Υλοποίηση της Διεπαφής OMPT για την Παρακολούθηση Εφαρμογών OpenMP

Γεώργιος Αγρογιάννης

με Α.Μ.: 3176

Διπλωματική Εργασία

Επιβλέπων: Βασίλειος Β. Δημακόπουλος

Ιωάννινα, Φεβρουάριος, 2026

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη μιας διεπαφής για τον μεταφραστή OMPi, η οποία διαθέτει, σε πραγματικό χρόνο, πληροφορίες σχετικά με την κατάσταση των νημάτων και με το περιβάλλον εκτέλεσης εφαρμογών οι οποίες κάνουν χρήση του προτύπου OpenMP. Η διεπαφή αυτή βασίζεται στο OpenMP Tool Interface (OMPT), το οποίο αποτελεί μέρος του επίσημου προτύπου OpenMP από την έκδοση 5.0 και μετά.

Σκοπός ήταν η ανάπτυξη όλων των συναρτήσεων της διεπαφής OMPT και η ενσωμάτωση τους στον μεταφραστή OMPi, ο οποίος έχει υλοποιηθεί και αναπτύσσεται από το Parallel Processing Group, εν συντομία PPG, της Σχολής Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων υπό την αιγίδα και καθοδήγηση του Καθηγητή Βασσίλειου Β. Δημακόπουλου. Είναι ένας ελαφρύς (lightweight) και ανοιχτού κώδικα ερευνητικός μεταφραστής που υλοποιεί το επίσημο πρότυπο του OpenMP και σύστημα χρόνου εκτέλεσης (runtime system) για τη γλώσσα προγραμματισμού C.

Υλοποιήθηκε το μεγαλύτερο μέρος της διεπαφής. Ένα μικρό μέρος των κλήσεων δεν είναι δυνατόν να υλοποιηθεί με την υπάρχουσα δομή του OMPi. Όμως, υπάρχει η υποδομή ώστε με τις κατάλληλες προσθήκες και τροποποιήσεις στη βιβλιοθήκη του χρόνου εκτέλεσης (runtime) του OMPi να μπορούν να προστεθούν χωρίς ιδιαίτερο κόπο.

Η διεπαφή χρησιμοποιείται από εργαλεία (1st/3rd party tools), τα οποία καταγράφουν, μετρούν και παρουσιάζουν την αξιοποίηση και απόδοση των πόρων του συστήματος, που επιτυγχάνεται από την εφαρμογή, κατά το χρόνο της εκτέλεσης της. Τα εργαλεία αποσκοπούν στη βελτιστοποίηση του κώδικα της εφαρμογής βάση του συστήματος και του προβλήματος, που απευθύνονται στην εφαρμογή.

Στη διπλωματική αυτή εργασία έγινε καταγραφή των διαθέσιμων εργαλείων και αξιολογήσαμε την υλοποίησή μας χρησιμοποιώντας ένα από αυτά.

Λέξεις Κλειδιά: Διεπαφή Χαμηλού Επιπέδου, OpenMP, OMPi, OMPT, Εργαλεία

Abstract

This diploma thesis focuses on the development of an interface for the OMPi compiler, which provides real-time information regarding the state of threads and the execution environment of applications that make use of the OpenMP standard. This interface is based on the OpenMP Tool Interface (OMPT), which has been part of the official OpenMP specification since version 5.0.

The goal was the development of all OMPT interface functions and their integration into the OMPi compiler, which is developed by the Parallel Processing Group (PPG) of the School of Computer Engineering and Science at the University of Ioannina, under the supervision of Professor Vasileios B. Dimakopoulos. OMPi is a lightweight, open-source research compiler that implements the official OpenMP specification and a runtime system for the C programming language.

The majority of the interface was implemented. A small subset of the interface functions cannot be implemented with the current structure of OMPi. However, the necessary infrastructure exists so that, once the appropriate additions and modifications are made to the OMPi runtime library, the remaining interface functions can be incorporated without significant effort.

The interface is used by tools (1st/3rd party tools), which record, measure, and present the utilization and performance of system resources achieved during the execution of an application. These tools aim to optimize application code based on the underlying system and the problem domain targeted by the application.

In this diploma thesis, the available tools were surveyed, and our implementation was evaluated using one of them.

Keywords: Low-Level Interface, OpenMP, OMPT, OMPi, 1st Party Tools

Πίνακας περιεχομένων

Κεφάλαιο 1. Εισαγωγή

1.1 Παράλληλα Συστήματα

1.1.1 Εξέλιξη των Η/Υ

Ο πρώτος ηλεκτρονικός υπολογιστής έκανε την εμφάνιση του την δεκαετία του 1940 με το όνομα ENIAC. Ο υπολογιστής αυτός καταλάμβανε μεγάλο χώρο, ήταν θορυβώδης, αποτελούταν από εκατοντάδες λυχνίες που καίγονταν συχνά και κατανάλωνε πολλή ενέργεια. Ήταν όμως ένα μεγάλο βήμα προς τους σημερινούς υπολογιστές.

Έκτοτε ο άνθρωπος έχει δαπανήσει πολλούς πόρους για την βελτιστοποίηση αυτού του εγχειρήματος. Είτε πρόκειται για το μέγεθος, είτε για την κατανάλωση ενέργειας και σημαντικότερα την χωρητικότητα μνήμης και την επεξεργαστική ισχύ ένας απλός σημερινός υπολογιστής είναι τάξεις μεγεθών καλύτερος από τον ENIAC.

Παρατηρώντας κάποιος αυτή την δραματική εξέλιξη των υπολογιστών, θα μπορούσε να ισχυριστεί ότι σε μερικά χρόνια η ταχύτητα των υπολογιστών θα είναι τέτοια ώστε θα μπορούσαν να επιλύσουν οποιοδήποτε υπολογίσιμο πρόβλημα σε σχεδόν μηδενικό χρόνο. Όμως ο ισχυρισμός αυτός φαίνεται να διαψεύδεται αφού έχουμε καταλήξει στο συμπέρασμα ότι η αύξηση της ταχύτητας των επεξεργαστών δεν είναι δυνατόν να συνεχιστεί επ' αορίστου.

Φυσικοί νόμοι, όπως η ταχύτητα των ηλεκτρονίων που δεν μπορεί να ξεπεράσει την ταχύτητα του φωτός, καθώς και περιορισμοί στην πυκνότητα των transistors ανά μονάδα επιφάνειας θέτουν ένα άνω όριο στην αύξηση της ταχύτητας.

Για να ξεπεράσουμε όμως αυτούς τους φυσικούς περιορισμούς, έρχεται στο προσκήνιο το παράλληλο μοντέλο υπολογισμού και η παράλληλη επεξεργασία που αποτελούν εξέλιξη του κλασικού μοντέλου προγραμματισμού/αρχιτεκτονικής von Neumann ή αλλιώς σειριακό.

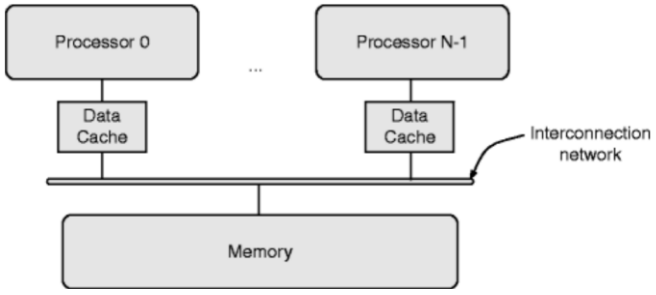
1.1.2 Παράλληλα συστήματα

Ως παράλληλο σύστημα μπορεί να οριστεί ένα σύνολο από πολλές επεξεργαστικές μονάδες (CPUs) που συνεργάζονται για την επίλυση ενός προβλήματος. Συγκεκριμένα ένα παράλληλο σύστημα διαιρεί το αρχικό πρόβλημα σε μικρότερα υποπροβλήματα, τα επιλύει ταυτόχρονα, και συγκεντρώνει τα αποτελέσματα, εξάγοντας το τελικό.

1.1.3 Αρχιτεκτονικές παράλληλων συστημάτων

Ένα παράλληλο σύστημα μπορεί να αποτελείται είτε από πολλούς πυρήνες σε ένα chip είτε από πολλούς διαφορετικούς υπολογιστές.

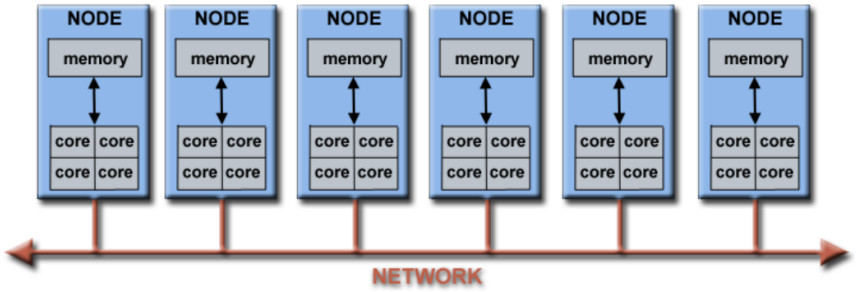
Στην πρώτη περίπτωση αναφερόμαστε στα παράλληλα συστήματα κοινόχρηστης μνήμης. Οι διαφορετικοί επεξεργαστές μοιράζονται κάποιο κοινό χώρο διευθύνσεων μνήμης και επικοινωνούν τροποποιώντας μεταβλητές. Η μνήμη οργανώνεται σε κεντρική θέση, όπως στα συστήματα SMP (symmetric multiprocessors), όπου κάθε επεξεργαστής συνδέεται σε ένα δίαυλο μέσω του οποίου έχει πρόσβαση σε αυτή, όπως απεικονίζεται στο σχήμα 1.1. Όλοι οι επεξεργαστές έχουν τον ίδιο χρόνο προσπέλασης στην κύρια μνήμη και για αυτόν τον λόγο αναφέρονται ως UMA (Uniform Memory Access). Επίσης κάθε επεξεργαστής διαθέτει και τοπική cache μνήμη με αποτέλεσμα να δημιουργούνται προβλήματα με την συνέπειά της. Εναλλακτικά υπάρχει και το μοντέλο NUMA (Non-Uniform Memory Access) όπου κάθε επεξεργαστής είναι συνδεδεμένος σε ιδιωτική μνήμη. Αυτά τα κομμάτια μνήμης όμως συνδυάζονται ώστε να προκύψει ο κοινός χώρος διευθύνσεων. Σε αντίθεση με το μοντέλο UMA, ο χρόνος προσπέλασης της μνήμης διαφέρει για κάθε επεξεργαστή καθώς εξαρτάται από την τοποθέτησή του. Οι πυρήνες ομαδοποιούνται σε κόμβους (nodes) στους οποίους κάθε ένας έχει τον δικό του ελεγκτή μνήμης.



Σχήμα 1.1 Μοντέλο Κοινόχρηστης Μνήμης

Η άλλη κατηγορία είναι τα συστήματα κατανεμημένης μνήμης. Τα συστήματα αυτά αποτελούνται από συστάδες διαφορετικών υπολογιστών (clusters) που διαθέτουν δικιά τους μνήμη και συνδέονται με δίκτυο διασύνδεσης για την ανταλλαγή

μηνυμάτων. Η διασύνδεση αυτή φαίνεται στο σχήμα 1.2. Τα πλεονεκτήματα αυτής της αρχιτεκτονικής είναι ότι κάθε μονάδα μπορεί να εκμεταλλευτεί όλο το διαθέσιμο εύρος ζώνης της τοπικής μνήμης. Επίσης λόγω της απουσίας διαύλου δεν υπάρχει περιορισμός στο μέγεθος του συστήματος αφού το δίκτυο διασύνδεσης υποστηρίζει μεγάλο αριθμό μονάδων. Ακόμα δεν εμφανίζονται προβλήματα συνέπειας cache καθώς κάθε μονάδα είναι υπεύθυνη για την συνέπεια της δικιάς της μνήμης. Παρά όλα αυτά τα πλεονεκτήματα το μεγάλο μειονέκτημα αυτής της αρχιτεκτονικής είναι ο χρόνος που δαπανάται για τις επικοινωνίες. Όταν ένας επεξεργαστής χρειάζεται δεδομένα από έναν άλλο θα πρέπει να γίνει ανταλλαγή μηνυμάτων, διαδικασία που απαιτεί χρόνο για την δημιουργία και αποστολή του μηνύματος καθώς και την λήψη του, που απαιτεί διακοπή της λειτουργίας του δέκτη για την επεξεργασία του μηνύματος.



Σχήμα 1.1 Μοντέλο Κατανεμημένης Μνήμης

1.2 Παράλληλος Προγραμματισμός και OpenMP

1.2.1 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 8 και συμφόρηση του δικτύου.

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).

1.2.2 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 variables) 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.

1.2.3 OpenMP

Για τη διευκόλυνση του προγραμματισμού, έχουν αναπτυχθεί εργαλεία πιο υψηλού επιπέδου όπως το OpenMP (Open Multi-Processing)[1]. 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 7 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 Αντικείμενο της Διπλωματικής

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη του μέρους OMPT της διεπαφής OMP-Tools του προτύπου OpenMP και η ενσωμάτωση του στον ερευνητικό μεταφραστή OMPi. 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 5.0 του OpenMP[1] προστέθηκε η διεπαφή OMP-Tools. Έχει δύο μέρη το ένα (OMPD) χρησιμοποιείται από εξωτερικά εργαλεία (3rd party tools), βιβλιοθήκες που εκτελούνται παράλληλα με την υλοποίηση του OpenMP, συχνά και ανεξάρτητα. Το δεύτερο (OMPT) χρησιμοποιείται από 1st party tools, βιβλιοθήκες που πρέπει να φορτωθούν και να συνδεθούν κατά τη μεταγλώττιση ή την εκτέλεση με την υλοποίηση του OpenMP.

Κατά την εκκίνηση της εκτέλσης, η υλοποίηση της διεπαφής OMPT αναζητά το σύστημα για εγκατεστημένα 1st party tools. Όταν βρεθεί το πρώτο συμβατό εργαλείο η διεπαφή το ενεργοποιεί. Ένα βήμα της ενεργοποίησης είναι η καταχώρηση των συναρτήσεων, που είναι υλοποιημένες από το εργαλείο, στα αντίστοιχα γεγονότα (trigger events) της διεπαφής. Αντίστοιχα στο ίδιο σημείο της ενεργοποίησης γίνεται η καταχώρηση των συναρτήσεων, που παρέχει η διεπαφή στο εργαλείο, καθώς οι συναρτήσεις της διεπαφής που επιτρέπουν στο εργαλείο να αποκτήσει πρόσβαση στους δείκτες των συναρτήσεων. Εφόσον η ενεργοποίηση ήταν επιτυχημένη το εργαλείο και η διεπαφή είναι ενεργά και επικοινωνούν μεταξύ τους καθ' όλη τη διάρκεια της εκτέλεσης του προγράμματος μέσω των συναρτήσεων αυτών, περισσότερες λεπτομέρειες για τα γεγονότα βρίσκονται στις ενότητες 3.1.1, 3.1.4 και 3.2.1 και 4.4. Κατά το χρόνο εκτέλεσης της εφαρμογής αν η διεπαφή είναι ενεργή και το εργαλείο έχει καταχωρήσει μια συνάρτηση για κάποιο γεγονός όταν συμβεί αυτό το γεγονός η διεπαφή είναι υπεύθυνη να καλέσει την καταχωρημένη συνάρτηση του εργαλείου. Έτσι ανταλλάσσονται οι πληροφορίες μεταξύ των εργαλείων και της διεπαφής σε πραγματικό χρόνο. Το εργαλείο μετά το πέρας της εκτέλεσης συγκεντρώνει και παρουσιάζει στον τελικό χρήστη τις μετρήσεις και την αξιολόγηση για το σύστημα και την εφαρμογή. Πιο αναλυτικές λεπτομέρειες για τα εργαλεία πρώτου και τρίτου βαθμού στις ενότητες 3.2 και 2.5.2, αντίστοιχα.

1.4 Οργάνωση του Κειμένου

Η διπλωματική εργασία είναι οργανωμένη με τον εξής τρόπο:

- **Κεφάλαιο 2**: Παρουσίαση και περιγραφή του προτύπου παράλληλου προγραμματισμού OpenMP. Επεξήγηση των οδηγιών, των δομών και των κλήσεων συστήματος που αποτελούν το πρότυπο. Παραδείγματα χρήσης του προτύπου σε μεταφραστές OpenMP. Αναφορά και περιγραφή στις δυνατότητες των εργαλείων και της διεπαφής OMP-Tools που εισήγαγε η έκδοση 5.0 του OpenMP. Αναφορά και επεξήγηση της διεπαφής OMPD και των εξωτερικών εργαλείων 3rd party tools.
- **Κεφάλαιο 3**: Αναφορά και επεξήγηση στις επιμέρους λειτουργίες της διεπαφής OMPT και των 1st party tools και στις δομές που χρησιμοποιούν. Περιγραφή της διαδικασίας διασύνδεσης εργαλείων στη διεπαφή και της χρήσης της διεπαφής από τα εργαλεία μέσω του OMPi.
- **Κεφάλαιο 4**: Παρουσίαση και περιγραφή του παραλληλοποιητικού μεταφραστή OMPi. Περιγραφή της υλοποίησης της διεπαφής εργαλείων OMPT για τον μεταφραστή OMPi. Επεξήγηση του τρόπου χρήσης από τον μεταφραστή OMPi.
- **Κεφάλαιο 5**: Επεξήγηση της λειτουργίας, χρήσης και υποστήριξης 1st party tools, από τη διεπαφή OMPT του μεταφραστή OMPi. Ανάλυση παραδείγματος εκτέλεσης εφαρμογής OpenMP με το Caliper.
- **Κεφάλαιο 6**: Σύνοψη διπλωματικής εργασίας, περιγραφή και αναφορά σε συμπεράσματα από τα παραδείγματα χρήσης 1st party tools. Αναφορά σε μελλοντικές υλοποιήσεις και βελτιστοποιήσεις για την υλοποίηση της διεπαφής OMP-Tools στον OMPi.

Κεφάλαιο 2. Εισαγωγή σε OpenMP & OMP-Tools

2.1 OpenMP

Όπως αναφέρθηκε ήδη στην ενότητα 1.2.3, 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): Απευθύνονται στον 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 που 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 να καθορίσουν διάφορα χαρακτηριστικά 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 τιμές μπορούν να τροποποιηθούν σε χρόνο εκτέλεσης χρησιμοποιώντας τις διαθέσιμες ρουτίνες 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 (`accelerators`). 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 υπόψιν 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 συστημάτων κοινόχρηστης μνήμης καθώς και στο προγραμματιστικό μοντέλο fork-join που συναντάται στις διεργασίες.

Στο μοντέλο fork-join, 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 η εξής:

#pragma omp *όνομα_οδηγίας [φράση[[,] φράση] …] <νέα-γραμμή>*

Γενικά για όλες τις οδηγίες ισχύουν οι εξής κανόνες:

1. Το όνομα της οδηγίας είναι υποχρεωτικό μετά το **`#pragma omp.`**
2. Το όνομα της οδηγίας είναι *case-sensitive*.

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 συνδυασμό 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 Παράλληλες Περιοχές

Ως περιοχή παραλληλίας ορίζεται το τμήμα κώδικα που εκτελείται παράλληλα από μία ομάδα νημάτων. Η περιοχή δηλώνεται μέσω της οδηγίας **`parallel`** και περιλαμβάνει ένα δομημένο τμήμα, που μπορεί να είναι είτε μπλοκ κώδικα μέσα σε άγκιστρα είτε μία μοναδική εντολή χωρίς άγκιστρα. Η γενική μορφή της οδηγίας **`parallel`** είναι η εξής:

#pragma omp *`parallel`* *[φράση[[,] φράση] …] νέα-γραμμή*

δομημένο τμήμα

Κατά την εκτέλεση, το κύριο νήμα δημιουργεί την ομάδα νημάτων που εκτελεί το δομημένο τμήμα. Τα νήματα μπορεί να τερματίζουν σε διαφορετικούς χρόνους, γι' αυτό στο τέλος της περιοχής υπάρχει ένα φράγμα (**`barrier`**) που συγχρονίζει τα νήματα, ώστε όλα να ολοκληρώσουν πριν συνεχίσει το κύριο πρόγραμμα. Ο χρήστης μπορεί να καθορίσει το μέγεθος της ομάδας νημάτων με τρεις βασικές μεθόδους:

1. Κλήση της συνάρτησης **omp_set_num_threads()** εκτός περιοχής παραλληλίας.
2. Ορισμός της μεταβλητής περιβάλλοντος **OMP_NUM_THREADS** πριν την εκτέλεση.
3. Χρήση της φράσης num_threads() στην οδηγία parallel με τον αριθμό των νημάτων που επιθυμεί.

