Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Ανάπτυξη Εργαλειοθήκης για την Πρόβλεψη του Χρόνου Επικοινωνίας Παράλληλων Εφαρμογών

Διπλωματικη Εργασια

του

ΦΟΙΒΟΥ ΚΟΤΟΜΑΤΑ

Επιβλέπων: Γεώργιος Γκούμας
Επίκουρος Καθηγητής Ε.Μ.Π.

Εργαστηριο Υπολογιστικων Συστηματων
Αθήνα, Νοέμβριος 2018

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Ανάπτυξη Εργαλειοθήκης για την Πρόβλεψη του Χρόνου Επικοινωνίας Παράλληλων Εφαρμογών

Διπλωματικη Εργασια

του

ΦΟΙΒΟΥ ΚΟΤΟΜΑΤΑ

Επιβλέπων: Γεώργιος Γκούμας
Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Νοεμβρίου 2018.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Νικόλαος Παπασπύρου
Επίκ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπλ. Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2018

(Υπογραφή)

.....

Φοιβος Κοτοματας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η πρόβλεψη του χρόνου επικοινωνίας μεταξύ των διεργασιών μιας παράλληλης εφαρμογής είναι μια πολύ σημαντική διαδικασία, κυρίως στα σύγχρονα μεγάλα υπολογιστικά συστήματα τα οποία αποτελούνται από εκατοντάδες χιλιάδες ή και εκατομμύρια πυρήνες. Τα αποτελέσματα της διαδικασίας πρόβλεψης της επικοινωνίας δίνουν τη δυνατότητα στον δημιουργό της εφαρμογής ή στο διαχειριστή του συστήματος να πάρει τις κατάλληλες αποφάσεις για να διορθώσει ή να βελτιώσει την απόδοση της εφαρμογής του.

Στα πλαίσια της συγκεκριμένης διπλωματικής εργασίας αναπτύχθηκαν τρία εργαλεία τα οποία εξάγουν τα χαρακτηριστικά της επικοινωνίας της εφαρμογής και το σχήμα της απεικόνισής της σε ένα υπολογιστικό σύστημα. Στόχος είναι τα εξαγόμενα δεδομένα να μπορεί, στη συνέχεια, ο χρήστης να τα μοντελοποιήσει εύκολα ανάλογα με το εκάστοτε μοντέλο πρόβλεψης που χρησιμοποιεί.

Κάθε ένα εργαλείο έχει διακριτό ρόλο και στοχεύει να εξάγει διαφορετικά χαρακτηριστικά της εφαρμογής. Το πρώτο εργαλείο είναι μία βιβλιοθήκη η οποία καταγράφει τα ίχνη της εφαρμογής και εξάγει το σχήμα της επικοινωνίας ανά διεργασία. Το δεύτερο εργαλείο αναλύει τα δεδομένα του προηγούμενου σταδίου και αποτυπώνει το συνολικό σχήμα επικοινωνίας της εφαρμογής. Τέλος το τρίτο εργαλείο, είναι ένας προσομοιωτής ο οποίος παράγει το σχήμα της απεικόνισης της εφαρμογής στο σύστημα. Συνολικά, από τα εξαγόμενα των τριών εργαλείων μπορεί να εξάγει κανείς το σχήμα κίνησης των δεδομένων μίας εφαρμογής.

Στα κεφάλαια της παρούσας εργασίας, η δομή, οι μηχανισμοί και ο τρόπος χρήσης κάθε εργαλείου θα μελετηθεί διεξοδικά και στο τέλος θα αξιολογήσουμε την απόδοσή τους χρησιμοποιώντας σύγχρονα μετροπρογράμματα ακολουθώντας μία προσέγγιση η οποία αποσυνδέει τη διαδικασία συλλογής των χαρακτηριστικών από το υπολογιστικό σύστημα που ο χρήστης έχει ως τελικό στόχο.

Λέξεις Κλειδιά

Παράλληλες εφαρμογές, MPI, πρόβλεψη χρόνου επικοινωνίας, συλλογή ιχνών, Mpirun

Abstract

Having the capability of predicting the time needed for process communication and interchange of data during an application's execution is becoming really important and crucial nowadays, bearing in mind the scaling of modern large computing systems, that consist of hundreds of thousands or even millions of computing cores. Equipped with the results of the prediction, application developers and system administrators will be able to correct or even optimize the performance of an application.

This works provides three (3) tools that will help a user to extract communication characteristics from an application as well as placement information on the target platform. The end goal is to provide the user with a complete set of extracted data, that he will then be able to easily model to a format suitable to his prediction models.

Each tool has a distinct role and targets to different parts and phases of the application lifecycle. The first tool, is a library that traces the execution of an application and aims to extract a per process communication schema. The second tool is a parser that analyzes and aggregates the previously extracted data in order to provide the complete application communication schema. Finally, the third tool is a simulator that produces a mapping of the placement or ranks on a target system. By combining the results of each tool, a user may deduce the traffic pattern of the application.

Through the chapters of this work, a reader will be presented with the structure, the internal mechanisms and the usage of each part of the toolchain. In the final chapters, we will evaluate the efficiency of the tools by testing them with modern benchmarks and using an approach that aims to decouple completely the process of data extraction from the real target machine.

Keywords

Parallel applications, MPI, Communication time prediction, tracing, Mpirun

Ευχαριστίες

Η συγκεκριμένη διπλωματική εργασία ολοκληρώνει ένα πολύ σημαντικό κύκλο σπουδών αλλά και ένα γενικότερο κύκλο ο οποίος ξεκίνησε το Σεπτέμβρη του 2011. Κατά τη διάρκεια αυτής της χρονικής περιόδου άλλαξαν πολλά και θα ήθελα να ευχαριστήσω όλα τα άτομα τα οποία συντέλεσαν στην ολοκλήρωσή του.

Η εργασία εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων. Θα ήθελα να ευχαριστήσω πρώτα τον υπεύθυνο μου, Επίκουρο Καθηγητή Γεώργιο Γκούμα, τόσο για τη ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα όσο και για τις γνώσεις που μου μετέφερε σε όλα μαθήματά του. Θα ήθελα να ευχαριστήσω ιδιαιτέρως τη Διδάκτορα Νικέλα Παπαδοπούλου η οποία με καθοδήγησε και με βοήθησε σε όλη τη διάρκεια και σε όλα τα στάδια αυτής της εργασίας.

Τέλος, δεν γίνεται να μην ευχαριστήσω την οικογένεια και τους φίλους μου, οι οποίοι ήταν δίπλα μου σε όλη τη διάρκεια, για την συμπαράσταση και την πάντα ευγενική τους παρότρυνση.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Αντικείμενο της διπλωματικής εργασίας

Τα τελευταία χρόνια η μεγάλη άναπτυξη που γνώρισε η τεχνολογία του υλικού των υπολογιστών συνέβαλλε στην αντίστοιχη ανάπτυξη των τεχνικών και των μοντέλων ανάπτυξης του λογισμικού οδηγώντας αναπόδραστα σε μοντέλα παράλληλης επεξεργασίας και προγραμμάτων τα οποία βοηθούν την καλύτερη εκμετάλευση των πόρων του υλικού. Πολλά πρότυπα, πρωτόκολλα και μοντέλα έχουν αναπτυχθεί έτσι ώστε να διευκολύνουν την ανάπτυξη παράλληλων εφαρμογών και να αποσυνδέσουν την διαδικασία επικοινωνίας των επεξεργαστικών πόρων με τη διαδικασία εκτέλεσης των υπολογισμών της εκάστοτε εφαρμογής. Κάθε ένα από τα μοντέλα που έχουν αναπτυχθεί είναι σχεδιασμένο έτσι ώστε να ανταποκρίνεται καλύτερα σε συγκεκριμένες αρχιτεκτονικές του υλικού πάνω στο οποίο εκτελείται μια εφαρμογή. Στα πλαίσια της συγκεκριμένης διπλωματικής εργασίας έχει αναπτυχθεί μια σειρά εργαλείων η οποία επιτρέπει στους προγραμματιστές παράλληλων εφαρμογών με χρήση του πρωτόκολλου επικοινωνίας MPI, ένα πρωτόκολλο που επιτρέπει την ανάπτυξη παράλληλων εφαρμογών πάνω από αρχιτεκτονικής κατανεμημένης μνήμης, να κατανοήσουν καλύτερα τη λειτουργία και την αλληλεπίδραση με το υλικό των εφαρμογών τους. Η παρούσα διπλωματική στοχεύει στο να παρουσιάσει εργαλεία τα οποία θα οδηγήσουν τους προγραμματιστές να αναβαθμίσουν τις εφαρμογές τους υποδυκνείοντας εμμέσως τρόπους με τους οποίους θα μπορούσε να επιτευχθεί μεγαλύτερη απόδοση.

1.2 Κίνητρο

Οι ταχύτητες με τις οποίες όχι μόνο δουλεύουν αλλά και εξελίσσονται τα σύγχρονα υπολογιστικά συστήματα είναι τόσο υψηλές, που αναμένεται ότι σύντομα θα πετύχουν τις επιδόσεις των ExaFlops, έχοντας δώσει τεράστια ώθηση στην επιστήμη της υπολογιστικής επίλυσης προβλημάτων. Πλέον είναι δυνατό να εκτελεστούν προσομοιώσεις φυσικών φαινομένων οι οποίες κλιμακώνουν σε χιλιάδες κόμβους υπερυπολογιστών οι οποίοι αριθμούν εκατομμύρια επεξεργαστικούς πυρήνες, με χαρακτηριστικό παράδειγμα τον Summit του Oak Ridge National Laboratory, ο οποίος τη στιγμή συγγραφής της παρούσας εργασίας βρίσκεται στη κορυφή

του top500[1] με 2.282.544 πυρήνες. Επιπλέον, η εξέλιξη των ίδιων των επεξεργαστών και η μείωση του χρόνου πρόσβασης στις μνήμες έχει καταστήσει εφικτή την δραστική μείωση του χρόνου υπολογισμού των εφαρμογών.

Με μία πρώτη ματιά η επικρατούσα κατάσταση φαντάζει ιδεατή, όμως δυστυχώς εμφανίζονται προβλήματα τα οποία μπορούν να ανακόψουν το σημερινό ρυθμό της προόδου. Οι τεχνολογία των επεξεργαστών έχει φτάσει σε ένα σημείο στο οποίο περαιτέρω εξέλιξη βασισμένη στις ήδη ευρέως χρησιμοποιούμενες τεχνικές είναι πολύ δύσκολη λόγω φυσικών προβλημάτων τόσο στη διαχείριση της ισχύος και της θερμότητας όσο και στις αλληλεπιδράσεις μεταξύ των μορίων των υλικών των τρανζίστορ. Επιπλέον, ο διαφορετικός ρυθμός εξέλιξης των επεξεργαστών σε σχέση με τα υπόλοιπα μέρη που αποτελούν ένα σύγχρονο επεξεργαστικό σύστημα, όπως οι μνήμες και τα δίκτυα διασύδεσης δημιουργεί ένα επιπλέον πρόβλημα καθώς πλέον το όριο στην ικανότητα κλιμάκωσης μίας εφαρμογής τίθεται από παράγοντες που δεν σχετίζονται με την υπολογιστική διαδικασία που καλείται να εκτελέσει. Κυρίως στην κατηγορία των δικτύων διασύνδεσης, εμφανίζεται ένα περίπλοκο πρόβλημα. Από τη μία πλευρά, η αύξηση του αριθμού των επεξεργαστικών πυρήνων επιφέρει βελτιώσεις στη φάση υπολογισμών μιας εφαρμογής. Από την άλλη πλευρά, η αύξηση του αριθμού των πυρήνων επιφέρει και αύξηση του χρόνου επικοινωνίας των εφαρμογών η οποία μπορεί να οφείλεται στον απαιτούμενο χρόνο για την ανταλλαγή των μηνυμάτων, στην πραγματική απόσταση μεταξύ των διεργασιών ή απλά στην ανάγκη για συγχρονισμό μεγάλου αριθμού διεργασιών με αποτέλεσμα ο άεργος χρόνος να μπορεί να αυξηθεί πολύ. Επιπλέον όσο αυξάνεται ο αριθμός των διεργασιών, γίνεται πιο επιτακτική η ανάγκη για σωστή και αποτελεσματική τοποθέτησή τους σε πυρήνες αφού πολύ εύκολα μια αστοχία θα οδηγήσει στη υπερφόρτωση κόμβων του δικτύου διασύνδεσης ή στη μεταφορά μηνυμάτων ανάμεσα σε απομακρυσμένες διεργασίες.

Για την αντιμετώπιση του τελευταίου προβλήματος, έχουν προταθεί στη βιβλιογραφία διάφορα θεωρητικά μοντέλα και τεχνικές για την πρόβλεψη του χρόνου ή της επίδοσης της επικοινωνίας, που αποσκοπούν στη βελτιστοποίηση της επίδοσης της επικοινωνίας και στη μείωση της ανισοκατανομής του χρόνου υπολογισμών και επικοινωνίας. Η επίδοση της επικοινωνίας είναι μία μετρική που εξαρτάται από όλα τα επίπεδα που έχουν αναφερθεί έως τώρα. Εξαρτάται από την ίδια την εφαρμογή και τα σχήματα επικοινωνίας που παρουσιάζει. Εξαρτάται από το σύστημα εκτέλεσης και το λογισμικό του και εξαρτάται ακόμα και από την απεικόνιση (placement) της εφαρμογής στο σύστημα κατά το χρόνο εκτέλεσης. Η γνώση των συγκριμένων χαρακτηριστικών οδηγεί στην εξαγωγή του σχήματος επικοινωνίας της εφαρμογής είτε συνολικά είτε ανά φάση. Το σχήμα επικοινωνίας προσφέρει πολύ χρήσιμες πληροφορίες καθώς σε συνδυασμό με την απεικόνιση της εφαρμογής στους πόρους του συστήματος μπορεί να εξαχθεί το σχήμα κίνησης δεδομένων (traffic pattern), μοναδικό ανά διεργασία και απεικόνιση, το οποίο καταδεικνύει τον όγκο των δεδομένων που μεταφέρονται στο δίκτυο διασύνδεσης, τα σημεία αποστολής και λήψης καθώς και σημεία του δικτύου διασύνδεσης στα οποία μπορεί να δημιουργηθεί συμφόρηση [5, 1, 8].

Ανεξαρτήτως του μοντέλου με το οποίο θα πραγματοποιηθεί η πρόβλεψη του χρόνου επικοινωνίας, είναι απαραίτητη αρχικά η εξαγωγή μετρήσιμων χαρακτηριστικών της επικοινωνίας τα

[1] top500.org

οποία στη συνέχεια θα μοντελοποιηθούν και θα χρησιμοποιηθούν για την εξαγωγή του σχήματος κίνησης δεδομένων και των υπόλοιπων προαναφερθέντων μετρικών. Η μοντέλοποίηση και ο βαθμός στον οποίο μπορεί να προβλεφθεί η επικοινωνία εξαρτάται σε πολύ μεγάλο βαθμό από το είδος και το πλήθος των χαρακτηριστικών που έχουμε στη διάθεσή μας. Μπορούμε να ομαδοποιήσουμε τα είδη των χαρακτηριστικών σε τρεις μεγάλες κατηγορίες [8]. Στην Κατηγορία Α, περιλαμβάνονται πληροφορίες σχετικές με τη μορφή της επικοινωνίας της εφαρμογής, όπως το πλήθος και το μέγεθος των μηνυμάτων καθώς και το είδος των κλήσεων. Επίσης, περιλαμβάνονται χαρακτηριστικά που έχουν σχέση με τα δεδομένα που έχει ζητήσει ο χρήστης για την εκτέλεση της εφαρμογής του, δηλαδή τον αριθμό των διεργασιών και το μέγεθος του προβλήματος. Στην Κατηγορία Β, προστίθονται επιπλέον χαρακτηριστικά. Περιέχονται πλέον πληροφορίες οι οποίες δείχνουν την ανάθεση των διεργασιών σε κόμβους ενός συστήματος. Σε συνδυασμό των δύο κατηγοριών λοιπόν μπορούν να εξαχθούν χαρακτηριστικά τα οποία δείχνουν και να ποσοτικοποιούν την επικοινωνία εντός του κόμβου ενός συστήματος και το μέγεθος (σε bytes) των δεδομένων που εισέρχονται ή εξέρχονται από έναν κόμβο. Ακόμα όμως δεν υπάρχει δυνατότητα να εξαχθεί το πλήρες σχήμα κίνησης δεδομένων καθώς δεν υπάρχουν διαθέσιμες πληροφορίες σχετικά με την πραγματική θέση των κόμβων στο σύστημα και την αρχιτεκτονική του δικτύου διασύνδεσης. Η Κατηγορία Γ, συμπληρώνει αυτό το κενό συλλέγοντας πληροφορίες ακριβώς πριν την εκτέλεση μιας εφαρμογής. Αξίζει να τονίσουμε εδώ, ότι οι τρεις Κατηγορίες μπορούν να λειτουργήσουν και ανεξάρτητα αφού κάθε μία παρέχει χαρακτηριστικά τα οποία μπορούν να συνεισφέρουν και σε τομείς εκτός της πρόβλεψης του χρόνου επικοινωνίας, αλλά μόνο ο συνδυασμών όλων θα δώσει όλα τα απαραίτητα χαρακτηριστικά για τη μοντελοποίηση του χρόνου επικοινωνίας μιας εφαρμογής.
Η παρούσα εργασία, αποσκοπεί στην ανάπτυξη μιας εργαλειοθήκης η οποία θα παρέχει τα μέσα για την εξαγωγή των χαρακτηριστικών που χρειάζεται να γνωρίζει ένα σύστημα πρόβλεψης του χρόνου εκτέλεσης από μία παράλληλη εφαρμογή. Αναπτύχθηκαν τρία διακριτά μεταξύ τους εργαλεία τα οποία συνδυάζονται είτε άμεσα είτε έμμεσα και παράγουν πληροφορίες οι οποίες καλύπτουν τις απαιτήσεις της Κατηγορίας Α και της Κατηγορίας Β.[8]

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Η εργασία συνεισφέρει στην εξαγωγή μετρήσιμων χαρακτηριστικών σχετικών με το χρόνο επικοινωνίας μιας παράλληλης εφαρμογής.

2. Υλοποιήθηκε εργαλείο συλλογής ιχνών από μία MPI εφαρμογή και καταγραφής τους σε αρχεία κειμένου.

3. Υλοποιήθηκε εργαλείο επεξεργασίας των παραγόμενων του παραπάνω εργαλείου έτσι ώστε να εξαχθεί το σχήμα επικοινωνίας της εφαρμογής.

