενο πρόγραμμα. Τέλος, από τη σύνδεση (linking) του αντικείμενου προγράμματος με τη βιβλιοθήκη χρόνου εκτέλεσης του OMPi και τις βιβλιοθήκες του συστήματος, προκύπτει το τελικό εκτελέσιμο αρχείο.

Σύστημα χρόνου εκτέλεσης (runtime)

Το σύστημα χρόνου εκτέλεσης παρέχει όλες τις απαραίτητες βοηθητικές συναρτήσεις που απαιτούνται σε χρόνο εκτέλεσης. Παράδειγμα τέτοιων συναρτήσεων είναι οι συναρτήσεις που υλοποιούν αμοιβαίο αποκλεισμό και συγχρονισμό όπως κλειδαριές και κλήσεις φραγής, συναρτήσεις δημιουργίας οντοτήτων εκτέλεσης, συναρτήσεις που διαβάζουν τις μεταβλητές περιβάλλοντος κλπ.

Αποτελείται από τα τμήματα host και devices. Το πρώτο τμήμα αφορά την υποστήριξη εκτέλεσης κώδικα στο κύριο σύστημα (host), δηλαδή εκεί όπου ξεκίνησε η εκτέλεση του προγράμματος, ενώ το δεύτερο τμήμα αφορά την εκτέλεση κώδικα σε συσκευές όπως κάρτες γραφικών γενικού σκοπού (devices). Το τμήμα host αποτελείται με τη σειρά του από το ORT (**O**MPi **R**un**T**ime) και τις βιβλιοθήκες οντοτήτων εκτέλεσης (EELIBs - **E**xecution **E**ntity **LIB**raries).

Η οντότητα εκτέλεσης είναι μία αφηρημένη έννοια που χρησιμοποιείται για την απόκρυψη των λεπτομερειών υλοποίησης των EELIBs ώστε αυτά να είναι εντελώς ανεξάρτητα από τον υπόλοιπο κώδικα του OMPi. Το ORT διαχειρίζεται και συντονίζει τις οντότητες εκτέλεσης σε υψηλό επίπεδο, δηλαδή μέσω μίας διεπαφής προγραμματισμού εφαρμογών (API) που παρέχουν όλα τα EELIBs. Από τη μεριά τους, τα EELIBs διαχειρίζονται τις οντότητες εκτέλεσης σε χαμηλό επίπεδο ανάλογα με τις λεπτομέρειες υλοποίησης της εκάστοτε βιβλιοθήκης. Για να γίνει πιο κατανοητό, το ORT με μία κλήση της συνάρτησης `othr_request()` ζητάει από το EELIB να δημιουργήσει ένα συγκεκριμένο πλήθος οντοτήτων εκτέλεσης, χωρίς όμως να γνωρίζει λεπτομέρειες για τις οντότητες αυτές (π.χ. αν είναι νήματα POSIX, νήματα PSTHREADS ή διεργασίες). Αντίθετα, το EELIB γνωρίζει όλες τις λεπτομέρειες υλοποίησης και πραγματοποιεί τις κατάλληλες κλήσεις για τη δημιουργία των οντοτήτων (π.χ. `pthread_create()` στην περίπτωση των νημάτων POSIX).

Η ιδιαίτερη αρχιτεκτονική σχεδίαση του συστήματος χρόνου εκτέλεσης και ο τρόπος αλληλεπίδρασής του με τα EELIBS καθιστούν δυνατή τη χρήση πολλών διαφορετικών τύπων οντοτήτων εκτέλεσης. Επίσης σημαντικό χαρακτηριστικό είναι ότι ο οποιοσδήποτε μπορεί να αναπτύξει και να χρησιμοποιήσει τη δική του EELIB χωρίς να ασχοληθεί με τον κώδικα του OMPi.

ΚΕΦΑΛΑΙΟ 3

Τοπολογία Συστήματος

Η δημιουργία φορητού (portable) λογισμικού, δηλαδή λογισμικού το οποίο μπορεί να χρησιμοποιηθεί σε συστήματα διαφορετικά μεταξύ τους, είναι δυνατή με τη χρήση αφαιρέσεων (abstractions) για το υποκείμενο σύστημα. Οι αφαιρέσεις αυτές αποκρύπτουν λεπτομέρειες της οργάνωσης του υλικού του εκάστοτε συστήματος, με την έννοια ότι από τη σκοπιά του προγραμματιστή δεν θα πρέπει να μας ενδιαφέρουν ζητήματα χαμηλού επιπέδου, όπως για παράδειγμα η οργάνωση της μνήμης από φυσική άποψη (κοινόχρηστη ή κατανεμημένη) που αναφέραμε στην Υποενότητα 1.2.2, αλλά το τι εικόνα έχουμε για αυτή.

Η αφαιρετική όμως εικόνα που έχει ο προγραμματιστής για το σύστημα που καλείται να προγραμματίσει δεν του επιτρέπει να εκμεταλλευτεί στο μέγιστο τις δυνατότητές του και άρα να επιτύχει την καλύτερη δυνατή επίδοση. Για παράδειγμα, σε ένα σύστημα δύο κόμβων με κατανεμημένη φυσική οργάνωση μνήμης που όμως παρέχει κοινόχρηστο χώρο διευθύνσεων, ναι μεν ο ένας κόμβος μπορεί να προσβεί οποιαδήποτε θέση μνήμης στον άλλο κόμβο, παρόλα αυτά το κόστος της απομακρυσμένης πρόσβασης θα είναι πολύ μεγαλύτερο από αυτό της πρόσβασης στην τοπική μνήμη.

Στον προγραμματισμό παράλληλων συστημάτων που στόχος μας είναι η επίτευξη όσο το δυνατόν καλύτερων επιδόσεων, συχνά καταφεύγουμε στην εκμετάλλευση πληροφοριών που σχετίζονται με την τοπολογία. Για το λόγο αυτό γίνονται προσπάθειες δημιουργίας μεταφέρσιμου λογισμικού που όμως λαμβάνει υπόψην του την οργάνωση του υλικού σε χρόνο εκτέλεσης.