2.3.3 Διαμοιρασμός εργασίας (Worksharing Regions)

Ο διαμοιρασμός της εργασίας στα νήματα μιας ομάδας αποτελεί βασική λειτουργία του OpenMP, επιτρέποντας την αποτελεσματική κατανομή του φόρτου εργασίας, όπως για παράδειγμα σε επαναληπτικούς βρόχους μέσα σε μια περιοχή παραλληλίας. Για το σκοπό αυτό, το OpenMP υποστηρίζει τις περιοχές διαμοιρασμού εργασίας, που παρέχουν στον προγραμματιστή ειδικές οδηγίες για να κατανείμει το υπολογιστικό φορτίο σε όλα τα νήματα της ομάδας. Αν και αυτές οι περιοχές μπορούν να χρησιμοποιηθούν και εκτός περιοχών παραλληλίας, η χρησιμότητά τους είναι μεγαλύτερη όταν εφαρμόζονται μέσα σε αυτές.

Όπως και στις περιοχές παραλληλίας, στο τέλος κάθε περιοχής διαμοιρασμού εργασίας υπάρχει ενσωματωμένο φράγμα (**barrier**) που συγχρονίζει τα νήματα, διασφαλίζοντας ότι όλα ολοκληρώνουν πριν συνεχιστεί η εκτέλεση. Ωστόσο, η χρήση της φράσης **nowait** στην οδηγία επιτρέπει την παράλειψη του φράγματος, προσφέροντας ευελιξία και βελτιστοποίηση σε κατάλληλες περιπτώσεις.

Υπάρχουν τρεις βασικοί τύποι περιοχών διαμοιρασμού εργασίας στο **OpenMP**, καθένας ορισμένος από συγκεκριμένες οδηγίες, που βοηθούν στην ορθή και αποδοτική κατανομή του φόρτου σε παράλληλα εκτελούμενα τμήματα του κώδικα.

1. Οδηγία **for**: Χρησιμοποιείται για τον καταμερισμό του φόρτου ενός επαναληπτικού βρόχου for. Έχει την εξής γενική μορφή:

   #pragma omp for *[φράση[, φράση] …]*

   for *(αρχικοποίηση; συνθήκη; βήμα) {*

   …

   }

2. Οδηγία **sections**: Επιτρέπει την παραλληλοποίηση ανεξάρτητων εργασιών εντός περιοχής παραλληλίας. Περιλαμβάνει διαδοχικές δηλώσεις #pragma omp section με δομημένα τμήματα κώδικα. Έχει την εξής γενική μορφή:

```
#pragma omp sections [φράση[, φράση] … ]

    #pragma omp section

        δομημένο τμήμα
```

3. Οδηγίας **single**: Επιτρέπει την εκτέλεση ενός δομημένου τμήματος μόνο από το πρώτο νήμα που το συναντά. Τα υπόλοιπα νήματα περιμένουν έως την ολοκλήρωση, για συγχρονισμό. Παραλλαγή: master, όπου το δομημένο τμήμα εκτελείται μόνο από το κύριο νήμα και δεν υπονοείται φράγμα συγχρονισμού. Έχει την εξής γενική μορφή:

```
#pragma omp single | master [φράση[, φράση] … ]

    δομημένο τμήμα
```

2.3.4 Συγχρονισμός Νημάτων στο OpenMP

Οι οδηγίες συγχρονισμού στο OpenMP εξασφαλίζουν τη σωστή και συνεπή εκτέλεση των νημάτων κατά την παράλληλη επεξεργασία. Οι βασικότερες είναι οι εξής:

1. Οδηγία **atomic**: Διασφαλίζει ότι μια πράξη σε μια μεταβλητή εκτελείται αδιαίρετα, χωρίς παρέμβαση άλλων νημάτων.
2. Οδηγία **barrier**: Φράγμα όπου όλα τα νήματα περιμένουν μέχρι να φτάσουν όλα, ώστε να συνεχίσουν ταυτόχρονα με ενημερωμένες κοινόχρηστες μεταβλητές.
3. Οδηγία **critical**: Ορίζει κρίσιμη περιοχή κώδικα που εκτελείται μόνο από ένα νήμα κάθε φορά, με τα υπόλοιπα να περιμένουν.
4. Οδηγία **flush**: συγχρονίζει τη μνήμη μεταξύ νημάτων, εξασφαλίζοντας ότι όλες οι αλλαγές σε κοινόχρηστα δεδομένα είναι ορατές σε όλα τα νήματα.
5. Οδηγία **ordered**: Χρησιμοποιείται για την σειριοποίηση της εκτέλεσης ενός τμήματος κώδικα εντός μιας παράλληλης περιοχής. Συνήθως τοποθετείται στο εσωτερικό ενός βρόχου επανάληψης, όταν χρειάζεται να τηρηθεί συγκεκριμένη σειρά στην εκτέλεση των επαναλήψεων.

Αυτές οι οδηγίες είναι απαραίτητες για την αποφυγή συγκρούσεων και την ορθή συνεργασία των νημάτων στο παράλληλο περιβάλλον.

2.3.5 Φράσεις οδηγιών

Οι φράσεις οδηγιών στο OpenMP τοποθετούνται μετά το όνομα της οδηγίας και καθορίζουν τον τρόπο εκτέλεσής της, καθώς και τη συμπεριφορά των νημάτων πριν, κατά τη διάρκεια και μετά το δομημένο τμήμα κώδικα. Βασικές φράσεις είναι:

- **if(**<συνθήκη>**)**: Εκτελεί την οδηγία μόνο αν η συνθήκη είναι αληθής.
- **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**.
- **nowait**: Αποφεύγει το φράγμα συγχρονισμού που υπονοείται μετά την οδηγία, επιτρέποντας στα νήματα να συνεχίσουν χωρίς αναμονή.
- **num_threads()**: Καθορίζει τον αριθμό των νημάτων που θα χρησιμοποιηθούν σε μια παράλληλη περιοχή.
- **schedule(<**τύπος>**[,** <μέγεθος κόκκου>**])**: Ρυθμίζει τον τρόπο κατανομής επαναλήψεων βρόχου στα νήματα, με πολιτικές όπως `static` (στατικός καταμερισμός), `dynamic` (δυναμικός) και `guided` (με μειούμενο μέγεθος τμημάτων). Υπάρχουν τρεις πολιτικές διαμοιρασμού των επαναλήψεων του βρόχου στους επεξεργαστές ή/και πυρήνες: η **static** που διαμοιράζει στατικά τον βρόχο σε τμήματα σταθερού μεγέθους ή ανά νήμα, η `dynamic` όπου τα διαθέσιμα νήματα αναλαμβάνουν δυναμικά τμήματα εργασίας, και η **guided** που λειτουργεί παρόμοια με τη `dynamic` αλλά μειώνει εκθετικά το μέγεθος των τμημάτων κατά την εκτέλεση.
- **reduction(**<πράξη> **:** <λίστα μεταβλητών>**)**: Επιτρέπει εκτέλεση αθροιστικών πράξεων σε κοινόχρηστες μεταβλητές, εξασφαλίζοντας τη συνέπεια των δεδομένων χωρίς επιπλέον κώδικα από τον προγραμματιστή.

Αυτές οι φράσεις προσφέρουν ευελιξία και έλεγχο στην παράλληλη εκτέλεση.

2.3.6 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.

2.3.7 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[1]:

- `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

[1] Διαθέσιμοι στο: https://openmp.org/resources/openmp-compilers-tools/

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.
- OMPi Research Compiler: Ο OMPi είναι ερευνητικός μεταφραστής ανοιχτού κώδικα για τη γλώσσα C που επιτρέπει τη χρήση διάφορων βιβλιοθηκών νημάτων και συσκευών. Υποστηρίζει πλήρως την εκτέλεση κώδικα σε συσκευές βάσει του OpenMP 4.5, καθώς επίσης και υποσύνολο των λειτουργιών του 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 υποστήριξη αφορά προϊόντα system on a chip (SoC) της Texas Instruments.

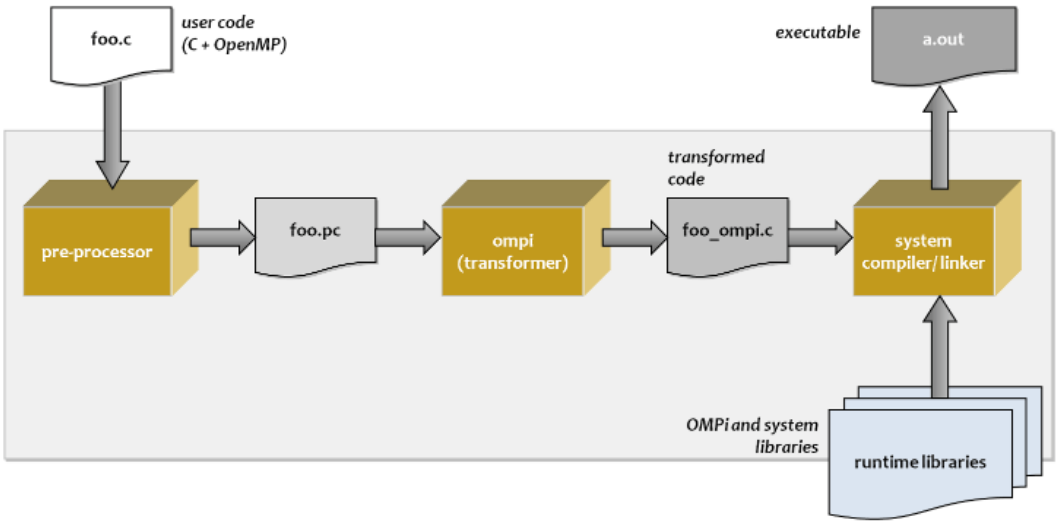
2.4.1 Ο μεταφραστής OMPi

Ο μεταφραστής OMPi[2] 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).

2.4.1.1 Διαδικασία μετάφρασης προγραμμάτων χρήστη

Η πλήρης διαδικασία μετάφρασης που ακολουθεί ο OMPi φαίνεται στο Σχήμα 2.1. 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.



Σχήμα 2.1: Η διαδικασία μετάφρασης του μεταφραστή OMPi.

2.4.1.2 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 (`OMPi RunTime`) και τις βιβλιοθήκες οντοτήτων εκτέλεσης (`EELIBs - Execution Entity LIBraries`)[3].

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.

2.5 OMP-Tools

Η διεπαφή OMP-Tools είναι μια τυποποιημένη διεπαφή προσανατολισμένη στην υποστήριξη εργαλείων ανάλυσης, παρακολούθησης και αποσφαλμάτωσης για εφαρμογές που χρησιμοποιούν το πρότυπο OpenMP. Στόχος της είναι να παρέχει ένα ενιαίο και φορητό σημείο αλληλεπίδρασης μεταξύ OpenMP runtime και των εργαλείων (1st/3rd party tools), χωρίς να απαιτείται τροποποίηση του πηγαίου κώδικα του χρήστη, τα εργαλεία αναλύονται περισσότερο στις ενότητες 3.4 και 2.5.2, αντίστοιχα. Μέσω της διεπαφής αυτής, τα εργαλεία αυτά αποκτούν πρόσβαση σε εσωτερικές πληροφορίες του OpenMP runtime, όπως η δομή και η κατάσταση των νημάτων, η διαχείριση των παράλληλων και task περιοχών (regions), οι μηχανισμοί συγχρονισμού, καθώς και οι δραστηριότητες που σχετίζονται με tasks, schedulers και συγχρονισμούς (synchronizations). Η διεπαφή OMP-Tools βασίζεται σε ένα σύστημα callbacks και δομών metadata που επιτρέπουν την ακριβή καταγραφή γεγονότων εκτέλεσης σε πραγματικό χρόνο, διατηρώντας παράλληλα χαμηλό overhead ώστε να είναι πρακτική η χρήση της σε περιβάλλοντα υψηλής απόδοσης.

Αποτελείται από ένα αρχείο κεφαλίδας (header file) το οποίο διαχωρίζεται σε δύο επιμέρους υποσυστήματα που εξυπηρετούν διαφορετικές κατηγορίες εργαλείων. Το πρώτο μέρος **OMPT (OpenMP Tool Interface)**, επικεντρώνεται στην παρακολούθηση κατά το χρόνο εκτέλεσης μέσω μηχανισμών callbacks καθώς είναι το αντικείμενο της διπλωματικής εργασίας, θα αναλυθεί εκτενώς στο κεφάλαιο 3. Το δεύτερο μέρος **OMPD (OpenMP Debugging Interface)** επικεντρώνεται στη δυναμική αποσφαλμάτωση και επιθεώρηση (debugging) κατά το χρόνο εκτέλεσης **OpenMP** προγραμμάτων ανεξάρτητα από την εκάστοτε υλοποίηση του παράλληλου προγράμματος, κάτι που το καθιστά βασικό εργαλείο για τον διαφανή και φορητό έλεγχο παράλληλων εφαρμογών. Και για τα δύο υποσυστήματα υπάρχουν αντίστοιχες

μεταβλητές περιβάλλοντος για την ενεργοποίηση/απενεργοποίηση τους αντίστοιχα, ώστε να αποφεύγεται το ανεπιθύμητο overhead όταν δεν υπάρχει κάποιο ενεργό εργαλείο που χρησιμοποιεί το αντίστοιχο υποσύστημα.

2.5.1 OMPD

Το **OMPD (OpenMP Debugging Interface)** αποτελεί το τμήμα της διεπαφής **OMP-Tools** που εστιάζει στη διευκόλυνση του εντοπισμού και της ανάλυσης σφαλμάτων σε προγράμματα που αξιοποιούν το πρότυπο OpenMP. Ορίζει μια τυποποιημένη διεπαφή που επιτρέπει στους αποσφαλματωτές, race/leak detectors, memory profilers και άλλα εργαλεία, να έχουν πρόσβαση σε κρίσιμες πληροφορίες που αφορούν την εσωτερική κατάσταση, κατά το χρόνο εκτέλεσης του παράλληλου προγράμματος, χωρίς να απαιτείται άμεση παρέμβαση στον κώδικα του προγράμματος.

Σε αντίθεση με το **OMPT**, το οποίο προσανατολίζεται στη συλλογή γεγονότων και δεδομένων κατά το χρόνο εκτέλεσης, το **OMPD** εστιάζει στη λεπτομερή παρακολούθηση της κατάστασης των νημάτων, των tasks και των συγχρονιστικών δομών, με τρόπο ανεξάρτητο από την υλοποίηση του runtime. Καθιστώντας δυνατή την απεικόνιση της ιεραρχίας των tasks, την παρακολούθηση των σημείων εισόδου σε περιοχές παράλληλου κώδικα και τη χαρτογράφηση της κατάστασης των νημάτων, παρέχοντας στο εκάστοτε εργαλείο φορητή και συνεπή πρόσβαση στις ίδιες πληροφορίες, ανεξαρτήτως συστήματος ή εκδόσεως runtime. Επίσης, παρέχει συναρτήσεις ανάγνωσης/ερώτησης της κατάστασης (threads, parallel/task regions, ICVs, κ.λπ.), callback-συμβάσεις που το εργαλείο πρέπει να υλοποιήσει και μερικά ειδικά σύμβολα που βοηθούν το εργαλείο να εντοπίσει την σωστή βιβλιοθήκη OMPD για την τρέχουσα υλοποίηση. Η μεταβλητή περιβάλλοντος **OMP_DEBUG**, ενεργοποιεί το OMPD.

2.5.1.1 Ενεργοποίηση OMPD runtime

Το **OMPD** συνδέεται με εξωτερικά εργαλεία (3rd party tools). Κάθε εργαλείο είναι μια ξεχωριστή βιβλιοθήκη και εκτελείται ως μια διεργασία, ανεξάρτητη από την εφαρμογή που παρακολουθεί/αναλύει το εργαλείο. Το εξωτερικό εργαλείο συνήθως μπορεί να εκτελέσει την εφαρμογή ή να προσαρτηθεί σε μια ήδη υπάρχουσα διεργασία. Ώστε να συγκρατεί και να εκθέτει τις απαραίτητες πληροφορίες για την εκτέλεση του προγράμματος προς τα εργαλεία, η βιβλιοθήκη του OpenMP runtime συλλέγει επιπλέον πληροφορίες μόνο όταν έχει ενεργοποιηθεί η σχετική υποστήριξη. Από το πρότυπο OpenMP το OMPD ενεργοποιείται όταν η μεταβλητή περιβάλλοντος OMP_DEBUG είναι

ορισμένη ως enabled. Η ενεργοποίηση αυτή μπορεί να προκαλέσει επιπλέον κόστος σε μνήμη/χρόνο εκτέλεσης, οπότε συνηθίζεται να ενεργοποιείται μόνο για sessions debugging/ανάλυσης. Πρακτικά εργαλεία, όπως TotalView και άλλοι απασφαλματωτές, απαιτούν ή συστήνουν την ρύθμιση αυτής της μεταβλητής πριν την εκτέλεση.

Η ενεργοποίηση του OMPD πραγματοποιείται μέσω της φόρτωσης ειδικών συμβόλων και calbacks που εκτίθενται από το OpenMP runtime. Ο αποσφαλματωτής, κατά την αρχικοποίηση της συνεδρίας του, αναζητά τη βιβλιοθήκη που περιέχει την υλοποίηση της διεπαφής OMPD (API) και συνδέεται με αυτήν μέσω προκαθορισμένων σημείων εισόδου (entry points). Τα βασικά στάδια είναι:

- **Δημιουργία συνεδρίας OMPD** (ompd_dll_entry), όπου καθιερώνεται το πλαίσιο επικοινωνίας με το runtime.
- **Προσάρτηση σε διεργασία** (attach) ή **εκκίνηση με παρακολούθηση** (launch), ανάλογα με το αν το πρόγραμμα εκτελείται ήδη.
- **Φόρτωση συμβολικών πινάκων και δομών του runtime**, ώστε να είναι διαθέσιμα στον αποσφαλματωτή όλα τα απαραίτητα σύμβολα για πρόσβαση στα εσωτερικά δεδομένα.

Η διαδικασία ενεργοποίησης είναι σχεδιασμένη ώστε να μην απαιτεί αλλαγές στον πηγαίο κώδικα της εφαρμογής, αλλά να βασίζεται αποκλειστικά στη συνεργασία αποσφαλματωτή–runtime.

2.5.1.2 Διαχείριση μνήμης για το OMPD

Η πρόσβαση στη μνήμη αποτελεί κρίσιμο τμήμα της διεπαφής OMPD, καθώς τα δεδομένα που περιγράφουν την κατάσταση του OpenMP runtime (π.χ. πίνακες νημάτων, στοίβες tasks, σηματοφόροι συγχρονισμού) βρίσκονται εντός των εσωτερικών δομών της βιβλιοθήκης runtime. Για το σκοπό αυτό, το OMPD καθορίζει ένα στρώμα αφαίρεσης που διαχωρίζει τον αποσφαλματωτή από τον άμεσο χειρισμό της μνήμης. Τα κύρια χαρακτηριστικά είναι:

- **Λειτουργίες ανάγνωσης μνήμης**: Ο αποσφαλματωτής αποκτά αντίγραφα από συγκεκριμένες περιοχές μνήμης μέσω συναρτήσεων όπως ompd_read_memory, χωρίς να έχει άμεση εγγραφή ή τροποποίηση.
- **Συνεπής απεικόνιση**: Το OMPD διασφαλίζει ότι οι αναγνώσεις μνήμης αποδίδουν συνεκτικά στιγμιότυπα της κατάστασης, ακόμα και όταν πολλαπλά νήματα βρίσκονται σε ταυτόχρονη εξέλιξη.

- **Πρόσβαση σε runtime handles**: Αντί ο αποσφαλματωτής να διαχειρίζεται ωμές διευθύνσεις, το OMPD ορίζει συναρτήσεις διαχείρισης, αλλιώς handles (π.χ. ompd_thread_handle_t, ompd_task_handle_t) που λειτουργούν ως αφαιρετικά αντικείμενα. Μέσω αυτών, ο αποσφαλματωτής μπορεί να ζητήσει πληροφορίες για ένα task ή ένα νήμα, χωρίς να γνωρίζει την πραγματική διάταξη της μνήμης στο runtime.
- **Ασφαλής αποδέσμευση**: Ο μηχανισμός διαχείρισης φροντίζει για την ασφαλή αποδέσμευση των handles και την αποτροπή διαρροών πόρων, ώστε η συνεδρία αποσφαλμάτωσης να παραμένει ελαφριά και σταθερή.