4. Υλοποιήθηκε εργαλείο προσομοίωσης της διαδικασίας ανάθεσης MPI διεργασιών σε επεξεργαστικούς πόρους από έναν διαχειριστή συστήματος.

5. Η εργασία καλείται να παρέχει τα μέσα ώστε να βοηθήσει το χρήστη να προβλέψει το χρόνο επικοινωνίας παράλληλων εφαρμογών MPI και να κατανοήσει τη συμπεριφορά των παράλληλων εφαρμογών του.

1.3 Οργάνωση του τόμου

Η συνέχεια του τόμου έχει οργανωθεί ως εξής: Στο Κεφάλαιο 2 αφού παρουσιαστούν οι θεωρητικές βάσεις πάνω στις οποίες στηρίχθηκε η εφαρμογή, δίνεται η αρχιτεκτονική και η δομή της εργαλειοθήκης που αναπτύχθηκε. Στα Κεφάλαια 3, 4, 5 παρουσιάζεται αναλυτικά κάθε ένα εργαλείο και δίνονται παραδείγματα της χρήσης του. Στο Κεφάλαιο 6 παρουσιάζονται τα πειραματικά αποτελέσματα από μετρήσεις που έγιναν πάνω στο εργαλείο συλλογής ιχνών. Τέλος, στο 7 καταγράφονται τα συμπεράσματα που εξήχθησαν, ολοκληρώνεται η παρούσα διπλωματική εργασία και προτείνονται πιθανές μελλοντικές επεκτάσεις των εργαλείων.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο Δομή της εργαλειοθήκης

2.1 Εισαγωγή

Στο συγκεκριμένο Κεφάλαιο θα παρουσιαστεί η αρχιτεκτονική και ο σκοπός της εργαλειοθήκης που αναπτύχθηκε. Στόχος είναι να καταλάβει ο αναγνώστης το σκοπό λειτουργίας κάθε εργαλείου και πώς αυτά μπορούν να συνδυαστούν μεταξύ τους για να συνεισφέρουν στη διαδικασία της πρόβλεψης του χρόνου επικοινωνίας μιας εφαρμογής. Πριν όμως αναφερθούμε σε αυτά, είναι χρήσιμο να τεθεί μία βάση πάνω στην οποία χτίστηκε όλη η εργασία. Είναι απαραίτητο, ο αναγνώστης να έχει γνώση της δομής ενός παράλληλου συστήματος και του τρόπου με τον οποίο τα μέρη του μπορούν να επηρεάσουν μία εφαρμογή. Ακόμα, πρέπει να γνωρίζει τις αρχές λειτουργίας του προτύπου MPI, στη χρήση του οποίου εστιάζει η παρούσα εργασία. Θα αναφερθούμε ακόμα στη διαδικασία συλλογής ιχνών από μια παράλληλη εφαρμογή με στόχο την εξαγωγή των χαρακτηριστικών που επιθυμούμε.

2.2 Αρχιτεκτονική παράλληλων συστημάτων

Στα σύγχρονα υπολογιστικά συστήματα συνδυάζονται διακριτά μεταξύ τους μέρη, το κάθε ένα με τις δικές του τεχνολογίες και ιδιαιτερότητες, με στόχο τη σωστή λειτουργία του συνόλου. Η δομική μονάδα ενός υπολογιστικού συστήματος είναι ο κόμβος. Οι κόμβοι αποτελούνται από έναν αριθμό από επεξεργαστές οι οποίοι συνδέονται μεταξύ τους και διαθέτουν τη δικιά τους μνήμη. Η μνήμη αυτή μπορεί να είναι κοινή σε όλους τους επεξεργαστές ή να είναι μοιραζεται ανάμεσα στους επεξεργαστές οι οποίοι επίσης να διατηρούν και έναν αριθμό από κρυφές μνήμες. Στο επίπεδο κόμβου ο χρόνος επικοινωνίας μεταξύ των επεξεργαστών εξαρτάται από δύο παράγοντες. Καταρχάς, από τον τρόπο της διασύνδεσης μεταξύ τους, αν και επειδή ο αριθμός των επεξεργαστών είναι σχετικά μικρός έχει επικρατήσει να συνδέονται όλοι με απευθείας συνδέσεις (point to point). Κατά δεύτερον, εξαρτάται και από την αρχιτεκτονική της μνήμης αφού σε σχήματα μοιραζόμενης μνήμης ο χρόνος πρόσβασης του επεξεργαστή στην τοπική του μνήμη είναι μικρότερος από αυτόν στις απομακρυσμένες (μοντέλα NUMA).

Οι κόμβοι του συστήματος συνδέονται μεταξύ τους με δίκτυα διασύνδεσης. Σημαντικό ρόλο στην επίδοση της επικοινωνίας μεταξύ των κόμβων έχει η τοπολογία του δικτύου διασύνδεσης που χρησιμοποιείται καθως καθορίζει τον αριθμό των διακοπτών του δικτύου και το χρόνο ζωής των πακέτων που ανταλλάσονται μέσα σε αυτό. Μας ενδιαφέρει, επίσης και η τεχνολογία που χρησιμοποιείται καθώς τα δικτυακά πρωτόκολλα που χρησιμοποιούνται και το εύρος ζώνης των συνδέσεων (bandwidth) καθορίζουν την ταχύτητα ανταλλαγής των μηνυμάτων. Ο χρόνος επικοινωνίας επηρεάζεται και από τους αλγόριθμους δρομολόγησης που χρησιμοποιεί το δίκτυο διασύνδεσης, οι οποίοι ανάλογα με την πολυπλοκότητά τους μπορεί να προκαλέσουν καθυστέρηση σε συγκεκριμένα σημεία του δικτύου ή στην προσπάθεια να λάβουν πιο σύνθετες αποφάσεις να εισάγουν μη αμελητέες καθυστερήσεις στην δρομολόγηση των δεδομένων.
Τη σωστή λειτουργία των συστημάτων που περιγράφηκαν την καθορίζει συνήθως ένα κοινό λειτουργικό σύστημα. Το σύστημα αρχείων είναι επίσης κοινό μεταξύ όλων των κόμβων τις περισσότερες φορές. Είναι πολλοί οι παράγοντες που επηρεάζουν την επικοινωνία των παράλληλων εφαρμογών και οφείλονται στο λειτουργικό σύστημα και το σύστημα αρχείων, όμως στα πλαίσια της παρούσας εργασίας το κομμάτι που μας ενδιαφέρει κατά κύριο λόγο είναι η συμπεριφορά του διαχειριστή πόρων (resource manager) ή/και του δρομολογητή (scheduler). Ο διαχειριστής λαμβάνοντας υπόψη την απεικόνιση που έχει ζητηθεί κατά την υποβολή μίας εργασίας για εκτέλεση από το πρότυπο MPI, για παράδειγμα με χρήση του Mpirun, θα λάβει τις αποφάσεις σχετικά με τη δέσμευση συγκεκριμένων πόρων για την απεικόνιση των διεργασιών σε κόμβους και επεξεργαστές. Οι αποφάσεις του λοιπόν θα είναι καθοριστικές σε σχέση με το τελικό σχήμα κίνησης των δεδομένων της εφαρμογής στο σύστημα.

2.3 MPI

Η παρούσα εργασία εστιάζει σε παράλληλες εφαρμογές που κάνου χρήση του ευρέως διαδεδομένου προτύπου MPI (Message Passing Interface [2], [7]). Μέσα στα χρόνια έχουν αναπτυχθεί πολλά μοντέλα και γλώσσες προγραμματισμού με στόχο να επιτρέψουν και να διευκολύνουν την υλοποίηση παράλληλων εφαρμογών. Τέτοια παραδείγματα είναι το OpenMP για προγραμματισμό σε κοινή μνήμη, η CUDA για προγραμματισμό σε κάρτες γραφικών γενικού σκοπού, ενώ τα τελευταία χρόνια μεγάλη άνθιση γνωρίζουν και οι γλώσσες PGAS. Κάθε μοντέλο είναι σχεδιασμένο έτσι ώστε να αλληλεπιδρά καλύτερα με διαφορετικές αρχιτεκτονικές. Εστιάζουμε στο MPI καθώς είναι ένα πρότυπο το οποίο χρησιμοποιείται ευρέως σε συστήματα μεγάλης κλίμακας και αλληλεπιδρά πολύ καλά με την αρχιτεκτονική κατανεμημένης μνήμης των σύγχρονων υπολογιστικών συστημάτων, λειτουργώντας ως προγραμματιστική διεπαφή για οποιοδήποτε δίκτυο διασύνδεσης.

Το MPI είναι ένα πρότυπο το οποίο ξεκίνησε να αναπτύσεται το 1991 και στοχεύει στην επικοινωνία επεξεργαστών σε αρχιτεκτονικές κατανεμημένης μνήμης. Υπάρχουν διάφορες υλοποιήσεις του προτύπου, τόσο εμπορικές όσο και ανοικτού κώδικα, το οποίο έχει σχεδιαστεί για να λειτουργεί απευθείας με τις γλώσσες C, C++, Fortran. Στην πορεία αναπτύχθηκαν διάφορες βιβλιοθήκες σε άλλες γλώσσες με αποτέλεσμα σήμερα να μπορεί να χρησιμοποιηθεί από γλώσσες που έχουν τη δυνατότητα να αλληλεπιδράσουν με βιβλιοθήκες γραμμένες σε

κάποια από τις αρχικές τρεις. Είναι ένα πρότυπο το οποίο επιτρέπει την ανάπτυξη παράλληλων εφαρμογών στα πρότυπα του μοντέλου SIMD (Single Instruction Multiple Data), εισάγοντας στη βάση του την έννοια της διεργασίας. Μία παράλληλη εφαρμογή που χρησιμοποιεί το πρότυπο MPI, κατά το χρόνο εκτέλεσης αποτελείται από έναν αριθμό διεργασιών οι οποίες λειτουργούν αυτόνομα, με ιδιωτική εικονική μνήμη, και αλληλεπιδρούν αυστηρά μέσω μηνυμάτων, όπως ορίζει ο προγραμματιστής. Οι διεργασίες οργανώνονται σε πιθανά επικαλυπτόμενα σύνολα και μπορούν να επικοινωνήσουν με διεργασίες είτε του δικού τους είτε ενός άλλου συνόλου κάνοντας αναφορά στο διαχειριστή κάθε συνόλου, ο οποίος ορίζεται ως communicator. Αφαιρετικά, το μοντέλο αυτό εκτελεί μεταφορά πακέτων δεδομένων ανάμεσα σε επεξεργαστές πάνω από το δίκτυο διασύνδεσης μέσω απλών κλήσεων αποστολής και λήψης με διαφανή για την εφαρμογή τρόπο. Το μοντέλο, μέσω εσωτερικών μηχανισμών, που διαφέρουν ανάλογα με την υλοποίηση, αναλαμβάνει να αντιγράψει τα περιεχόμενα ενός μέρους της μνήμης που καθορίζει η διεργασία αποστολέας και να το μεταφέρει, πάλι μέσω εσωτερικών κανόνων και λειτουργιών, σε κάποιο σημείο της μνήμης της διεργασίας παραλήπτη, που η τελευταία έχει υποδείξει.

Το μοντέλο προσφέρει στο χρήστη διάφορα είδη επικοινωνίας. Η επικοινωνία μπορεί να είναι από σημείο σε σημείο (pointtopoint), στην οποία συμμετέχουν μόνο δύο διεργασίες, συλλογική (collective) που συμμετέχει ένας αυθαίρετος αριθμός διεργασιών ή ακόμα και μονομερής (single-sided) όπου μια διεργασία αναλαμβάνει με δική της πρωτοβουλία να γράψει δεδομένα στη μνήμη μιας άλλης διεργασίας. Όσον αφορά τη συλλογική επικοινωνία, ο σωστός ορισμός για το πλήθος των διεργασιών που συμμετέχουν σε αυτή είναι ότι ισούται με το πλήθος των διεργασιών που ανήκουν στο περιβάλλον του communicator στον οποίο εκτελείται η κλήση. Η επικοινωνία μπορεί ακόμα να είναι σύγχρονη είτε ασύγχρονη. Δηλαδή κάθε διεργασία έχει τη δυνατότητα να επιλέξει αν θα περιμένει μέχρι να ολοκληρωθεί επιτυχώς μια ανταλλαγή δεδομένων ή αν θα καταχωρήσει το αίτημά της και μετά από κάποιο χρονικό διάστημα θα έχει τη δυνατότητα να ελέγξει αν αυτό ολοκληρώθηκε επιτυχώς.

2.4 Συλλογή Ιχνών - Tracing

Η συλλογή ιχνών είναι η διαδικασία κατά την οποία εξάγονται χρήσιμες πληροφορίες από μία εφαρμογή είτε κατά την εκτέλεσή της είτε από τον πηγαίο της κώδικα. Είναι μια πολύ διαδεδομένη τεχνική καθώς τα αποτελέσματα μπορούν στη συνέχεια να επεξεργαστούν και να βοηθήσουν στην καλύτερη κατανόηση των λειτουγιών της εφαρμογής, να συνεισφέρουν στη δημιουργία προσομοιώσεων της εφαρμογής, να εντοπίσουν σφάλματα ορθότητας ή σφάλματα επίδοσης. Η συλλογή ιχνών ([3]) στην περίπτωση των παράλληλων εφαρμογών, για τις οποίες ενδιαφερόμαστε, είναι μια πολύ περίπλοκη διαδικασία καθώς εξαρτάται σε μεγάλο βαθμό από την αλληλεπίδραση μεταξύ των διεργασιών αλλά και από το ίδιο το σύστημα στο οποίο θα εκτελεστεί. Συνεπώς, είναι αρκετά δύσκολο να εξαχθεί σημαντική πληροφορία από τον πηγαίο κώδικα της παράλληλης εφαρμογής και τα εργαλεία συλλογής ίχνους συμβάλουν στη συλλογή του ίχνους της εφαρμογής κατά την εκτέλεσή της (online tracing). Διάφορα υπάρχοντα

εργαλεία όπως το SST/DUMPI [1] και το score-p της σουίτας Scalasca [4] ή το mpiP [2] επιτρέπουν τη συλλογή ιχνών από παράλληλες εφαρμογές κατά την εκτέλεσή τους. Τα ίχνη που συλλέγονται είναι δυνατό να οπτικοποιηθούν για μελέτη της συμπεριφοράς της εφαρμογής, να χρησιμοποιηθούν για προσομοίωση της εφαρμογής ή να δώσουν χρήσιμα στατιστικά στοιχεία για το πού καταναλώνεται ο χρόνος εκτέλεσης και για τον εντοπισμό σφαλμάτων επίδοσης.
Ανάλογα με τον τελικό στόχο κάθε ενός, αυτά τα εργαλεία συλλέγουν περισσότερη ή λιγότερη πληροφορία από την εκτελούμενη εφαρμογή. Κάποια από αυτά είναι συγκεντρωτικά (summative) και συλλέγουν λιγότερα δεδομένα τα οποία ομαδοποιούν ενώ άλλα κάνουν πλήρη καταγραφή (imperative or functional tracing) και καταγράφουν κάθε κίνηση δεδομένων της εφαρμογής. Όσο αναλυτικότερη είναι η καταγραφή τόσο σημαντικότερη είναι η επιβράδυνση και η επιβάρυνση που προκαλεί το εκάστοτε εργαλείο στην εφαρμογή. Ενδεικτικά, η επίδραση του score-p κατά το χρόνο εκτέλεσης μίας εφαρμογής κυμαίνεται από 124% έως 219%. Ακόμα και το εργαλείο mpiP το οποίο χρησιμοποιείται μόνο για στατιστικούς λόγους έχει επίδραση τάξης μεγαλύτερης του 100% στην εκτέλεση της εφαρμογής, [6]. Βέβαια από την άλλη μεριά, η αναλυτικότερη καταγραφή προσφέρει περισσότερες επιλογές στα πιο ισχυρά εργαλεία, όπως το Vampir, τα οποία έχουν τη δυντότητα να αναπαράγουν εξ ολοκλήρου την εκτέλεση μίας εφαρμογής από παλαιότερα ίχνη (trace replay).
Ο στόχος της συγκεκριμένης εργασίας όμως είναι η εξαγωγή μέσω της χρήσης των ιχνών του σχήματος επικοινωνίας μίας εφαρμογής. Για το σκοπό αυτό χρειαζόμαστε ένα εργαλείο το οποίο να παράγει αναλυτική πληροφορία, δηλαδή κάθε κίνηση δεδομένων στο δίκτυο. Συνεπώς τα συγκεντρωτικά εργαλεία που αναφέρθηκαν, αν και ελαφριά, δεν καλύπτουν τις ανάγκες της εργαλειοθήκης. Δεύτερον, έχουμε ορίσει από την αρχή ότι η εργαλειοθήκη πρέπει να μπορεί να χρησιμοποιηθεί σε συστήματα διαφορετικά, συνήθως αρκετά μικρότερα, από το τελικό. Όπως θα αναφερθεί στην αμέσως επόμενη υποενότητα αυτή η διαδικασία μπορεί να επιβαρύνει αρκετά αυτά τα συστήματα, με αποτέλεσμα να είναι επιτακτική η χρήση ενός εργαλείου το οποίο να είναι όσο το δυνατόν πιο ελαφρύ. Ως αποτέλεσμα λοιπόν, δημιουργήθηκε ένα νέο εργαλείο καταγραφής το οποίο θα παρουσιαστεί στην πορεία και το οποίο πιστεύουμε ότι είναι πολύ κοντά στις ανάγκες μας.

2.4.1 Trace extrapolation vs Oversubscription

Ένα από τα πρώτα προβλήματα που συναντάμε κατά την διαδικασία συλλογής ιχνών από μια παράλληλη εφαρμογή είναι το ίδιο το υπολογιστικό σύστημα στο οποίο θα εκτελέσουμε τη συλλογή. Ανάλογα με το είδος των δεδομένων που θέλουμε να συλλέξουμε μπορεί να χρειαστεί να αναγκαστούμε να χρησιμοποιήσουμε το πραγματικό παράλληλο σύστημα ή να σε άλλες περιπτώσεις να εκτελέσουμε την εφαρμογή σε κάποιο μικρότερο. Ακόμα, μπορεί σε ορισμένες περιπτώσεις να μην έχουμε πρόσβαση στο πραγματικό σύστημα, όπως μπορεί να συμβεί για παράδειγμα όταν ο στόχος είναι εμπορικός και η συλλογή ιχνών γίνεται για να ελέγξουμε αν το υπολογιστικό σύστημα στο οποίο πιθανά θα επενδύσουμε καλύπτει τις ανάγκες μας.