3.1 Η εξέλιξη των βασικών στοιχείων ενός συστήματος

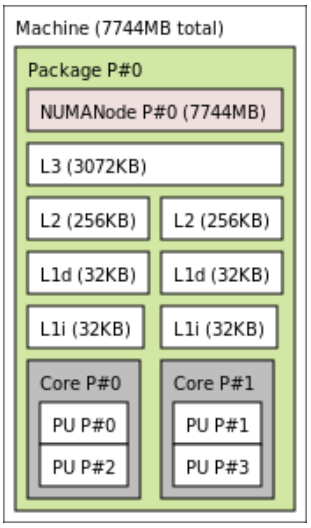
Τα πιο σημαντικά στοιχεία που αποτελούν ένα υπολογιστικό σύστημα είναι αυτά που αφορούν τα υποσυστήματα επεξεργαστή και μνήμης.

Ένα απλό σειριακό σύστημα αποτελούνταν από έναν επεξεργαστή (ΚΜΕ - Κεντρική Μονάδα Επεξεργασίας, CPU - Central Processing Unit) ο οποίος περιείχε έναν επεξεργαστικό πυρήνα (ή απλώς πυρήνας, core) και επικοινωνούσε με την κύρια μνήμη μέσω ενός διαύλου. Με την εμφάνιση των πολυπύρηνων (multicore) οργανώσεων περισσότεροι του ενός πυρήνες ενσωματώθηκαν στον ίδιο επεξεργαστή, ενώ με την πάροδο του χρόνου προστέθηκε και ιεραρχία κρυφών μνημών.

Στα πλαίσια μιας άλλης αρχιτεκτονικής βελτίωσης, με μικρή αύξηση του μεγέθους του κυκλώματος, μπόρεσαν να ενσωματωθούν μέσα σε έναν πυρήνα 2 ή περισσότερα νήματα υλικού (H/W threads, logical processors - λογικοί επεξεργαστές), ώστε να αξιοποιηθούν καλύτερα οι λειτουργικές μονάδες του επεξεργαστή, προσφέροντας με αυτό τον τρόπο τη δυνατότητα παράλληλης εκτέλεσης.

Τη σημερινή εποχή, μπορούμε να συναντήσουμε μεγάλα συστήματα που διαθέτουν πολλαπλούς επεξεργαστές, όπου κάθε επεξεργαστής είναι πολυπύρηνος, περιλαμβάνει ιεραρχία κρυφών μνημών και τοπική μνήμη, ενώ κάθε πυρήνας διαθέτει πολλαπλά H/W threads.

Στο Σχήμα 3.1 φαίνεται η οργάνωση ενός επεξεργαστή που διαθέτει 2 πυρήνες με 2 H/W threads και μία ιεραρχία κρυφών μνημών τριών επιπέδων (L1, L2, L3) στην οποία η κρυφή μνήμη του πρώτου επιπέδου διακρίνεται σε κρυφή μνήμη δεδομένων (L1d) και εντολών (L1i). Η οπτικοποίηση της τοπολογίας έγινε με το λογισμικό Portable Hardware Locality (hwloc) με το οποίο θα ασχοληθούμε περαιτέρω στην Υποενότητα 3.3.1.



Σχήμα 3.1: Η τοπολογία ενός Intel® Core™ i3-7100U.

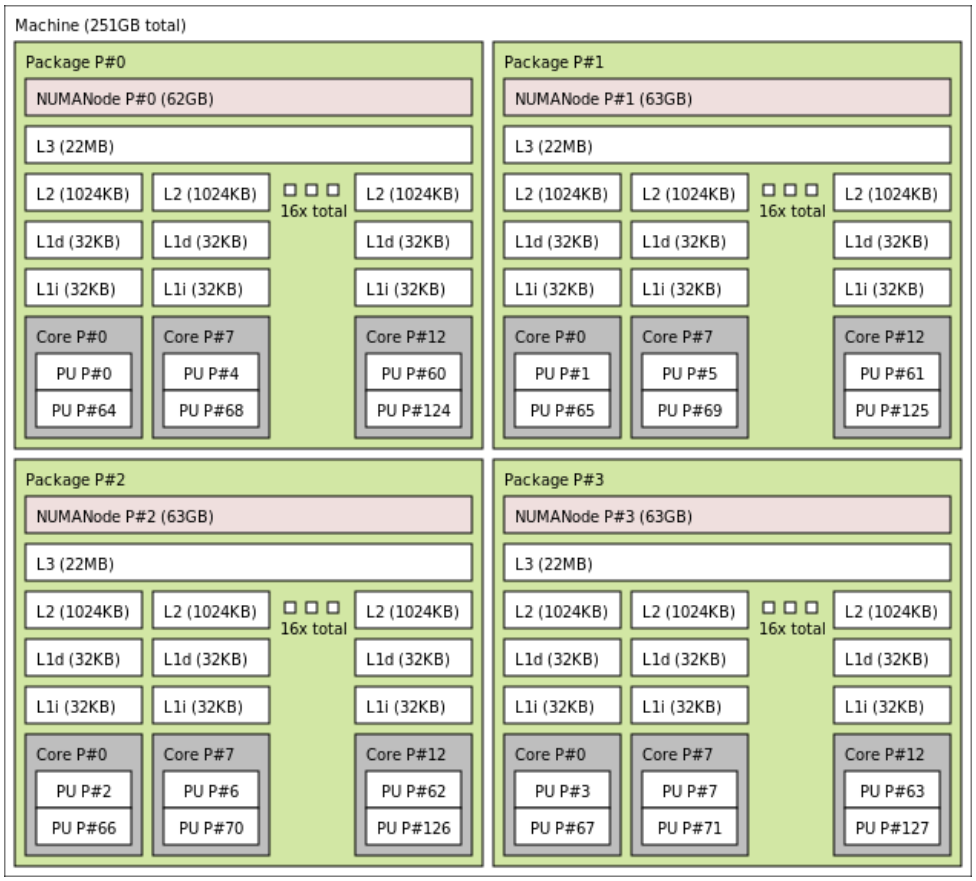
3.2 Συστήματα NUMA

Στην Υποενότητα 1.2.2 περιγράφηκε η οργάνωση των συστημάτων ανομοιόμορφης προσπέλασης μνήμης (NUMA), τα οποία αποτελούν αρχιτεκτονική εξέλιξη των Συμμετρικών Πολυεπεξεργαστών (SMPs) που επιτρέπει την κατασκευή μεγαλύτερων παράλληλων συστημάτων.