Με τον τρόπο αυτό, το OMPD παρέχει στον αποσφαλματωτή πρόσβαση σε κρίσιμες πληροφορίες όπως η τοπολογία των νημάτων, οι τρέχουσες στοίβες κλήσεων και τα ενεργά tasks, χωρίς να εκθέτει τις λεπτομέρειες της εσωτερικής υλοποίησης του runtime. Για να επιτευχθεί αυτό το runtime πρέπει να εκθέτει καλά ορισμένα callbacks για όλα τα σημεία όπου αλλάζει η κατάσταση της μνήμης, δηλαδή συγκεκριμένα υλοποιημένες κλήσεις **alloc** και **free**, που ακολουθούν το πρότυπο OpenMP, όπως **ompd_callback_memory_alloc_fn** και **ompd_callback_memory_free_fn**.

2.5.2 3rd Party Tools

Τα εξωτερικά εργαλεία (3rd party tools), που αξιοποιούν τη διεπαφή OMPD, αποτελούν το πρακτικό μέσο με το οποίο οι δυνατότητες αποσφαλμάτωσης και παρακολούθησης του runtime γίνονται άμεσα διαθέσιμες στον τελικό χρήστη. Μέσω της διασύνδεσής τους με το OMPD, εργαλεία όπως debuggers, analyzers, leak/race detectors και profilers αποκτούν πρόσβαση σε συνεπή και τυποποιημένη πληροφόρηση σχετικά με την εκτέλεση προγραμμάτων OpenMP. Με τον τρόπο αυτό καθίσταται εφικτή η απομόνωση σφαλμάτων, η λεπτομερής μελέτη της χρήσης πόρων και η βελτιστοποίηση της απόδοσης, χωρίς να απαιτείται τροποποίηση του ίδιου του προγράμματος.

2.5.2.1 Αρχικοποίηση διεπαφής με OMPD

Τα εξωτερικά εργαλεία συνδέονται στο OMPD μέσω της δημόσιας συνάρτησης που εκθέτει το runtime—συνήθως με έναν συνδυασμό δυναμικής φόρτωσης και handshake: εντοπισμός συμβατής βιβλιοθήκης, κλήση initialization routine, διαπραγμάτευση έκδοσης πρωτοκόλλου, και καταχώρηση callbacks/συνδρομών για τα γεγονότα ενδιαφέροντος. Η αρχικοποίηση περιλαμβάνει επίσης ρύθμιση φίλτρων

(π.χ. μόνο allocation events ή μόνο thread events), επιλογή τρόπου επικοινωνίας (in-process callbacks, socket για remote debugging) και κατάρτιση των ACLs/permissions που απαιτούνται. Κατά την αρχικοποίηση, τα εργαλεία καλό είναι να ζητούν ελάχιστα στοιχεία και να προσαρμόζουν το επίπεδο λεπτομέρειας αναλόγως του διαθέσιμου overhead.

2.5.2.2 Παραδειγματα χρήσης εργαλείων

Ένα χαρακτηριστικό παράδειγμα χρήσης του OMPD σε συνδυασμό με τον **GNU Debugger (gdb)** είναι η αποσφαλμάτωση μιας παράλληλης εφαρμογής OpenMP, όπου ο debugger εκμεταλλεύεται τη διεπαφή για να έχει ορατότητα στην εσωτερική κατάσταση του runtime. Ακολουθεί ένα απλό παράδειγμα εκτέλεσης μέσω τερματικού:

```
$ gcc -g -fopenmp -o omp_app omp_app.c
$ gdb omp_app
(gdb) run
Starting program: /home/cs03176/sandbox/report_ss/omp_app
[New Thread 0x7ffff739a700 (LWP 1394511)]
[New Thread 0x7ffff6b99700 (LWP 1394512)]
[New Thread 0x7ffff6398700 (LWP 1394513)]
Thread 3 is in the parallel region.
Thread 3 starting work.
Thread 0 is in the parallel region.
Thread 0 starting work.
Thread 1 is in the parallel region.
Thread 1 starting work.
Thread 2 is in the parallel region.
Thread 2 starting work.
Thread 3 finishing work.
Thread 3 starting work.
Thread 0 finishing work.
Thread 0 starting work.
Thread 1 finishing work.
Thread 1 starting work.
Thread 2 finishing work.
Thread 2 starting work.
Thread 3 finishing work.
Thread 1 finishing work.
Thread 1 starting work.
Thread 2 finishing work.
Thread 0 finishing work.
Thread 0 starting work.
Thread 1 finishing work.
Thread 0 finishing work.
Parallel region finished.
[Thread 0x7ffff6398700 (LWP 1394513) exited]
[Thread 0x7ffff6b99700 (LWP 1394512) exited]
[Thread 0x7ffff739a700 (LWP 1394511) exited]
[Inferior 1 (process 1394507) exited normally]
```

Τερματικό 2.1 Αρχή παραδείγματος εκτέλεσης GDB με OMPD

Ο debugger, μέσω του OMPD, μπορεί να εκτελέσει εντολές που εκθέτουν την κατάσταση του OpenMP runtime, όπως:

- **ompd-threads**: εμφάνιση όλων των OpenMP threads και της κατάστασής τους.
- **ompd-tasks**: λίστα ενεργών tasks με ταυτοποίηση ιδιοκτήτη και σχέσεις εξάρτησης.
- **ompd-state**: έλεγχος σημείων συγχρονισμού (barriers,critical sections).

Με αυτόν τον τρόπο, το gdb αξιοποιεί το OMPD για να προσφέρει στον προγραμματιστή πληροφορίες που διαφορετικά δεν θα ήταν διαθέσιμες μέσω των κλασικών μηχανισμών αποσφαλμάτωσης, διευκολύνοντας τον εντοπισμό αδιεξόδων, μη ισορροπημένης φόρτωσης ή σφαλμάτων συγχρονισμού. Άλλα παραδείγματα χρήσης είναι σε εφαρμογές όπως:

- **Leak detector**: εγγραφή στα allocation/free events, καταχώριση metadata για κάθε allocation, και παραγωγή αναφοράς κατά την έξοδο ή σε snapshot. Χρήση per-thread caches για επιτάχυνση και batch reporting για μείωση overhead.
- **Race detector**: συνδυασμός memory access instrumentation (όπου δυνατό) με OMPD πληροφορία για συγχρονισμούς (barriers, critical regions, task dependencies) ώστε να περιορίσει false positives. Χρειάζεται tight integration με μηχανισμούς παρακολούθησης μνήμης και δυνατότητα να θέτει watchpoints σε περιοχές με υποψία σύγκρουσης.
- **Profiler / memory profiler**: sampling allocations/usage μέσω OMPD events για στατιστική ανάλυση χρήσης μνήμης ανά σύναρξη, thread ή task, με γραφικά timelines και heatmaps.
- **Offload inspector**: αναγνώριση mappings host↔device, επαλήθευση consistency μεταφορών και προβολή backtraces για allocations στο device.

Κεφάλαιο 3. OMPT & 1st Party Tools

3.1 OMPT

Η διεπαφή **OMPT (OpenMP Tool Interface)** αποτελεί ένα από τα δύο βασικά υποσυστήματα του προτύπου **OpenMP-Tools**, το οποίο αρχικά μελετήθηκε σε ερευνητικό επίπεδο[4] και αργότερα εισήχθη στις προδιαγραφές του προτύπου **OpenMP**, από την έκδοση 5.0 και ύστερα, προκειμένου να διευκολύνει την ανάπτυξη 1st party tools παρακολούθησης, ανάλυσης και διάγνωσης προβλημάτων κλιμάκωσης της εκτέλεσης εφαρμογών OpenMP. Σε αντίθεση με την παραδοσιακή προσέγγιση που βασίζεται σε μηχανισμούς ανάλυσης μέσω βιβλιοθηκών ή επεξεργασίας ιχνών (traces) μετά το πέρας της εκτέλεσης, το **OMPT** παρέχει πρόσβαση σε λεπτομερή γεγονότα του χρόνου εκτέλεσης **σε πραγματικό χρόνο**. Με αυτόν τον τρόπο, καθίσταται δυνατή η δημιουργία εργαλείων profiling, tracing και task/thread analysis, που μπορούν να λειτουργούν δυναμικά, χωρίς τροποποίηση του πηγαίου κώδικα του χρήστη.

Η λειτουργία του **OMPT** βασίζεται σε ένα **σύστημα callbacks**, μέσω του οποίου το **OpenMP runtime** ειδοποιεί το εργαλείο για την εμφάνιση συγκεκριμένων γεγονότων, όπως δημιουργία/τερματισμός μιας παράλληλης περιοχής ή εκκίνηση/ολοκλήρωση ενός task. Η προσέγγιση αυτή προσφέρει υψηλό βαθμό επεκτασιμότητας και φορητότητας, καθώς το ίδιο εργαλείο μπορεί να χρησιμοποιηθεί με διαφορετικές υλοποιήσεις OpenMP, αρκεί να υποστηρίζουν την ίδια έκδοση του προτύπου OpenMP και της διεπαφής OMPT με αυτή που υποστηρίζει η βιβλιοθήκη του runtime. Επιπλέον, η χρήση ενός ενιαίου API μειώνει σημαντικά την ανάγκη για εσωτερική γνώση της δομής του εκτελέσιμου προγράμματος ή της υλοποίησης της βιβλιοθήκης του runtime, διευκολύνοντας την ανάπτυξη ανεξάρτητων εργαλείων παρακολούθησης. Έτσι το OMPT καθιστά δυνατή τη δημιουργία εργαλείων profiling και tracing που μπορούν να λειτουργούν δυναμικά, χωρίς τροποποίηση του πηγαίου

κώδικα της εφαρμογής. Σε αυτό το κεφάλαιο θα αναλυθούν επεξηγηματικά οι διαδικασίες που λαμβάνουν χώρα για τη λειτουργία και συνεργασία της διεπαφής OMPT και του εργαλείου, όπως παρέχονται από τις προδιαγραφές του προτύπου OpenMP. Αναλύονται σε μεγαλύτερο βαθμό στο κεφάλαιο 4.

3.1.1 Αρχιτεκτονική και βασικές έννοιες

Η αρχιτεκτονική του OMPT είναι σχεδιασμένη ώστε να επιτρέπει την αποδοτική και χαμηλού κόστους αλληλεπίδραση μεταξύ του OpenMP runtime και του εργαλείου. Η επικοινωνία αυτή βασίζεται σε σαφώς καθορισμένα στάδια αρχικοποίησης, καταχώρησης callbacks και ανταλλαγής δεδομένων κατά τη διάρκεια της εκτέλεσης.

Κεντρικός στόχος του σχεδιασμού είναι η ελαχιστοποίηση του overhead, ώστε η ενεργοποίηση και εκτέλεση/χρήση της υλοποίησης της διεπαφής του OMPT να μην επηρεάζει σημαντικά τη συμπεριφορά του προγράμματος, επιτρέποντας ταυτόχρονα στο εργαλείο να συλλέγει επαρκείς πληροφορίες για ανάλυση επιδόσεων, εντοπισμό συμφόρησης (bottlenecks) και μελέτη του παραλληλισμού.

3.1.1.1 Callbacks και γεγονότα

Η επικοινωνία μεταξύ της βιβλιοθήκης runtime και του εργαλείου, κατά το χρόνο εκτέλεσης, υλοποιείται μέσω ενός συνόλου **callbacks**, δηλαδή συναρτήσεις που το εργαλείο υλοποιεί και καταχωρεί , με το αντίστοιχο γεγονός, στη βιβλιοθήκη χρόνου εκτέλεσης (runtime library), η οποία θα καλέσει το **καταχωρημένο callback** κατά την εκτέλεση της εφαρμογής κάθε φορά που λαμβάνει χώρα το **γεγονός** για το οποίο καταχωρήθηκε (trigger event). Τα γεγονότα που μπορεί να παρακολουθήσει το εργαλείο περιλαμβάνουν, μεταξύ άλλων:

- τη δημιουργία και ολοκλήρωση **παράλληλων περιοχών (parallel regions)**,
- τη δημιουργία, ενεργοποίηση και καταστροφή **tasks**,
- γεγονότα **συγχρονισμού** όπως barriers, critical ή atomic περιοχές,
- καθώς και την **εκκίνηση και τερματισμό νημάτων (threads)**.

Κάθε callback λαμβάνει ως ορίσματα δομές δεδομένων που περιέχουν πληροφορίες για το γεγονός (όπως αναγνωριστικά, pointers ή χρόνους εκτέλεσης), επιτρέποντας στο εργαλείο να τα αποθηκεύσει ή να τα αναλύσει σε πραγματικό χρόνο. Η προσέγγιση αυτή επιτρέπει τον λεπτομερή συσχετισμό μεταξύ των γεγονότων που προκύπτουν από διαφορετικά νήματα ή tasks, σχηματίζοντας έτσι μια πλήρη εικόνα της δυναμικής συμπεριφοράς του προγράμματος.

3.1.1.2 Συσχέτιση threads και tasks

Μια από τις σημαντικότερες έννοιες του OMPT είναι η συσχέτιση των threads με τα tasks που εκτελούν. Κατά την εκτέλεση ενός OpenMP προγράμματος, κάθε νήμα μπορεί να δημιουργήσει ή να εκτελέσει ένα ή περισσότερα tasks, ενώ τα ίδια tasks μπορεί να μεταφερθούν μεταξύ νημάτων, ανάλογα με την πολιτική του runtime scheduler.

Για την ομαλή και συνεπή παρακολούθηση αυτής της σχέσης, το OMPT παρέχει μοναδικά αναγνωριστικά (identifiers) για κάθε thread, task και parallel region. Μέσω αυτών, το εργαλείο μπορεί να ανασυνθέσει τη δομή του DAG (Directed Acyclic Graph) των tasks, να μελετήσει τις εξαρτήσεις μεταξύ τους και να εντοπίσει σημεία πιθανής ανισορροπίας φόρτου ή καθυστέρησης.

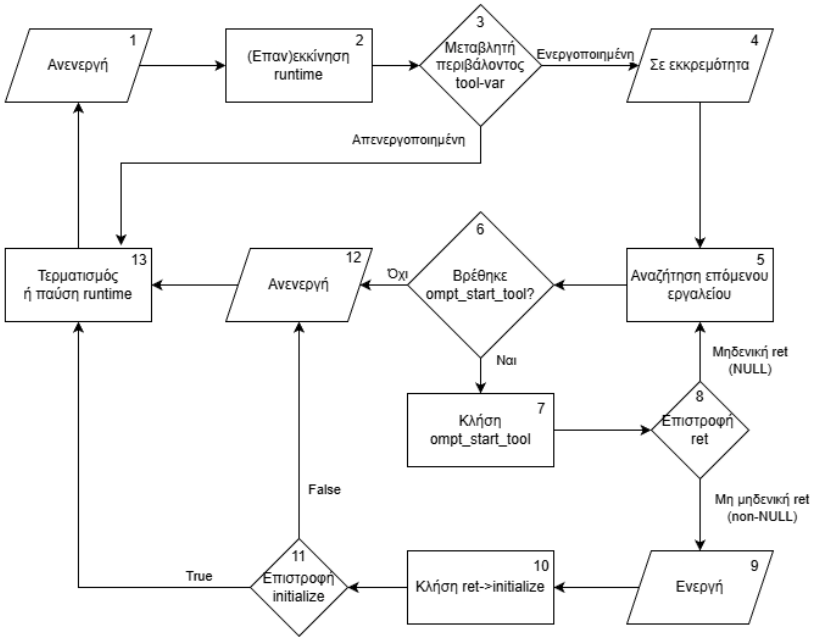
Η δυνατότητα αυτή είναι κρίσιμη για εργαλεία ανάλυσης απόδοσης, καθώς επιτρέπει την κατανόηση του τρόπου με τον οποίο τα threads εκτελούν τα tasks σε σχέση με την προγραμματισμένη λογική του χρήστη, αναδεικνύοντας αποκλίσεις ή ανεπάρκειες στον προγραμματισμό του παράλληλου προγράμματος.

3.1.2 Ενεργοποίηση OMPT runtime

Για να μπορέσει ένα εργαλείο να αλληλεπιδράσει με τη διεπαφή OMPT, απαιτείται η σωστή ενεργοποίηση και των δύο κατά την εκκίνηση μιας εφαρμογής OpenMP. Η διαδικασία αυτή περιλαμβάνει τέσσερα βασικά στάδια, βλ. Σχήμα 3.1:

- Ορισμός της μεταβλητής περιβάλλοντος **OMP_TOOL** ως **enabled** (κόμβος 3)
- Έκθεση της συνάρτησης **ompt_start_tool** από το εργαλείο στην βιβλιοθήκη του runtime (κόμβος 6)
- Έλεγχος συμβατότητας της έκδοσης μεταξύ εργαλείου και runtime (κόμβος 8)
- Ανάκτηση των συναρτήσεων της διεπαφής και καταχώρηση των **callback** που το εργαλείο υλοποιεί για την ανάλυση της εφαρμογής OpenMP. (κόμβος 10)

Ο χρήστης μπορεί να ενεργοποιήσει το εργαλείο είτε στατικά είτε δυναμικά, μέσω μεταβλητών περιβάλλοντος ή μέσω εντολές τερματικού κατά τη φόρτωση του εκτελέσιμου αρχείου, εξασφαλίζοντας ευελιξία στην ενσωμάτωση εργαλείων τρίτων ή ερευνητικών πρωτοτύπων. Παρακάτω βρίσκεται το διάγραμμα με τα βήματα της διαδικασίας ενεργοποίησης του πρώτου συμβατού εργαλείου από την διεπαφή OMPT, βάση των προδιαγραφών του προτύπου OpenMP.



Σχήμα 3.1 Διάγραμμα ροής ενεργοποίησης εργαλείου

3.1.3 Διαχείριση μνήμης

Η αποτελεσματική διαχείριση μνήμης αποτελεί κρίσιμη παράμετρο για την αξιόπιστη λειτουργία και την απόδοση των εργαλείων που βασίζονται στη διεπαφή `OMPT`. Δεδομένου ότι η παρακολούθηση γεγονότων σε περιβάλλοντα παράλληλης εκτέλεσης συνεπάγεται με τη συνεχή ανταλλαγή δεδομένων μεταξύ του `OpenMP` runtime και του εργαλείου, η οργάνωση και η προστασία της μνήμης απαιτούν ιδιαίτερη προσοχή ώστε να αποφεύγονται φαινόμενα συμφόρησης, ασυνέπειες ή συνθήκες/καταστάσεις ανταγωνισμού (`race conditions`).

3.1.3.1 Δομές δεδομένων callback

Κάθε `callback` που δηλώνεται μέσω της διεπαφής `OMPT` συνοδεύεται από συγκεκριμένες δομές δεδομένων που περιγράφουν το πλαίσιο του γεγονότος (`event context`). Οι δομές αυτές περιλαμβάνουν δείκτες προς αντικείμενα `runtime`, όπως ταυτοποιητές νημάτων (`ompt_thread_t`), ταυτοποιητές `tasks` (`ompt_task_id_t`) και αναφορές σε παράλληλες περιοχές (`ompt_parallel_id_t`). Ο σχεδιασμός τους είναι ελαφρύς, ώστε να ελαχιστοποιείται το επιπλέον κόστος κατά τη μετάδοση των δεδομένων, ενώ η ευθύνη για τη διαχείριση της μνήμης (δέσμευση, αποδέσμευση και ανακύκλωση) ανήκει εξ ολοκλήρου στο εργαλείο. Η σωστή χρήση των δομών `callback` είναι απαραίτητη για τη διατήρηση της συνέπειας των μετρήσεων, καθώς οι ίδιες δομές μπορούν να ανακυκλώνονται μεταξύ διαφορετικών γεγονότων, εφόσον δεν υπάρχουν ενεργές αναφορές σε αυτές.