[1] `https://github.com/sstsimulator/sst-dumpi`
[2] `http://llnl.github.io/mpiP/`

Σε αυτές τις περιπτώσεις καταφεύγουμε σε δύο διαφορετικές τεχνικές. Στα πλαίσια της πρώτης εκτελούμε την εφαρμογή σε ένα μικρότερο σύστημα από το σύστημα στόχο και μετά τα αποτελέσματα της συλλογής τα ανάγουμε στην περίπτωση του πραγματικού συστήματος με βάση τη σχέση των παραμέτρων μεταξύ των δύο. Αυτή η διαδικασία ονομάζεται trace extrapolation και χρησιμοποιείται πολλές φορές όταν ο στόχος είναι να δημιουργήσουμε μια προσομοίωση της εφαρμογής όταν θα εκτελεστεί με πολύ μεγαλύτερο αριθμό διεργασιών. Είναι όμως μία προσεγγιστική τεχνική η οποία εξαρτάται σε μεγάλο βαθμό και από τις συναρτήσεις που χρησιμοποιούνται για την αναγωγή, με αποτέλεσμα το ρίσκο να παραποιηθούν τα δεδομένα να είναι υψηλό. Η δεύτερη τεχνική που χρησιμοποιείται βασίζεται πάλι στη χρήση ενός μικρού συστήματος ως πλατφόρμα για την εκτέλεση της διαδικασίας συλλογής ιχνών. Αυτή τη φορά όμως, εκμεταλευόμενοι τις τεχνολογίες του context switching και της ύπαρξης νημάτων υλικού hardware threads, μπορούμε να εκτελέσουμε την εφαρμογή με αρκετά μεγαλύτερο αριθμό επεξεργαστικών πυρήνων από αυτούς που διαθέτει πραγματικά το σύστημα. Προφανώς αυτή η υπερφόρτωση (oversubscription) των πόρων του συστήματος έχει ένα όριο το οποίο εξαρτάται από το βαθμό που αντέχει να κλιμακώσει το σύστημα όσον αφορά τους πυρήνες του και τη μνήμη που διαθέτει. Ένα δεύτερο μειονέκτημα της δεύτερης τεχνικής είναι ότι εφαρμόζεται μόνο σε περιπτώσεις όπου δεν θέλουμε να πάρουμε time-dependent ίχνη (όπως παρουσιάζονται στην επόμενη ενότητα) καθώς οι χρόνοι που θα αποτυπωθούν στην έξοδο είναι πλασματικοί και επηρεάζονται από την υπερφόρτωση του συστήματος.

2.4.2 Time-dependent vs Time-independent Tracing

Τα ίχνη που συλλέγονται από μια εφαρμογή χωρίζονται σε δύο μεγάλες κατηγορίες. Η πρώτη είναι η κατηγορία όπου σε κάθε ίχνος καταγράφεται και η χρονική στιγμή, με τη μορφή χρονοσφραγίδας (timestamp), που συνέβη το γεγονός (time dependent traces). Με αυτό το μοντέλο, στη συνέχεια τα δεδομένα μπορούν να ταξινομηθούν και να αναπαραχθεί σχεδόν πλήρως η πραγματική εκτέλεση του προγράμματος. Στη δεύτερη κατηγορία κατατάσουμε τα ίχνη που δεν περιέχουν πληροφορίες σχετικά με τη χρονική στιγμή κατά την οποία έλαβαν χώρα τα γεγονότα (time independent traces). Συνεπώς προκύπτει μια αφηρημένη εικόνα της εκτέλεσης της εφαρμογής.
Ανάλογα με τις εκάστοτε ανάγκες επιλέγεται ποιο από τα δύο μοντέλα θα χρησιμοποιηθεί. Προφανώς στην πρώτη κατηγορία μπορεί να εξαχθεί μία πιο λεπτομερής εικόνα της εκτέλεσης της εφαρμογής, από την άποψη ότι μπορεί στη συνέχεια να αναπαραχθεί (trace replay). Είναι λοιπόν, ο ενδεδειγμένος τρόπος καταγραφής ιχνών όταν ο απώτερος στόχος του χρήστη είναι η αναπαραγωγή της εκτέλεσης και η δημιουργία προσομοιωτών. Όμως ταυτόχρονα για να είναι εφικτή η σωστή καταγραφή είναι απαραίτητη η πρόσβαση στο πραγματικό περιβάλλον εκτέλεσης. Διαφορετικές τεχνολογίες ανάμεσα στα συστήματα και διαφορές στα μεγέθη των υπολογιστικών συστημάτων επηρεάζουν τα αποτελέσματα της διαδικασίας. Η δεύτερη κατηγορία είναι γενικότερη, αφού καταγράφονται μόνο γεγονότα τα οποία είναι ανεξάρτητα της πλατφόρμας στην οποία γίνεται η εκτέλεση[3]. Από τα συγκεκριμένα ίχνη είναι δυνατό να

[3]Το πλήθος των εφαρμογών οι οποίες προσαρμόζονται και αλλάζουν τον τρόπο εκτέλεσής τους ανάλογα με το σύστημα στο οποίο εκτελούνται, είναι πολύ μικρό. Τέτοιες εφαρμογές δεν καλύπτονται από όσα

εξαχθεί μια λογική σειρά των γεγονότων, δεδομένου κιόλας ότι γνωρίζουμε τους βασικούς μηχανισμούς συγχρονισμού του MPI, αλλά και τις διακριτές φάσεις της εφαρμογής. Συνεπώς, καταλήγουμε σε μία μοντελοποίηση της επικοινωνίας από την οποία μπορεί να εξαχθεί το σχήμα επικοινωνίας της εφαρμογής, το οποίο είναι και το ζητούμενο στα πλαίσια της συγκεκριμένης εργασίας.

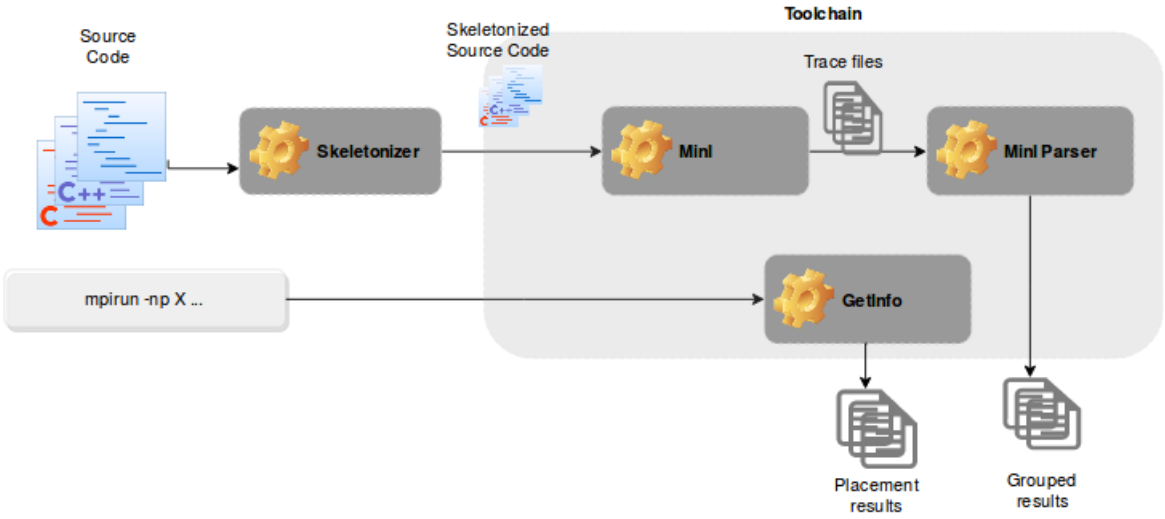
Με βάση τα παραπάνω, το εργαλείο καταγραφής που αναπτύχθηκε στα πλαίσια της παρούσας, καταγράφει ίχνη ανεξάρτητα του χρόνου και βασίζεται στην τεχνική της υπερφόρτωσης για την εξαγωγή χαρακτηριστικών τα οποία να αντικατοπτρίζουν κατά το δυνατό τη συμπεριφορά μεγάλων υπολογιστικών συστημάτων.

2.5 Η εργαλειοθήκη

Στις παραπάνω ενότητες αναπτύχθηκαν οι διάφοροι τομείς που επηρεάζουν την παρούσα εργασία και αναλύθηκαν τα προβλήματα τα οποία έχει να αντιμετωπίσει κανείς κατά τη διαδικασία εξαγωγής των απαραίτητων δεδομένων από μία εφαρμογή έχοντας ως στόχο την πρόβλεψη του χρόνου επικοινωνίας της.
Στόχος της παρούσας διπλωματικής, όπως έχει αναφερθεί, είναι η ανάπτυξη μιας σειράς εργαλείων, η χρήση των οποίων θα επιτρέπει στο χρήστη να εξάγει πληροφορίες από μια εφαρμογή η οποία βασίζεται στο πρότυπο επικοινωνίας MPI. Στη συνέχεια ο χρήστης μπορεί να επεξεργαστεί τα αποτελέσματα και να βγάλει συμπεράσματα σχετικά με τη απόδοση της εφαρμογής του, τις αλληλεπιδράσεις της με το σύστημα στο οποίο εκτελείται και τελικά να εξάγει το σχήμα της επικοινωνίας. Ιδανικά, σε περίπτωση που ο χρήστης έχει τη δυνατότητα, με τις πληροφορίες αυτές θα τροφοδοτήσει τις εισόδους ενός ευφυούς συστήματος [8] . Το σύστημα αυτό στη συνέχεια, θα παράγει μετρικές και μια βελτιστοποιημένη εικόνα ανάθεσης της εφαρμογής σε επεξεργαστικούς κόμβους έτσι ώστε ο χρόνος που χρειάζονται οι διεργασίες για τη μεταξύ τους επικοινωνία να μειώνεται.

Στο σχήμα 2.1 φαίνεται ολοκληρωμένη η αρχιτεκτονική ιδέα που θέλουμε να πετύχουμε και μέσα στο γκρι πλαίσιο τα εργαλεία τα οποία επεκτάθηκαν ή αναπτύχθηκαν στα πλαίσια της παρούσας διπλωματικής. Πρόκειται για μια δομή σωλήνωσης στην οποία κάθε μέρος λαμβάνει είσοδο από το προηγούμενο και παράγει έξοδο που χρησιμοποιείται από το επόμενο στάδιο. Στο πρώτο στάδιο της πάνω ροής ο πηγαίος κώδικας της εφαρμογής παραδίδεται στο επίπεδο του σκελετοποιητή. Στη συνέχεια ο σκελετοποιημένος πηγαίος κώδικας περνάει στο στάδιο της συλλογής ιχνών, το οποίο εκτελείται σε ένα πειραματικό περιβάλλον και συλλέγονται τα δεδομένα της επικοινωνίας. Το επόμενο στάδιο παραλαμβάνει τα αρχεία που περιέχουν τα δεδομένα της επικοινωνίας ανά διεργασία και τα κατηγοριοποιεί ανά φάση του προγράμματος (η «φάση» ενός προγράμματος εξηγείται σε επόμενα κεφάλαια), εξάγωντας ταυτόχρονα και το συνολικό σχήμα της επικοινωνίας. Υπάρχει και μια δεύτερη πιο σύντομη παράλληλη ροή η οποία δέχεται ως είσοδο έναν αριθμό ορισμάτων που αναπαριστούν τα ορίσματα που θα δώσει ο χρήστης όταν θα εκτελέσει την εφαρμογή στο πραγματικό σύστημα (π.χ. όταν χρησιμοποιεί

αναφέρονται στην παρούσα ενότητα αλλά και γενικότερα δεν εμπίπτουν στην κατηγορία των εφαρμογών που η εργαλειοθήκη μπορεί να χειριστεί



Σχήμα 2.1: Δομή εργαλειοθήκης

Mpirun) και δίνει στην έξοδό της έναν χάρτη της τοπολογικής ανάθεσης των διεργασιών σε επεξεργαστικούς κόμβους.

Τέλος, οι δύο παραπάνω ροές συνδυάζονται για να αποδώσουν πληροφορίες που κατατάσσονται στις Κατηγορίες μοντέλων Α και Β, όπως περιγράφηκαν. Τα αποτελέσματά τους, όπως αναφέρθηκε, μπορούν να τροφοδοτήσουν την είσοδο του ευφυούς δικτύου το οποίο και θα παραδώσει τελικά στο χρήστη τις προβλέψεις του χρόνου επικοινωνίας της εφαρμογής. Είναι σημαντικό ο αναγνώστης να έχει επίγνωση ότι η συγκεκριμένη εργαλειοθήκη αποκλειστικά παρέχει πληροφορίες σχετικές με την εφαρμογή όπως αυτές ήδη περιγράφηκαν και δεν προβαίνει σε επεξεργασία των αποτελεσμάτων η οποία θα οδηγούσε στη λήψη αποφάσεων ή στην εξαγωγή μετρικών. Ο χρήστης καλείται να χειριστεί ο ίδιος τα αποτελέσματα αυτά με βάση της εκάστοτε ανάγκες του.

2.5.1 Σκελετοποίηση

Με τη χρήση της συγκεκριμένης τεχνικής, ο προγραμματιστής μπορεί είτε χειροκίνητα είτε με τη χρήση κάποιου εργαλείου να παράγει από τον αρχικό πηγαίο κώδικα της εφαρμογής εκδόσεις από τις οποίες έχουν αφαιρεθεί ακριβοί μαθηματικοί υπολογισμοί, αναθέσεις στη μνήμη και γενικά όλα τα στοιχεία τα οποία δεν σχετίζονται με την προετοιμασία και εκτέλεση της παράλληλης επικοινωνίας. Με αυτόν τον τρόπο θα αποκτήσει τη δυνατότητα να εκτελεί online tracing της εφαρμογής ελαχιστοποιώντας τόσο τις απαιτήσεις του συστήματος όσο και το χρόνο εκτέλεσης. Συνήθως, για τέτοιες ενέργειες αναζητείται κάποιο εργαλείο που αυτοματοποιεί αυτή τη διαδικασία. Ιδανικά θα πρέπει να πληρεί τις εξής προϋποθέσεις:

- Να εκτελεί μετατροπές από πηγαίο κώδικα σε πηγαίο κώδικα (sourcetosource). Η βασικότερη ιδιότητα καθώς επιτρέπει να διατηρείται μόνο μία εκδοχή του πηγαίου κώδικα και να μην χρειάζονται να συντηρούνται και να ανανεώνται οι διαφορετικές εκδοχές.

- Να προσφέρει ευκολία χρήσης και να ελαχιστοποιεί κατά το δυνατό την ανάγκη για επέμβαση του προγραμματιστή.

- Να μην αφαιρεί απλά από τον αρχικό κώδικα τις αναθέσεις στη μνήμη και τους αχρείαστους υπολογισμούς αλλά να τους αντικαθιστά με δομές οι οποίες θα προσομοιάζουν την επίδρασή τους στη μνήμη (heap and stack) καθώς και μοντέλα που μιμούνται την καθυστέρηση.

Στα πλαίσια της παρούσας διπλωματικής δεν υλοποιήθηκε κάποιος νέος σκελετοποιητής καθώς θεωρήθηκε ότι ξέφευγε από τα πλαίσια αυτής. Χρησιμοποιήθηκε για κάποιο χρονικό διάστημα ο ερευνητικός μεταγλωττιστής ROSE Compiler, αλλά καθώς προέκυψαν διάφορα προβλήματα στη χρήση του τελικά δεν συμπεριλήφθηκε ως μέρος της εργαλειοθήκης, Ο συγκεκριμένος μεταγλωττιστής, από τις επιθυμητές ιδιότητες που αναφέρονται παραπάνω, κάλυπτε μόνο την πρώτη. Είχε περίπλοκη παραμετροποίηση και πραγματοποιούσε συντηρητικές μετατροπές. Συγκεκριμένα, ο κώδικας δεν απλοποιούταν επαρκώς καθώς ο μεταγλωττιστής εκτελούσε τη μετατροπή ακολουθώντας την εξής απλή οδηγία "να μη επηρεάσει τις παραμέτρους των κλήσεων του MPI". Συνεπώς, αυτό οδηγούσε στη διατήρηση όλου του κώδικα που επηρέαζε τις παραμέτρους των κλήσεων του προτύπου ανεξαρτήτως του αν τα τμήματα αυτά ήταν ακριβά ή όχι. Επιπλέον δεν είχε τη δυνατότητα να εισάγει τα κατάλληλα μοντέλα που θα προσομοίαζαν τον χρόνο εκτέλεσης του κώδικα που αφαιρέθηκε, αν και αυτή η λειτουργία δεν επηρεάζει την παρούσα διπλωματική. Κοντά στη χρονική στιγμή της ολοκλήρωσης της συγγραφής της παρούσας διπλωματικής παρουσιάστηκε ένας σκελετοποιητής ο οποίος καλύπτει σε αρκετά μεγάλο βαθμό τα στοιχεία που αναφέρθηκαν παραπάνω με μοναδική εξαίρεση την αυτοματοποίηση καθώς είναι σχεδιασμένος για υποβοηθούμενη σκελετοποίηση, ζητώντας δηλαδή την συνδρομή του προγραμματιστή για διευκρινιστικές σημειώσεις όπου χρειάζεται. ([9]) Συνεπώς, θεωρείται ότι αν ο χρήστης το επιθυμεί, πρέπει να παραδώσει στο στάδιο της ιχνηλασίας κώδικα ο οποίος να είναι ήδη σκελετοποιημένος.