Η πιο συχνή υλοποίηση των συστημάτων NUMA είναι ως ένα σύνολο κόμβων τύπου SMP που διαθέτουν ιεραρχία κρυφών μνημών συνδεδεμένους μεταξύ τους με ένα δίκτυο διασύνδεσης [4]. Το δίκτυο διασύνδεσης μπορεί να είναι δακτύλιος (ring), διακοπτικό δίκτυο τύπου crossbar, από σημείο σε σημείο (point-to-point) ή πλέγμα (mesh).

Η εξασφάλιση της συνοχής των κρυφών μνημών συνήθως επιτυγχάνεται με τη χρήση ενός πρωτοκόλλου εντός του κόμβου και ενός δεύτερου πρωτοκόλλου μεταξύ των κόμβων. Όπως αναφέρθηκε νωρίτερα, ένας κόμβος SMP βασίζεται σε δίαυλο, ενώ η επικοινωνία μεταξύ των κόμβων γίνεται με πιο πολύπλοκα δίκτυα διασύνδεσης. Για το λόγο αυτό, το πρωτόκολλο συνοχής εντός του κόμβου είναι τύπου παρακολούθησης (snooping protocol), ενώ αυτό μεταξύ των κόμβων βασίζεται σε καταλόγους (directory-based protocol). Φυσικά, σε διαφορετικές οργανώσεις συστημάτων NUMA μπορούμε να συναντήσουμε και άλλες στρατηγικές επίτευξης συνοχής, όπως αυτή της μη αποθήκευσης κοινόχρηστων δεδομένων στις κρυφές μνήμες.

Όπως γίνεται γρήγορα αντιληπτό, ο προγραμματισμός αυτών των συστημάτων



Σχήμα 3.2: Η τοπολογία ενός Dell PowerEdge R840 με 4 Intel® Xeon® Gold 6130.

συνήθως απαιτεί να εστιάσουμε στην τοπικότητα, δηλαδή τη χρήση πόρων που βρίσκονται στη μικρότερη δυνατή απόσταση και την μείωση της επικοινωνίας μεταξύ των κόμβων [5]. Για την επίτευξη της τοπικότητας και άρα τη συγγραφή πιο αποδοτικών προγραμμάτων, η εκμετάλλευση πληροφορίας που σχετίζεται με την τοπολογία είναι απαραίτητη.

Στο Σχήμα 3.2 φαίνεται η οργάνωση του `parade`, ενός NUMA συστήματος με τέσσερις κόμβους τύπου SMP που διαθέτει η Ομάδα Παράλληλης Επεξεργασίας του Πανεπιστημίου Ιωαννίνων και η οργάνωσή του αποτελεί την πλέον συνηθισμένη.

3.3 Βοηθητικά Εργαλεία

Για την εκμετάλλευση της τοπολογίας ενός συστήματος έχουν αναπτυχθεί διάφορα εργαλεία όπως η βιβλιοθήκη Portable Hardware Locality (hwloc) και η `libnuma`.

3.3.1 hwloc

Η βιβλιοθήκη Portable Hardware Locality παρέχει μία φορητή αφαίρεση (abstraction) της τοπολογίας που διαθέτουν υπολογιστικά συστήματα με σύγχρονες αρχιτεκτονικές. Ο κύριος λόγος χρήσης της είναι η συγκέντρωση πληροφοριών σχετικά με πολύπλοκα παράλληλα συστήματα, με στόχο την κατάλληλη και αποδοτική εκμετάλλευσή τους [6].
Τα λειτουργικά συστήματα που υποστηρίζονται είναι τα εξής:

- Linux
- Solaris, AIX, HP-UX
- NetBSD, FreeBSD, kFreeBSD/GNU
- Darwin / OS X
- Microsoft Windows
- IBM BlueGene/Q Compute Node Kernel (CNK)

Τα στοιχεία που απαρτίζουν την τοπολογία

Η hwloc μπορεί να αναγνωρίσει τα εξής στοιχεία υλικού:

- Κόμβους συστημάτων NUMA (NUMA nodes)
- Υποδοχές επεξεργαστών (packages, πρώην sockets)
- Κρυφές μνήμες (caches)
- Πυρήνες (cores)
- H/W threads (PUs - Processing Units)
- Σύσκευές εισόδου-εξόδου όπως:
  - Συσκευές PCI
  - Επιταχυντές OpenCL, CUDA και Xeon Phi
  - Προσαρμογείς δικτύου (network interfaces) όπως InfiniBand

Με την ορολογία της hwloc στη διάθεση μας, ας ανατρέξουμε στο Σχήμα 3.2 που απεικονίζεται η τοπολογία του συστήματος parade. Στο σχήμα αυτό βλέπουμε 4 επεξεργαστές ο καθένας εκ των οποίων διαθέτει 12 πυρήνες, και ιεραρχία κρυφών μνημών τριών επιπέδων (L1i & L1d, L2, L3). Τα επίπεδα ένα και δύο των κρυφών μνημών είναι κοινά ανά πυρήνα, ενώ το επίπεδο τρία είναι κοινό για όλους τους πυρήνες του επεξεργαστή. Επίσης, κάθε πυρήνας διαθέτει δύο H/W threads τα οποία μοιράζονται τους πόρους του πυρήνα (π.χ. κρυφές μνήμες).