3.1.3.2 Per-thread storage και buffers

Σε ένα σύστημα OpenMP, κάθε νήμα εκτέλεσης μπορεί να ενεργοποιεί πλήθος callbacks, καθιστώντας αναγκαία την ύπαρξη τοπικού χώρου αποθήκευσης ανά νήμα (thread-local storage) για την απομόνωση των δεδομένων. Το OMPT δεν επιβάλλει συγκεκριμένο μηχανισμό διαχείρισης δεδομένων ανά νήμα, παρέχοντας στα εργαλεία την ευελιξία να υλοποιήσουν προσαρμοσμένους buffers ανάλογα με τις ανάγκες τους. Μια συνήθης προσέγγιση είναι η δέσμευση κυκλικών buffers για κάθε νήμα, στους οποίους αποθηκεύονται προσωρινά οι πληροφορίες γεγονότων προτού μεταφερθούν σε κεντρική δομή συλλογής ή γραφτούν σε αρχείο καταγραφής. Με τον τρόπο αυτό μειώνεται η ανάγκη συγχρονισμού μεταξύ νημάτων και βελτιώνεται η απόδοση, ειδικά σε εφαρμογές με μεγάλο πλήθος γεγονότων ανά μονάδα χρόνου.

3.1.3.3 Ασφάλεια και συνέπεια δεδομένων

Η παράλληλη φύση του OpenMP καθιστά απαραίτητη την προστασία της πρόσβασης σε κοινόχρηστες δομές δεδομένων του εργαλείου. Ο σχεδιασμός των εργαλείων OMPT πρέπει να προβλέπει μηχανισμούς συγχρονισμού, όπως κλειδώματα (locks) ή atomic λειτουργίες, μόνο όταν αυτό είναι απολύτως αναγκαίο, ώστε να αποφεύγεται η επιβάρυνση της απόδοσης.

Επιπλέον, προτείνεται η χρήση αμετάβλητων (immutable) δομών για δεδομένα που παράγονται από γεγονότα τα οποία δεν τροποποιούνται στη συνέχεια. Αυτό διευκολύνει την ασφαλή παράλληλη ανάγνωση και μειώνει τον κίνδυνο λανθασμένων εξαρτήσεων μεταξύ νημάτων.

Η τήρηση της συνέπειας μεταξύ των αναγνωριστικών threads και tasks (όπως παρέχονται από το runtime) είναι κρίσιμη για την ορθή αντιστοίχιση των γεγονότων και την αναπαράσταση της ιεραρχίας εκτέλεσης κατά την ανάλυση ή την οπτικοποίηση των δεδομένων.

3.1.4 Παρακολούθηση γεγονότων

Η δυνατότητα παρακολούθησης γεγονότων (event monitoring) αποτελεί τον πυρήνα της λειτουργικότητας της διεπαφής OMPT, καθώς μέσω αυτής της δυνατότητας το εργαλείο αποκτά πρόσβαση στις πληροφορίες της εκτέλεσης ενός προγράμματος OpenMP, που το ενδιαφέρουν. Τα γεγονότα αντιστοιχούν σε κρίσιμες δραστηριότητες του runtime, όπως η δημιουργία παράλληλων περιοχών, η εκκίνηση/ολοκλήρωση tasks, οι μηχανισμοί συγχρονισμού μεταξύ των νημάτων και άλλα.

Αξιοποιώντας τις δυνατότητες που παρέχει η διεπαφή μέσω των callback, τα εργαλεία μπορούν να ανιχνεύσουν και να καταγράψουν λεπτομερείς ακολουθίες γεγονότων, επιτρέποντας την ανάλυση της χρονικής συμπεριφοράς και της αλληλεπίδρασης μεταξύ threads και tasks. Το OMPT δεν επιβάλλει συγκεκριμένο ρυθμό δειγματοληψίας ή μορφή αποθήκευσης, αφήνοντας στο εργαλείο την ευχέρεια να υλοποιήσει προσαρμοσμένους μηχανισμούς συλλογής δεδομένων ανάλογα με τον σκοπό του.

3.1.4.1 Parallel regions

Κάθε φορά που μια εφαρμογή OpenMP εισέρχεται ή εξέρχεται από μια παράλληλη περιοχή (parallel region), το runtime ενεργοποιεί/καλεί τη καταχωρημένη συνάρτηση callback στο αντίστοιχο γεγονός, όπως τα γεγονότα ompt_callback_parallel_begin και ompt_callback_parallel_end. Μέσω αυτών των συναρτήσεων, το εργαλείο μπορεί να αναγνωρίσει τη δημιουργία μιας νέας ομάδας νημάτων, να καταγράψει το πλήθος τους, καθώς και να συσχετίσει τη συγκεκριμένη περιοχή με το νήμα που την ενεργοποίησε.

Οι πληροφορίες αυτές είναι ιδιαίτερα χρήσιμες για την ανάλυση απόδοσης (performance profiling), καθώς επιτρέπουν τη μέτρηση του χρόνου ζωής κάθε παράλληλης περιοχής, την εκτίμηση της κλιμάκωσης (scalability) και την αναγνώριση πιθανών ανισορροπιών φόρτου μεταξύ των νημάτων. Επιπλέον, η γνώση της ιεραρχίας των parallel regions επιτρέπει τη χαρτογράφηση της συνολικής δομής του προγράμματος σε επίπεδο παραλληλισμού.

3.1.4.2 Tasks και εξαρτήσεις

Τα tasks αποτελούν θεμελιώδες στοιχείο του προτύπου OpenMP, προσφέροντας έναν ευέλικτο μηχανισμό ασύγχρονης εκτέλεσης εργασιών. Το OMPT επιτρέπει την παρακολούθηση του κύκλου ζωής κάθε task μέσω των εξής callbacks:

- ompt_callback_task_create
- ompt_callback_task_schedule
- ompt_callback_task_end

Κάθε task λαμβάνει μοναδικό αναγνωριστικό (ompt_task_id_t), το οποίο μπορεί να συσχετιστεί με το νήμα που το δημιούργησε και με το task από το οποίο προήλθε (parent task). Με αυτόν τον τρόπο, το εργαλείο μπορεί να αναδομήσει το γράφημα εξαρτήσεων (DAG) των tasks, εντοπίζοντας τις σχέσεις δημιουργίας και εκτέλεσης.

Η πληροφορία αυτή είναι πολύτιμη για εργαλεία ανάλυσης εξαρτήσεων και βελτιστοποίησης προγραμματισμού, καθώς επιτρέπει την κατανόηση του τρόπου με τον οποίο ο runtime εκτελεί τα tasks σε σχέση με τη δηλωμένη λογική του χρήστη. Επιπλέον, η καταγραφή γεγονότων αλλαγής κατάστασης (π.χ. από "suspended" σε "executing") επιτρέπει την αναγνώριση καθυστερήσεων ή ανισοκατανομής φόρτου μεταξύ νημάτων.

3.1.4.3 Συγχρονισμοί (barriers, critical sections)

Οι μηχανισμοί συγχρονισμού του OpenMP αποτελούν κρίσιμα σημεία ελέγχου που επηρεάζουν σημαντικά την απόδοση και τη συμπεριφορά της παράλληλης εκτέλεσης. Η διεπαφή OMPT παρέχει ειδοποιήσεις μέσω callback για γεγονότα όπως barrier_begin, barrier_end, critical_begin, critical_end και atomic, τα οποία επιτρέπουν στο εργαλείο να παρακολουθεί πότε τα νήματα εισέρχονται ή εξέρχονται από περιοχές συγχρονισμού.

Η καταγραφή αυτών των γεγονότων δίνει τη δυνατότητα εντοπισμού περιοχών συμφόρησης (bottlenecks), όπου η απόδοση του προγράμματος περιορίζεται από τη συχνότητα ή τη διάρκεια των συγχρονισμών. Παράλληλα, επιτρέπει την οπτικοποίηση της αλληλουχίας των γεγονότων μεταξύ νημάτων, προσφέροντας πολύτιμες πληροφορίες για τον εντοπισμό προβλημάτων όπως false sharing, lock contention ή υπερβολικός συγχρονισμός.

3.1.4.4 Thread lifecycle

Η διεπαφή OMPT επιτρέπει την πλήρη παρακολούθηση του κύκλου ζωής των νημάτων (thread lifecycle) μέσω των γεγονότων ompt_callback_thread_begin και ompt_callback_thread_end. Καταχωρώντας την κατάλληλη συνάρτηση σε αυτά τα γεγονότα, το εργαλείο μπορεί να ανιχνεύσει τη δημιουργία κάθε νήματος, τον τύπο του (π.χ. master, worker) και τη χρονική στιγμή εκκίνησης και τερματισμού του.

Οι πληροφορίες αυτές είναι κρίσιμες για τη χαρτογράφηση της εκτέλεσης σε πολυπύρηνα συστήματα, καθώς επιτρέπουν την ακριβή μέτρηση του χρόνου δραστηριότητας ανά νήμα και την ανίχνευση ανενεργών φάσεων (idle times). Σε συνδυασμό με τα δεδομένα των parallel regions και των tasks, η ανάλυση του κύκλου ζωής των νημάτων συμβάλλει στη δημιουργία ενός πλήρους χρονοδιαγράμματος εκτέλεσης, χρήσιμου για προφίλ απόδοσης και βελτιστοποίηση παράλληλων εφαρμογών.

3.2 1st Party Tools

Η διεπαφή OMPT παρέχει τη θεμελιώδη υποδομή για την ανάπτυξη εργαλείων τα οποία συνδέονται απευθείας στο περιβάλλον εκτέλεσης OpenMP. Αξιοποιώντας τις κλήσεις συναρτήσεων callback στα υποστηριζόμενα γεγονότα και τις δομές δεδομένων που προσφέρει η διεπαφή OMPT, βάση του προτύπου OpenMP, μπορούν να παρακολουθούν και να αναλύουν την εκτέλεση εφαρμογών OpenMP σε πραγματικό χρόνο και χωρίς να απαιτείται καμία τροποποίηση ή εκ νέου μετάφραση του πηγαίου κώδικα της εκάστοτε εφαρμογής. Τα 1st party tools αναπτύσσονται ειδικά για να αξιοποιήσουν πλήρως τις δυνατότητες της διεπαφής, συνήθως από ερευνητές ή προγραμματιστές που έχουν άμεση πρόσβαση στο OpenMP runtime. Τα εργαλεία αυτά μεταφράζονται ή συνδέονται (link) με την εφαρμογή OpenMP και εξαρτούνται από αυτή για να λειτουργήσουν, σε αντίθεση με τα εξωτερικά εργαλεία (3rd party tools), τα οποία μπορούν να εκτελεστούν ανεξάρτητα από την εφαρμογή.

Τα 1st party tools χρησιμοποιούνται τόσο profiling και tracing, επιτρέποντας τη μελέτη της συμπεριφοράς της εφαρμογής με λεπτομέρεια και ακρίβεια και σε σημεία δυσπρόσιτα εξωτερικά της βιβλιοθήκης του runtime, όπως η παρακολούθηση των tasks, των parallel regions και των συγχρονισμών. Παράλληλα, αποτελούν τη βάση για πιο εξελιγμένα συστήματα οπτικοποίησης και αυτόματης βελτιστοποίησης απόδοσης. Η ανάπτυξή τους προϋποθέτει τη κατανόηση του τρόπου με τον οποίο ενεργοποιείται και αλληλεπιδρά η διεπαφή OMPT με το OpenMP runtime. Στόχος αυτής της ενότητας είναι να παρουσιαστεί ο τρόπος με τον οποίο ένα τέτοιο εργαλείο (1st party) βασισμένο στη διεπαφή OMPT ενεργοποιείται, αρχικοποιείται και καταχωρεί τους απαραίτητους μηχανισμούς παρακολούθησης. Ιδιαίτερη έμφαση δίνεται στη διαδικασία αρχικοποίησης του εργαλείου, καθώς αυτή αποτελεί το κρίσιμο στάδιο που καθορίζει την επιτυχή και ομαλή επικοινωνία με την υλοποίηση της διεπαφής OMPT και τη σωστή λειτουργία όλων των μηχανισμών συλλογής δεδομένων.

3.2.1 Αρχικοποίηση εργαλείου

Η αρχικοποίηση ενός 1st party tool από την υλοποίηση της διεπαφής αποτελεί το πρώτο και καθοριστικό στάδιο για τη λειτουργία του. Σε αυτό το στάδιο το εργαλείο συνδέεται με το OpenMP runtime, καταχωρεί τις συναρτήσεις παρακολούθησης και προετοιμάζει τις δομές δεδομένων που θα χρησιμοποιήσει σε όλη τη διάρκεια της εκτέλεσης της εφαρμογής. Η διαδικασία αυτή είναι πλήρως

καθορισμένη από το πρότυπο κάτι που, εξασφαλίζει τη φορητότητα και τη συμβατότητα των εργαλείων με διαφορετικές υλοποιήσεις του OpenMP. Αποτελείται από δύο βασικά μέρη, τη συνάρτηση ενεργοποίησης του εργαλείου `ompt_start_tool` και την καταχώρηση των `callback` στα κατάλληλα γεγονότα.

3.2.1.1 ompt_start_tool

Η συνάρτηση `ompt_start_tool` είναι το σημείο εισόδου ενός εργαλείου βασισμένο στη διεπαφή OMPT αφού αποτελεί τον μηχανισμό μέσω του οποίου το OpenMP runtime αναγνωρίζει και ενεργοποιεί το εργαλείο. Κατά την εκκίνηση του προγράμματος, η διεπαφή αναζητά τη συγκεκριμένη συνάρτηση μέσα στις βιβλιοθήκες πιθανών συμβατών εργαλείων (κόμβος 5 – Σχήμα 3.1). Η συνάρτηση αυτή έχει αποκλειστικά διαμορφωτικό ρόλο καθώς σε αυτό το σημείο της εκκίνησης δεν πραγματοποιείται καμία παρακολούθηση γεγονότων το εργαλείο. Το εργαλείο θα πρέπει να επιβεβαιώσει την συμβατότητα με την έκδοση του OpenMP που υποστηρίζεται από τη βιβλιοθήκη του runtime και εφόσον είναι συμβατό να επιστρέψει στη διεπαφή τις συναρτήσεις εκκίνησης και τερματισμού (κόμβος 8 – Σχήμα 3.1). Η γενική μορφή όπως δίνεται από το πρότυπο OpenMP είναι η εξής:

```c
ompt_start_tool_result_t *ompt_start_tool(unsigned int omp_version, const char *runtime_version)
{
    ... // Έλεγχος συμβατότητας έκδοσης
    static ompt_start_tool_result_t ompt_tool_result = {
        &ompt_initialize,     // Συνάρτηση αρχικοποίησης
        &ompt_initialize,     // Συνάρτηση τερματισμού
        ompt_data_none        // Προαιρετικά δεδομένα εργαλείου
    }
}
```

Κώδικας 3.1 Παράδειγμα γενικής δομής συνάρτησης ompt_start_tool

3.2.1.2 Έλεγχος συμβατότητας έκδοσης

Για να εξασφαλιστεί η ορθή λειτουργία της βιβλιοθήκης του OpenMP runtime και του εργαλείου, η διεπαφή OMPT χρησιμοποιεί έναν μηχανισμό ελέγχου συμβατότητας έκδοσης (`version compatibility`). Κατά την αρχικοποίηση (κόμβο 7 – Σχήμα 3.1), το runtime παρέχει στο εργαλείο την έκδοση (ή τις εκδόσεις) του πρότυπου OpenMP που υποστηρίζει. Στη περίπτωση ασυμβατότητας το εργαλείο ενημερώνει τη διεπαφή, ώστε να σταματήσει την ενεργοποίηση του, αποτρέποντας πιθανές αστοχίες ή απροσδόκητες συμπεριφορές, και να συνεχίσει να αναζητά άλλα συμβατά εργαλεία στο σύστημα (κόμβος 5 - Σχήμα 3.1).

Ο μηχανισμός αυτός είναι ιδιαίτερα σημαντικός σε περιβάλλοντα όπου συνυπάρχουν διαφορετικές εκδόσεις μεταφραστών ή βιβλιοθηκών OpenMP, καθώς

προλαμβάνει λάθη που θα μπορούσαν να προκύψουν από ασυμβατότητες στη δομή ή τη σημασιολογία των γεγονότων.

3.2.1.3 Καταχώρηση callbacks

Μετά την επιτυχή επιστροφή της ompt_start_tool (κόμβος 9 – Σχήμα 3.1), το OpenMP runtime αρχικοποιεί το συμβατό εργαλείο που βρέθηκε μέσω της συνάρτηση εκκίνησης (initialize), που αυτό υλοποιεί και είναι υπεύθυνη για την καταχώρηση των callbacks που θα χρησιμοποιηθούν κατά την εκτέλεση της εφαρμογής για την ανάλυση της (κόμβος 10 – Σχήμα 3.1).

Σε αυτό το σημείο το εργαλείο για κάθε γεγονός, που ενδιαφέρεται να παρακολουθήσει κατά τον χρόνο εκτέλεσης της εφαρμογής, καταχωρεί την κατάλληλη συνάρτηση callback, η οποία θα καλείται από την διεπαφή όταν αυτό το γεγονός συμβαίνει (event triggered). Η διαχείριση των καταχωρημένων callbacks γίνεται μέσω των συναρτήσεων ompt_set_callback και ompt_get_callback.

Η σωστή καταχώρηση των callbacks μπορεί να επιβεβαιωθεί μέσω του αποτελέσματος που επιστρέφει η ompt_set_callback και τη συνάρτηση ompt_get_callback. Εφόσον η τιμή επιστροφής υποδεικνύει επιτυχία, η συνάρτηση του εργαλείου είναι πλέον καταχωρημένη επιτυχώς με το αντίστοιχο γεγονός από το runtime και θα κληθεί όταν συμβεί το γεγονός.

Με την ολοκλήρωση της καταχώρησης, το εργαλείο και η διεπαφή παραμένουν ενεργά και συνδεδεμένα μεταξύ τους σε όλη τη διάρκεια της εκτέλεσης της εφαρμογής OpenMP ή μέχρι την κλήση της συνάρτησης τερματισμού (finalize), η οποία τερματίζει το εργαλείο και τη διεπαφής, αποδεσμεύοντας τους πόρους τους.

3.2.2 Δυνατότητες εργαλείων μέσω OMPT

Η διεπαφή OMPT αποτελεί τον βασικό μηχανισμό μέσω του οποίου 1st party tools μπορούν να παρατηρούν, να αναλύουν και να καταγράφουν τη συμπεριφορά εφαρμογών OpenMP κατά τον χρόνο εκτέλεσης. Σε αντίθεση με παραδοσιακές τεχνικές στατικής ή δυναμικής οργάνωσης κώδικα, η διεπαφή OMPT παρέχει τυποποιημένη πρόσβαση στον εσωτερικό μηχανισμό του OpenMP runtime, επιτρέποντας την ανάπτυξη εργαλείων υψηλής φορητότητας και χαμηλού επιπέδου παρεμβατικότητας.

Οι δυνατότητες αυτές καλύπτουν ένα ευρύ φάσμα αναγκών ανάλυσης, από την αξιολόγηση της απόδοσης παράλληλων περιοχών έως τη λεπτομερή καταγραφή γεγονότων εκτέλεσης και τη μελέτη της δυναμικής συμπεριφοράς των threads και των

tasks. Στο παρόν κεφάλαιο παρουσιάζονται οι βασικές κατηγορίες δυνατοτήτων που μπορούν να υλοποιηθούν μέσω OMPT.

3.2.2.1 Ανάλυση απόδοσης παράλληλων περιοχών OpenMP

Η ανάλυση απόδοσης των παράλληλων περιοχών (Parallel Regions Profiling) αποτελεί μία θεμελιώδη δυνατότητα που παρέχεται μέσω της διεπαφής OMPT. Μέσω των callbacks που ενεργοποιούνται κατά την είσοδο και έξοδο από παράλληλες περιοχές (parallel regions), είναι δυνατή η ακριβής καταγραφή της διάρκειας εκτέλεσης κάθε περιοχής, καθώς και η συσχέτισή της με τα συμμετέχοντα νήματα. Τα δύο βασικά παραδείγματα αυτών είναι τα εξής;

- ompt_callback_parallel_begin
- ompt_callback_parallel_end

Η δυνατότητα αυτή επιτρέπει την ποσοτική αξιολόγηση του βαθμού παραλληλισμού που επιτυγχάνει μια εφαρμογή, καθώς και τον εντοπισμό περιοχών όπου ο παραλληλισμός δεν αξιοποιείται αποτελεσματικά. Ιδιαίτερη σημασία έχει η ανάλυση της κατανομής χρόνου ανά νήμα, η οποία μπορεί να αποκαλύψει ανισορροπία φόρτου ή υπερβολικό κόστος συγχρονισμού. Η συγκεκριμένη λειτουργικότητα βασίζεται στους μηχανισμούς που έχουν ήδη παρουσιαστεί σε προηγούμενες ενότητες, αλλά εδώ αποκτά πρακτική αξία ως εργαλείο βελτιστοποίησης απόδοσης.