2.5.2 MinI και παρόμοια εργαλεία

Το πρώτο εργαλείο που αναπτύχθηκε στα πλαίσια της συγκεκριμένης εργασίας επιτρέπει στο χρήστη να καταγράφει ίχνη ανεξάρτητα χρόνου από την εκτέλεση μίας παράλληλης εφαρμογής που χρησιμοποιεί το πρότυπο MPI. Το συγκεκριμένο εργαλείο αποθηκεύει τα παραγόμενα ίχνη σε αναγνώσιμη μορφή. Έχουν αναπτυχθεί πολλά εργαλεία που εκτελούν παρόμοιες διαδικασίες με τη MinI, όπως το SST/Macro και το DUMPI. Το δεύτερο εργαλείο συγκεκριμένα το οποίο αναπτύχθηκε ως μέρος του γενικότερου SST εκτελεί πολύ συναφή με τη βιβλιοθήκη λειτουργία αφού παράγει ίχνη αντίστοιχης μορφής. Όμως είναι ένα αρκετά πιο βαρύ εργαλείο και επίσης αρχικά τα δεδομένα του αποθηκεύονται σε μη αναγνώσιμη μορφή και είναι απαραίτητη η χρήση επιπλέον εργαλείων για την επεξεργασία τους. Ακόμα καταγράφει και πληροφορίες οι οποίες δεν είναι απαραίτητες στα πλαίσια της προσέγγισης μας με στόχο την πρόβλεψη του χρόνου επικοινωνίας.
Είναι προφανές ότι η MinI είναι ένα εργαλείο που προσφέρει λιγότερες δυνατότητες σε σχέση

με τα πλήρως παραγικά εργαλεία που περιγράφηκαν, όμως θεωρείται ότι υπερτερεί σε μερικά βασικά χαρακτηριστικά. Καταρχάς, δεδομένου ότι τα δεδομένα αποθηκεύονται σε αναγνώσιμη μορφή προσφέρει δυνατότητες πέρα από την απλή συλλογή ιχνών. Τα αποτελέσματα αποθηκεύονται σε απλή αλλά πλήρη μορφή έτσι ώστε ένας χρήστης με απλά εργαλεία (scripts) να έχει τη δυνατότητα να εξάγει χρήσιμα στατιστικά της χρήσης του MPI για την εφαρμογή. Επιπλέον, το εργαλείο μπορεί πολύ εύκολα να ενσωματωθεί στον κώδικα τον οποίο ο χρήστης θέλει να μελετήσει. Το μέγεθος της βιβλιοθήκης είναι μικρό και κυριότερα η επίδρασή της στην εκτέλεση του κώδικα δεν είναι σημαντική, όπως αποδικνύεται στο 7.
Τα παραγόμενα ίχνη περιέχουν τα δεδομένα της επικοινωνίας ανά διεργασία και μπορούν να χρησιοποιηθούν για τη εξαγωγή αρκετών χαρακτηριστικών της Κατηγορίας Α.

2.5.3 MinI Parser

Το δεύτερο εργαλείο που αναπτύχθηκε είναι ένας αναλυτής ο οποίος επεξεργάζεται τα αρχεία καταγραφής και παράγει το συλλογικό σχήμα ομαδοποιημένο και χωρισμένο ανά φάση. Είναι ένα εργαλείο πολύ στενά συνδεδεμένο με τη βιβλιοθήκη. Η χρήση του όμως είναι εξίσου σημαντική αφού παράγει το σχήμα επικοινωνίας της εφαρμογής το οποίο είναι απαραίτητο σε οποιαδήποτε μετέπειτα προσπάθεια πρόβλεψης του χρόνου.
Ακόμα και αν η παρούσα εργαλειοθήκη χρησιμοποιηθεί για σκοπούς πέραν της πρόβλεψης του χρόνου, η χρήση του συνίσταται καθώς βοηθά σημαντικά στη σύμπτυξη του μεγέθους των καταγραφόμενων δεδομένων χωρίς όμως να χάνει πληροφορία.

2.5.4 GetInfo

Το τελευταίο εργαλείο που αναπτύχθηκε είναι ένας προσομοιωτής σε επίπεδο γραμμής εντολών ο οποίος μιμείται τη συμπεριφορά ενός διαχειριστή των πόρων του συστήματος (resource manager). Είναι βασισμένος πάνω στον Mpirun. Σε αντίθεση όμως με έναν πραγματικό διαχειριστή, δεν ξεκινάει κάποια εκτέλεση στους πόρους του συστήματος, αλλά παράγει μια εικόνα της απεικόνισης που θα γινόταν σε μία πραγματική κλήση.
Θεωρούμε ότι μπορεί να χρησιμοποιηθεί με δύο τρόπους. Ο πρώτος είναι για να εξάγουμε απλά τις πληροφορίες της απεικόνισης που θα έχουμε και στο πραγματικό σύστημα έτσι ώστε να συλλέξουμε τα χαρακτηριστικά που χρειάζόμαστε για να έχουμε ένα πλήρες μοντέλο της Κατηγορίας Α ή Β και να προχωρήσουμε στην πρόβλεψη του χρόνου επικοινωνίας. Μπορεί όμως να χρησιμοποιηθεί και ως ένας παραγωγός διαφορετικών απεικονίσεων με στόχο να δούμε ποια παραμετροποίηση θα οδηγήσει σε καλύτερα αποτελέσματα.

2.5.5 Πηγαίοι Κώδικες

Στο συγκεκριμένο Κεφάλαιο έγινε μία σύντομη περιγραφή της εργαλειοθήκης και παρουσιάστηκαν αφαιρετικά οι λειτουργίες κάθε εργαλείου. Στα Κεφάλαια που ακολουθούν θα εμβαθύνουμε σε λεπτομέρειες για κάθε ένα από αυτά και θα παρουσιάσουμε παραδείγματα χρήσης τους.

Όλοι οι πηγαίοι κώδικες των εργαλείων βρίσκονται στο GitHub και προτείνουμε στον αναγνώστη να ανατρέξει σε αυτά παράλληλα με την ανάγνωση της παρούσας εργασιάς σε περίπτωση που επιθυμεί να αποκτήσει μία σφαιρικότερη εικόνα της διαδικασίας. Συγκεκριμένα, τα εργαλεία μπορεί να τα βρει κανείς ως εξής:

Εργαλείο	**Git Repository**
MinI	https://github.com/fkoto/mini-extension.git
MinI Parser	https://github.com/fkoto/MiniParser.git
GetInfo	https://github.com/fkoto/getInfo.git

Πίνακας 2.1: Πηγαίος Κώδικας Εργαλείων

Κεφάλαιο 3

Minimal Implementation (MinI) Library

3.1 Εισαγωγή

Το πρώτο εργαλείο που αναπτύχθηκε είναι η MinI. Είναι μία βιβλιοθήκη γραμμένη στη γλώσσα προγραμματισμού C την οποία κάνουμε link στις εφαρμογές MPI για τις οποίες θέλουμε να πάρουμε μετρήσεις σχετικές με την επικοινωνία. Από όλα τα εργαλεία της διπλωματικής αυτής, είναι το μόνο το οποίο δεν υλοποιήθηκε εξ' ολοκλήρου, αλλά προέκυψε ως επέκταση ενός ήδη υπάρχοντος εργαλείου. Η πρώτη έκδοση της MinI βρίσκεται στο GitHub και είναι ένα έργο του χρήστη gmarkomanolis. Στην αρχική της μορφή πρόσφερε πληροφορίες σχετικά με κάποιες μεθόδους point to point επικοινωνίας και με κάποιες συλλογικής και επίσης έδινε τη δυνατότητα να καταγράφονται και δεδομένα χρόνου στα αποτελέσματα των διεργασιών. Η λειτουργικότητα που παρουσιάστηκε συνοπτικά παραπάνω επεκτάθηκε και αναπτύχθηκε τόσο ως προς τον αριθμό των υποστηριζόμενων MPI κλήσεων όσο και ως προς την πληροφορία η οποία εξάγεται και ως προς τις δυνατότητες που δίνει στο χρήστη να προσαρμόζει το είδος της συλλογής ιχνών, έχοντας πάντα ως βασικούς στόχους το τελικό αποτέλεσμα:

1. Να είναι μια μικρή σε μέγεθος βιβλιοθήκη.
2. Να μην επηρεάζει σε μεγάλο βαθμό το χρόνο εκτέλεσης της εφαρμογής.
3. Να μην επηρεάζει σε μεγάλο βαθμό τον απαιτούμενο χώρο μνήμης που χρειάζεται η εφαρμογή κατά το χρόνο εκτέλεσης.
4. Να είναι εύκολα επεκτάσιμο τόσο ως προς την προσθήκη νέων υποστηριζόμενων κλήσεων MPI, όσο και ως προς την ενσωμάτωση νέων λειτουργιών.
5. Τα αποτελέσματα του προγράμματος να είναι κατανοητά και εύκολα αναγνώσιμα στο χρήστη, χωρίς να χρειάζεται απαραίτητα η χρήση κάποιου αναλυτή.
6. Να μπορεί εύκολα να μεταγλωττίζεται και να συνδέεται στις διάφορες παράλληλες εφαρμογές.

Στη συνέχεια, αναλύεται διεξοδικά η λειτουργία, οι υποστηριζόμενες μέθοδοι και ο τρόπος χρήσης της βιβλιοθήκης. Σε επόμενο κεφάλαιο, ο αναγνώστης μπορεί να βρει αποτελέσματα και μετρήσεις τόσο ως προς το χρόνο εκτέλεσης όσο και ως προς την απαιτούμενη μνήμη κατά την εκτέλεση, τα οποία συλλέχθηκαν από μία σειρά κατάλληλα επιλεγμένων μετροπρογραμμάτων. Απο αυτά στο τέλος θα αξιολογήσουμε αν το τελικό αποτέλεσμα ανταποκρίνεται στους στόχους που τέθηκαν παραπάνω.

3.2 Μεταγλώττιση και λειτουργία

Η βιλβιοθήκη αποτελείται από τα εξής τέσσερα αρχεία: mini.c, utils.c, compile.sh, README.md, όπου τα δύο πρώτα περιέχουν τον πηγαίο κώδικα της βιβλιοθήκης, το τρίτο είναι υπεύθυνο για τη μεταγλώττιση του κώδικα και τη μετατροπή του σε μοιραζόμενη βιβλιοθήκη(shared library), ενώ το τελευταίο περιέχει τις άδειες και οδηγίες για την εκτέλεση του κώδικα. Ο χρήστης μπορεί να προσαρμόσει το αρχείο compile.sh, προσδιορίζοντας τις παραμέτρους όπως περιγράφονται στην αντίστοιχη ενότητα έτσι ώστε η εκδοχή της βιβλιοθήκης που θα δημιουργηθεί να προσφέρει τις ζητούμενες λειτουργίες και στη συνέχεια να το εκτελέσει. Για τη σύνδεση της βιβλιοθήκης με την εκάστοτε εφαρμογή απαιτείται να συνδεθεί κατά το στάδιο της σύνδεσης (linking) με το εκτελέσιμό της και επιπλέον η θέση της βιβλιοθήκης στο σύστημα να είναι διαθέσιμη κατά το χρόνο εκτέλεσης μέσω των κατάλληλων μεταβλητών περιβάλλοντος. Τέλος, πρέπει ο χρήστης να δημιουργήσει ένα φάκελο με την ονομασία ti_traces στο επίπεδο του συστήματός του από το οποίο θα κληθεί η εφαρμογή, καθώς η MinI χρησιμοποιεί αυτό το φάκελο για να αποθηκεύσει τα αρχεία εξόδου.

Αφού τα παραπάνω βήματα έχουν εκτελεστεί και η εφαρμογή την οποία ο χρήστης επιθυμεί να συλλέξει ίχνη ολοκληρωθεί, στο φάκελο ti_traces θα έχουν δημιουργηθεί τα αρχεία καταγραφής. Ο αριθμός των αρχείων ισούται με τον αριθμό των διεργασιών του MPI που δημιουργήθηκαν για τις ανάγκες εκτέλεσης της εφαρμογής. Το όνομα κάθε αρχείου είναι ti_trace ακολουθούμενο από το βαθμό (rank) της διεργασίας και την κατάληξη .txt (για παράδειγμα ti_trace16.txt). Κάθε εγγραφή σε ένα αρχείο καταγραφής περιέχεται σε μία γραμμή, ενώ διαδοχικές εγγραφές χωρίζονται μεταξύ τους με το χαρακτήρα newline ("\n").

3.3 Υλοποίηση

Η ιδέα για την ανάπτυξη του εργαλείου βασίζεται στη ιεραρχία των κλήσεων των διεπαφών και μεθόδων του MPI. Το πρότυπο λειτουργεί με βάση την παρακάτω ιεραρχία (όπως αναγράφεται και στην επίσημη σελίδα του προτύπου), όπου με τον όρο ασθενής σύμβολο (weak symbol) αναφερόμαστε σε συναρτήσεις οι οποίες κατά το στάδιο της σύνδεσης μπορούν να υπερκαλυφθούν από τις ισχυρές (strong) εκδοχές τους οι οποίες φέρουν την ίδια υπογραφή.

Fortran MPI_ interfaces are weak symbols for ...
Fortran PMPI_ interfaces, which call ...
C MPI_ interfaces, which are weak symbols for ...
C PMPI_ interfaces, which provide the specified functionality.

Συνεπώς η MinI αποτελείται από περιτυλίγματα (wrappers) των διεπαφών του MPI στη γλώσσα C οι οποίοι αφού εκτελέσουν την επιθυμητή επιπρόσθετη λειτουργία καλούν με τη σειρά τους το τελευταίο στάδιο που είναι οι διεπαφές του PMPI στη γλώσσα C για να πραγματοποιηθεί τελικά η επικοινωνία μεταξύ των διεργασιών. Ένα βασικό πλεονέκτημα αυτής της δομής είναι ότι έχουμε τη δυνατότητα να συλλέγουμε ίχνη από MPI εφαρμογές υλοποιημένες σε διάφορες προγραμματιστικές γλώσσες (π.χ. σε FORTRAN όπως φαίνεται και παραπάνω), χωρίς να χρειαζόμαστε υλοποίηση της βιβλιοθήκης στην εκάστοτε γλώσσα.

Στην τελική εκδοχή της η MinI προσφέρει περιτυλίγματα για τις εξής μεθόδους:

Ακόμα η βιβλιοθήκη χρησιμοποιεί διάφορες βοηθητικές συναρτήσεις και δομές εσωτερικά οι οποίες τις επιτρέπουν να διατηρεί δικά της δεδομένα και να μετασχηματίζει τις παραμέτρους των κλήσεων του MPI σε καταλληλότερη μορφή για εμφάνιση. Από αυτές, οι πιο σημαντικές παρουσιάζονται παρακάτω:

1. endode_datatype() : η βιβλιοθήκη στα αποτελέσματα παρουσιάζει τον τύπο παραμέτρου που χρησιμοποιείται σε μία κλήση κωδικοποιημένο. Συγκεκριμένα η δομή MPI_Datatype μετασχηματίζεται σε έναν ακέραιο αριθμό, σύμφωνα με τον παρακάτω πίνακα. Όπως φαίνεται, η MinI αναγνωρίζει τους περισσότερους τύπους του MPI και έχει τη δυνατότητα να αναγνωρίζει αν ένας τύπος είναι συνεχής (contiguous) ή όχι. Αυτό είναι ενδιαφέρον γιατί ο υπολογιστικός φόρτος για μια κλήση του προτύπου είναι διαφορετικός αν χρειάζεται μία ασυνεχής δομή να αντιγραφεί πρώτα πριν σταλεί. Ο περιορισμός στη συγκεκριμένη δυνατότητα είναι ότι για να μπορέσει η βιβλιοθήκη να αναγνωρίσει εάν ένας νέος τύπος είναι συνεχής, πρέπει να έχει δημιουργηθεί με τη χρήση της MPI_Type_contiguous() έτσι ώστε να προστεθεί στη λίστα με τους τύπους που διατηρεί η βιβλιοθήκη κατά το χρόνο εκτέλεσης της εφαρμογής. Οι τύποι που αναγνωρίζονται φαίνονται στον πίνακα 3.2.

2. Λιστα από custom MPI_Datatypes: όπως ήδη αναφέρθηκε παραπάνω η βιβλιοθήκη διατηρεί μία λίστα με τους διάφορους τύπους που δημιουργεί το MPI που δημιουργούνται κατά το χρόνο εκτέλεσης και είναι συνεχείς. Η λίστα αυτή αρχικοποιείται κατά την έναρξη λειτουργίας του προτύπου (MPI_Init()), αποδεσμεύεται κατά τη λήξη του και διατηρείται στη μνήμη καθε διεργασίας. Σε κάθε νέα δομή δίνεται ένα μοναδικό όνομα από τη βιβλιοθήκη και το οποίο χρησιμοποιείται σαν ταυτότητα της νέας δομής. Τα ονόματα είναι της μορφής mini_internal_{id}.

3. Λίστα από custom communicators : Η ίδια ακριβώς διαδικασία ακολουθείται και για τους communicators που δημιουργούνται με χρήση της MPI_Comm_split(). Τα ονόματα που χρησιμοποιούνται είναι της μορφής mini_comm_{id}_{color}.

4. Λίστα από custom async_metadata: η βιβλιοθήκη διατήρει σε μία λίστα στοιχεία για κάποιες ασύγχρονες κλήσεις που έχουν κληθεί αλλά δεν είναι ακόμα δεδομένο ότι έχουν ολοκληρωθεί. Η λειτουργία της συγκεκριμένης δομής περιγράφεται αναλυτικότερα αργότερα στο κεφάλαιο όπου αναλύεται ο χειρισμός των ασύγχρονων κλήσεων από τη βιβλιοθήκη.

3.3.1 Μέθοδοι επικοινωνίας σημείο προς σημείο

Η πρώτη κατηγορία συναρτήσεων που υποστηρίζονται είναι αυτές της επικοινωνίας σημείο προς σημείο όπου η εργασία που εκτελείται είναι της μορφής "ο αποστολέας στέλνει προς τον παραλήπτη ένα φορτίο συγκεκριμένου τύπου". Όλες οι συναρτήσεις της συγκεκριμένης κατηγορίας καταγράφουν την κλήση τους στην παρακάτω μορφή:
{αποστολέας} {λειτουργία} {παραλήπτης} {πλήθος} *(of* ***{μέγεθος}*** *bytes)* ***{κωδικός MPI_Type}*** *on comm* ***{όνομα communicator}***

όπου

- αποστολέας: η ταυτότητα του αποστολέα. Χρησιμοποιείται η παγκόσμια ταυτότητα της διεργασίας (global rank) ανεξαρτήτως του communicator που χρησιμοποιείται στη συγκεκριμένη κλήση. Σημειώνεται ότι ο λόγος που δεν χρησιμοποιείται η τοπική (local rank) είναι για να μην εμφανίζονται σε κάθε αρχείο εξόδου πολλαπλοί αποστολείς, δεδομένου ότι η τοπική ταυτότητα κάθε διεργασίας ανά communicator μπορεί να προσδιοριστεί από τις υπόλοιπες εγγραφές στο αρχείο.

- λειτουργία: το όνομα της κλήσης MPI. Αποτελείται από το όνομα της μεθόδου αφού αφαιρεθεί το πρόθεμα "MPI_". Για παράδειγμα στην κλήση της "MPI_Isend" χρησιμοποιείται το λεκτικό "Isend."

- παραλήπτης: η ταυτότητα του παραλήπτη. Εδώ χρησιμοποιείται η τοπική ταυτότητα της διεργασίας καθώς είναι σημαντική η πληροφορία.

- πλήθος: το πλήθος των αντικειμένων που θα μεταφερθούν. Αντιστοιχεί στο όρισμα count της μεθόδου.