Οι εντολές lstopo και lstopo-no-graphics

Οι εντολές `lstopo` και `lstopo-no-graphics` συμπεριλαμβάνονται στην hwloc και χρησιμοποιούνται για την προβολή της τοπολογίας ενός συστήματος σε διαφορετικές μορφές, όπως κείμενο ASCII, SVG, PDF, Postscript, PNG, XML και αρκετές άλλες.

Τα Σχήματα 3.1 και 3.2 είναι αποτέλεσμα αυτών των δύο εντολών. Η αναπαράσταση του Σχήματος 3.1 σε μορφή κειμένου ASCII είναι η ακόλουθη[1]:

```
Machine (7744MB total)
  Package L#0
    NUMANode L#0 (P#0 7744MB)
    L3 L#0 (3072KB)
      L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
        PU L#0 (P#0)
        PU L#1 (P#2)
      L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1
        PU L#2 (P#1)
        PU L#3 (P#3)
```

Η διαφορά των δύο εντολών είναι ότι η δυνατότητα γραφικής αναπαράστασης (π.χ. PNG) υπάρχει μόνο στην πρώτη. Αυτό συμβαίνει για τη μείωση των εξαρτήσεων με τις σχετικές βιβλιοθήκες γραφικών που χρησιμοποιούνται για τη δημιουργία των οπτικοποιήσεων. Για παράδειγμα, στο σύστημα parade που χρησιμοποιείται για τη διεξαγωγή πειραματικών μετρήσεων δεν υπάρχουν οι απαραίτητες βιβλιοθήκες γραφικών. Για το λόγο αυτό χρησιμοποιήθηκε η εντολή `lstopo-no-graphics` για την αποθήκευση της τοπολογίας σε μορφή XML και σε διαφορετικό σύστημα με τις

[1]Για λόγους απλότητας έχουν παραλειφθεί οι συσκευές Εισόδου/Εξόδου και από τις δύο αναπαραστάσεις.

κατάλληλες βιβλιοθήκες εγκατεστημένες χρησιμοποιήθηκε η μορφή XML ως είσοδος στην εντολή `lstopo` για την παραγωγή αρχείου PNG.

Η διεπαφή προγραμματισμού εφαρμογών (API)

Η βασική διεπαφή προγραμματισμού εφαρμογών (API) της hwloc είναι διαθέσιμη μέσω του αρχείου `hwloc.h`, ενώ για την επιτυχή μετάφραση χρειάζεται το όρισμα `-lhwloc`.

Η τοπολογία του υποκείμενου συστήματος αναπαρίσταται ως μία δεντρική δομή από αντικείμενα και αποθηκεύεται σε μία μεταβλητή τύπου `hwloc_topology_t`. Στη ρίζα του δέντρου υπάρχει πάντα ένα αντικείμενο τύπου Machine που αναπαριστά ολόκληρο το υποκείμενο μηχάνημα/σύστημα, ενώ στα φύλλα του δέντρου υπάρχουν πάντα PUs ή αλλιώς H/W threads που αποτελούν το μικρότερο διαθέσιμο επεξεργαστικό στοιχείο. Καμία άλλη παραδοχή δεν είναι δυνατόν να γίνει για τη μορφή της τοπολογίας (π.χ. βάθος, τύποι αντικειμένων που την απαρτίζουν κλπ).

Για τη διαχείριση της μεταβλητής τύπου `hwloc_topology_t` χρησιμοποιούνται οι εξής βασικές συναρτήσεις των οποίων η λειτουργία είναι εύκολα κατανοητή από το όνομά τους:

- `hwloc_topology_init()`
- `hwloc_topology_load()`
- `hwloc_topology_destroy()`

Αφού αρχικοποιηθεί και φορτωθεί η τοπολογία με τις δύο πρώτες κλήσεις, μπορεί να γίνει αναδρομική διάσχιση του δέντρου για την ανακάλυψη των διαφορετικών συστατικών του στοιχείων (packages, NUMA nodes, caches, cores, PUs). Η hwloc διαθέτει πλήθος χρήσιμων συναρτήσεων που διευκολύνουν είτε την αναδρομική διάσχιση, είτε τη διάσχιση ενός μόνο επιπέδου και επιτρέπουν:

- την εύρεση του αντικειμένου στη ρίζα της τοπολογίας (`hwloc_get_root_obj()`).
- την εύρεση του πλήθους των αντικειμένων που έχουν συγκεκριμένο τύπο ή βάθος (`hwloc_get_nbobjs_by_type()` και `hwloc_get_nbobjs_by_depth()`).
- την εύρεση αντικειμένων με συγκεκριμένο τύπο ή σε συγκεκριμένο βάθος (`hwloc_get_obj_by_type()` και `hwloc_get_obj_by_depth()`).

- την εύρεση του επόμενου στη σειρά αντικειμένου με συγκεκριμένο τύπο ή βάθος (`hwloc_get_next_obj_by_type()` και `hwloc_get_next_obj_by_depth()`).

Άλλες δυνατότητες της hwloc

Πέρα από τη συγκέντρωση πληροφοριών που σχετίζονται με την τοπολογία του υποκείμενου συστήματος, η hwloc μπορεί να χρησιμοποιηθεί για την ανάθεση νημάτων (thread binding) σε επεξεργαστές και τη διαχείριση μνήμης με δυνατότητα δέσμευσης τμημάτων μνήμης σε συγκεκριμένους κόμβους NUMA (memory binding). Παρόλα αυτά, η χρήση των διαθέσιμων λειτουργιών υπόκειται στις δυνατότητες του εκάστοτε λειτουργικού συστήματος.

3.3.2 libnuma

Η libnuma είναι μία βιβλιοθήκη που παρέχει μία διεπαφή προγραμματισμού εφαρμογών (API) για την πολιτική NUMA (NUMA policy) του Linux Kernel.