3.2.2.2 Ανάλυση και παρακολούθηση OpenMP tasks

Η υποστήριξη δυναμικών task στο OpenMP εισάγει αυξημένη πολυπλοκότητα στην ανάλυση της εκτέλεσης, καθώς η σειρά και ο τόπος εκτέλεσης των tasks δεν είναι προκαθορισμένα. Το OMPT αντιμετωπίζει το ζήτημα αυτό παρέχοντας callbacks που καλύπτουν τη δημιουργία, τον προγραμματισμό και την ολοκλήρωση των tasks. Μερικά βασικά παραδείγματα είναι τα εξής:

- ompt_callback_task_create
- ompt_callback_task_schedule
- ompt_callback_task_end

Μέσω αυτής της δυνατότητας, τα εργαλεία μπορούν να ανασυνθέσουν την πραγματική ροή εκτέλεσης των tasks και να εκτιμήσουν τον πραγματικό βαθμό παράλληλης εκτέλεσης που επιτυγχάνεται. Η ανάλυση αυτή είναι κρίσιμη για τον εντοπισμό περιπτώσεων όπου οι εξαρτήσεις περιορίζουν τον παραλληλισμό ή όπου η δημιουργία υπερβολικά μικρών tasks οδηγεί σε αυξημένο runtime overhead. Οι πληροφορίες που μπορεί να συλλέξει ένα εργαλείο για κάθε task, είναι οι εξής:

- το μοναδικό του αναγνωριστικό (ompt_task_id_t),
- τον γονέα του (parent task),
- την κατάσταση εκτέλεσης (π.χ. ompt_task_queued, ompt_task_running, ompt_task_complete),
- καθώς και τη χρονική στιγμή δημιουργίας και ολοκλήρωσής του.

Με βάση τα δεδομένα αυτά, το εργαλείο μπορεί να αναδομήσει το κατευθυνόμενο άκυκλο γράφημα (DAG) των tasks, αποτυπώνοντας τις εξαρτήσεις μεταξύ τους και τη ροή εκτέλεσης. Η οπτικοποίηση του γράφου αυτού αποκαλύπτει κρίσιμα μοτίβα, όπως σημεία συμφόρησης, αλληλεξαρτήσεις ή ανισορροπία στη διανομή εργασιών.

Επιπλέον, το εργαλείο μπορεί να υπολογίζει μετρικές όπως το βάθος του DAG, τον αριθμό ενεργών tasks ανά χρονική στιγμή, ή τον λόγο συγχρονισμού/υπολογισμού, επιτρέποντας τη λεπτομερή αξιολόγηση του βαθμού παραλληλισμού και της αποδοτικότητας του task scheduler.

3.2.2.3 Ανάλυση συμπεριφοράς μνήμης σε περιβάλλον OpenMP

Η συμπεριφορά μνήμης αποτελεί καθοριστικό παράγοντα απόδοσης σε παράλληλες εφαρμογές, ιδιαίτερα σε περιβάλλοντα με έντονη χρήση tasks και δυναμικών κατανομών. Μέσω του OMPT, είναι δυνατή η συσχέτιση πληροφοριών εκτέλεσης με μηχανισμούς αποθήκευσης δεδομένων ανά νήμα, επιτρέποντας την ανάλυση της χρήσης μνήμης σε επίπεδο threads , tasks και παράλληλων περιοχών.

Η δυνατότητα αυτή καθιστά εφικτή τη βαθύτερη κατανόηση του αποτυπώματος στην μνήμη της εφαρμογής. Κάτι που διευκολύνει τον εντοπισμό ανισορροπιών στη χρήση μνήμης, αυξημένου overhead λόγω κατανομών, καθώς και προβλημάτων που σχετίζονται με τον κατακερματισμό. Η ανάλυση συμπεριφοράς της μνήμης συμπληρώνει τις μετρικές απόδοσης και παρέχει μια πιο ολοκληρωμένη εικόνα της πραγματικής συμπεριφοράς, όπως και στην συνολική αξιολόγηση της αποδοτικότητας και της κλιμάκωσης, της παράλληλης εφαρμογής OpenMP.

3.2.3 Παραδείγματα εργαλείων

Στην παρούσα ενότητα παρουσιάζονται αντιπροσωπευτικά 1st party tools που αξιοποιούν τη διεπαφή OMPT για την ανάλυση και παρακολούθηση εφαρμογών OpenMP. Στόχος δεν είναι η εξαντλητική παρουσίαση όλων των διαθέσιμων εργαλείων, αλλά η ανάδειξη χαρακτηριστικών παραδειγμάτων που είναι διαδεδομένα και δημοφιλή. Κάθε εργαλείο χρησιμοποιεί όλες τις δυνατότητες της διεπαφής OMPT ως

βασικό μηχανισμό συλλογής πληροφορίας διαφοροποιούμενο όμως ως προς το είδος της ανάλυσης και τον τρόπο παρουσίασης των αποτελεσμάτων σε σχέση με τα υπόλοιπα εργαλεία.

3.2.3.1 Score-P

Το Score-P[5] αποτελεί ένα ολοκληρωμένο εργαλείο μέτρησης απόδοσης για εφαρμογές υψηλών επιδόσεων και υποστηρίζει OpenMP μέσω της διεπαφής OMPT. Στο πλαίσιο αυτό, το εργαλείο αξιοποιεί τη δυνατότητα profiling παράλληλων περιοχών, καταγράφοντας τη διάρκεια και τη συχνότητα εκτέλεσης των parallel regions, καθώς και τη συμμετοχή των threads σε αυτές.

Επιπλέον, το Score-P υποστηρίζει execution tracing, επιτρέποντας τη συλλογή λεπτομερών ιχνών εκτέλεσης που περιλαμβάνουν γεγονότα σχετικά με παράλληλες περιοχές, tasks και συγχρονισμούς. Μέσω αυτών των δεδομένων, είναι δυνατή η ανάλυση της χρονικής αλληλεπίδρασης των επιμέρους στοιχείων της εφαρμογής, καθώς και ο εντοπισμός σημείων συμφόρησης. Το εργαλείο μπορεί επίσης να συσχετίσει πληροφορίες χρήσης μνήμης με threads και περιοχές εκτέλεσης, καλύπτοντας σε βασικό επίπεδο τη δυνατότητα ανάλυσης συμπεριφοράς μνήμης.

3.2.3.2 Extrae

Το Extrae[6] είναι εργαλείο δυναμικής καταγραφής γεγονότων, σχεδιασμένο για τη συλλογή ιχνών εκτέλεσης σε παράλληλες εφαρμογές. Η υποστήριξη OpenMP μέσω OMPT επιτρέπει στο Extrae να αξιοποιεί σε βάθος τη δυνατότητα καταγραφής γεγονότων εκτέλεσης, παράγοντας χρονοσειρές γεγονότων που περιλαμβάνουν parallel regions, tasks, συγχρονισμούς και δραστηριότητα threads.

Ιδιαίτερη έμφαση δίνεται στην ανάλυση και παρακολούθηση OpenMP tasks, καθώς το εργαλείο καταγράφει τη δημιουργία, τον προγραμματισμό και την ολοκλήρωση των tasks, επιτρέποντας την ανακατασκευή της δυναμικής συμπεριφοράς τους και του γράφου εξαρτήσεων (DAG). Τα παραγόμενα ίχνη μπορούν να χρησιμοποιηθούν για την κατανόηση του βαθμού ταυτόχρονης εκτέλεσης και της επίδρασης των εξαρτήσεων στην απόδοση της εφαρμογής.

Μέσω αυτής της ανάλυσης, το εργαλείο επιτρέπει τον εντοπισμό περιπτώσεων υπερβολικής σειριοποίησης, μη αναμενόμενων εξαρτήσεων ή αναποτελεσματικής κατανομής εργασίας στα threads.

3.2.3.3 Intel VTune Profiler

Το Intel VTune Profiler[7] αποτελεί εμπορικό εργαλείο ανάλυσης απόδοσης, το οποίο υποστηρίζει OpenMP μέσω της διεπαφής OMPT σε σύγχρονες υλοποιήσεις του OpenMP runtime. Το εργαλείο αξιοποιεί κυρίως τη δυνατότητα profiling παράλληλων περιοχών, παρέχοντας συγκεντρωτικά και αναλυτικά στατιστικά για τη χρήση των threads, τη διάρκεια των parallel regions και τον βαθμό αξιοποίησης των διαθέσιμων πυρήνων.

Παράλληλα, το VTune υποστηρίζει τη παρακολούθηση του κύκλου ζωής των νημάτων OpenMP, επιτρέποντας τον εντοπισμό περιόδων αδράνειας και ανισορροπίας φόρτου. Μέσω της ενοποίησης των δεδομένων αυτών με πληροφορίες χαμηλού επιπέδου από το σύστημα, το εργαλείο παρέχει μια συνολική εικόνα της απόδοσης της εφαρμογής, χωρίς να απαιτείται άμεση παρέμβαση στον κώδικα.

3.2.3.4 Caliper

Το Caliper[8] είναι ένα ελαφρύ εργαλείο σχολιασμού (annotation) και συλλογής μετρικών απόδοσης, σχεδιασμένο ώστε να ενσωματώνεται εύκολα σε εφαρμογές υψηλών επιδόσεων και να συνεργάζεται με άλλα εργαλεία ανάλυσης. Στο πλαίσιο του OpenMP, το Caliper υποστηρίζει τη διεπαφή OMPT και αξιοποιεί τις δυνατότητές της, κυρίως για profiling παράλληλων περιοχών OpenMP και για παρακολούθηση του κύκλου ζωής των νημάτων.

Μέσω των OMPT callbacks, το εργαλείο μπορεί να συσχετίσει τις παράλληλες περιοχές και τη δραστηριότητα των threads με λογικές περιοχές (regions) μέτρησης που ορίζονται από τον χρήστη, επιτρέποντας την ιεραρχική οργάνωση των μετρικών απόδοσης. Επιπλέον, το Caliper μπορεί να αξιοποιήσει πληροφορίες καταγραφής γεγονότων εκτέλεσης σε συνοπτική μορφή, με στόχο τη χαμηλού κόστους συλλογή δεδομένων και τη μεταγενέστερη ανάλυσή τους από εργαλεία, όπως το Score-P.

Αν και δεν εστιάζει στην εκτενή ανάλυση OpenMP tasks ή στη λεπτομερή διερεύνηση της συμπεριφοράς μνήμης, το Caliper λειτουργεί συμπληρωματικά προς άλλα 1st party tools, προσφέροντας μια ευέλικτη και επεκτάσιμη προσέγγιση για στοχευμένο profiling και πειραματική αξιολόγηση εφαρμογών OpenMP.

Κεφάλαιο 4. Υλοποίηση Διεπαφής OMPT στον OMPi

4.1 Περιγραφή Υλοποίησης

Όπως αναφέρθηκε ο OMPi διαχωρίζεται σε δύο κύρια μέρη τον μεταφραστή (compiler) και το σύστημα χρόνου εκτέλεσης (runtime). Η διεπαφή υλοποιείται στο σύστημα χρόνου εκτέλεσης ή αλλιώς ORT (OMPi RunTime). Καθώς η διεπαφή χρειάζεται πρόσβαση σε δεδομένα που είναι μέρος του ORT ώστε να αποσταλούν τα κατάλληλα δεδομένα στο εργαλείο.

Σημαντική έμφαση δόθηκε ώστε να μην προκαλείται overhead όταν η διεπαφή είναι ανενεργή. Με τη χρήση μεταβλητών περιβάλλοντος και οδηγιών προεπεξεργαστή (preprocessor instructions) το κύριο αρχείο της υλοποίησης της διεπαφής (ompt.c) με το αρχείο κεφαλίδας (omp-tools.h) όπως και όλες οι γραμμές κώδικα, σε αρχεία του ORT εκτός του ompt.c, που υλοποιούνται για τη διεπαφή OMPT δεν μεταφράζονται καθόλου. Στη περίπτωση που μεταφραστούν, δηλαδή η μεταβλητή περιβάλλοντος **OMP_TOOL** είναι ενεργή και δεν υπάρχει κάποιο εργαλείο ενεργοποιημένο, με χρήση της οδηγίας προ-επεξεργαστή **OMPT_ACTIVE** του ORT η γραμμές αυτές δεν εκτελούνται καθόλου. Οπότε σε αυτή τη περίπτωση το μόνο overhead που υπάρχει είναι ελάχιστο και επιβαρύνει μόνο στη μετάφραση, όχι κατά την εκτέλεση. *(βρίσκονται στο τέλος του αρχείου libort/ort.h)*

4.2 Αρχείο ompt.c

Το κύριο μέρος της διεπαφής υλοποιείται στο αρχείο **ompt.c**, του ORT. Οι συναρτήσεις που υλοποιήθηκαν ως μέρος της διπλωματικής εργασίας είναι κυρίως αυτές που υποστηρίζονται από το πρότυπο του OpenMP για τη διεπαφή OMPT,

περισσότερες πληροφορίες για αυτές βρίσκονται στην ενότητα 4.4. Οι πιο σημαντικές συναρτήσεις είναι της αρχικοποίησης και του τερματισμού, οι οποίες αναλύονται στις ενότητες 4.3 και 4.7, αντίστοιχα. Στον μεταφραστή `OMPi` υλοποιήθηκαν επίσης οι εξής "βοηθητικές" συναρτήσεις, που δεν είναι μέρος του προτύπου:

- `ompi_ompt_init`
- `check_verb_init`
- `add_log`
- `open_tool_lib`
- `tool_initialize`
- `ompi_ompt_lookup`
- `load_symbol`
- `ompt_invoke_callback`
- `find_state_index`
- `check_before_exec`
- `ompt_state_pop/peep/push`

Οι συναρτήσεις που υλοποιήθηκαν για τον έλεγχο της ομαλής εκτέλεσης από το τελικό χρήστη είναι οι εξής:

- **check_verb_init**, που ελέγχει αν η εκτέλεση πρέπει να γίνει verbose
- **add_log**, η οποία αν η εκτέλεση είναι verbose αναφέρει μέσω του τερματικού στον τελικό χρήστη σημαντικές πληροφορίες κατά την εκτέλεση, ώστε να μπορεί να τη παρακολουθήσει με ακρίβεια και συνέπεια.

Οι πιο σημαντικές από τις "βοηθητικές" συναρτήσεις, που δεν θα αναλυθούν σε άλλες ενότητες περαιτέρω είναι οι εξής:

- **ompt_state_pop/peep/push**: Διαχειρίζεται τις καταστάσης των νημάτων κατά το χρόνο της εκτέλεσης, δηλαδή καλούνται από το `runtime (ORT)` όταν τα νήματα αλλάζουν κατάσταση.
- **find_state_index**: Μετατρέπει το αναγνωριστικό της κατάστασης (`enum`) σε αλφαριθμητικό (`string`).

```
int find_state_index(int current_state)
{
    int len = sizeof(ompi_ompt_thr_states) / sizeof(ompi_ompt_thr_states[0]);

    for (int i=0; i<len; i++)
        if (ompi_ompt_thr_states[i].state == current_state)
            return i;

    return -1;
}
```

Κώδικας 4.1 Υλοποίηση συνάρτηση find_state_index

4.3 Αρχικοποίηση διεπαφής και εργαλείου

Ένα από τα κρίσιμα σημεία κατά την εκκίνηση της εκτέλεσης του ORT είναι η αρχικοποίηση της υλοποίησης της διεπαφής OMPT και του εργαλείου. Για τη λειτουργία αυτή υλοποιήθηκε η "βοηθητική" συνάρτηση ompi_ompt_init, η οποία καλείται κατά την εκκίνηση του ORT, εφόσον η μεταβλητή περιβάλλοντος **OMP_TOOL** είναι ενεργή. Είναι υπεύθυνη για το έλεγχο ροής του διαγράμματος στο Σχήμα 3.1, αναζητάει και ενεργοποιεί το πρώτο συμβατό εργαλείο. Αρχικά ελέγχει αν υπάρχει verbose εκτέλεση έπειτα ξεκινάει τη διαδικασία αναζήτησης του πρώτου συμβατού εργαλείου για ενεργοποίηση. Σε περίπτωση που η εκτέλεση είναι verbose η διεπαφή ενημερώνει το τελικό χρήστη μέσω του τερματικού για την εξέλιξη της διαδικασίας αυτής με τα κατάλληλα μηνύματα.

Αξιοσημείωτο είναι πως οι συναρτήσεις για την διαχείριση καταχώρησης των callback και η συνάρτηση που δίνει πρόσβαση στο lookup_table είναι διαθέσιμες στο εργαλείο αποκλειστικά κατά την αρχικοποίησης του εργαλείου (κόμβος 10 – Σχήμα 3.1). Ενώ οι συναρτήσεις που υποστηρίζονται από την υλοποίηση της διεπαφής OMPT και παρέχονται στο εργαλείο μέσω του lookup_table, παραμένουν διαθέσιμες σε όλη τη διάρκεια της εκτέλεσης, εφόσον το εργαλείο και η διεπαφή είναι ενεργά, βάσει των προδιαγραφών του προτύπου OpenMP. Έτσι, ο μηχανισμός φόρτωσης του εργαλείου είναι υπεύθυνος ώστε να διατηρήσει το εργαλείο οποιαδήποτε συνάρτηση επιθυμεί για την ανάλυση και καταγραφή των δεδομένων που χρειάζεται, κατά την εκτέλεση.

Ο μηχανισμός φόρτωσης εργαλείου βασίζεται σε δύο συναρτήσεις που οφείλει να υλοποιεί κάθε εργαλείο, το οποίο επιθυμεί να είναι συμβατό με την διεπαφή OMPT, τις ompt_start_tool, initialize και μία συνάρτηση lookup που υλοποιεί η διεπαφή, η οποία είναι οφείλει να ακολουθεί τον τύπο ompt_function_lookup_t, βάσει των προδιαγραφών του προτύπου OpenMP.

4.3.1 ompt_start_tool

Η συνάρτηση ompt_start_tool είναι υπεύθυνη για το έλεγχο συμβατότητας, πρέπει να υλοποιείται και να συνδέεται με το runtime από το εργαλείο.

Εφόσον, το εργαλείο είναι συμβατό με την έκδοση, που υποστηρίζει η βιβλιοθήκη του runtime, επιστρέφει μια μη-μηδενική δομή (struct) με τον τύπο **ompt_start_tool_result_t**, η οποία περιέχει δείκτες προς δύο κρίσιμες συναρτήσεις:

- **initialize**: Χρησιμοποιείται για την καταχώρηση των `callbacks` και τη δέσμευση των απαιτούμενων πόρων του εργαλείου
- **finalize**: Καλείται κατά τον τερματισμό του runtime για την απελευθέρωση της μνήμης και την ολοκλήρωση των διαδικασιών καταχώρησης ή σε περίπτωση σφάλματος για την ομαλή ολοκλήρωση του τερματισμού διεπαφής και εργαλείου.

Κατά την εκκίνηση του `OpenMP runtime`, εφόσον η μεταβλητή περιβάλλοντος "**`OMP_TOOL`**" είναι ενεργή, η διεπαφή `OMPT` αναζητά τη συνάρτηση `ompt_start_tool`, που ένα συμβατό εργαλείο πρέπει να έχει συνδέσει (`link`) στο runtime με έναν από τους εξής τρόπους και με την αντίστοιχη σειρά προτεραιότητας:

- **Στατικά** με το εργαλείο να θέτει τη συνάρτηση ως καθολική (`global`). Έτσι είναι διαθέσιμη για το `runtime` ώστε να κληθεί από το `runtime`.
- **Δυναμικά** μέσω βιβλιοθήκης (`dynamically-linked library`) που παραθέτει (`include`) τον ορισμό της υλοποίησης της συνάρτησης, στο χώρο διευθύνσεων της εφαρμογής `OpenMP`.
- Επίσης **δυναμικά**, ο χρήστης παρέχει το μονοπάτι της βιβλιοθήκης του εργαλείου που υπάρχει η υλοποίηση της συνάρτησης, μέσω της μεταβλητής περιβάλλοντος **`OMP_TOOL_LIBRARIES (ICV)`**. Η διεπαφή φορτώνει και συνδέει τη συνάρτηση στο runtime, μέσω των συναρτήσεων `dlopen` και `load_symbol`.