- μέγεθος: το μέγεθος σε bytes του τύπου αντικειμένου που θα χρησιμοποιηθεί στη συγκεκριμένη κλήση. Υπολογίζεται με χρήση της μεθόδου MPI_Type_size().

- κωδικός MPI_Type: ένας αριθμός ο οποίος δείχνει το είδος του αντικειμένου που θα μεταφερθεί στη συγκεκριμένη κλήση. Υπολογίζεται με χρήση της μεθόδου encode_datatype().

- όνομα communicator: το όνομα του communicator στο περιβάλλον του οποίου εκτελείται η συγκεκριμένη κλήση. Υπολογίζεται με χρήση της μεθόδου MPI_Comm_get_name(). Σε περίπτωση που χρησιμοποιείται κάποιος custom communicator, τότε τυπώνεται στο αρχείο καταγραφής το όνομα που του έχει δώσει η MinI τη στιγμή που δημιουργήθηκε. Το όνομα υπολογίζεται με βάση τη λίστα από communicators που διατηρεί η εφαρμογή.

Μοναδική εξαίρεση στην παραπάνω μορφή των κλήσεων σημείου προς σημείο αποτελεί η MPI_Sendrecv. Όταν μια διεργασία MPI εκτελεί τη συγκεκριμένη εντολή παράγονται δύο μηνύματα, ένα με παραλήπτη τη συγκεκριμένη διεργασία και ένα όπου αυτή είναι αποστολέας. Χρησιμοποιείται συχνά από τους προγραμματιστές που κάνουν χρήση του προτύπου, όταν είναι επιθυμητό η σειρά των μηνυμάτων να καθορίζεται από τη ίδια τη βιβλιοθήκη και όχι από τον

προγραμματιστή απλοποιώντας έτσι περιπτώσεις όπου τα μηνύματα μεταδίδονται σύγχρονα και ένα λάθος στη σειρά μπορεί να οδηγήσει σε αμοιβαίο κλείδωμα deadlock των διεργασιών. Δεδομένου, λοιπόν, ότι παράγονται δύο μηνύματα, η MinI καταγράφει επίσης δύο εγγραφές στο αρχείο εξόδου, μία για κάθε μήνυμα, όπου κάθε εγγραφή ακολουθεί το πρότυπο εξόδου που έχει ήδη περιγραφεί.

3.3.2 Μέθοδοι συλλογικής επικοινωνίας

Από τη δεύτερη κατηγορία συναρτήσεων που υποστηρίζει η βιβλιοθήκη, προκύπτουν πολλαπλές μορφές αποτελεσμάτων καθώς υπάρχουν διαφορετικά είδη κλήσεων. Συγκεκριμένα, οι κλήσεις χωρίζονται με βάση δύο σημαντικά χαρακτηριστικά τους. Πρώτα, αν υπάρχει κάποια διεργασία ρίζα η οποία στέλνει σε όλες τις υπόλοιπες ή λαμβάνει από όλες τις υπόλοιπες, όπως οι MPI_Bcast() και MPI_Reduce(), ή όχι χαρακτηριστικό παράδειγμα των οποίων είναι η MPI_Allgather(). Στη συνέχεια χωρίζονται με βάση αν στέλνονται διανύσματα (vectors) ως δεδομένα καθώς για αυτές τις κλήσεις παρουσιάζονται τα μεγέθη συγκεντρωτικά. Με βάση τα παραπάνω προκύπτουν τα εξής τέσσερα είδη αποτελεσμάτων:

1. Multicast:
   {αποστολέας} {λειτουργία} {πλήθος} *(of* ***{μέγεθος}*** *bytes)* ***{ρίζα}*** *{* ***{κωδικός MPI Type}*** *or of types* ***{κωδικός MPI Type1}, {κωδικός MPI Type2}****}* *on comm* ***{όνομα communicator} {τοπικός αποστολέας}***

2. MulticastV:
   {αποστολέας} {λειτουργία} *min=****{min}*** *median=****{median}*** *max=****{max}*** *(of* ***{μέγεθος}*** *bytes)* ***{ρίζα}*** *{of type* ***{κωδικός MPI Type}*** *or of types* ***{κωδικός MPI Type1}, {κωδικός*** *MPI Type2}****}*** *on comm* ***{όνομα communicator} {τοπικός αποστολέας}***

3. AllMulticast:
   {αποστολέας} {λειτουργία} {πλήθος} *(of* ***{μέγεθος}*** *bytes) {of type* ***{κωδικός MPI Type}*** *or of types* ***{κωδικός MPI Type1}, {κωδικός MPI Type2}****}* *on comm* ***{όνομα communicator} {id}***

4. AllMulticastV:
   {αποστολέας} {λειτουργία} *min=****{min}*** *median=****{median}*** *max=****{max}*** *(of* ***{μέγεθος}*** *bytes) {of type* ***{κωδικός MPI Type}*** *or of types* ***{κωδικός MPI Type1}, {κωδικός*** *MPI Type2}****}*** *on comm* ***{όνομα communicator} {id}***

όπου σε όλα τα παραπάνω,

- αποστολέας: η ταυτότητα του αποστολέα. Όπως περιγράφεται και στις κλήσεις σημείο προς σημείο.
- λειτουργία: το όνομα της κλήσης MPI. Όπως περιγράφεται και στις κλήσεις σημείο προς σημείο.

- πλήθος: το πλήθος των αντικειμένων που θα μεταφερθούν. Όπως περιγράφεται και στις κλήσεις σημείο προς σημείο.

- μέγεθος: το μέγεθος σε bytes του τύπου αντικειμένου που θα χρησιμοποιηθεί στη συγκεκριμένη κλήση. Όπως περιγράφεται και στις κλήσεις σημείο προς σημείο.

- κωδικός MPI_Type: ένας αριθμός ο οποίος δείχνει το είδος του αντικειμένου που θα μεταφερθεί στη συγκεκριμένη κλήση. Υπολογίζεται με χρήση της μεθόδου encode_datatype(). Σε κάποιες κλήσεις εμφανίζονται δύο τιμές καθώς η συγκεκριμένη κλήση περιλαμβάνει ένα κομμάτι αποστολής και ένα κομμάτι παραλαβής δεδομένων και σε κάθε ένα χρησιμοποιείται διαφορετικός τύπος.

- ρίζα: η ταυτότητα της διεργασίας που λειτουργεί ως κόμβος ρίζα για τη συγκεκριμένη κλήση. Χρησιμοποιείται η τοπική ταυτότητα της διεργασίας μέσα στο περιβάλλον του communicator στον οποίο γίνεται η κλήση.

- min: το ελάχιστο πλήθος δεδομένων που θα σταλούν μεταξύ δύο διεργασιών κατά τη συγκεκριμένη κλήση. Χρησιμοποιείται μόνο όταν η κλήση είναι διανυσματική.

- median: η μέση τιμή των δεδομένων που θα σταλούν μεταξύ δύο διεργασιών κατά τη συγκεκριμένη κλήση. Χρησιμοποιείται μόνο όταν η κλήση είναι διανυσματική.

- max: το μέγιστο πλήθος δεδομένων που θα σταλούν μεταξύ δύο διεργασιών κατά τη συγκεκριμένη κλήση. Χρησιμοποιείται μόνο όταν η κλήση είναι διανυσματική.

- όνομα communicator: το όνομα του communicator στο περιβάλλον του οποίου εκτελείται η συγκεκριμένη κλήση. Όπως περιγράφεται και στις κλήσεις σημείο προς σημείο.

- id: στις κλήσεις στις οποίες δεν υπάρχει κάποια διεργασία ρίζα (δηλαδή τις AllMulticast κλήσεις, όπως για παράδειγμα η MPI_AllGather(», δεν υπάρχει κάποιος τρόπος να συσχετίσουμε τις διαφορετικές εγγραφές των αρχείων καταγραφής μεταξύ τους. Συνεπώς η βιβλιοθήκη για κάθε κλήση προσθέτει στο τέλος μια τιμή (έναν ακέραιο αριθμό) η οποία στη συνέχεια θα χρησιμοποιηθεί κατά το στάδιο της επεξεργασίας για να συγκεντρωθούν οι τιμές από όλα τα αρχεία καταγραφής σε ένα τελικό αρχείο εξόδου (αναλυτικότερα περιγράφεται παρακάτω). Αυτή η τιμή δημιουργείται και διατηρείται εσωτερικά στη βιβλιοθήκη και αυξάνεται σε κάθε κλήση.

- τοπικός αποστολέας: η ταυτότητα του αποστολέα στον communicator στον οποίο θα λάβει χώρα η κλήση MPI. Αυτή τιμή εμφανίζεται μόνο σε κλήσεις που χρησιμοποιούν κάποια διεργασία ως κόμβο ρίζα και δεν χρησιμοποιούν τον καθολικό communicator. Ο λόγος που υπάρχει είναι καθαρά τεχνικός, καθώς πρέπει τα επόμενα στάδια της επεξεργασίας να αναγνωρίζουν ποια από τις εγγραφές της κλήσης είναι αυτή του κόμβου ρίζα. Με τα υπόλοιπα στοιχεία της εγγραφής δεν ήταν δυνατό να γίνει αυτή η αντιστοίχιση καθώς ο αποστολέας χρησιμοποιεί την παγκόσμια ταυτότητά του ενώ η ρίζα την τοπική της.

3.3.3 Λοιπές μέθοδοι

Οι υπόλοιπες συναρτήσεις που υποστηρίζει η βιβλιοθήκη δεν έχουν άμεση σχέση με την αποστολή δεδομένων μεταξύ διεργασιών, αλλά συμπεριλαμβάνονται είτε λόγω της πληροφορίας που μπορούν να προσφέρουν στο αρχείο εξόδου είτε γιατί η βιβλιοθήκη εκτελεί εσωτερικά κώδικα κατά την κλήση τους έτσι ώστε να διασφαλιστεί η σωστή εκτέλεση. Από αυτές παρουσιάζονται παρακάτω οι πιο σημαντικές.

- MPI_Init(), MPI_Finalize(): είναι οι κλήσεις που σηματοδοτούν την έναρξη και λήξη της λειτουργίας του προτύπου. Η λειτουργία τους είναι σημαντική για τη βιβλιοθήκη, καθώς δεσμεύονται και ελευθερώνονται όλοι οι πόροι που θα χρειαστούν στη πορεία και επίσης καθορίζονται οι παράμετροι οι οποίες θα καθορίσουν τη μορφή των αποτελεσμάτων. Οι δύο αυτές κλήσεις διαχειρίζονται την αρχικοποίηση και τον κλείσιμο των αρχείων εξόδου.

- MPI_Comm_split(): Κατά τη συγκεκριμένη κλήση δημιουργείται ένας αριθμός από νέους communicators και οι διεργασίες χωρίζονται σε υποσύνολα. Από την πλευρά της MinI, εκτός από το να καλέσει την πραγματική κλήση του προτύπου, δημιουργείται και η αντίστοιχη εγγραφή στη δομή των communicators που διατηρεί εσωτερικά σε κάθε διεργασία. Κάθε νέος communicator λαμβάνει ένα όνομα, όπως έχει ήδη περιγραφεί. Σημειώνεται ότι η συγκεκριμένη κλήση είναι ο μόνος τρόπος που υποστηρίζεται από τη MinI για να δημιουργούνται νέοι communicators.

- MPI_Comm_free(): Η συγκεκριμένη κλήση χρησιμοποιείται για να καταργηθεί ένας communicator. Η βιβλιοθήκη αντίστοιχα διαγράφει μια εγγραφή από την λίστα που διατηρεί.

- MPI_Type_contiguous(): Η MinI καταγράφει, όπως έχει προαναφερθεί, τον τύπο των δεδομένων που χρησιμοποιείται σε κάθε αποστολή μηνύματος. Ενσωματώνοντας τη συγκεκριμένη κλήση μέσα στη βιβλιοθήκη, αυξάνεται η ικανότητά της να αναγνωρίζει επιπλέον τύπους. Συγκεκριμένα, πραγματοποιείται ένας έλεγχος αν ο νέος τύπος που θα δημιουργηθεί αποτελείται από έναν προηγούμενο συνεχή τύπο και αν ο έλεγχος είναι αληθής, τότε προστίθεται ο νέος τύπος στη δομή που διατηρεί.

- MPI_Wait(), MPI_Waitall(), MPI_Waitany(): Μια βασική λειτουργία του MPI είναι οι ασύγχρονες ανταλλαγές μηνυμάτων. Η βιβλιοθήκη επίσης υποστηρίζει ασύγχρονες κλήσεις (όπως οι MPI_Isend() και MPI_Irecv()). Στις συγκεκριμένες δύο συναρτήσεις, η βιβλιοθήκη αφού εκτελέσει την πραγματική κλήση, τυπώνει και όσα ασύγχρονες παραλαβές δεδομένων (MPI_Isend) ολοκληρώθηκαν.

- mini_annotate_phase_start(), mini_annotate_phase_end(): Οι συγκεκριμένες δύο συναρτήσεις είναι οι μοναδικές οι οποίες δεν αντιστοιχούν σε κάποια κλήση του προτύπου MPI και ο προγραμματιστής πρέπει να τις εισάγει μόνος του στον κώδικά του αν το επιθυμεί. Η χρήση τους δημιουργεί στο εκάστοτε αρχείο εξόδου μια εγγραφή της μορφής

"Phase {id} start.", "Phase {id} end." αντίστοιχα όπου το id είναι μία συμβολοσειρα που ορίζεται από το χρήστη και δίνεται ως παράμετρος εισόδου σε αυτές. Με τη χρήση τους επιτυγχάνεται να χωριστεί το αποτέλεσμα της συλλογής ιχνών σε λογικές ενότητες οι οποίες αντιπροσωπεύουν και διαφορετικές φάσεις του προγράμματος. Ακόμα επιτρέπεται και υποστηρίζεται τόσο από τη βιβλιοθήκη όσο και από τα επόμενα στάδια της εργαλειοθήκης οι φάσεις αυτές να είναι εμφωλευμένες μεταξύ τους, αλλά δεν επιτρέπεται να ξεκινάει μια νέα φάση εντός μίας άλλης αλλά να ολοκληρώνεται μετά τη λήξη της αρχικής. Ο λόγος που η βιβλιοθήκη δίνει αυτή τη δυνατότητα στο χρήστη είναι διότι είναι συχνό φαινόμενο οι παράλληλες εφαρμογές να αλλάζουν το σχήμα της επικοινωνίας του ανάλογα με τη φάση στην οποία βρίσκονται. Αυτό το φαινόμενο λαμβάνεται υπόψη και από τα διάφορα μοντέλα πρόβλεψης του χρόνου επικοινωνίας.

3.4 Επιπρόσθετες λειτουργίες

Εκτός από τη δυνατότητα η βιβλιοθήκη να συνδεθεί με εφαρμογές οι οποίες είναι υλοποιημένες σε διαφορετική γλώσσα προγραμματισμού, η τελική έκδοση της βιβλιοθήκης προσφέρει ακόμα τις εξής δυνατότητες, οι οποίες επιλέγονται κατά το χρόνο μεταγλώττισης της βιβλιοθήκης:

- Προαιρετική χρήση των σφραγίδων χρόνου (timestamps). Η MinI κάνει χρήση των PAPI counters για να μπορέσει να μετρήσει και στη συνέχεια να εμφανίσει τους κύκλους ρολογιού που απαιτούνται ανάμεσα σε δύο διαδοχικές κλήσεις MPI. Σημειώνεται εδώ ότι η βιβλιοθήκη υποστηρίζει ακόμα τη χρήση των μετρητών αυτών, αλλά για την εκτέλεση του συνόλου της ροής της εργαλειοθήκης, απαιτείται η μεταγλώττιση της βιβλιοθήκης με τους μετρητές απενεργοποιημένους. Η συγκεκριμένη λειτουργία καθορίζεται από τη σημαία PAPI.

- Προαιρετική εκτέλεση των εντολών MPI. Σε περίπτωση που θέλουμε ταχύτερο εκτέλεση ή το περιβάλλον προσομοίωσης έχει πολύ περιορισμένους πόρους μνήμης και η εφαρμογή το υποστηρίζει μπορεί να συνδεθεί μια έκδοση της βιβλιοθήκης στην εφαρμογή η οποία θα καταγράφει κανονικά τα απαιτούμενα στατιστικά αλλά δεν θα εκτελεί τις πραγματικές κλήσεις MPI. Αυτός ο τύπος εκτέλεσης υποστηρίζεται μόνο από μία κατηγορία εφαρμογών και συγκεκριμένα από εφαρμογές των οποίων οι κλήσεις MPI εκτελούν μόνο μεταφορές δεδομένων και υπολογισμούς οι οποίοι δεν επηρεάζουν τη ροή εκτέλεσης του προγράμματος (για παράδειγμα συνθήκες τερματισμού). Η συγκεκριμένη λειτουργία καθορίζεται από τη σημαία WITH_MPI.

- Παραμετροποίηση ενδιάμεσων πινάκων: Οι βιβλιοθήκη χρησιμοποιεί μερικούς ενδιάμεσους πίνακες για να αποθηκεύει προσωρινά τις εγγραφές της για λόγους που εξηγούνται στην αμέσως επόμενη ενότητα. Το μέγεθος των συγκεκριμένων πινάκων και ο τρόπος χρήσης τους μπορεί να καθοριστεί κατά το στάδιο της μεταγλώττισης χρησιμοποιώντας της σημαίες BUFSIZE, BUFCNT και TMPSIZE.

3.5 Λεπτομέρειες Υλοποίησης

Για να προκύψει η τελική έκδοση της βιβλιοθήκης, λήφθηκαν διάφορες αρχιτεκτονικές αποφάσεις και ακολουθήθηκαν οι περισσότερες από τις υπάρχουσες στην αρχική έκδοση. Οι βασικότερες από αυτές περιγράφονται στη συγκεκριμένη ενότητα. Αρχικά όλες οι επιπρόσθετες λειτουργίες όπως έχουν ήδη περιγραφεί καθορίζονται κατά το χρόνο μεταγλώττισης της εφαρμογής και ο αντίστοιχος κώδικας που υλοποιεί τη λειτουργικότητά τους περικλείεται από τη μακροεντολή #ifdef. Συνεπώς τα τμήματα αυτά ανάλογα με την τιμή της αντίστοιχης σημαίας συμπεριλαμβάνονται ή αποκόπτονται από τον προεπεξεργαστή.