Η πολιτική NUMA του Linux αφορά τη δέμευση μνήμης σε συγκεκριμένους κόμβους NUMA ώστε να μπορεί το πρόγραμμα που εκτελείται να την προσβεί όσο το δυνατόν πιο γρήγορα [7]. Η προκαθορισμένη πολιτική του Linux είναι η δέσμευση μνήμης να πραγματοποιείται στον τοπικό κόμβο του νήματος που την προκάλεσε [8, `mm/mempolicy.c`]. Ένα νήμα προκαλεί δέσμευση μνήμης όταν γράψει για πρώτη φορά σε μία εικονική διεύθυνση[2] που ανήκει σε σελίδα[3] (page) της εικονικής μνήμης η οποία δεν έχει αντιστοιχιστεί σε σελίδα φυσικής μνήμης. Αυτή η πολιτική είναι γνωστή και ως first-touch.

Η πολιτική first-touch είναι αποδοτική όταν οι προσβάσεις πραγματοποιούνται από νήματα που εκτελούνται στον κόμβο όπου έχει δεσμευτεί η μνήμη. Υπάρχουν όμως περιπτώσεις που μπορεί να οδηγηθούμε σε κακές επιδόσεις. Μία τέτοια περίπτωση είναι όταν μία πολυνηματική εφαρμογή εκτελείται σε πολλαπλούς κόμβους αλλά μόνο ένα νήμα κάνει τις αρχικοποιήσεις της μνήμης. Κάτι τέτοιο θα οδηγήσει στη δέσμευση της μνήμης μόνο σε ένα κόμβο και άρα τα νήματα που είναι τοπο-

[2] Το λειτουργικό σύστημα χρησιμοποιεί έναν μηχανισμό γνωστό ως εικονική μνήμη (virtual memory) ώστε να μπορεί να παρέχει σε κάθε διεργασία ένα συνεχόμενο χώρο (εικονικών) διευθύνσεων ικανοποιητικού μεγέθους (π.χ. 4 Gigabytes για συστήματα 32 bit), ακόμα και αν αυτός ο χώρος δεν είναι διαθέσιμος στη φυσική μνήμη (π.χ. 8 Gigabytes).

[3] Η εικονική και φυσική μνήμη είναι χωρισμένες σε τμήματα, συνήθως μεγέθους 4,096 Bytes, που ονομάζονται σελίδες (pages).

θετημένα στους υπόλοιπους κόμβους θα αναγκαστούν να πραγματοποιούν απομακρυσμένες προσβάσεις με αποτέλεσμα να παρατηρηθούν αυξημένες καθυστερήσεις. Τη λύση σε αυτό το πρόβλημα μπορεί να δώσει η libnuma, καθώς επιτρέπει στο νήμα που κάνει όλες τις αρχικοποιήσεις να να δεσμεύσει μνήμη σε οποιοδήποτε κόμβο και συνεπώς να κατανείμει τη μνήμη λαμβάνοντας υπόψην από ποιό κόμβο πρόκειται να προέλθουν οι περισσότερες προσπελάσεις.

Η προκαθορισμένη πολιτική μπορεί να αλλάξει για συγκεκριμένα τμήματα μνήμης ή σε επίπεδο νήματος/διεργασίας. Στην πρώτη περίπτωση η πολιτική είναι ίδια για όλα τα νήματα/διεργασίες. Στη δεύτερη περίπτωση, η αλλαγή επηρεάζει μόνο το τρέχον νήμα/διεργασία και η νέα πολτική κληρονομείται σε νήματα/διεργασίες-παιδιά που ίσως δημιουργηθούν μεταγενέστερα. Οι διαθέσιμες πολιτικές είναι οι εξής:

- page interleaving: Οι σελίδες δεσμεύονται κυκλικά (round-robin) σε όλους τους διαθέσιμους κόμβους (ή ένα υποσύνολό τους).
- preferred node allocation: Οι σελίδες δεσμεύονται στον επιθυμητό κόμβο.
- local allocation: Οι σελίδες δεσμεύονται στον τοπικό κόμβο όπου εκτελείται η διεργασία ή το νήμα.
- allocation only on specific nodes: Οι σελίδες δεσμεύονται μόνο σε συγκεκριμένους κόμβους.

Είναι σημαντικό να διευκρινιστεί ότι η διαχείριση μνήμης στη libnuma γίνεται σε επίπεδο σελίδων. Για παράδειγμα, όταν ο χρήστης αιτείται δέσμευση μνήμης συγκεκριμένου μεγέθους, αυτό το μέγεθος στρογγυλοποιείται προς τα πάνω ώστε να είναι πολλαπλάσιο του μεγέθους σελίδας (page size, συνήθως 4,096 Bytes). Επίσης, η ανομοιόμορφη προσπέλαση μνήμης που προκύπτει λόγω της ύπαρξης ιεραρχίας κρυφών μνημών αγνοείται κατά τη διαδικασία ορισμού των κόμβων NUMA [9].

Η διεπαφή προγραμματισμού εφαρμογών (API)

Η βασική διεπαφή προγραμματισμού εφαρμογών (API) της libnuma είναι διαθέσιμη μέσω του αρχείου `numa.h`, ενώ για την επιτυχή μετάφραση χρειάζεται το όρισμα `-lnuma`.

Πριν οποιαδήποτε κλήση συνάρτησης που παρέχει η libnuma, είναι υποχρεωτικό να κληθεί η συνάρτηση `numa_available()` και σε περίπτωση που επιστραφεί η τιμή -1 οι συναρτήσεις της libnuma είναι απροσδιόριστες (undefined).