Στη περίπτωση που, η κλήση της συνάρτησης `ompt_start_tool` επιστρέψει έγκυρο αποτέλεσμα `ompt_start_tool_result` (ή `ret`, κόμβος 9 – Σχήμα 3.1), δηλαδή μη-μηδενικό (`non-null`), η διεπαφή σταματάει να ψάχνει για εργαλεία και καλεί τη συνάρτηση αρχικοποίησης του εργαλείου (`initialize`), που παρέχεται στο αποτέλεσμα (κόμβος 10 – Σχήμα 3.1).

4.3.2 Initialize

Η συνάρτηση **`initialize`** είναι υπεύθυνη για την αποθήκευση των συναρτήσεων της διεπαφής από το εργαλείο, οι οποίες παρέχονται μέσω του πίνακα `lookup_table` και την καταχώρηση των συναρτήσεων `callback`. Τα δύο αυτά αντικείμενα αναλύονται περισσότερο στις ενότητες 4.4 και 4.5, αντίστοιχα. Η "βοηθητική" συνάρτηση `tool_initialize` υλοποιήθηκε με σκοπό να καλέσει τη συνάρτηση αρχικοποίησης του εργαλείου (`ret->initialize`) και να ελέγξει για την επιτυχή ενεργοποίηση του εργαλείου. Εφόσον η συνάρτηση αρχικοποίησης του εργαλείου επιστρέψει έγκυρο (`non-null`) αποτέλεσμα (`True`, κόμβος 11 – Σχήμα 3.1)

σημαίνει πως το εργαλείο ενεργοποιήθηκε, αποθήκευσε τις συναρτήσεις του πίνακα `lookup_table` και καταχώρησε τις συναρτήσεις `callback`, που επιθυμεί να χρησιμοποιήσει κατά την εκτέλεση, με την διεπαφή με απόλυτη επιτυχία. Σε αυτό το σημείο το εργαλείο και η υλοποίηση της διεπαφής θα παραμείνουν συνδεδεμένα και ενεργά για το υπόλοιπο της εκτέλεσης του προγράμματος ή μέχρι να κληθεί η συνάρτηση τερματισμού του εργαλείου (κόμβος 13 – Σχήμα 3.1). Η συνάρτηση αρχικοποίησης ακολουθεί την εξής γενική μορφή, βάσει των προδιαγραφών του προτύπου OpenMP:

```
int ompt_initialize(ompt_function_lookup_t lookup,   // Συνάρτηση διεπαφής για το lookup_table
                    int initial_device_num,          // Αριθμητικό αναγνωριστικό συσκευής εκτέλεσης
                    ompt_data_t *tool_data)          // Δείκτης στα δεδομένα του εργαλείου
```

Κώδικας 4.2 Παράδειγμα γενικής δομής συνάρτησης αρχικοποίησης εργαλείου

Η πρώτη παράμετρος της συνάρτησης (`ompt_function_lookup_t lookup`) παρέχει στο εργαλείο τον δείκτη (`pointer`) σε μια εξίσου σημαντική συνάρτηση, τη συνάρτηση `lookup`, η οποία είναι υπεύθυνη ώστε να παρέχει στο εργαλείο οποιαδήποτε συνάρτηση, από τον πίνακα με τις συναρτήσεις που υποστηρίζονται (`lookup_table`) από την υλοποίηση της διεπαφής `OMPT`, που επιθυμεί να χρησιμοποιήσει, κατά την εκτέλεση της εφαρμογής `OpenMP`.

Στη περίπτωση που δεν βρεθεί συμβατό εργαλείο, η διεπαφή απενεργοποιείται και παραμένει ανενεργή για το υπόλοιπο της εκτέλεσης του `runtime`. Αφαιρώντας ολοκληρωτικά οποιοδήποτε `overhead` δημιουργείται από την υλοποίηση της διεπαφής κατά την εκτέλεση της εφαρμογής `OpenMP`.

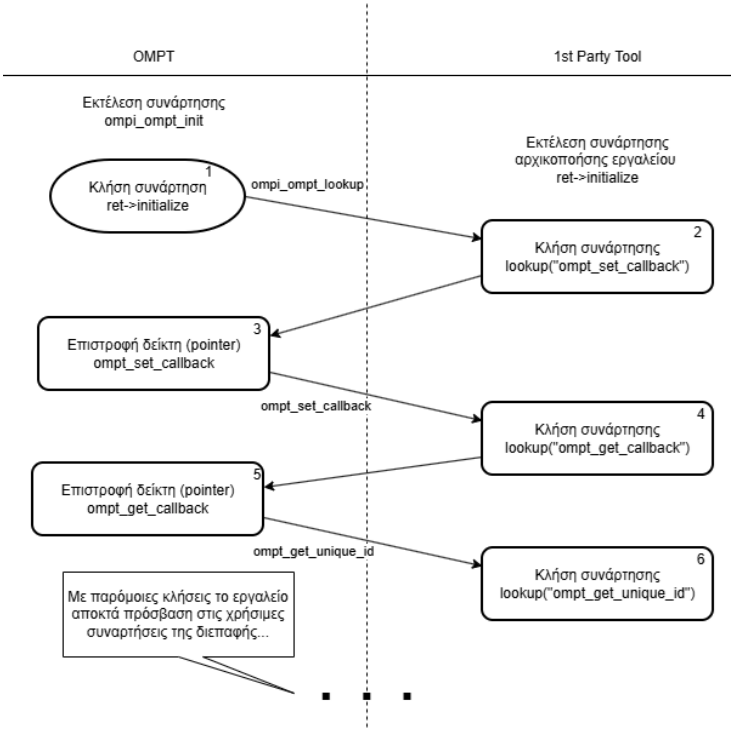
Το μονοπάτι που ψάχνει ο μηχανισμός φόρτωσης εργαλείου μπορεί να ελεγχθεί από τον χρήστη μέσω της μεταβλητής περιβάλλοντος **`OMP_TOOL_LIBRARIES (ICV)`**, η οποία καθορίζει το μονοπάτι για τη βιβλιοθήκη του εργαλείου. Η βιβλιοθήκη πρέπει να είναι σε μορφή `shared object` (`.so`). Αν δεν οριστεί, το `runtime` μπορεί να χρησιμοποιήσει προεπιλεγμένες βιβλιοθήκες ή να απενεργοποιήσει πλήρως τη λειτουργία της διεπαφής `OMPT`.

Αυτή η ευέλικτη προσέγγιση καθιστά δυνατή την ενεργοποίηση εργαλείων χωρίς εκ νέου μεταγλώττιση (`re-compile`) της εφαρμογής `OpenMP`, διευκολύνοντας τον πειραματισμό και την ανάλυση διαφορετικών σεναρίων εκτέλεσης.

4.4 Συναρτήσεις διεπαφής βάσει OpenMP

Οι προδιαγραφές του προτύπου `OpenMP` παραθέτουν κάποιες συναρτήσεις που είναι βασικές για την υποστήριξη των δυνατοτήτων που παρέχονται από την διεπαφή `OMPT`. Αυτές οι συναρτήσεις παρέχονται από την διεπαφή στο εργαλείο μέσω του

πίνακα συναρτήσεων (lookup_table), όπως αυτές υποστηρίζονται από την υλοποίηση της διεπαφής OMPT. Το εργαλείο αποκτά πρόσβαση σε αυτές βάσει του εξής διαγράμματος:



Σχήμα 4.1 Διάγραμμα ροής ανάκτησης συναρτήσεων διεπαφής

Μερικές συναρτήσεις του lookup_table υπήρχαν ήδη υλοποιημένες εσωτερικά στο ORT και απλά προωθούνται από τη διεπαφή στο εργαλείο:

- ompt_get_num_places και omp_get_num_places
- ompt_get_place_num και omp_get_place_num

Ακολουθεί ένας πίνακας με όλες τις συναρτήσεις των προδιαγραφών του προτύπου OpenMP με στήλες που υποδεικνύουν την υποστήριξη της αντίστοιχης συνάρτησης από την υλοποίηση της διεπαφής στον OMPi και την αντιστοιχία τους σε συσκευές host και target/device:

Function Name	Implemented	Host	Device
ompt_enumerate_states	✓	✓	X
ompt_enumerate_mutex_impls	✓	✓	X
ompt_set_callback	✓	✓	X
ompt_get_callback	✓	✓	X
ompt_get_thread_data	✓	✓	X
ompt_get_num_places	✓	✓	X

ompt_get_place_proc_ids	✓	✓	X
ompt_get_place_num	✓	✓	X
ompt_get_partition_place_nums	✓	✓	X
ompt_get_proc_id	✓	✓	X
ompt_get_state	✓	✓	X
ompt_get_parallel_info	✓	✓	X
ompt_get_task_info	✓	✓	X
ompt_get_task_memory	✓	✓	X
ompt_get_num_devices	✓	✓	X
ompt_get_num_procs	✓	✓	X
ompt_get_target_info	✓	✓	X
ompt_get_unique_id	✓	✓	X
ompt_finalize_tool	✓	✓	X
ompt_get_device_num_procs	X	X	✓
ompt_get_device_time	X	X	✓
ompt_translate_time	X	X	✓
ompt_set_trace_ompt	X	X	✓
ompt_set_trace_native	X	X	✓
ompt_start_trace	X	X	✓
ompt_pause_trace	X	X	✓
ompt_flush_trace	X	X	✓
ompt_stop_trace	X	X	✓
ompt_advance_buffer_cursor	X	X	✓
ompt_get_record_type	X	X	✓
ompt_get_record_ompt	X	X	✓
ompt_get_record_native	X	X	✓
ompt_get_record_abstract	X	X	✓

Πίνακας 4.1 Συναρτήσεις διεπαφής

Οι συναρτήσεις που είναι βασισμένες στις συσκευές (devices) δεν ήταν εφικτό να υλοποιηθούν καθώς δεν υπάρχει η κατάλληλη υποδομή για να ανακτηθούν τα δεδομένα που χρειάζονται. Δυστυχώς οι προσθήκες και αλλαγές για να γίνει εφικτή η

υλοποίηση αυτών των συναρτήσεων είναι σε ένα κρίσιμο και περίπλοκο σημείο του ORT (device offload), στο οποίο απαιτείται βαθιά γνώση και κατανόηση του OMPi και της διαχείρισης των συσκευών. Κάτι που είναι εκτός της έκτασης και του σκοπού αυτής της διπλωματικής εργασίας. Περισσότερες σχετικές πληροφορίες βρίσκονται στην Ενότητα 6.2.1.

Μερικές από τις πιο σημαντικές συναρτήσεις που υλοποιήθηκαν είναι οι εξής:

- **ompt_set_callback**

Καταχωρεί στο δοσμένο, ως παράμετρο, γεγονός τη συνάρτηση callback που δόθηκε ως παράμετρος. Επιστρέφει μεταβλητή τύπου ompt_set_callback_result_t που παίρνει τις τιμές ompt_set_sometimes, ompt_set_sometimes_paired, ompt_set_always, ompt_set_never και ενημερώνουν το εργαλείο για την κατάσταση της καταχώρησης του γεγονότος αντίστοιχα με την υποστήριξη της υλοποίησης του runtime.

- **ompt_get_callback**

Επιστρέφει τον δείκτη (pointer) της συνάρτησης, που είναι καταχωρημένη για το γεγονός, που δόθηκε, ως παράμετρος. Επιστρέφει μηδενική τιμή, αν δεν υπάρει καταχωρημένη συνάρτηση για αυτό το γεγονός. Έτσι, ένα εργαλείο μπορεί να επιβεβαιώσει την επιτυχή καταχώρηση ενός callback για το γεγονός, που το ενδιαφέρει να παρακολουθήσει, κατά την εκτέλεση.

- **ompt_enumerate_states**

Είναι υπεύθυνη για την αρίθμηση των καταστάσεων, που υλοποιεί η διεπαφή, από το εργαλείο ώστε να έχει επίγνωση και να μπορεί να τα αντιστοιχεί κατά την εκτέλεση της εφαρμογής. Καθώς οι καταστάσεις είναι ορισμένες ως enum μπορούν να χρησιμοποιηθούν τα αναγνωριστικά τους για την αρίθμηση τους. Στη πρώτη κλήση της θα πρέπει να γίνει με τη αρχική κατάσταση, που έχει αναγνωριστικό το μηδέν (0). Με κάθε επόμενη κλήση επιστρέφει την επόμενη στη σειρά κατάσταση από αυτή που δόθηκε, ως παράμετρος.

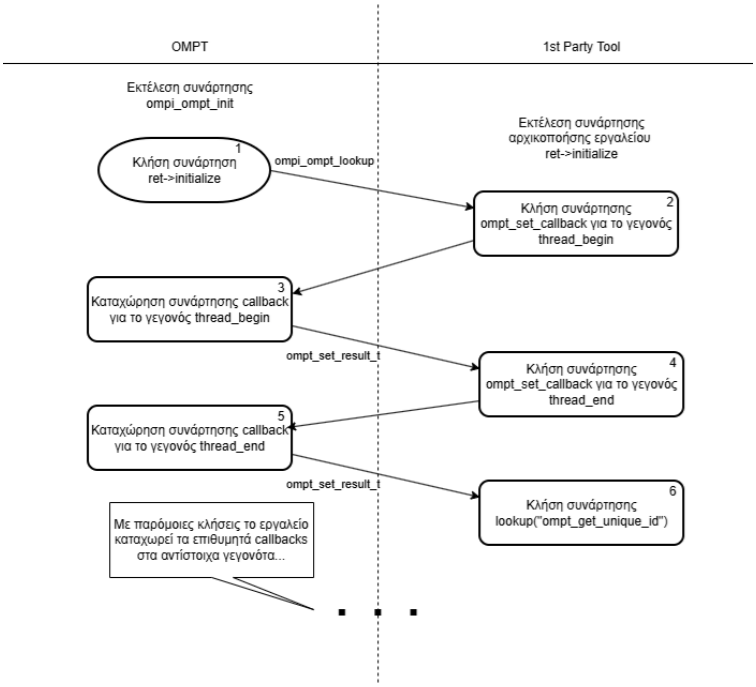
- **ompt_get_proc_id**

Επιστρέφει το αριθμητικό αναγνωριστικό (identifier) του πυρήνα στον οποίο βρίσκεται το τρέχων νήμα της εκτέλεσης.

4.5 Callbacks

Οι συναρτήσεις callbacks είναι θεμελιώδη για την άμεση και απευθείας επικοινωνία μεταξύ ενός εργαλείου (1st party tool) και της υλοποίησης της διεπαφής OMPT. Η συνάρτηση callback υλοποιείται από το εργαλείο, την οποία

καταχωρεί στην υλοποίηση της διεπαφής, ως δείκτης (pointer), μαζί με το αντίστοιχο γεγονός (trigger event), που το εργαλείο επιθυμεί να παρακολουθήσει, κατά την εκτέλεση. Όταν κατά την εκτέλεση συμβεί το γεγονός αυτό (event triggered), η υλοποίηση της διεπαφής θα καλέσει τη συνάρτηση callback, που καταχώρησε το εργαλείο κατά την αρχικοποίηση του.



Σχήμα 5.1 Διάγραμμα ροής καταχώρησης callback

Μια κοινή χρήση από εργαλεία για τα γεγονότα ompt_parallel_begin και ompt_parallel_end είναι να καταχωρούν σε αυτά συναρτήσεις callback που καταγράφουν την κατάσταση των νημάτων, της μνήμης και υπολογίζουν το χρόνο εκτέλεσης μιας παράλληλης περιοχής ή και παραπάνω παράλληλων περιοχών μιας εφαρμογής OpenMP. Με χρήση περισσότερων γεγονότων ένα εργαλείο μπορεί να επιτύχει όλες τις δυνατότητες που αναφέρονται στην ενότητα 3.2.2. Σε αυτή την ενότητα θα αναλύσουμε τις δύο κύριες διαδικασίες καταχώρησης και κλήσης των συναρτήσεων callback, που παρέχει η διεπαφή OMPT για χρήση από τα εργαλεία. Τα γεγονότα είναι τα εξής:

Callback signature (event)	Implemented	Mandatory
ompt_callback_thread_begin	✓	✓
ompt_callback_thread_end	✓	✓
ompt_callback_parallel_begin	✓	✓
ompt_callback_parallel_end	✓	✓
ompt_callback_task_create	✓	✓

ompt_callback_task_schedule	✓	✓
ompt_callback_implicit_task	✓	✓
ompt_callback_target	✓	✓
ompt_callback_target_emi	X	✓
ompt_callback_target_data_op	X	✓
ompt_callback_target_data_op_emi	X	✓
ompt_callback_target_submit	X	✓
ompt_callback_target_submit_emi	X	✓
ompt_callback_control_tool	X	✓
ompt_callback_device_initialize	X	✓
ompt_callback_device_finalize	X	✓
ompt_callback_device_load	X	✓
ompt_callback_device_unload	X	✓
ompt_callback_error	✓	✓
ompt_callback_sync_region_wait	X	X
ompt_callback_mutex_released	X	X
ompt_callback_dependences	X	X
ompt_callback_task_dependence	X	X
ompt_callback_work	X	X
ompt_callback_master (deprecated)	X	X
ompt_callback_masked	X	X
ompt_callback_target_map	X	X
ompt_callback_target_map_emi	X	X
ompt_callback_sync_region	X	X
ompt_callback_reduction	X	X

Πίνακας 4.2 Υποστηριζόμενα γεγονότα/callacks

Όπως φαίνεται κάποια από τα υποχρεωτικά `callbacks` δεν υποστηρίζονται στην υλοποίηση της διεπαφής `OMPT` στον `OMPi`. Όλα αυτά είναι `callbacks` που απευθύνονται σε γεγονότα των συσκευών (`devices`). Αυτό συμβαίνει καθώς δεν υπάρχει η απαραίτητη υποστήριξη/υποδομή στο ORT ώστε να μπορεί να γίνει η κλήση των `callback` τη κατάλληλη στιγμή, που συμβαίνει το γεγονός (`event triggered`), κατά την εκτέλεση στη συσκευή. Ενώ οι αλλαγές στην υλοποίηση της διεπαφής που χρειάζονται είναι απλές και ελάχιστες, το σημείο που πρέπει να υλοποιηθεί η υποδομή

για την κλήση των callback, βρίσκεται σε ένα κρίσιμο σημείο του ORT για τις συσκευές, οπότε απαιτεί βαθιά γνώση και κατανόηση του OMPi, που είναι εκτός της έκτασης και του σκοπού αυτής της διπλωματικής εργασίας. Περισσότερες πληροφορίες βρίσκονται στην ενότητα 6.2.1.

Δύο βασικά γεγονότα από αυτά που υλοποιήθηκαν είναι τα ompt_callback_parallel_begin και ompt_callback_parallel_end, τα οποία συμβαίνουν όταν η εκτέλεση της εφαρμογής εισέρχεται και εξέρχεται από μια παράλληλη περιοχή (parallel region), αντίστοιχα. Σε κάθε ένα από αυτά τα γεγονότα ένα εργαλείο μπορεί να καταχωρήσει από μια συνάρτηση callback, η οποία θα εκμεταλλεύεται τις δυνατότητες, που παρέχει, η υλοποίηση της διεπαφής OMPT ώστε να παρακολουθήσει και να αναλύσει την εκτέλεση της εφαρμογής. Για παράδειγμα, τα callbacks που καταχωρούνται μπορούν να καλούν τις κατάλληλες συναρτήσεις της διεπαφής OMPT, ώστε να αποκτήσει το εργαλείο πληροφορίες για την παράλληλη περιοχή που εκτελείται, την κατάσταση των νημάτων και της εκτέλεσης. Επίσης, το εργαλείο μέσα στα ίδια callback μπορεί να κρατάει χρονικές μεταβλητές, οι οποίες είναι άμεσα συνδεδεμένες με την αντίστοιχη παράλληλη περιοχή, ώστε να μετρηθεί ο χρόνος εκτέλεσης της κάθε παράλληλης περιοχής με απόλυτη ακρίβεια.

4.5.1 Καταχώρηση callback

Η πρώτη διαδικασία είναι η καταχώρηση των συναρτήσεων callback στην υλοποίηση της διεπαφής από το εργαλείο. Μετά την επιτυχή επιστροφή της ompt_start_tool (κόμβος 9 – Σχήμα 3.1), το OpenMP runtime καλεί τη συνάρτηση **initialize** (κόμβος 10 – Σχήμα 3.1), που δόθηκε από το εργαλείο μέσα στην μεταβλητή ompt_start_tool_result (ret, κόμβος 9 – Σχήμα 3.1). Η συνάρτηση initialize, είναι υπεύθυνη για την **καταχώρηση των callbacks** που επιθυμεί να χρησιμοποιήσει το εργαλείο κατά την εκτέλεση της εφαρμογής OpenMP.