Σε κάθε κλήση του προτύπου όπως αναφέρθηκε τυπώνεται το είδος του MPI_Datatype κωδικοποιημένο και το μέγεθος του σε byte. Η συγκεκριμένες πληροφορίες συγκεντρώνονται κάνοντας χρήση την κλήσεων MPI_Type_size() και MPI_Type_get_name(). Η πρώτη δίνει το μέγεθος σε bytes, ενώ η δεύτερη αφού επιστρέφει μία συμβολοσειρά με το όνομα του τύπου η οποία στη συνέχεια κωδικοποιείται με χρήση της encode_datatype(). Επιπλέον, ο αριθμός των μεταφερόμενων αντικειμένων που τυπώνεται στο αρχείο εξόδου προκύπτει φυσικά από την τιμή της παραμέτρου sendcount (ή recvcount σε κάποιες περιπτώσεις) της εκάστοτε κλήσεις. Στις κλήσεις που περιλαμβάνουν διανυσματικό αριθμό παραμέτρων επιλέχθηκε να εμφανίζεται στο αρχείο καταγραφής μία σύνοψη των δεδομένων που δείχνει το μικρότερο, το μέγιστο και τη διάμεσο του πλήθους των αντικειμένων που μεταφέρονται. Οι τιμές αυτές για τα δύο πρώτα χαρακτηριστικά προκύπτουν με χρήση βοηθητικών συναρτήσεων οι οποίες διατρέχουν γραμμικά (O(n)) τον πίνακα των sendcounts (ή recvcounts), ενώ για τη διάμεσο ο πίνακας αυτός πρώτα ταξινομείται (O(nlogn)) και στη συνέχεια με μια απλή πράξη επιλέγεται η διάμεσος ανάλογα με το πλήθος των στοιχείων. Για την ταξινόμηση επιλέχθηκε ο αλγόριθμος της mergesort. Οι δομές που διατηρεί η βιβλιοθήκη για να γνωρίζει πόσοι communicators υπάρχουν ενεργοί, πόσοι ορισμένοι από το χρήστη συνεχείς τύποι δεδομένων είναι διαθέσιμοι και πόσες ασύγχρονες κλήσεις δεν έχουν ακόμα ολοκληρωθεί, υλοποιήθηκαν με χρήση απλά συνδεδεμένων λιστών που υλοποιούνται από δομές (structs) της γλώσσας C. Οι δομές που δημιουργήθηκαν φαίνονται στον πίνακα 3.3.

Όπως παρατηρούμε, η δομή για τους communicators διατηρεί επίσης και τη τιμή του id που θα χρησιμοποιηθεί στις συλλογικές κλήσεις που δεν διαθέτουν κάποιο κόμβο ρίζα όπως περιγράφηκε. Η πολυπλοκότητα για τη εισαγωγή στοιχείου, διαγραφή στοιχείου και αναζήτηση ενός στοιχείου σε αυτές τις λίστες είναι O(1), O(n) και O(n) αντίστοιχα. Καθώς όμως, συνήθως, το πλήθος των στοιχείων της κάθε λίστας σε μια τυπική εφαρμογή είναι μικρό, δεν θεωρήθηκε αναγκαίο να δημιουργηθεί κάποια πιο περίπλοκη δομή η οποία θα βελτιστοποιούσε το χρόνο αναζήτησης. Τέλος, διευκρινίζεται ότι κάθε διεργασία διατηρεί το δικό της αντίγραφο αυτών των λιστών, όπως είναι φυσικό.

Ένα από τα βασικότερα προβλήματα προς επίλυση της βιβλιοθήκης είναι η ανάγκη για συνεχή αλληλεπίδραση με το σύστημα αρχείων και η διαρκής καταγραφή των αποτελεσμάτων. Μάλιστα, το πρόβλημα αυτό εντείνεται όταν ο αριθμός των διεργασιών αυξάνεται με πιθανή συνέπεια να οδηγήσει σε πολύ μεγάλη χρονική καθυστέρηση ή ακόμα και σε σφάλμα εκτέλεσης όταν το σύστημα δεν είναι ικανό να διαθέσει τους απαραίτητους πόρους να εξυπηρετήσει

όλα τα αιτήματα. Για να ανταπεξέλθει σε αυτό το φαινόμενο η βιβλιοθήκη χρησιμοποιεί δύο επίπεδα εγγραφής. Σε πρώτη φάση κάθε διεργασία διατηρεί τοπικά στη μνήμη της ένα πίνακα που καταγράφονται τα δεδομένα εξόδου (buffer). Μόλις αυτός ο πίνακας γεμίσει, η βιβλιοθήκη αντιγράφει το περιεχόμενό του στο αρχείο εγγραφής και καθαρίζει τον πίνακα για νέα χρήση. Συνεπώς, αντί να χρησιμοποιεί το αρχείο καταγραφής σε κάθε κλήση του προτύπου, το χρησιμοποιεί μόνο όταν ο πίνακας συμπληρωθεί. Χρησιμοποιείται μια καθολική μεταβλητή (global variable) για να ελέγχει η βιβλιοθήκη πότε ο πίνακας αυτός συμπληρώθηκε. Σε κάθε κλήση η μεταβλητή αυτή αυξάνει κατά ένα για κάθε τμήμα μηνύματος που προστίθεται. Παρόλο που αυτός ο τρόπος μπορεί να μην είναι απόλυτα ακριβής, καθώς το μέγεθος σε bytes των μηνυμάτων δεν είναι πάντα το ίδιο, πειραματικά προέκυψε ότι είναι αποδεκτός καθώς δεν ανιχνεύτηκαν σφάλματα.

Όπως αναφέρθηκε η MinI υποστηρίζει και ασύγχρονες κλήσεις. Θεωρείται ότι ο χρόνος ζωής μιας ασύγχρονης κλήσης είναι από τη στιγμή που το μήνυμα καταχωρείται για αποστολή (π.χ. MPI_Isend()) μέχρι τη στιγμή που το μήνυμα επιβεβαιώνεται ότι λήφθηκε (π.χ. MPI_Wait()). Η βιβλιοθήκη ακολουθεί την εξής στρατηγική για να καταγράφει τις συγκεκριμένες κλήσεις. Η κλήση MPI_Isend() καταγράφεται κανονικά όπως όλες οι κλήσεις σημείο προς σημείο. Όταν καλείται η MPI_Irecv() δεν καταγράφεται κάποιο μήνυμα στο αρχείο εξόδου, αλλά αλλάζει η κατάσταση της βιβλιοθήκης σε ασύγχρονη (μέσω της καθολικής μεταβλητής i_mode) και ακόμα αυξάνεται η καθολική μεταβλητή i_counter κατά 1, η οποία μετράει τις ασύγχρονες κλήσεις που δεν έχουν ακόμα ολοκληρωθεί. Ακόμα δημιουργείται μια νέα δομή τύπου async_metadata στην οποία καταχωρούνται το πλήθος, το είδος και το μέγεθος των δεδομένων που θα μεταφερθούν καθώς και το όνομα του communicator στο περιβάλλον του οποίου θα εκτελεστεί η μεταφορά. Αυτά είναι τα απαραίτητα δεδομένα για τη σωστή εγγραφή στο αρχείο εξόδου και τα οποία στη συνέχεια δεν θα είναι διαθέσιμα. Στη συνέχεια αυτή η δομή προστίθεται στην αντίστοιχη λίστα.Για όση διάρκεια η βιβλιοθήκη βρίσκεται σε ασύγχρονη λειτουργία τα περισσότερα μηνύματα των κλήσεων σημείου προς σημείο καταγράφονται σε ένα δεύτερο προσωρινό πίνακα έτσι ώστε να μην διαταραχθεί η τελική σειρά εμφάνισης στο αρχείο εξόδου. Μόλις γίνει μία κλήση στην MPI_Wait(), στην MPI_Waitall() ή στη MPI_Waitany(), τότε για κάθε MPI_Request για το οποίο το πρότυπο θα ολοκληρωθεί και το οποίο αντιστοιχεί σε κλήση της MPI_Irecv, η βιβλιοθήκη τυπώνει το αντίστοιχο Irecv μήνυμα που δεν είχε τυπώσει νωρίτερα και μειώνει το i_counter κατά 1. Για να βρεθούν τα απαραίτητα δεδομένα για τη σωστή καταγραφή, η βιβλιοθήκη κάνει αναζήτηση στη λίστα με τα async_metadata με βάση το MPI_Request το οποίο χρησιμοποιείται ως κλειδί. Αν βρεθεί καταχώρηση στη λίστα τότε η βιβλιοθήκη την τυπώνει και την αφαιρεί. Σε αντίθετη περίπτωση θα τυπώσει ένα απλό μήνυμα Isend όπου θα εμφανίζεται μόνο το αποστολέας και ο παραλήπτης. Σε περίπτωση που δεν εκκρεμεί κάποια άλλη ασύγχρονη κλήση, δηλαδή ο μετρητής μηδενιστεί, τότε μεταφέρει όλα τα μηνύματα από τον προσωρινό πίνακα στον κύριο πίνακα και αλλάζει ξανά την κατάσταση της βιβλιοθήκης σε σύγχρονη λειτουργία.

Τέλος, αφού διαβάσει τις παραπάνω παραγράφους σχετικά με τη σημαντικότητα και τη χρήση των ενδιάμεσων πινάκων καταγραφής, ο χρήστης της βιβλιοθήκης είναι σημαντικό να γνωρίζει άλλη μια παράμετρο σχετικά με τον τρόπο λειτουργίας τους. Το μέγεθος τόσο του

κύριου πίνακα όσο και του προσωρινού είναι παραμετροποιήσιμο κατά το χρόνο μεταγλώττισης. Συγκεκριμένα η βιβλιοθήκη επιτρέπει στον προγραμματιστή να θέσει τρεις παραμέτρους με ονόματα BUFSIZE, BUFCNT και TMPSIZE. Οι παράμετροι αυτοί επιτρέπουν στο χρήστη να ορίσει το μέγεθος του κύριου πίνακα, το όριο το οποίο αν ο πίνακας ξεπεράσει θα γράψει το περιεχόμενό του στο δίσκο και το μέγεθος του προσωρινού πίνακα αντίστοιχα. Για την παράμετρο BUFCNT ορίζεται ότι μετράει τον αριθμό από οχτάδες χαρακτήρων που αποθηκεύονται στον κύριο πίνακα. Για παράδειγμα μια τιμή 100 σημαίνει ότι ο κύριος πίνακας θα δεχτεί 800 χαρακτήρες πριν μεταφέρει το περιεχόμενό του στο δίσκο. Είναι αρκετά σημαντικό ο χρήστης να κατανοεί τον αντίκτυπο των παραμέτρων αυτών στη λειτουργία της βιβλιοθήκης καθώς αν τεθούν ελλειπώς μπορεί να προκαλέσουν καθυστερήσεις στο χρόνο εκτέλεσης ή ακόμα και να οδηγήσουν στην αποτυχία της εκτέλεσης. Θέτοντας μεγάλο μέγεθος πίνακα ο χρήστης κερδίζει σε ασφάλεια καθώς είναι δύσκολο να ξεμείνει από χώρο αποθήκευσης και επίσης κερδίζει σε απόδοση καθώς, με ταυτόχρονη αύξηση του ορίου, μειώνεται η ανάγκη αλληλεπίδρασης με το λειτουργικό και το σύστημα αρχείων. Από την άλλη όμως, δεδομένου ότι κάθε MPI διεργασία θα διατηρεί αυτούς τους πίνακες στη μνήμη της αυξάνονται πιθανά οι απαιτήσεις σε μνήμη. Αντίστοιχα μια μικρή τιμή του ορίου θα οδηγήσει σε ασφάλεια ότι πιο συχνά θα εγγράφονται δεδομένα στο δίσκο αλλά θα έχει και αντίκτυπο στην απόδοση αφού θα αυξηθούν τα I/O. (Σημείωση: όσον αφορά τον προσωρινό πίνακα δεν υπάρχει αντίστοιχο όριο για έλεγχο, συνεπώς ο χρήστης θα πρέπει να είναι προσεκτικός με την τιμή του. Στα πειράματα που εκτελέστηκαν στα πλαίσια της διπλωματικής οι παράμετροι BUFSIZE και TMPSIZE είχαν μεταξύ τους μια τάξη μεγέθους διαφορά. Σε περίπτωση που ένας χρήστης γνωρίζει ότι η εφαρμογή του θα βρίσκεται συχνά σε ασύγχρονη λειτουργία, όπως αυτή έχει περιγραφεί, ίσως χρειαστεί να μεγαλώσει το μέγεθος του προσωρινού πίνακα)

3.6 Εξαγωγή στατιστικών

Η βιβλιοθήκη αν και δεν έχει σχεδιαστεί με αυτό το σκοπό μπορεί να εκτελέσει μια επιπλέον εργασία. Δεδομένου ότι τα παραγόμενα αρχεία καταγραφής είναι σε αναγνώσιμη μορφή, είναι αρκετά εύκολο με χρήση απλών εντολών που προσφέρει το UNIX να εξαχθούν βασικά στατιστικά σχετικά με τον τρόπο χρήσης του προτύπου MPI από μία συγκεκριμένη εκτέλεση μίας εφαρμογής.

Αυτά μπορεί να είναι αρκετά απλά, όπως για παράδειγμα να εξετάσει ο χρήστης αν μία συγκεκριμένη κλήση έχει χρησιμοποιηθεί από την εφαρμογή, όπως φαίνεται στο πρώτο μέρος της εικόνας, 3.1, όμως μπορούν να είναι και αρκετά πιο σύνθετα όπως το να μετρήσει τον αριθμό των χρήσεων μίας συγκεκριμένης κλήσης από μία εκτέλεση, δεύτερο μέρος της ίδιας εικόνας, ή ακόμα και να παράξει στατιστικά σχετικά με το φορτίο των κλήσεων. Προφανώς οι επιλογές είναι πολλές και εξαρτώνται από το χρήστη. Την ίδια δυνατότητα προσφέρουν και τα αρχεία εξόδου του επόμενου σταδίου, όμως κυρίως όσον αφορά τις συλλογικές κλήσεις, θεωρούμε ότι η εξαγωγή των στατιστικών θα είναι πιο χρήσιμη σε επίπεδο διεργασιών και όχι εφαρμογής.

Για την εξαγωγή στατιστικών χρησιμοποιήθηκε μία εκτέλεση του μετροπρογράμματος

CoMD με 256 διεργασίες, το οποίο θα παρουσιαστεί και στο κεφάλαιο της αξιολόγησης. Είναι μία εφαρμογή η οποία γνωρίζουμε ότι δεν χρησιμοποιεί ασύγχρονες κλήσεις και βασίζεται κυρίως σε κλήσεις σημείο σε σημείο, χρησιμοποιώντας την MPI_Sendrecv(). Καθώς γνωρίζουμε ότι η συγκεκριμένη κλήση δημιουργεί δύο εγγραφές στο αρχείο καταγραφής, μπορούμε να μετρήσουμε πόσες φορές χρησιμοποιήθηκε.

```
fkotomat@scirouter:~/clones/CoMD/bin$ cd ti_traces/
fkotomat@scirouter:~/clones/CoMD/bin/ti_traces$
fkotomat@scirouter:~/clones/CoMD/bin/ti_traces$
fkotomat@scirouter:~/clones/CoMD/bin/ti_traces$ grep "Isend" ./* | wc -l
0
fkotomat@scirouter:~/clones/CoMD/bin/ti_traces$
fkotomat@scirouter:~/clones/CoMD/bin/ti_traces$
fkotomat@scirouter:~/clones/CoMD/bin/ti_traces$
fkotomat@scirouter:~/clones/CoMD/bin/ti_traces$ grep "Sendrecv" ./* | wc -l
310272
fkotomat@scirouter:~/clones/CoMD/bin/ti_traces$
```

Σχήμα 3.1: Παράδειγμα εξαγωγής στατιστικών

3.7 Παραδείγματα χρήσης

Τα παραδείγματα της χρήσης της MinI, τελικά προστέθηκαν στο επόμενο κεφάλαιο και παρουσιάζονται συγκεντρωτικά μαζί με αυτά του MinI Parser καθώς δεδομένου ότι αυτά τα δύο εργαλεία έχουν σχεδιαστεί για να λειτουργούν αλληλένδετα, θεωήθηκε καλύτερο να ενωθούν σε μία ενότητα. Με αυτό τον τρόπο ο αναγνώστης θα κατανοήσει καλύτερα τη χρήση τους και τις διαφορες των εργαλείων. Υπενθυμίζεται ότι η MinI είναι σχεδιασμένη για να εξάγει την εικόνα επικοινωνίας ανά διεργασία, ενώ ο MinI Parser εξάγει το συνολικό σχήμα της επικοινωνίας ανά φάση.

Είδος επικοινωνίας	**Υποστηριζόμενες μέθοδοι**
Σημείο προς σημείο (p2p)	MPI_Send() MPI_Bsend() MPI_Rsend() MPI_Isend() MPI_Recv() MPI_Irecv() MPI_Sendrecv()
Συλλογική (collective)	MPI_Allreduce() MPI_Alltoallv() MPI_Reduce() MPI_Bcast() MPI_Reduce_scatter() MPI_Allgatherv() MPI_Allgather() MPI_Gatherv() MPI_Alltoall() MPI_Gather()
Λοιπές (other)	MPI_Init() MPI_Finalize() MPI_Comm_size() MPI_Comm_rank() MPI_Wait() MPI_Waitall() MPI_Waitany() MPI_Test() MPI_Type_contiguous() MPI_Comm_free() MPI_Comm_split() MPI_Initialized() MPI_Init_thread() MPI_Barrier() mini_annotate_phase_start() mini_annotate_phase_end()

Πίνακας 3.1: Υποστηριζόμενες Κλήσεις Βιβλιοθήκης

MPI_Datatype	**Κωδικός**
MPI_DOUBLE_PRECISION, MPI_DOUBLE	0
MPI_INTEGER, MPI_INT	1
MPI_CHARACTER and MPI_CHAR	2
MPI_SHORT	3
MPI_LONG	4
MPI_REAL, MPI_FLOAT	5
MPI_BYTE	6
custom contiguous MPI_Datatype	100
custom non contiguous MPI_Datatype	101

Πίνακας 3.2: Κωδικοποίηση τύπων MPI

Δομές		
Communicator	Συνεχής Τύπος	Ασύγχρονο
typedef struct comm{ int cnt; char name[100]; struct comm *next; }communicator;	typedef struct contig{ char name[100]; struct contig *next; }contiguous;	typedef struct mpi_request_meta{ int id; int count; int size; int data_code; char comm_name[100]; struct mpi_request_meta *next; }mpi_request_metadata;

Πίνακας 3.3: Εσωτερικές δομές βιβλιοθήκης

Κεφάλαιο 4

MinI Parser

4.1 Εισαγωγή

Το δεύτερο στοιχείο της εργαλειοθήκης που αναπτύχθηκε είναι ένας αναλυτής με τη χρήση του οποίου επιτυγχάνεται η ανάγνωση και μετατροπή των αρχείων εξόδου του προηγούμενου σταδίου (MinI). Είναι μία εφαρμογή που αναπτύχθηκε εξ ολοκλήρου στα πλαίσια της συγκεκριμένης διπλωματικής και υλοποιήθηκε σε γλώσσα Python. Βασική της λειτουργία είναι να ξεχωρίσει και να ομαδοποιήσει τα αποτελέσματα του προηγούμενου σταδίου και να παράγει το συνολικό σχήμα επικοινωνίας μίας εφαρμογής σε μια μορφή κατάλληλη να τροφοδοτήσει την είσοδο του νευρωνικού δικτύου.