Οι ακόλουθες συναρτήσεις δεσμεύουν μνήμη χρησιμοποιώντας κάποια από τις διαθέσιμες πολιτικές, χωρίς όμως να αλλάξουν την προκαθορισμένη πολιτική:

- `numa_alloc_onnode()`: Δεσμεύει μνήμη σε συγκεκριμένο κόμβο.
- `numa_alloc_local()`: Δεσμεύει μνήμη στον τοπικό κόμβο.
- `numa_alloc_interleaved()` και `numa_alloc_interleaved_subset()`: Δεσμεύουν μνήμη κυκλικά σε όλους τους κόμβους ή σε ένα υποσύνολό τους αντίστοιχα.
- `numa_alloc()`: Δεσμεύει μνήμη βάσει της τρέχουσας πολιτικής.

Η απελευθέρωση της μνήμης που δεσμεύτηκε με τις παραπάνω συναρτήσεις γίνεται μέσω της συνάρτησης `numa_free()`, ενώ η συνάρτηση `numa_realloc()` επιτρέπει την αλλαγή του μεγέθους της ήδη δεσμευμένης μνήμης.

Συναρτήσεις αλλαγής της προκαθορισμένης πολιτικής του τρέχοντος νήματος ή διεργασίας:

- `numa_set_localalloc()`: Αλλάζει την πολιτική σε local allocation.
- `numa_set_preferred()`: Αλλάζει την πολιτική σε preferred node allocation και θέτει ως προτιμητέο κόμβο αυτόν που περνιέται ως παράμετρος.
- `numa_set_interleave_mask()`: Αλλάζει την πολιτική σε page interleaving και η δέσμευση πραγματοποιείται κυκλικά στους κόμβους που καθορίζονται μέσω της μάσκας που περνιέται ως παράμετρος.
- `numa_set_membind()`: Αλλάζει την πολιτική ώστε η δέσμευση μνήμης να πραγματοποιείται μόνο από τους κόμβους που καθορίζονται μέσω της μάσκας που περνιέται ως παράμετρος.

Άλλες χρήσιμες συναρτήσεις:

- `numa_max_node()`: Επιστρέφει το μέγιστο αναγνωριστικό κόμβου που είναι διαθέσιμο στο υποκείμενο σύστημα.

- `numa_num_configured_nodes()`: Επιστρέφει το πλήθος των κόμβων του υποκείμενου συστήματος συμπεριλαμβανομένων και των απενεργοποιημένων (disabled) κόμβων.
- `numa_get_mems_allowed()`: Επιστρέφει μία μάσκα που αναπαριστά τους κόμβους στους οποίους η τρέχουσα διεργασία μπορεί να δεσμεύσει μνήμη.
- `numa_num_configured_cpus()`: Επιστρέφει το πλήθος των επεξεργαστών του υποκείμενου συστήματος συμπεριλαμβανομένων και όσων είναι απενεργοποιημένοι (disabled).
- `numa_node_size()`: Επιστρέφει το μέγεθος της μνήμης ενός κόμβου και προεραιτικά (μέσω παραμέτρου) το μέγεθος της διαθέσιμης προς δέσμευση μνήμης.
- `numa_pagesize()`: Επιστρέφει το μέγεθος της σελίδας σε bytes.
- `numa_node_of_cpu()`: Επιστρέφει το αναγνωριστικό του κόμβου στον οποίο ανήκει ένας επεξεργαστής[4].

Άλλες δυνατότητες της libnuma

Η libnuma υποστηρίζει μεταξύ άλλων, λειτουργίες όπως η αλλαγή της πολιτικής ήδη δεσμευμένης μνήμης, περιορισμού της εκτέλεσης ενός νήματος/διεργασίας σε συγκεκριμένο σύνολο κόμβων ή επεξεργαστών, εύρεσης της απόστασης μεταξύ δύο κόμβων, μετακίνησης σελίδων μεταξύ κόμβων, καθώς και πληθώρα συναρτήσεων διαχείρισης μασκών που χρησιμοποιούνται για την αναπαράσταση συνόλων από κόμβους ή επεξεργαστές.

[4]Οτιδήποτε ορίζει το λειτουργικό σύστημα ως επεξεργαστή. Στην περίπτωση του Linux Kernel, οποιοδήποτε επεξεργαστικό στοιχείο που μπορεί να εκτελέσει μία διεργασία, όπως για παράδειγμα ένα H/W thread.

3.4 Χρήση τοπολογίας στο OpenMP

Στην έκδοση 4.0 του OpenMP προστέθηκε η δυνατότητα ανάθεσης (binding) των νημάτων σε σύνολα από συγκεκριμένους επεξεργαστές, τα λεγόμενα OpenMP places. Η ανάθεση αυτή γίνεται βάσει των διάφορων διαθέσιμων πολιτικών που περιγράφονται στις προδιαγραφές του OpenMP και είναι γνωστές ως processor binding policies. Με αυτό τον τρόπο το OpenMP δίνει τη δυνατότητα εκμετάλλευσης της τοπολογίας του υποκείμενου συστήματος ώστε, για τους λόγους που αναφέραμε στην αρχή του κεφαλαίου, να μπορούν να συγγραφούν πιο αποδοτικά παράλληλα προγράμματα.

Σε αυτό το σημείο χρειάζεται να διευκρινιστεί ότι επεξεργαστής (processor) είναι μία αυθαίρετη έννοια που προσδιορίζεται από την εκάστοτε υλοποίηση του OpenMP και αφορά μονάδα του υλικού στην οποία μπορούν να εκτελεστούν ένα ή περισσότερα νήματα. Τυπικά, ένας επεξεργαστής αντιστοιχεί σε ένα H/W thread.

Κατά το binding η εκτέλεση των νημάτων περιορίζεται σε ένα υποσύνολο των διαθέσιμων επεξεργαστών και μπορεί να χρησιμοποιηθεί για την επίτευξη καλύτερων επιδόσεων. Καλύτερες επιδόσεις μπορούμε να πετύχουμε, για παράδειγμα, περιορίζοντας τις μετακινήσεις των νημάτων μεταξύ διαφορετικών επεξεργαστών και άρα μειώνοντας[5] το πλήθος χρονοβόρων διαδικασιών όπως διαχείριση αστοχιών κρυφής μνήμης (cache misses). Επίσης, το binding μπορεί να χρησιμοποιηθεί σε πολύπλοκα συστήματα (π.χ. NUMA) για την καλύτερη κατανομή των νημάτων ανάλογα την παράλληλη εφαρμογή.