Κατά την αρχικοποίηση του εργαλείου (κόμβος 10 - Σχήμα 3.1), η υλοποίηση της διεπαφής επιτρέπει στο εργαλείο, μέσω των συναρτήσεων ompt_set_callback και ompt_get_callback, να διαχειριστεί την καταχώρηση των συναρτήσεων callback στα κατάλληλα γεγονότα, ώστε να παρακολουθήσει και να αναλύσει την εκτέλεση της εφαρμογής OpenMP.

Στο σημείο αυτό (κόμβος 10 - Σχήμα 3.1), το εργαλείο δηλώνει ποια γεγονότα του χρόνου εκτέλεσης επιθυμεί να παρακολουθεί, καταχωρώντας τη κατάλληλη συνάρτηση callback σε κάθε ένα από αυτά τα γεγονότα. Η καταχώρηση

πραγματοποιείται μέσω της συνάρτησης ompt_set_callback, η οποία ακολουθεί, βάση του προτύπου, τη γενική μορφή:

```
ompt_set_callback(ompt_callbacks_t event_type, (ompt_callback_t) handler_function);
```

Κώδικας 4.3 Παράδειγμα γενικής δομής συνάρτησης set_callback

Όπου event_type καθορίζει το είδος του γεγονότος και handler_function είναι η συνάρτηση που θα καταχωρηθεί για να καλείται όποτε συμβαίνει το γεγονός.

Για κάθε γεγονός η διεπαφή, μέσω της συνάρτησης ompt_set_callback, λαμβάνει τις συναρτήσεις callback, ως δείκτες, που το εργαλείο θέλει να καταχωρήσει μαζί με το αντίστοιχο γεγονός. Η υλοποίηση της διεπαφής όταν παραλάβει κάποιο τέτοιο ζεύγος event και callback, ενημερώνει μια εσωτερική δομή καταγράφοντας το ζευγάρι αυτό. Με αποτέλεσμα όταν συμβεί το γεγονός αυτό θα κληθεί η συνάρτηση callback, που είναι καταχωρημένη για το γεγονός αυτό, παρέχοντας στο εργαλείο πρόσβαση σε κρίσιμες πληροφορίες κατά την εκτέλεση, όπως:

- Αναγνωριστικά (identifiers) νημάτων και tasks,
- Δείκτες (pointers) προς parallel regions,
- Χρόνους εκκίνησης και ολοκλήρωσης,
- Μεταδεδομένα (metadata) που σχετίζονται με το περιβάλλον εκτέλεσης.

Το εργαλείο μπορεί να επιλέξει υποσύνολο των διαθέσιμων callbacks, ανάλογα με τις ανάγκες του. Για παράδειγμα, ένα profiler εργαλείο μπορεί να καταχωρήσει μόνο συναρτήσεις callback σε γεγονότα που σχετίζονται με παράλληλες περιοχές (parallel regions) και tasks, ενώ ένα tracing εργαλείο μπορεί να καταχωρήσει συναρτήσεις callback σε επιπλέον γεγονότα συγχρονισμού ή αλλαγής κατάστασης των νημάτων.

Η σωστή καταχώρηση των συναρτήσεων callbacks επιβεβαιώνεται μέσω του αποτελέσματος που επιστρέφει η συνάρτηση ompt_set_callback. Εφόσον, η τιμή επιστροφής υποδεικνύει επιτυχία, το εργαλείο είναι πλέον πλήρως συνδεδεμένο με την υλοποίηση της διεπαφής και έτοιμο να παρακολουθήσει τα επιθυμητά γεγονότα.

Με την ολοκλήρωση της καταχώρησης, η υλοποίηση της διεπαφής και το εργαλείο θεωρούνται ενεργά και η επικοινωνία μεταξύ τους θα συνεχίσει καθ' όλη τη διάρκεια ζωής του προγράμματος ή μέχρι την κλήση της συνάρτησης finalize, η οποία πραγματοποιεί την αποδέσμευση των πόρων και τον τερματισμό της διεπαφής.

Παρακάτω βρίσκεται ο κώδικας των δύο συναρτήσεων της διεπαφής ompt_get_callback και ompt_set_callback, χωρίς βέβαια τις γραμμές κώδικα, που είναι υπεύθυνες για την εκτύπωση των μηνυμάτων στο τερματικό:

```c
ompt_set_result_t ompt_set_callback(ompt_callbacks_t event, ompt_callback_t callback)
{
    /* Έλεγχος ενεργής διεπαφής και ενεργού εργαλείου */
    if (check_before_exec(TOOLINIT) == 0) return ompt_set_error;

    /* Καταχώρηση συνάρτησης callback */
    ompi_ompt_callbacks[event-1].callback = callback;

    /* Επιστροφή κατάλληλου ompt_set_result, ανάλογα της υλοποίησης */
    return ompi_ompt_callbacks[event-1].result;
}

int ompt_get_callback(ompt_callbacks_t event, ompt_callback_t *callback)
{
    /* Έλεγχος ενεργής διεπαφής και ενεργού εργαλείου */
    if (check_before_exec(TOOLINIT) == 0) return 0;

    /* Έλεγχος καταχωρημένης συνάρτησης για το γεγονός event */
    if (ompi_ompt_callbacks[event-1].callback != NULL) {
        /* Επιστροφή της καταχωρημένης συνάρτησης ως δείκτη, μέσω αναφοράς */
        *callback = ompi_ompt_callbacks[event-1].callback;
        return 1;
    }

    /* Δεν υπάρχει καταχωρημένη συνάρτηση */
    return 0;
}
```

Κώδικας 4.4 Κώδικας υλοποίησης συναρτήσεων ompt_set_callback και ompt_get_callback

4.5.2 Ενεργοποίηση (Invoke) callback

Η δεύτερη διαδικασία είναι η ενεργοποίηση/κλήση (invoke) των καταγεγραμμένων callback, τη στιγμή της εκτέλεσης που θα συμβούν τα αντίστοιχα γεγονότα για τα οποία καταγράφηκαν, στην αρχικοποίηση του εργαλείου. Στη "βοηθητική" συνάρτηση ompt_invoke_callback υλοποιήθηκε η διαδικασία ενεργοποίησης των συναρτήσεων callback από την υλοποίηση της διεπαφής. Έχει μεταβλητές παραμέτρους με την μοναδική αναγκαία να είναι το γεγονός που μόλις συνέβη, ώστε να μπορεί να κληθεί η συνάρτηση callback που είναι καταχωρημένη για το γεγονός αυτό. Αυτές οι μεταβλητές παράμετροι πρέπει να αντιστοιχούν ένα προς ένα με την κλήση της συνάρτησης callback καθώς ο τύπος (signature) της κάθε συνάρτησης είναι διαφορετικός και πρέπει να ακολουθεί συγκεκριμένη γενική μορφή, όπως παρέχεται στις προδιαγραφές του προτύπου OpenMP. Για παράδειγμα η συνάρτηση που καταχωρείται για το γεγονός ompt_callback_thread_begin πρέπει να έχει την εξής γενική μορφή:

```c
typedef void (*ompt_callback_thread_begin_t) (
    ompt_thread_t thread_type,    // Είδος νήματος (initial, worker)
    ompt_data_t *thread_data      // Δείκτης στα δεδομένα του εργαλείου για το νήμα
);
```

Κώδικας 4.5 Παράδειγμα γενικής δομής συνάρτησης callback

Έτσι ο μεγαλύτερος όγκος γραμμών κώδικα της συνάρτησης αυτής είναι ένα switch case. Χρησιμοποιήθηκε switch αντί ενός βρόγχου for καθαρά για λόγους

επίδοσης. Στο κώδικα που χρησιμοποιείται ως παράδειγμα έχουν αφαιρεθεί όλες εκτός από μια περίπτωση (case) για λόγους σύμπτυξης, καθώς η μόνη λειτουργία του κάθε case είναι να ανακτήσει τις μεταβλητές παραμέτρους μέσω va_arg και να τις περάσει στην κλήση (invokation) της συνάρτησης callback. Στον κώδικα 4.7, συνάρτηση ompt_invoke_callback, αρχικά ελέγχεται πως η διεπαφή και το εργαλείο είναι ενεργά, μετά ελέγχεται πως υπάρχει καταχωρημένη συνάρτηση για το γεγονός, που μόλις συνέβη, και εισέρχεται στο βρόγχο switch ώστε από τις μεταβλητές παραμέτρους να ανακτήσει τις κατάλληλες μεταβλητές για την καταχωρημένη συνάρτηση callback και να την καλέσει. Στο τέλος του βρόγχου switch αν δεν είναι αληθή καμία περίπτωση (case) σημαίνει πως το γεγονός που δόθηκε ως παράμετρος δεν υποστηρίζεται και επιστρέφεται μηδέν. Αν μια από τις περιπτώσεις, του βρόγχου switch, είναι αληθή, τότε σημαίνει πως έχει κληθεί η καταχωρημένη συνάρτηση callback για το γεγονός, που συνέβη, και επιστρέφεται ένα.

```
int ompt_invoke_callback(ompt_callbacks_t event, ...)
{
    /* Έλεγχος ενεργής διεπαφής και ενεργού εργαλείου */
    if (check_before_exec(NOTOOLINIT) == 0) return 0;

    /* Έλεγχος καταχωρημένης συνάρτησης για το γεγονός event */
    if (ompi_ompt_callbacks[event-1].callback == NULL) return 0;
    ort_eecb_t *t = __MYCB;

    /* Ανάκτηση των κατάλληλων δεδομένων και κλήση της καταχωρημένης συνάρτησης */
    switch (event)
    {
        /* Γεγονός εκκίνησης νήματος (ompt_callback_thread_begin) */
        case ompt_callback_thread_begin: {
            /* Ανάκτηση δείκτη καταχωρημένης συνάρτησης */
            GETCALLBACK(ompt_callback_thread_begin_t);
            /* Ανάκτηση δεδομένων από τις μεταβλητές παραμέτρους */
            va_list args;
            va_start(args, event);
            ompt_thread_t type = va_arg(args, ompt_thread_t);
            ompt_data_t *thread_data = va_arg(args, ompt_data_t *);
            va_end(args);
            /* Κλήση της καταχωρημένης συνάρτησης */
            callback(type, thread_data);
            break;
        }

        ... // Υπόλοιπες περιπτώσεις (cases)

        /* Δεν υποστηρίζεται το γεγονός που δόθηκε ως παράμετρος */
        default: return 0; break;
    }
    /* Επιστροφή σήματος επιτυχίας */
    return 1;
}
```

Κώδικας 4.6 Υλοποίηση "βοηθητικής " συνάρτησης ompt_invoke_callback

Η "βοηθητική" συνάρτηση ompt_invoke_callback είναι διαθέσιμη στο ευρύτερο κομμάτι του ORT, μέσω αρχείων κεφαλίδας (header), ώστε να μπορεί να κληθεί εσωτερικά στο ORT τις στιγμές που συμβαίνουν τα γεγονότα, τα οποία υποστηρίζονται από την υλοποίηση, όπως αυτά αναλύονται στον Πίνακα 4.2. Σημαντικό μέλημα της υλοποίησης είναι η αποφυγή ανεπιθύμητου overhead λόγο της λειτουργίας της διεπαφής κατά τον χρόνο μετάφρασης και εκτέλεσης της εφαρμογής OpenMP. Στο επόμενο παράδειγμα κώδικα φαίνονται οι δύο μηχανισμοί υπεύθυνα για την αποφυγή ανεπιθύμητου overhead, περισσότερα στην ενότητα 4.6:

- Κατά τη **μετάφραση**: Αν η μεταβλητή περιβάλλοντος OMP_TOOL είναι ανενεργή τότε η μεταβλητή προ-επεξεργαστή OMPT_ENABLED δεν ορίζεται, αποτρέποντας τη μετάφραση του αρχείοτ ompt.c και των γραμμών κώδικα της υλοποίησης, στα υπόλοιπα αρχεία του ORT, που είναι υπεύθυνες για την κλήση των callback.
- Κατά την **εκτέλεση**: Αν η μεταβλητή περιβάλλοντος OMP_TOOL είναι ενεργή αλλά δεν ενεργοποιήθηκε επιτυχώς ένα εργαλείο. Με τη συνάρτηση στον Κώδικα 4.10, και μια οδηγία if, αποτρέπεται η εκτέλεση των συναρτήσεων που παρέχονται από την υλοποίηση της διεπαφής OMPT όταν αυτή είναι ανενεργή.

Το παράδειγμα εκτός αυτών των μηχανισμών αναδεικνύει πόσο εύκολο είναι να ανακτηθούν τα δεδομένα, που χρειάζεται να παρέχει η διεπαφή στο εργαλείο, καθώς υπάρχει η κατάλληλη υποδομή στο runtime του OMPi. Το παράδειγμα βρίσκεται στο σημείο εκκίνησης ενός νέου νήματος από το runtime το οποίο αναθέτει τα απαραίτητα δεδομένα (flags, variables) ανάλογα το είδος του task (initial/worker) και τη συγκεκριμένη περιοχή (παράλληλη/σειριακή), που δημιουργείται το νήμα, και καλεί τα κατάλληλα callbacks.

```
#ifdef OMPT_ENABLED
    if (OMPTACTIVE) {
        /* Ανάκτηση πληροφοριών για το νήμα */
        ort_eecb_t *t = __MYCB;
        ompt_thread_t type = (t->thread_num == 0) ?
                             ompt_thread_initial : ompt_thread_worker;
        /* Κλήση της καταχωρημένης συνάρτησης για το γεγονός εκκίνησης νήματος */
        ompt_invoke_callback(ompt_callback_thread_begin, type, &(t->thread_data));
    }
#endif /* OMPT_ENABLED */
```

Κώδικας 4.7 Υλοποίηση κλήσης καταχωρημένου callback

4.6 Μηχανισμοί αποφυγής overhead

Όπως φαίνεται από τα παραδείγματα κώδικα υπάρχουν δύο κύριοι μηχανισμοί αποφυγής ανεπιθύμητου overhead. Η χρήση οδηγιών προ-επεξεργαστή, εντολών if και εσωτερικών μεταβλητών του ORT, επιβεβαιώνουμε πως η διεπαφή και το εργαλείο

είναι ενεργά. Ο κάθε μηχανισμός είναι υπεύθυνος για να ελέγχει και να αποφεύγει το ανεπιθύμητο overhead κατά τη μεταγλώττιση και την εκτέλεση, αντίστοιχα.

4.6.1 Οδηγίες προ-επεξεργαστή (pre-processor)

Η μεταβλητή προ-επεξεργαστή OMPT_ENABLED ορίζεται μόνο αν η μεταβλητή περιβάλλοντος OMP_TOOL είναι enabled και η οδηγία προ-επεξεργαστή OMPTACTIVE ορίζεται ώστε να ελέγχεται αν υπάρχει ενεργό εργαλείο, άρα η διεπαφή είναι ενεργή. Όλες οι γραμμές του κώδικα, που υλοποιήθηκαν στο ORT και είναι υπεύθυνες για τις λειτουργίες της υλοποίησης της διεπαφής OMPT, όπως και ολόκληρο το αρχείο ompt.c, είναι μέσα σε οδηγίες προ-επεξεργαστή #ifdef OMPT_ENABLED. Έτσι αποφεύγουμε οποιοδήποτε ανεπιθύμητο overhead κατά την μετάφραση.

```
#ifdef OMPT_ENABLED
/* Επιστρέφει True (1) αν η διεπαφή και το εργαλείο είναι ενεργά, αλλιίως False (0) */
#define OMPTACTIVE (ort->icvs.tool == 1 && ompi_ompt_is_active())
/* Έκθεση της συνάρτησης στα αρχεία του ORT που θα χρησιμοποιήσουν το macro */
int ompi_ompt_is_active();
#endif /* OMPT_ENABLED */
```

Κώδικας 4.8 Υλοποίηση οδηγιών προ-επεξεργαστή

4.6.2 Έλεγχος ενεργοποίησης εργαλείου

Η "βοηθητική" συνάρτηση ompi_ompt_is_active είναι μια απλή συνάρτηση που επιστρέφει ένα (1) αν υπάρχει ενεργό συμβατό εργαλείο, αλλιώς επιστρέφει μηδέν (0). Όλες οι γραμμές κώδικα που υλοποιήθηκαν στο ORT και είναι υπεύθυνες για τις λειτουργίες της υλοποίησης της διεπαφής OMPT είναι μέσα σε βρόγχους if(OMPTACTIVE). Συγκεκριμένα, οι συναρτήσεις που υλοποιήθηκαν στο αρχείο ompt.c όταν καλούνται ελέγχουν αν υπάρχει ενεργό εργαλείο με την "βοηθητική" συνάρτηση check_before_exec. Η συνάρτηση αυτή επιστρέφει True (1) αν η διεπαφή είναι συνδεδεμένη με ένα ενεργό εργαλείο, αλλιώς επιστρέφει False (0). Έτσι αποφεύγουμε τυχών ανεπιθύμητο overhead κατά την εκτέλεση όταν η διεπαφή είναι ανενεργή.

```
int ompi_ompt_is_active()
{
    return (ompi_ompt_state == ompi_tool_active) ? 1 : 0;
}
```

Κώδικας 4.9 Υλοποίηση συνάρτησης ompi_ompt_is_active

4.7 Τερματισμός διεπαφής και εργαλείου

Ο τερματισμός γίνεται μέσω της "βοηθητικής" συνάρτησης του OMPT ompt_finalize_tool, που μπορεί να κληθεί από το εργαλείο καθώς είναι μια συνάρτηση του lookup_table. Το ORT επίσης μπορεί να τη καλέσει ανά πάσα στιγμή,

καθώς σε περίπτωση σφάλματος πρέπει η διεπαφή και το εργαλείο να τερματίσουν ομαλά. Ενώ είναι προγραμματισμένη βάσει των προδιαγραφών του προτύπου OpenMP να κληθεί από το εργαλείο κατά την λήξη της εκτέλεσης. Δεν είναι κάτι περίπλοκο απλά αποδεσμεύει τους πόρους του συστήματος που δεσμεύτηκαν για τα δεδομένα και την επικοινωνία του εργαλείου και της υλοποίησης της διεπαφής OMPT, κατά την εκτέλεση.

Κεφάλαιο 5. Παραδείγμα Χρήσης της Διεπαφής

Στο κεφάλαιο αυτό θα αναλυθεί ένα παράδειγμα χρήσης της υλοποίησης της διεπαφής OMPT από ένα εργαλείο (1st party tool). Το εργαλείο που χρησιμοποιήθηκε είναι το **Caliper**[8].

Το Caliper αποτελεί ένα ευέλικτο πλαίσιο ενορχήστρωσης και ανάλυσης απόδοσης για παράλληλες εφαρμογές υψηλών επιδόσεων και υποστηρίζει το OpenMP μέσω της διεπαφής OMPT. Βασιζόμενο στους μηχανισμούς καταχώρησης και ενεργοποίησης callbacks της διεπαφής, είναι σε θέση να παρακολουθεί τη δημιουργία και εκτέλεση παράλληλων περιοχών, τη δραστηριότητα των νημάτων και τη συμπεριφορά των tasks OpenMP, χωρίς να απαιτείται εκτενής τροποποίηση του πηγαίου κώδικα της εφαρμογής OpenMP. Η σχεδίασή του δίνει έμφαση στη χαμηλή επιβάρυνση (low overhead) και στη δυνατότητα συνδυασμού πληροφοριών χρόνου εκτέλεσης από τη βιβλιοθήκη του runtime με επιπλέον μέτα-δεδομένα (metadata) που εισάγει ο τελικός χρήστης. Μέσω αυτής της προσέγγισης, το Caliper υποστηρίζει τόσο ανάλυση (profiling) παράλληλων περιοχών όσο και καταγραφή επιλεγμένων γεγονότων εκτέλεσης, επιτρέποντας την εξαγωγή μετρικών απόδοσης προσαρμοσμένων στις ανάγκες της ανάλυσης. Η χρήση της διεπαφής καθιστά το εργαλείο φορητό ως προς τις διάφορες υλοποιήσεις OpenMP και κατάλληλο για ενσωμάτωση σε ευρύτερες ροές ανάλυσης απόδοσης περιβάλλοντων υψηλής υπολογιστικής/επεξεργαστικής απόδοσης (High Performance Computing – HPC).