4.2 Μεταγλώττιση και τρόπος χρήσης

Ο αναλυτής περιέχεται στο αρχείο parser.py και εφόσον αναπτύχθηκε στη γλώσσα προγραμματισμού Python δεν χρειάζεται μεταγλώττιση. Μόνη απαίτηση είναι να είναι εγκατεστημένος στο σύστημα ο διερμηνέας της γλώσσας. Ως είσοδο η εφαρμογή δέχεται την τοποθεσία στο σύστημα αρχείων του φακέλου που περιέχει τα αρχεία εξόδου της MinI (συνήθως δίνεται απευθείας το απόλυτο μονοπάτι προς το φάκελο ti_traces που έχει προκύψει από τη διαδικασία συλλογής ιχνών). Ως έξοδο η εφαρμογή παράγει τα εξής αρχεία:

- communicators.txt: είναι ένα αρχείο κειμένου που περιέχει όλους τους communicators που κλήθηκαν και καταγράφηκαν στα αρχεία της MinI. Συγκεκριμένα αποτελείται από ζεύγη κλειδιών-τιμών, όπου κλειδί είναι το όνομα του communicator και τιμή είναι μια δομή (σε Json αναπαράσταση) που περιέχει το μέγεθος τους συγκεκριμένου communicator και μια λίστα με τις παγκόσμιες διευθύνσεις των διεργασιών που τον αποτελούν.

- global.txt: είναι ένα αρχείο κειμένου που περιέχει τα διαμορφωμένα αποτελέσματα της εκτέλεσης της MPI εφαρμογής. (θα αναλυθεί περισσότερο στη συνέχεια)

- {phase_name}.txt: είναι αρχεία της ίδιας ακριβώς μορφής με το global.txt. Δημιουργείται ένα τέτοιο αρχείο για κάθε φάση της MPI εφαρμογής, όπου φάση ορίζεται το μπλοκ

αποτελεσμάτων που περικλείεται από μηνύματα αρχής και τέλους μιας φάσης (χρήση mini_annotate_phase_start() και mini_annotate_phase_end()).

Σημείωση: Αν δεν υπάρχουν επιπλέον φάσεις, τότε θα παραχθεί μόνο το αρχείο global.txt.

4.3 Υλοποίηση

Η κατασκευή του αναλυτή μπορεί να διαχωριστεί σε τρεις μεγάλες λογικές ενότητες. Η πρώτη είναι η φάση αρχικοποίησης του αναλυτή στην οποία θα συμπεριληφθεί και η φάση τερματισμού καθώς είναι απλή τόσο σε λογική όσο και σε μέγεθος. Η δεύτερη είναι η διαδικάσία της ανάγνωσης από τα αρχεία καταγραφής της MinI και η τρίτη είναι η διαδικασία επεξεργασίας κάθε γραμμής και η καταγραφή στη νέα μορφή.

4.3.1 Αρχικοποίηση και τερματισμός

Όπως αναφέρθηκε ο αναλυτής δέχεται ως όρισμα κατά την κλήση του το μονοπάτι του φακέλου μέσα στον οποίο βρίσκονται τα αρχεία ti_tracesXX.txt που έχουν δημιουργηθεί στο προηγούμενο στάδιο. Σε πρώτο βήμα ο αναλυτής μεταφέρει το περιβάλλον εκτέλεσης στο συγκεκριμένο φάκελο. Στη συνέχεια ανοίγει όλα τα αρχεία κειμένου που περιέχονται σε αυτόν για ανάγνωση και τα προσθέτει στο σύνολο των αρχείων εισόδου. Ακόμα αρχικοποιεί το σύνολο των communicators που θα καταγραφούν κατά την ανάλυση και εισάγει τον MPI_COMM_WORLD σε αυτό. Τέλος αρχικοποιούνται και δημιουργούνται στο σύστημα αρχείων οι φάκελοι και τα ελάχιστα απαραίτητα αρχεία που θα προκύψουν από την εκτέλεση. Συγκεκριμένα δημιουργείται ο φάκελος "parserOutput" μέσα στον οποίο θα περιέχονται όλα τα αρχεία εξόδου του συγκεκριμένου αρχείου και μέσα σε αυτόν δημιουργούνται τα αρχεία "global.txt" που θα καταγραφεί το αποτέλεσμα της ανάλυσης (περιγράφεται αναλυτικότερα στη συνέχεια) και "communicators.txt" στο οποίο θα καταγραφούν όλοι οι communicators που θα ανακαλυφθούν.
Στη φάση τερματισμού, αρχικά αποδεσμεύονται όλα τα αρχεία που απαρτίζουν το σύνολο εισόδου. Στη συνέχεια τα περιεχόμενα της λίστας των communicators μετατρέπονται σε μορφή Json και περνάνε στο αντίστοιχο αρχείο το οποίο αποδεσμεύεται με τη σειρά του. Τέλος κλείνει και το αρχείο "global.txt" μόλις αποθηκευτεί και η τελευταία εγγραφή σε αυτό.

4.3.2 Ανάγνωση από αρχεία καταγραφής

Ο αναλυτής συλλέγει δεδομένα από τα αρχεία καταγραφής με Round Robin τρόπο, δηλαδή κυκλικά διαβάζει μια γραμμή από κάθε αρχείο και προχωράει στο επόμενο. Επιλέχθηκε αυτός ο τρόπος γιατί παρόλο που το επόμενο στάδιο δεν ασχολείται ιδιαίτερα με την ακριβή σειρά των μηνυμάτων αλλά περισσότερο με την πυκνότητα ροής και ανταλλαγής τους κατά τη διάρκεια μίας φάσης, τα αποτελέσματα που προκύπτουν είναι πιο ακριβή και πιο αντιπροσωπευτικά της πραγματικής ροής. Για την εκτέλεση του αλγορίθμου χρησιμοποιούνται τρεις λίστες μέσα στις οποίες κατανέμονται τα αρχεία καταγραφής. Η πρώτη είναι το τρέχον σύνολο ανάγνωσης,

στην οποία θα αναφερόμαστε ως ReadSet. Η δεύτερη είναι το σύνολο ανάγνωσης της επόμενης φάσης, στην οποία θα αναφερόμαστε ως NextPhaseSet και η οποία περαιτέρω χωρίζεται σε δύο λίστες (nextPhaseStartSet και nextPhaseEndSet για τη αρχή και λήξη μιας φάσης αντίστοιχα) . Η τελευταία περιέχει το σύνολο των ολοκληρωμένων αρχείων, στην οποία θα αναφερόμαστε ως completedSet.
Όλα τα αρχεία που ανοίχθηκαν στην προηγούμενη φάση, αρχικά τοποθετούνται στο ReadSet. Ό αναλυτής διαβάζει μια γραμμή από το πρώτο αρχείο στο ReadSet. Σε περίπτωση που αυτή είναι κενή, τότε το συγκεκριμένο αρχείο έχει επεξεργαστεί πλήρως και συνεπώς ο αναλυτής το μεταφέρει στο completedSet. Αν η γραμμή είναι δείκτης που σηματοδοτεί την έναρξη μιας νέας φάσης ή τη λήξη της τρέχουσας τότε ο αναλυτής το χειρίζεται στο συγκεκριμένο στάδιο, όπως αναλύεται αμέσως παρακάτω. Σε οποιαδήποτε άλλη περίπτωση, η γραμμή μεταφέρεται στο επόμενο στάδιο προς περαιτέρω επεξεργασία και το τρέχων αρχείο ανάγνωσης τοποθετείται ξανά στο τέλος του ReadSet.
Όταν μια γραμμή σηματοδοτεί αλλαγή κατάστασης στην τρέχουσα φάση, τότε ο αναλυτής εκτελεί μια σύνθετη επεξεργασία σε αυτό το στάδιο για να προετοιμάσει την επόμενη φάση. Καταρχάς, υπενθυμίζεται σε αυτό το σημείο ότι τα μηνύματα αλλαγής φάσης πρέπει να είναι παγκοσμίως (ενγλοβαλλψ) ορατά, δηλαδή όλες οι διεργασίες πρέπει να καλούν τις αντίστοιχες μεθόδου ακόμα και αν κατά τη διάρκεια της φάσης δεν εκτελούν καμία ενέργεια. Συνεπώς, όταν αναγνωρίζεται ένα μήνυμα το οποίο σηματοδοτεί την έναρξη μιας νέας φάσης για την τρέχουσα διεργασία τότε το αντίστοιχο αρχείο καταγραφής μεταφέρεται από το ReadSet στο nextPhaseStartSet και με βάση το αναγνωριστικό (id) της νέας φάσης δημιουργείται το όνομα του νέου αρχείου εξόδου που θα δημιουργηθεί. Δεδομένου ότι όλες οι διεργασίες θα έχουν την ίδια καταγραφή σε κάποιο σημείο του αρχείου καταγραφής τους, τελικά όλες οι μη τερματισμένες διεργασίες θα μεταφερθούν στο nextPhaseStartSet και το ReadSet θα μείνει κενό. Σε αυτό το σημείο, ο αναλυτής αντιλαμβάνεται ότι πρέπει να αλλάξει κατάσταση. Θα δημιουργήσει συνεπώς αρχικά το νέο αρχείο εξόδου με βάση το όνομα που έχει ήδη υπολογιστεί. Στη συνέχεια θα τυπώσει επεξηγηματικά σχόλια στο τρέχων αρχείο καταγραφής και στο αρχείο της νέας φάσης. Συγκεκριμένα θα τυπώσει “Going to {new phase filename}” και “Coming from {current phase filename}” στο τρέχων και στο επόμενο αρχείο αντίστοιχα. Τέλος, θα αποθηκεύσει σε μία στοίβα το τρέχων αρχείο εξόδου και το όνομά του έτσι ώστε να μπορεί να επιστρέψει μόλις ολοκληρωθεί η φάση που πρόκειται να ξεκινήσει και θα ξεκινήσει τη νέα φάση με τον ίδιο ακριβώς τρόπο λειτουργίας του αναλυτή.
Από την άλλη όταν αναγνωρίζεται ένα μήνυμα το οποίο σηματοδοτεί τη λήξη της τρέχουσας φάσης τότε το αρχείο μεταφέρεται στο nextPhaseEndSet. Όπως είναι προφανές, κάποια στιγμή όλα τα αρχεία μεταφέρονται από το ReadSet στο nextPhaseEndSet. Τότε ο αναλυτής εκκινεί τη διαδικασία επιστροφής σε προηγούμενη φάση. Αρχικά βρίσκει μέσω των δομών που διατηρεί το όνομα και το αρχείο εξόδου της προηγούμενης φάσης. Στη συνέχεια καταγράφει ξανά επεξηγηματικά μηνύματα. Συγκεκριμένα στο αρχείο της τρέχουσας φάσης προσθέτει την εγγραφή “Returning to {previous phase filename}” και κλείνει το αρχείο, ενώ στο αρχείο της προηγούμενης φάσης τοποθετεί την εγγραφή “Returned from {current phase filename}”. Στη συνέχεια όπως και προηγουμένως περνάει τα υπόλοιπα αρχεία στο ReadSet

ξανά και συνεχίζει να εκτελεί τον αλγόριθμο ανάλυσης κανονικά.

4.3.3 Επεξεργασία εγγραφής

Κάθε γραμμή που εισάγετε στον αναλυτή από οποιοδήποτε αρχείο καταγραφής και δεν σηματοδοτεί αλλαγή της τρέχουσας φάσης, προωθείται στο στάδιο της επεξεργασίας. Το συγκεκριμένο στάδιο θα αναγνωρίσει το είδος της MPI λειτουργίας που εκτελείται, θα την κωδικοποιήσει κατάλληλα και είτε άμεσα είτε σε μεταγενέστερο στάδιο θα την εγγράψει στο κατάλληλο αρχείο εξόδου του αναλυτή. Όπως είναι λογικό το συγκεκριμένο στάδιο είναι πιο στενά συνδεδεμένο με το προηγούμενο εργαλείο δεδομένου ότι επεξεργάζεται τις εγγραφές που παράγει η βιβλιοθήκη. Συνεπώς, η ομαδοποίηση των εγγραφών ακολουθεί αυτήν της βιβλιοθήκης και κυρίως οποιαδήποτε αλλαγή στη μορφή των παραγόμενων εγγραφών μπορεί να επηρεάσει την επεξεργασία τους από τον αναλυτή. Ο αναλυτής διαχωρίζει τις εγγραφές σε εφτά (7) κατηγορίες για την ανάλυσή του, οι οποίες θα περιγραφούν ξεχωριστά. Η κατηγοριοποίηση γίνεται κυρίως με βάση τη μορφή της γραμμής προς επεξεργασίας και ακολουθεί την ίδια ομαδοποίηση που κάνει και η βιβλιοθήκη σε προηγούμενο στάδιο. Τα στοιχεία που κυρίως προσπαθεί να εξάγει ο αναλυτής είναι ο αποστολέας και ο παραλήπτης κάθε μηνύματος και το μέγεθος του φορτίου.

1. **Κατηγορία παράβλεψης (ignore)**: Κάποιες εγγραφές δεν έχουν χρήσιμη πληροφορία, σχετική δηλαδή με κίνηση στο δίκτυο και δεν χρειάζεται να παράγουν κάποιο αποτέλεσμα, συνεπώς ο αναλυτής θα τις αγνοήσει. Για παράδειγμα τέτοιες εγγραφές είναι αυτές που σηματοδοτούν την έναρξη της λειτουργίας του προτύπου (MPI_Init()) ή την αναμονή των διεργασιών για να συγχρονιστούν (MPI_Wait()).
   Επίσης παραβλέπονται εγγραφές που περιέχουν πληροφορία που εξάγεται και από άλλες εγγραφές. Συγκεκριμένα στα μηνύματα σημείο προς σημείο καταγράφεται από την βιβλιοθήκη τόσο η αποστολή όσο και η λήψη ενός μηνύματος. Αν ο αναλυτής επεξεργαζόταν και τις δύο εγγραφές, θα προέκυπτε ότι η κίνηση είναι διπλάσια. Για να αποφευχθεί συνεπώς αυτό το φαινόμενο, ο αναλυτής επεξεργάζεται μόνο τις εγγραφές της αποστολής και όχι αυτές της λήψης.

2. **Κατηγορία σημείου προς σημείο**: Περιλαμβάνονται οι απλές κλήσεις τύπου MPI_Send() του προτύπου. Κάθε εγγραφή που ανήκει σε αυτή τη κατηγορία αφού αποτελεί μια ολοκληρωμένη ανταλλαγή φορτίου επεξεργάζεται κατευθείαν. Συγκεκριμένα παράγεται μια νέα εγγραφή τύπου:
   ({αποστολέας} {παραλήπτης} {φορτίο} {είδος φορτίου} {communicator})
   όπου '*αποστολέας*' είναι η παγκόσμια ταυτότητα (global id) της διεργασίας που στέλνει δεδομένα, '*παραλήπτης*' είναι η ταυτότητα της διεργασίας που δέχεται τα δεδομένα στο περιβάλλον του συγκεκριμένου communicator, το '*φορτίο*' υπολογίζεται ως το γινόμενο του πλήθους των MPI_Datatype που ανταλλάσσονται επί το μέγεθος σε bytes τους. Το '*είδος φορτίου*' μπορεί να πάρει δύο διαφορετικές τιμές, "contig" ή "no_contig" και καθορίζεται από την παράμετρο που δείχνει την κωδικοποίηση του τύπου και δηλώνει

αν ο τύπος που χρησιμοποιείται κατά την ανταλλαγή του μηνύματος είναι συνεχής ή όχι, με την επιπρόσθετη πολυπλοκότητα που συνεπάγεται. Τέλος, '*ενσομμυνισατορ*' είναι προφανώς το όνομα του communicator στο περιβάλλον του οποίου εκτελείται η συγκεκριμένη μεταφορά δεδομένων.

3. **Κατηγορία πολλαπλής αποστολής με διεργασία ρίζα**: Περιλαμβάνονται οι συλλογικές κλήσεις του προτύπου, τύπου MPI_Bcast(), κλήσεις δηλαδή οι οποίες εκκινούν από ή καταλήγουν σε έναν κόμβο ρίζα αλλά πολλές διεργασίες λαμβάνουν μέρος στην ανταλλαγή μηνυμάτων. Δεδομένου ότι όλες οι διεργασίες που συμμετέχουν στην κλήση θα έχουν την αντίστοιχη εγγραφή στο αρχείο καταγραφής, ο αναλυτής τις λαμβάνει όλες, αλλά θα επεξεργαστεί και θα καταγράψει μόνο αυτή η οποία θα προέρχεται από τον κόμβο ρίζα έτσι ώστε να αποφύγει τα διπλότυπα. Ο αναλυτής για να αναγνωρίσει τη διεργασία ρίζα αρχικά ελέγχει σε ποιον communicator εκτελείται η κλήση. Αν αυτός είναι ο παγκόσμιος (MPI_COMM_WORLD) τότε αρκεί να ελέγξει τα πεδία του αποστολέα και της ρίζας της εγγραφής, αν είναι κάποιος άλλος καταφεύγει στη σύγκριση της ρίζας της εγγραφής με την επιπλέον τιμή που έχει προσθέσει η MinI. Η νέα εγγραφή που παράγεται έχει την εξής μορφή:
({κλήση} {ρίζα} {φορτίο} {είδος φορτίου} {communicator})
όπου η '*κλήση*' δείχνει το είδος της διεργασίας που εκτελείται όπως καταγράφεται από τη MinI. Το '*φορτίο*' υπολογίζεται ως το γινόμενο του πλήθους των MPI_Datatype που ανταλλάσσονται επί το μέγεθος σε bytes τους. Σημειώνεται ότι σε αυτό το είδος κλήσεων ανταλλάσσονται τουλάχιστον τόσα μηνύματα όσα και ο αριθμός των διεργασιών που ανήκουν στο περιβάλλον του communicator συνεπώς το φορτίο απεικονίζει το μέγεθος κάθε μηνύματος και για να υπολογιστεί το συνολικό φορτίο στο δίκτυο πρέπει να πολλαπλασιαστεί αυτός ο αριθμός επί το πλήθος των διεργασιών που ανήκουν στον communicator. Το '*είδος φορτίου*' μπορεί να πάρει δύο διαφορετικές τιμές, "contig" ή "no_contig" και καθορίζεται από την παράμετρο που δείχνει την κωδικοποίηση του τύπου και δηλώνει αν ο τύπος που χρησιμοποιείται κατά την ανταλλαγή του μηνύματος είναι συνεχής ή όχι. Τέλος, '*communicator*' είναι προφανώς το όνομα του communicator στο περιβάλλον του οποίου εκτελείται η συγκεκριμένη μεταφορά δεδομένων.