Υποθέτοντας ένα σύστημα NUMA, αν η παράλληλη εφαρμογή απαιτεί συνεργασία μεταξύ των νημάτων για την εκτέλεση μίας εργασίας, τότε συμφέρει τα νήματα να τοποθετηθούν κοντά[6] το ένα στο άλλο ώστε να μπορούν να ανταλλάσσουν πιο γρήγορα δεδομένα μέσω της κρυφής και τοπικής μνήμης. Αν τα νήματα τοποθετηθούν σε διαφορετικούς κόμβους NUMA, το επικοινωνιακό φορτίο στο δίκτυο διασύνδεσης θα είναι αυξημένο λόγω του συντονισμού, συγχρονισμού και της μεταξύ τους επικοινωνίας, καθώς και λόγω της κίνησης πο δημιουργεί το πρωτοκόλλο συνοχής των κρυφών μνημών. Σε αντίθετη περίπτωση, αν οι εργασίες που ανατίθενται στα νήματα είναι ανεξάρτητες μεταξύ τους, τότε βολεύει η διασπορά των νημάτων στους κόμβους ώστε να μπορεί το κάθε νήμα να εκμεταλλευτεί τους πόρους

[5] Υπό προϋποθέσεις, όπως για παράδειγμα, κατά τη διάρκεια διακοπής (interupt) να μην εκτελεστεί στον ίδιο επεξεργαστή άλλο νήμα που θα γεμίσει την κρυφή μνήμη με δικά του δεδομένα.

[6] Βάσει της τοπολογίας του συστήματος.

ολόκληρου κόμβου.

3.4.1 OpenMP Places

Όπως αναφέρθηκε ήδη, τα OpenMP places, στα οποία από εδώ και στο εξής θα αναφερόμαστε απλά ως places, είναι σύνολα επεξεργαστών τα οποία χρησιμοποιούνται για την ανάθεση των νημάτων OpenMP σε αυτά βάσει των processor binding policies.

Κάθε place είναι ένα υποσύνολο των διαθέσιμων επεξεργαστών του υποκείμενου συστήματος και κάθε επεξεργαστής αναπαρίσταται μέσω ενός μοναδικού μη αρνητικού ακέραιου αριθμητικού αναγνωριστικού, η σημασία του οποίου καθορίζεται από την εκάστοτε υλοποίηση του OpenMP.

Όλα τα places μαζί αποτελούν το αρχικό place partition, το οποίο είναι η λίστα με τα places που είναι διαθέσιμα στο σύστημα χρόνου εκτέλεσης του OpenMP. Κάθε νήμα διαθέτει δικό του αντίγραφο του place partition και το οποίο ανάλογα την πολιτική ανάθεσης των νημάτων σε επεξεργαστές, μπορεί να είναι υποσύνολο του αρχικού place partition. Με τις πολιτικές ανάθεσης θα ασχοληθούμε στην Υποενότητα 3.4.2.

Η μεταβλητή περιβάλλοντος `OMP_PLACES` χρησιμοποιείται για την αρχικοποίηση του αρχικού place partition. Οι έγκυρες τιμές της `OMP_PLACES` μπορεί να έχουν δύο μορφές:

- Αφηρημένο όνομα (abstract name): Περιγράφει με αφηρημένο τρόπο ένα σύνολο places (π.χ. `cores`).
- Ρητή λίστα (explicit list): Περιγράφει κάθε place με ρητό τρόπο χρησιμοποιώντας μη αρνητικούς ακέραιους αριθμούς.

Η ρητή λίστα, μπορεί να οριστεί ως ένα σύνολο ενός ή περισσότερων places χωρισμένων μεταξύ τους με κόμμα. Κάθε place μπορεί να περιγραφεί είτε ως ένας αριθμός, είτε ως ένα σύνολο αριθμών που περικλείονται από άγκιστρα.

Για τον ορισμό των places μπορούν επίσης να χρησιμοποιηθούν διαστήματα (intervals) μέσω του συμβολισμού `<lower-bound>:<length>:<stride>` για να περιγράψουν την εξής ακολουθία αριθμών:

- `<lower-bound>`
- `<lower-bound>` + `<stride>`

- ...
- <lower-bound> + (<length> -1) * <stride>

Αν η τιμή του <stride> παραλείπεται, τότε θεωρείται ίση με τη μονάδα. Τα διαστήματα μπορούν να χρησιμοποιηθούν και για την περιγραφή ακολουθιών από places.

Ο τελεστής "!" μπορεί να χρησιμοποιηθεί για την εξαίρεση του αριθμού/place που ακολουθεί αμέσως μετά.
Εναλλακτικά, τα ακόλουθα αφηρημένα ονόματα μπορούν να χρησιμοποιηθούν για την περιγραφή των places:

- threads: Κάθε place αντιστοιχεί σε ένα H/W thread.
- cores: Κάθε place αντιστοιχεί σε έναν πυρήνα αποτελούμενος από ένα ή περισσότερα H/W threads.
- ll_caches: Κάθε place αντιστοιχεί σε ένα σύνολο πυρήνων που μοιράζονται το τελευταίο επίπεδο κρυφής μνήμης.
- numa_domains: Κάθε place αντιστοιχεί σε ένα σύνολο πυρήνων των οποίων η πιο κοντινή μνήμη σε αυτούς:
  - είναι η ίδια μνήμη και
  - σε παρόμοια απόσταση από τους πυρήνες.
- sockets: Κάθε place αντιστοιχεί σε μία υποδοχή ολοκληρωμένου κυκλώματος επεξεργαστή (socket).