Επίσης, για τον σκοπό του παραδείγματος υλοποιήθηκε μια απλή παράλληλη εφαρμογή OpenMP primes.c, η οποία υπολογίζει τους πρώτους αριθμούς που είναι μικρότεροι από το ένα εκατομμύριο. Το Caliper αναλύει και μετράει πληροφορίες για την κατάσταση των νημάτων και της εκτέλεσης της εφαρμογής και στο τέλος της

εκτέλεσης εκτυπώνει στο τερματικό μια αναφορά για τους χρόνους εκτέλεσης των περιοχών εργασίας (work) και αναμονής (barrier) συνολικά για τα νήματα.

```c
void serial_primes(long int n) {
    long int i, num, divisor, quotient, remainder;;

    if (n < 2) return;
    count = 1;                      /* 2 is the first prime */
    lastprime = 2;

    for (i = 0; i < (n-1)/2; ++i) {    /* For every odd number */
        num = 2*i + 3;

        divisor = 1;
        do
        {
            divisor += 2;                /* Divide by the next odd */
            quotient  = num / divisor;
            remainder = num % divisor;
        } while (remainder && divisor <= quotient);  /* Don't go past sqrt */

        if (remainder || divisor == num) /* num is prime */
        {
            count++;
            lastprime = num;
        }
    }
}
```

```c
void openmp_primes(long int n) {
    long int i, num, divisor, quotient, remainder;

    if (n < 2) return;
    count = 1;                      /* 2 is the first prime */
    lastprime = 2;

    #pragma omp parallel for private(divisor,num,quotient,remainder) shared(i,count,lastprime) schedule(runtime)
    for (i = 0; i < (n-1)/2; ++i) {    /* For every odd number */
        num = 2*i + 3;

        divisor = 1;
        do
        {
            divisor += 2;                /* Divide by the next odd */
            quotient  = num / divisor;
            remainder = num % divisor;
        } while (remainder && divisor <= quotient);  /* Don't go past sqrt */

        if (remainder || divisor == num) /* num is prime */
        {
            #pragma omp critical
            count++;
            if (lastprime < num) lastprime = num;
        }
    }
}
```

Κώδικας 5.1 Σειριακή (αριστερά) και Παράλληλη (δεξιά) υλοποίηση εφαρμογής primes.c

Όπως αναλύθηκε, το OpenMP επιτρέπει τον παραλληλισμό του βρόγχου for με τη προσθήκη ελάχιστων γραμμών κώδικα, στη προκειμένη μόνο δύο (2). Η πρώτη "#pragma omp parallel for", διατάζει τη βιβλιοθήκη του runtime (ORT) να εκτελέσει παράλληλα τον κόμβο for, που ακολουθεί. Τα clauses "shared(…)" και "private(…)", απευθύνονται στις μεταβλητές που υπάρχουν σαν παράμετροι και τις ορίζουν ως κοινόχρηστες ή ιδιωτικές ανά νήμα, αντίστοιχα. Τέλος, το clause, "schedule (runtime)" καθορίζει τον τρόπο διαμοιρασμού των επαναλήψεων του βρόγχου στα νήματα του συστήματος.

Ο κύριος λόγος ύπαρξης της εφαρμογής primes.c είναι η σύγκριση μεταξύ του παράλληλου και του σειριακού χρόνου εκτέλεσης του ίδιου υπολογισμού. Στη προκειμένη, υπολογίζουμε τους πρώτους αριθμούς μέχρι το ένα εκατομμύριο.

```c
double timeit(void (*func)(), long int n) {
    struct timeval start, end;
    gettimeofday(&start, NULL);
    func(n);
    gettimeofday(&end, NULL);
    return (double) (end.tv_usec - start.tv_usec) / 1000000
        + (double) (end.tv_sec - start.tv_sec);
}


int main()
{
    double s_time, p_time;

    printf("Serial and parallel prime number calculations:\n\n");

    s_time = timeit(serial_primes, UPTO);
    printf("[serial] count = %ld, last = %ld (time = %fs)\n", count, lastprime, s_time);

    p_time = timeit(openmp_primes, UPTO);
    printf("[openmp] count = %ld, last = %ld (time = %fs)\n\n", count, lastprime, p_time);

    return 0;
}
```

Κώδικας 5.2 Υλοποίηση σύγκρισης σειριακού και παράλληλου χρόνου εκτέλεσης

5.1 Παράδειγμα φόρτωσης εργαλείου

Όπως αναλύθηκε στην ενότητα 4.3 η ενεργοποίηση του 1st party tool και της υλοποίησης της διεπαφής OMPT, είναι υψίστης σημασίας για την ομαλή εκτέλεση του προγράμματος και λειτουργία του εργαλείου. Συγκεκριμένα στο τέλος της ενότητας 4.3.1 αναλύονται, επιγραμματικά, οι τρόποι με τους οποίους ένα εργαλείο συνδέεται με το runtime και ενεργοποιείται μέσω της συνάρτησης ompt_start_tool. Εδώ θα αναλυθούν οι υλοποιήσεις αυτών των τρόπων, αντίστοιχα με την προτεραιότητα και σειρά που ελέγχονται από τη διεπαφή, βάσει των προδιαγραφών του προτύπου OpenMP:

- **Στατικά:** Η εφαρμογή πρέπει να μεταφράζεται με έναν από τους εξής τρόπους:
    1. Περιέχει την οδηγία #include στο αρχείο πηγαίου κώδικα του εργαλείου.
       #include "/path/to/tool"
    2. Μετάφραση μαζί με το αρχείο πηγαίου κώδικα του εργαλείου.

       ompicc omp_app.c tool.c (or /path/to/tool.c)

- **Δυναμικά:** Η διεπαφή παρέχει τους εξής δύο μηχανισμούς:
    1. Μέσω των παραμέτρων μετάφρασης (compilation arguments):

       -L<path/to/tool_lib> -ltool_lib

    2. Αφού το εργαλείο υπάρχει ή μεταφραστεί, σε αρχείο βιβλιοθήκης shared object (.so). Η μεταβλητή περιβάλλοντος **OMP_TOOL_LIBRARIES (ICV)** ορίζεται ως το μονοπάτι αυτού του αρχείου.

       export OMP_TOOL_LIBRARIES="/path/top/tool.so"

Αν ένα εργαλείο έχει υλοποιήσει έναν από τους παραπάνω μηχανισμούς ώστε, η συνάρτηση ompt_start_tool να είναι διαθέσιμη και συνδεδεμένη με την υλοποίηση της διεπαφής κατά την εκκίνηση της εκτέλεσης και εφόσον ο μηχανισμός λειτούργησε με επιτυχία τότε το εργαλείο και η υλοποίηση της διεπαφής OMPT, θα παραμείνουν ενεργά σε όλη τη διάρκεια της εκτέλεσης.

Όταν η εκτέλεση της υλοποίησης της διεπαφής OMPT είναι verbose, τότε κατά την εκκίνηση της διεπαφής, η βιβλιοθήκη του runtime ενημερώνει, μέσω του τερματικού, τον τελικό χρήστη για την πρόοδο της διαδικασίας αρχικοποίησης του εργαλείου. Στο παρακάτω παράδειγμα, οι πρώτοι αριθμοί που υπολογίστηκαν ήταν μέχρι το εκατό χιλιάδες, καθώς ο υπολογισμός σειριακά με άνω όριο το ένα εκατομμύριο είναι πολύ χρονοβόρος.

```
[OMPT] CHECKING FOR TOOLS TO INITIALIZE...
[OMPT] Trying first way...
[OMPT] First way found an ompt_start_tool implementation, calling it...
[OMPT] First way ompt_start_tool returned non-NULL, initializing tool...
[OMPT] OMPT is trying to initialize tool...
[OMPT] Callback registered for event "ompt_callback_thread_begin" (0x7f052cf45a9d)
[OMPT] Callback registered for event "ompt_callback_thread_end" (0x7f052cf45c47)
[OMPT] Callback registered for event "ompt_callback_parallel_begin" (0x7f052cf45d43)
[OMPT] Callback registered for event "ompt_callback_parallel_end" (0x7f052cf45e0c)
[OMPT] Callback registered for event "ompt_callback_work" (0x7f052cf45ffb)
[OMPT] Callback registered for event "ompt_callback_target_map" (0x7f052cf46192)
[OMPT] Callback registered for event "ompt_callback_implicit_task" (0x7f052cf45ea3)
[OMPT] Tool initialized successfully! OMPT is active!
Serial and parallel prime number calculations:

[serial] count = 9592, last = 99991 (time = 0.063087s)
[openmp] count = 9592, last = 99991 (time = 0.033382s)

[OMPT] Finalizing the active tool
[OMPT] OMPT is finalized and inactive
```

Τερματικό 5.1 Παράδειγμα στατικής σύνδεσης εργαλείου Caliper (verbose)

5.2 Εκτέλεση με το Caliper

Αφού το εργαλείο ενεργοποιήθηκε με επιτυχία τότε κατά την εκτέλεση η υλοποίηση της διεπαφής `OMPT` θα το ενημερώνει καλώντας τις συναρτήσεις `callback`, που καταχωρήθηκαν από το εργαλείο, κατά την εκκίνηση. Έτσι το εργαλείο μπορεί να παρακολουθήσει με λεπτομέρεια την εκτέλεση του προγράμματος και με τις κατάλληλες εντολές εκτυπώνει μια σύντομη αλλά χρήσιμη αναφορά για τους χρόνους κάθε περιοχής που αναλύθηκε. Στη προκειμένη περίπτωση υπολογίστηκαν οι πρώτοι αριθμοί μέχρι το ένα εκατομμύριο και χωρίς να είναι `verbose` η υλοποίηση της διεπαφής `OMPT`.

```
[cs03176@paragon new-test]$ ompicc primes.c -L/home/cs03176/opt/lib64 -lcaliper
[cs03176@paragon new-test]$ CALI_CONFIG=openmp-report CALI_USE_OMPT=1 ./a.out
Serial and parallel prime number calculations:

[serial] count = 78498, last = 999983 (time = 1.673876s)
[openmp] count = 78498, last = 999983 (time = 0.135630s)

Path          #Threads Time (thread) Time (total) Work %     Barrier % Time (work) Time (barrier)
openmp_primes               0.134195     0.135619 100.000000
  parallel for       24     0.134195     0.135577  86.342647 13.657353    0.117061       0.018516
serial_primes                          1.673809
```

Τερματικό 5.2 Παράδειγμα εκτέλεσης primes.c και αναφορά Caliper

Κεφάλαιο 6. Συμπεράσματα & Μελλοντική Εργασία

6.1 Συμπεράσματα

Όπως αποδεικνύεται η διεπαφή OMP-Tools και οι δυνατότητες που παρέχει αποτελούν βάση ώστε εργαλεία να μπορούν να αναλύσουν, καταγράψουν και παρουσιάσουν λεπτομέρειες και συγκεκριμένα δεδομένα για τα νήματα, τη μνήμη και την κατάσταση του συστήματος χρόνου εκτέλεσης, σε πραγματικό χρόνο. Με αποτέλεσμα να αναδεικνύουν σημεία και τρόπους βελτίωσης των περιοχών παραλληλοποίησης του κώδικα οποιασδήποτε εφαρμογής OpenMP, υποστηρίζει έκδοση του OpenMP, νεότερη από την έκδοση 5.0. Όλα αυτά επιτυγχάνονται με το ελάχιστο επεξεργαστικό overhead και με ελάχιστες έως καθόλου αλλαγές στον αρχικό κώδικα της εφαρμογής, ανάλογα το εργαλείο και την υλοποίηση του OpenMP που χρησιμοποιείται.

Η ανάπτυξη 1st party tools, που εκμεταλλεύονται τις δυνατότητες της διεπαφής OMPT αναδεικνύει τη σημαντική πρόοδο που έχει επιτευχθεί στην παρακολούθηση και ανάλυση παράλληλων εφαρμογών OpenMP. Η ύπαρξη ενός τυποποιημένου, φορητού και επεκτάσιμου μηχανισμού επικοινωνίας μεταξύ της υλοποίησης του OpenMP runtime και των εργαλείων (1st party tools) καθιστά δυνατή τη δημιουργία εργαλείων που συνδυάζουν ακρίβεια, χαμηλό υπολογιστικό κόστος και ευελιξία ενσωμάτωσης σε διαφορετικά περιβάλλοντα εκτέλεσης. Τα παραδείγματα που παρουσιάστηκαν, στις ενότητες 3.2.3 και 5.2, δείχνουν ότι η διεπαφή OMPT καλύπτει ένα ευρύ φάσμα αναγκών, από την απλή μέτρηση επιδόσεων έως την πλήρη καταγραφή γεγονότων και συμπεριφοράς μνήμης.

Παράλληλα, η ενσωμάτωση με εργαλεία οπτικοποίησης και πλατφόρμες ανάλυσης επιδόσεων διευκολύνει και ενισχύει τη χρησιμότητα των δεδομένων που

παράγονται μέσω του OMPT, επιτρέποντας στους ερευνητές και τους προγραμματιστές να μετατρέπουν τον μεγάλο όγκο δεδομένων σε κατανοητές και αξιοποιήσιμες πληροφορίες. Έτσι, η διεπαφή OMPT συμβάλλει ουσιαστικά στην κατανόηση της παράλληλης συμπεριφοράς των εφαρμογών και στην βελτιστοποίηση της απόδοσης τους.

Συνολικά, η χρήση των δυνατοτήτων της διεπαφής OMPT μέσω των 1st party tools αποδεικνύει ότι το πρότυπο δεν αποτελεί απλώς ένα σύνολο μηχανισμών παρακολούθησης, αλλά ένα ολοκληρωμένο πλαίσιο ανάπτυξης εργαλείων ανάλυσης, το οποίο προάγει τη διαφάνεια και τη φορητότητα στο οικοσύστημα του OpenMP.

Η ωριμότητα και η σταθερότητά του το καθιστούν θεμέλιο για τη μελλοντική εξέλιξη των εργαλείων ανάλυσης (profiling), ενώ η συνδυαστική του χρήση με τη διεπαφή OMPD, που επικεντρώνεται στην αποσφαλμάτωση, διαμορφώνει ένα πλήρες και συνεκτικό περιβάλλον υποστήριξης ανάπτυξης για εφαρμογές υψηλών επιδόσεων.

6.2 Μελλοντική Εργασία

Ενώ η υλοποίηση που πραγματοποιήθηκε είναι επαρκή για να υποστηρίξει τις περισσότερες και πιο βασικές δυνατότητες της διεπαφής OMPT, υπάρχουν μερικά μέρη τα οποία δεν υλοποιήθηκαν ως μέρος της διπλωματικής εργασίας. Αυτά τα μέρη και μερικές ακόμα αλλαγές που θα μπορούσαν να προστεθούν στην υλοποίηση της διεπαφής OMPT ώστε να γίνει η ανάλυση και καταγραφή δεδομένων από εργαλεία με ακόμα μεγαλύτερη ακρίβεια και σε περισσότερα γεγονότα, άρα και σημεία, του χρόνου εκτέλεσης μιας παράλληλης εφαρμογής OpenMP. Στις επόμενες ενότητες θα αναλυθούν επεξηγηματικά μερικές από αυτές τις αλλαγές, προσθήκες ή επεκτάσεις της υλοποίησης.

6.2.1 Ολοκλήρωση υλοποίησης OMPT (devices, target etc)

Ώστε να ολοκληρωθεί πλήρως η υποστήριξη όλων των δυνατοτήτων της διεπαφής OMPT βάση του προτύπου OpenMP, θα πρέπει να γίνουν μικρές προσθέσεις και αλλαγές στον κώδικα του OMPi, οι οποίες είναι εκτός της έκτασης και του σκοπού αυτής της διπλωματικής εργασίας, καθώς βρίσκονται σε πολύ κρίσιμα σημεία του ORT (OMPi runtime). Οι περισσότερες έχουν να κάνουν με τις συσκευές που χρησιμοποιούνται για τις οποίες δυστυχώς δεν υπάρχει ακόμα η απαραίτητη υποδομή για να υποστηρίξει την ανάκτηση των απαραίτητων πληροφοριών και τις κλήσεις των callback στα σημεία που ζητούνται από τις προδιαγραφές του προτύπου OpenMP.

6.2.2 Υλοποίηση διεπαφής OMPD (Debugging Interface)

Η διεπαφή OMP-Tools του προτύπου OpenMP περιέχει δύο μέρη. Η διπλωματική αυτή εργασία υλοποίησε ένα μεγάλο κομμάτι της διεπαφής OMPT, που είναι το ένα μέρος. Για την πλήρη υποστήριξη της διεπαφής OMP-Tools, θα χρειαστεί να υλοποιηθεί το έτερο μέρος η διεπαφή OMPD, που δεν αναλύθηκε σε μεγάλο βαθμό, καθώς είναι εκτός της έκτασης και του σκοπού της διπλωματικής αυτής εργασίας. Περισσότερες σχετικές πληροφορίες στην ενότητα 2.5.

6.2.3 Υποστήριξη νεότερων εκδόσεων του OpenMP

Η υλοποίηση της διεπαφής έγινε βάσει της έκδοσης του προτύπου OpenMP 5.0. Με παρόμοιο σκεπτικό με την ενότητα 6.2.2, για την ολοκλήρωση της υποστήριξης νεότερων εκδόσεων του προτύπου μπορούν να γίνουν οι κατάλληλες προσθήκες και αλλαγές στην υλοποίηση της διεπαφής, ώστε να αντιστοιχούν στις προδιαγραφές των νεότερων εκδόσεων του προτύπου OpenMP (5.1 ή νεότερη).

6.2.4 AfterOMPT

Το άρθρο[9] παρουσιάζει το AfterOMPT, ένα νέο εργαλείο (1st party tool) ανάλυσης εκτέλεσης παραλληλίας σε εφαρμογές OpenMP, βασισμένο στη διεπαφή OMPT. Η συμβολή της εργασίας έγκειται στη μεθοδολογική επέκταση της Aftermath πλατφόρμας ανάλυσης με υποστήριξη για καταγεγραμμένα συμβάντα, μέσω OMPT, καθώς και στην πρόταση επεκτάσεων της σημερινής διεπαφής OMPT ώστε να υπάρχουν περισσότερα γεγονότα διαθέσιμα για καταχώρηση σε κρίσιμα σημεία παραλληλοποίησης.

Οι προτεινόμενες επεκτάσεις για την διεπαφή OMPT είναι μία ακόμα πιθανή προσθήκη που θα μπορούσε να υλοποιηθεί για τον OMPi, στο μέλλον. Παρέχοντας στα εργαλεία που θα την εκμεταλλευτούν μεγαλύτερη διαφάνεια, ακρίβεια και συνέπεια στην ανάλυση και καταγραφή των δεδομένων για την εκτέλεση της παράλληλης εφαρμογής OpenMP.

Βιβλιογραφία

[1] OpenMP Architecture Review Board, "*OpenMP Application Programming Interface v5.0*"', 2018.

[2] V. V. Dimakopoulos, E. Leontiadis and G. Tzoumas, "*A Portable C Compiler for OpenMP V.2.0*", in *Proceedings of the 5th European Workshop on OpenMP (EWOMP 2003)*, Aachen, Germany, September 2003.

[3] G. Ch. Philos, V. V. Dimakopoulos, P. E. Hadjidoukas, "*A runtime system architecture for ubiquitous support of OpenMP*", in Proceedings of the 7th International Symposium on Parallel and Distributed Computing (ISPDC 2008), Krakow, Poland, July 2008.

[4] A. Eichenberger, J. Mellor-Crummey, M. Schulz, N. Copty, J. DelSignore, R. Dietrich, X. Liu, E. Loh, D. Lorenz *et al.*, "*OMPT and OMPD: OpenMP Tools Application Programming Interfaces for Performance Analysis and Debugging*", OpenMP Tools Working Group, Technical Report, Apr. 24 2013.

[5] RWTH Aachen University, Technische Universität Dresden, University of Oregon, Forschungszentrum Jülich et al., "*SCORE-P: User Manual*", Revision f07997df4, Dec. 18 2025.

[6] Barcelona Supercomputing Center, "*Extrae: User and Instrumentation Guide*".

[7] Intel Corporation, "*Intel® oneAPI VTune™ Profiler Documentation*".

[8] Lawrence Livermore National Laboratory, "*Caliper: A Flexible Annotation and Performance Measurement Framework*", Technical Report LLNL-TR-666778, 2014.

[9] I. Wodiany, A. Drebes, R. Neill, and A. Pop, "*AfterOMPT: An OMPT-based tool for fine-grained tracing of tasks and loops*", in *Proceedings of the 16th International Workshop on OpenMP (IWOMP 2020)*, Austin, TX, USA, Sep. 2020, Lecture Notes in Computer Science, Springer.