Τα αφηρημένα ονόματα μπορούν να ακολουθούνται από ένα ακέραιο νούμερο μέσα σε παρενθέσεις για τον προσδιορισμό του πλήθους των places που θα αποτελέσουν το αρχικό place partition (π.χ. threads(8)). Σε περίπτωση που ζητηθούν λιγότερα places σε σχέση με τα διαθέσιμα, η υλοποίηση του OpenMP αποφασίζει ποιά places θα περιληφθούν στο place partition. Αν ζητηθούν περισσότερα places, η υλοποίηση του OpenMP καθορίζει το μέγεθος του place partition, δηλαδή το πλήθος των places που το απαρτίζουν.

Η υλοποίηση του OpenMP καθορίζει την ακριβή σημασία των αφηρημένων ονομάτων, μπορεί να ορίσει και επιπλέον ονόματα.

Οι παρακάτω εντολές export[7] αναθέτουν τιμή στη μεταβλητή OMP_PLACES, ορίζοντας και οι πέντε ακριβώς τα ίδια τέσσερα places:

[7]Σύνταξη συμβατή με το Bourne Again SHell (BASH).

- `export OMP_PLACES=threads`
- `export OMP_PLACES="threads(4)"`
- `export OMP_PLACES="0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15"`
- `export OMP_PLACES="0:4,4:4,8:4,12:4"`
- `export OMP_PLACES="0:4:4:4"`

Η σύνταξη της μεταβλητής περιβάλλοντος OMP_PLACES

Το σύνολο των έγκυρων τιμών που μπορούν να ανατεθούν στη μεταβλητή `OMP_PLACES` σύμφωνα με το OpenMP 5.1 [10] προκύπτουν βάσει των ακόλουθων συντακτικών κανόνων:

```
<list>         |= <p-list> | <aname>
<p-list>       |= <p-interval> | <p-list>,<p-interval>
<p-interval>   |= <place>:<len>:<stride> | <place>:<len> | <place> | !<place>
<place>        |= {<res-list>} | <res>
<res-list>     |= <res-interval> | <res-list>,<res-interval>
<res-interval> |= <res>:<num-places>:<stride> | <res>:<num-places> | <res>
                  | !<res>
<aname>        |= <word>(<num-places>) | <word>
<word>         |= sockets | cores | ll_caches | numa_domains | threads
                  | <implementation-defined abstract name>
<res>          |= non-negative integer
<num-places>   |= positive integer
<stride>       |= integer
<len>          |= positive integer
```

3.4.2 OpenMP Processor Binding Policies

ΚΕΦΑΛΑΙΟ 4

Συγχρονισμός με Barriers

4.1 Τι είναι οι Barriers

Η διπλωματική εργασία περιέχει ν κεφάλαια.

4.2 Barriers στο OpenMP

Η διπλωματική εργασία περιέχει ν κεφάλαια.

4.3 Ο Barrier του OMPi

Η διπλωματική εργασία περιέχει ν κεφάλαια.

4.4 Βελτιώσεις που έγινα στον Barrier του OMPi

Η διπλωματική εργασία περιέχει ν κεφάλαια.

ΚΕΦΑΛΑΙΟ 5

Πειραματική Αξιολόγηση

5.1 Εισαγωγή

Η διπλωματική εργασία περιέχει ν κεφάλαια.

5.2 Περιγραφή Συστημάτων

Η διπλωματική εργασία περιέχει ν κεφάλαια.

5.3 Τοπολογία

Η διπλωματική εργασία περιέχει ν κεφάλαια.

5.4 Barrier

Η διπλωματική εργασία περιέχει ν κεφάλαια.

ΚΕΦΑΛΑΙΟ 6

Συμπεράσματα και Μελλοντική Εργασία

6.1 Ανακεφαλαίωση

Η διπλωματική εργασία περιέχει ν κεφάλαια.

6.2 Μελλοντική Εργασία

Η διπλωματική εργασία περιέχει ν κεφάλαια.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "Cpu db: recording microprocessor history," *Communications of the ACM*, vol. 55, no. 4, pp. 55–63, 2012.

[2] T. Rauber and G. Rünger, *Parallel Programming: for Multicore and Cluster Systems*. Springer Science & Business Media, 2010.

[3] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[4] M. Dobson, P. Gaughen, M. Hohnbaum, and E. Focht, "Linux support for numa hardware," in *Proc. Linux Symposium*, vol. 2003. Citeseer, 2003.

[5] M. J. Bligh, M. Dobson, D. Hart, and G. Huizenga, "Linux on numa systems," in *Proceedings of the Linux Symposium*, vol. 1, 2004, pp. 89–102.

[6] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in hpc applications," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 180–186.

[7] A. Kleen, "A numa api for linux," *Novel Inc*, 2005.

[8] L. Torvalds, "Linux kernel," https://github.com/torvalds/linux.

[9] "numa(3) - linux manual page," https://man7.org/linux/man-pages/man3/numa.3.html.

[10] "Openmp application programming interface v5.1," https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-1.pdf, 2020.

ΠΑΡΑΡΤΗΜΑ Α

Απαιτήσεις Λογισμικού του μεταφραστή OMPi

Για τη μετάφραση του OMPi απαιτούνται τα εξής πακέτα λογισμικού:

- `autoconf`
- `automake1.16` (Debian-based distributions), `automake` (CentOS) v1.16
- `bison`
- `flex`
- `gcc` (ή κάποιος άλλος μεταφραστής για τη γλώσσα C)
- `libtool` v2.4.6
- (Προεραιτικά) `libhwloc-dev` (Debian-based distributions), `hwloc-devel` (CentOS)

Για την υποστήριξη των OpenMP places και binding των νημάτων OpenMP στα places απαιτείται επιπλέον το πακέτο:

- `libhwloc-dev` (Debian-based distributions), `hwloc-devel` (CentOS)

Για την υποστήριξη δενδρικού barrier σε NUMA συστήματα απαιτείται επιπλέον το πακέτο:

- `libnuma-dev` (Debian-based distributions), `numactl-devel` (CentOS)