4. **Κατηγορία διανυσματικής πολλαπλής αποστολής με διεργασία ρίζα**: Περιλαμβάνονται οι διανυσματικές συλλογικές κλήσεις του προτύπου τύπου MPI_Gatherv(), οι οποίες εκκινούν από ή καταλήγουν σε ένα κόμβο ρίζα και τα δεδομένα που μεταφέρονται είναι σε διανυσματική μορφή. Η συγκεκριμένη κατηγορία ακολουθεί το πρότυπο της απλής κατηγορίας πολλαπλής αποστολής, δηλαδή όλες οι εγγραφές από τα αρχεία καταγραφής που αναφέρονται σε μία κλήση θα ληφθούν από τον αναλυτή αλλά μόνο αυτή που προέρχεται από τη διεργασία ρίζα θα επεξεργαστεί έτσι ώστε να αποφευχθούν τα διπλότυπα. Επιπλέον η αναγνώριση της σωστής εγγραφής εκτελείται με τον αλγόριθμο που περιγράφηκε και στην προηγούμενη κατηγορία, ελέγχοντας τον communicator της κλήσης. Η νέα εγγραφή που παράγεται έχει την εξής μορφή:
({κλήση} {ρίζα} {min}/{median}/{max} {είδος φορτίου} {communicator})

όπου η *΄κλήση΄* δείχνει το είδος της διεργασίας όπως καταγράφεται από τη MinI. Οι τρεις μετρικές απεικονίζουν στοιχεία σχετικά με το ελάχιστο, μέσο και μέγιστο φορτίο που μεταφέρεται σε κάθε ανταλλαγή μεταξύ μιας διεργασίας και της ρίζας. Για το συνολικό φορτίο στο δίκτυο πρέπει να πολλαπλασιαστεί αυτός ο αριθμός με τον αριθμό των διεργασιών στο συγκεκριμένο communicator. Το '*είδος φορτίου*' μπορεί να πάρει δύο διαφορετικές τιμές, "contig" ή "no_contig" και καθορίζεται από την παράμετρο που δείχνει την κωδικοποίηση του τύπου και δηλώνει αν ο τύπος που χρησιμοποιείται κατά την ανταλλαγή του μηνύματος είναι συνεχής ή όχι. Τέλος, '*communicator*' είναι προφανώς το όνομα του communicator στο περιβάλλον του οποίου εκτελείται η συγκεκριμένη μεταφορά δεδομένων.

5. **Κατηγορία πολλαπλής αποστολής χωρίς διεργασία ρίζα**: Περιλαμβάνονται οι συλλογικές κλήσεις του προτύπου της μορφής MPI_Allreduce(), οι οποίες δεν χρησιμοποιούν κάποιο κόμβο ως ρίζα, αλλά τα αποτελέσματα της κλήσης είναι γνωστά σε όλους τους κόμβους. Όπως και στις προηγούμενες δύο κατηγορίες, για κάθε κλήση αυτής της μορφής υπάρχει εγγραφή σε κάθε διεργασία που θα λάβει μέρος. Συνεπώς, για να αποφευχθούν τα διπλότυπα ο αναλυτής φιλτράρει και ομαδοποιεί τα δεδομένα αυτά. Κάθε εγγραφή ομαδοποιείται με βάση το μοναδικό αναγνωριστικό (id) που έχει προσθέσει η MinI. Ο αναλυτής λοιπόν αποθηκεύει εσωτερικά σε μια δομή αυτές τις εγγραφής χρησιμοποιώντας ως κλειδί το μοναδικό αναγνωριστικό και ως τιμή των αριθμό των επαναλήψεων που έχει εμφανιστεί το συγκεκριμένο κλειδί. Μόλις η τιμή γίνει ίση με τον αριθμό των διεργασιών που ανήκουν στον communicator της κλήσης, τότε η εγγραφή καταγράφεται στο αρχείο εξόδου και το κλειδί αφαιρείται από τη δομή. Οι εγγραφές έχουν την μορφή:
({κλήση} {φορτίο} {είδος φορτίου} {communicator})
όπου η '*κλήση*' δείχνει το είδος της διεργασίας που εκτελείται όπως καταγράφεται από τη MinI. Το '*φορτίο*' υπολογίζεται ως το γινόμενο του πλήθους των MPI_Datatype που ανταλλάσσονται επί το μέγεθος σε bytes τους. Σημειώνεται ότι σε αυτό το είδος κλήσεων ανταλλάσσονται τουλάχιστον τόσα μηνύματα όσα και ο αριθμός των διεργασιών που ανήκουν στο περιβάλλον του communicator συνεπώς το φορτίο απεικονίζει το μέγεθος κάθε μηνύματος και για να υπολογιστεί το συνολικό φορτίο στο δίκτυο πρέπει να πολλαπλασιαστεί αυτός ο αριθμός επί το πλήθος των διεργασιών που ανήκουν στον communicator. Το '*είδος φορτίου*' μπορεί να πάρει δύο διαφορετικές τιμές, "contig" ή "no_contig" και καθορίζεται από την παράμετρο που δείχνει την κωδικοποίηση του τύπου και δηλώνει αν ο τύπος που χρησιμοποιείται κατά την ανταλλαγή του μηνύματος είναι συνεχής ή όχι. Τέλος, '*communicator*' είναι προφανώς το όνομα του communicator στο περιβάλλον του οποίου εκτελείται η συγκεκριμένη μεταφορά δεδομένων.

6. **Κατηγορία διανυσματικής πολλαπλής αποστολής χωρίς διεργασία ρίζα**: Περιλαμβάνει τις συλλογικές κλήσεις του προτύπου οι οποίες δεν χρησιμοποιούν κάποιον κόμβο ως ρίζα και τα δεδομένα που μεταφέρονται είναι διανυσματικά. Χαρακτηριστικό παράδειγμα της συγκεκριμένης κατηγορίας είναι η κλήση MPI_Alltoallv(). Η συγκεκρι-

μένη κατηγορία ακολουθεί τα πρότυπα της προηγούμενης κατηγορίας καθώς το μόνο που αλλάζει είναι το είδος των δεδομένων που μεταφέρονται. Συνεπώς, οι εγγραφές αποθηκεύονται με βάση το μοναδικό αναγνωριστίκό που προσθέτει η βιβλιοθήκη και όταν αναγνωριστούν όλες, τότε γίνεται η εγγραφή στο αρχείο εξόδου. Τα αποτελέσματα που προκύπτουν είναι της μορφής:
({κλήση} {min}/{median}/{max} {είδος φορτίου} {communicator})
όπου η *΄κλήση΄* δείχνει το είδος της διεργασίας όπως καταγράφεται από τη MinI. Οι τρεις μετρικές απεικονίζουν στοιχεία σχετικά με το ελάχιστο, μέσο και μέγιστο φορτίο που μεταφέρεται σε κάθε ανταλλαγή. Για το συνολικό φορτίο στο δίκτυο πρέπει να πολλαπλασιαστεί αυτός ο αριθμός με τον αριθμό των διεργασιών στο συγκεκριμένο communicator. Το '*είδος φορτίου*' μπορεί να πάρει δύο διαφορετικές τιμές, "contig" ή "no_contig" και καθορίζεται από την παράμετρο που δείχνει την κωδικοποίηση του τύπου και δηλώνει αν ο τύπος που χρησιμοποιείται κατά την ανταλλαγή του μηνύματος είναι συνεχής ή όχι. Τέλος, '*communicator*' είναι προφανώς το όνομα του communicator στο περιβάλλον του οποίου εκτελείται η συγκεκριμένη μεταφορά δεδομένων.

7. **Κατηγορία communicator**: Η τελευταία κατηγορία δεδομένων που καταγράφονται, δεν αναφέρεται σε πραγματική μεταφορά δεδομένων στο δίκτυο αλλά στην δημιουργία νέων communicator οι οποίοι είναι απαραίτητοι για την επικοινωνία των διεργασιών. Ο αναλυτής αναγνωρίζει και επεξεργάζεται, τις εγγραφές της κλήσης MPI_Comm_split(). Ο αναλυτής διατηρεί μια εσωτερική δομή στην οποία αποθηκεύει όλους τους γνωστούς communicators. Κατά την επεξεργασία μιας εγγραφής αυτού του είδους, ο αναλυτής θα ελέγξει αρχικά αν υπάρχει ήδη καταχώρηση του συγκεκριμένου communicator. Αν υπάρχει, θα προσθέσει την παγκόσμια ταυτότητα (global id) της διεργασίας στη λίστα με τα μέλη του. Αν δεν υπάρχει, θα δημιουργήσει μια νέα εγγραφή, θα καταγράψει το όνομα του communicator και το μέγεθός του και θα προσθέσει στη λίστα με τα μέλη του την παγκόσμια ταυτότητα της διεργασίας. Κατά το τέλος της ανάλυσης όλων των εγγραφών, ο αναλυτής όπως έχει προαναφερθεί καταγράφει όλους τους communicators σε ξεχωριστό αρχείο σε μορφή Json ως εξής:
όνομα communicator {'members': λίστα από ταυτότητες, 'size': μέγεθος}

4.4 Παραδείγματα χρήσης

Πλέον αφού έχουν παρουσιαστεί τα δύο πρώτα εργαλεία, στη συγκεκριμένη ενότητα θα παρουσιάσουμε παραδείγματα χρήσης τους. Έχουν επιλεχθεί δύο απλά παραδείγματα τα οποία στόχο έχουν να αναδείξουν τη λειτουργικότητα των εργαλείων και τον τρόπο αναπαράστασης των αποτελεσμάτων σε κλήσεις MPI διαφορετικού τύπου. Το πρώτο παράδειγμα, θα δείξει τον τρόπο απεικόνισης των κλήσεων σημείου προς σημείο σε κάθε εργαλείο, καθώς και τη χρήση διαφορετικών φάσεων, ενώ το δεύτερο θα χρησιμοποιήσει συλλογικές κλήσεις και θα κάνει χρήση δύο communicator.

4.4.1 Παράδειγμα δακτυλίου

Ως πρώτο παράδειγμα υλοποιήθηκε μια εφαρμογή στην οποία οι MPI διεργασίες στέλνουν μία αριθμητική τιμή στη διεργασία με το αμέσως μεγαλύτερο αναγνωριστικό και λαμβάνουν μία τιμή από τη διεργασία με το αμέσως μικρότερο αναγνωριστικό. Η εφαρμογή ακόμα κλείνει τα άκρα αυτής της αλυσίδας μεταφοράς κυκλικά, δημιουργώντας έτσι ένα δακτύλιο. Η διαδικασία αποστολής και λήψης δεδομένων εκτελείται για ένα συγκεκριμένο αριθμό βημάτων, ο οποίος διατηρήθηκε επαρκώς μικρός έτσι ώστε να μην μεγαλώσουν ασκόπως τα αρχεία εξόδου. Για καλύτερη κατανόηση των μηχανισμών που έχουν περιγραφεί στα προηγούμενα κεφάλαια, στο παράδειγμα χρησιμοποιούνται ασύγχρονες λήψεις και σύγχρονες αποστολές ενώ ακόμα υπάρχει συγχρονισμός ανάμεσα σε κάθε βήμα.

Algorithm 1 Circular data exchange example

```
1: ...
2: for i in 1..4 do
3:     begin("roundi")
4:     MPI_Irecv(buf[1], 1, MPI_INT, previous,..., request)
5:     MPI_Send(buf[0], 1, MPI_INT, next,...)
6:     MPI_Wait(request)
7:     end("roundi)
8: end for
9: ...
```

Η παραπάνω εφαρμογή αφού μεταγλωττιστεί και συνδεθεί με τη MinI μπορεί να εκτελεστεί όπως φαίνεται στην παρακάτω εικόνα. Καταρχάς, δημιουργείται ο φάκελος ti_traces στον οποίο θα αποθηκευθούν τα αρχεία εξόδου της εφαρμογής και στη συνέχεια εκτελείται όπως οποιαδήποτε MPI εφαρμογή. Παρατηρούμε ακόμα ότι μετά την εκτέλεσή της στον φάκελο εξόδου έχουν δημιουργηθεί τα αρχεία εξόδου της βιβλιοθήκης. Δεδομένου ότι επιλέξαμε εκτέλεση με τέσσερις(4) MPI διεργασίες, έχουν δημιουργηθεί και τέσσερα(4) αρχεία εξόδου.

```
fkotomat@clone6:~/diplTest/p2p$ ls
circular  circular.c
fkotomat@clone6:~/diplTest/p2p$ mkdir ti_traces
fkotomat@clone6:~/diplTest/p2p$ ls
circular  circular.c  ti_traces
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$ mpirun -np 4 ./circular 1>out.txt 2> err.txt
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$ ls
circular  circular.c  err.txt  out.txt  ti_traces
fkotomat@clone6:~/diplTest/p2p$ ls ti_traces/
ti_trace0.txt  ti_trace1.txt  ti_trace2.txt  ti_trace3.txt
```

Σχήμα 4.1: Εκτέλεση εφαρμογής-1 συνδεδεμένη με τη βιβλιοθήκη

Για λόγους πληρότητας πριν προχωρήσουμε στην ανάλυση των αποτελεσμάτων της βιβλιοθήκης, παρατίθεται το αποτέλεσμα της εκτέλεσης στην εικόνα 4.2 όπου φαίνεται ότι η βιβλιοθήκη έχει προσθέσει στην αρχή της εκτέλεσης και στο τέλος δύο συγκεκριμένες γραμ-

μές που υποδυκνείουν κιόλας την έναρξη και τη λήξη της λειτουργίας του προτύπου MPI.

```
fkotomat@clone6:~/diplTest/p2p$ cat out.txt
MINI STARTING!(buffersize=420000, buffercount=200, tempsize=100000)
3 sent 3, received 2 on round 0
0 sent 0, received 3 on round 0
1 sent 1, received 0 on round 0
2 sent 2, received 1 on round 0
0 sent 1, received 4 on round 1
2 sent 3, received 2 on round 1
1 sent 2, received 1 on round 1
3 sent 4, received 3 on round 1
0 sent 2, received 5 on round 2
1 sent 3, received 2 on round 2
3 sent 5, received 4 on round 2
2 sent 4, received 3 on round 2
0 sent 3, received 6 on round 3
2 sent 5, received 4 on round 3
3 sent 6, received 5 on round 3
1 sent 4, received 3 on round 3
MINI ENDING!
```

Σχήμα 4.2: Αποτελέσματα εκτέλεσης εφαρμογής-1

Στη συνέχεια, στην εικόνα 4.3 βλέπουμε τα ίχνη που κατέγραψε η βιβλιοθήκη MinI από την εκτέλεση του παραδείγματος για την MPI διεργασία με αναγνωριστικό 0. Όπως φαίνεται, κάθε μία κλήση του προτύπου έχει καταγραφεί και σε κάθε γραμμή του αρχείου εμφανίζεται και μία συγκεκριμένη κλήση. Παρουσιάζεται η εικόνα μόνο από τη διεργασία 0 αλλά και στις υπόλοιπες διεργασίες, παρατηρείται μια αντίστοιχη καταγραφή.

```
fkotomat@clone6:~/diplTest/p2p$ cat ti_traces/ti_trace0.txt
0 init 1
Phase round0 start.
0 Irecv 3 1 (of 4 bytes) 1 on comm MPI_COMM_WORLD
0 send 1 1 (of 4 bytes) 1 on comm MPI_COMM_WORLD
0 wait
Phase round0 end.
Phase round1 start.
0 Irecv 3 1 (of 4 bytes) 1 on comm MPI_COMM_WORLD
0 send 1 1 (of 4 bytes) 1 on comm MPI_COMM_WORLD
0 wait
Phase round1 end.
Phase round2 start.
0 Irecv 3 1 (of 4 bytes) 1 on comm MPI_COMM_WORLD
0 send 1 1 (of 4 bytes) 1 on comm MPI_COMM_WORLD
0 wait
Phase round2 end.
Phase round3 start.
0 Irecv 3 1 (of 4 bytes) 1 on comm MPI_COMM_WORLD
0 send 1 1 (of 4 bytes) 1 on comm MPI_COMM_WORLD
0 wait
Phase round3 end.
0 finalize
```

Σχήμα 4.3: Αποτέλεσμα καταγραφής ιχνών εφαρμογής-1

Όπως έχει τονιστεί και στα προηγούμενα κεφάλαια η εικόνα που έχουμε μέχρι στιγμής για την εφαρμογή που εκτελέστηκε είναι αρκετά χρήσιμη και αναλυτική όμως είναι αρκετά δύσκολο να εξαχθεί άμεσα η γενική εικόνα του σχήματος επικοινωνίας. Σε αυτό το σημείο

με τη χρήση του δεύτερου εργαλείου που έχει αναπτυχθεί, ο χρήστης θα μπορέσει να μαζέψει τα επιμέρους ίχνη και να δημιουργήσει μια συνολική εικόνα καταγραφής η οποία θα χωρίζεται ανά φάση δεδομένου ότι ο χρήστης έχει ήδη ορίσει διακριτές φάσεις λειτουργίας του προγράμματος. Για τη χρήση του δεύτερου εργαλείου αρκεί απλά η κλήση του δίνοντας ως όρισμα το μονοπάτι προς τα αρχεία καταγραφής της βιβλιοθήκης. Όπως φαίνεται στην παρακάτω εικόνα, λοιπόν, με μια απλή κλήση δημουργείται ο φάκελος parserOutput ο οποίος στη συγκεκριμένη περίπτωση περιέχει πληροφορίες για τους communicators που χρησιμοποιήθηκαν, τη γενική φάση εκτέλεσης global και τις τέσσερις επιμέρους όπως εμφανίζονται και παραπάνω.

```
fkotomat@clone6:~/diplTest/p2p$ ls ti_traces/
ti_trace0.txt  ti_trace1.txt  ti_trace2.txt  ti_trace3.txt
fkotomat@clone6:~/diplTest/p2p$ ../../parser/parser.py ./ti_traces/
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$ ls ti_traces/
parserOutput  ti_trace0.txt  ti_trace1.txt  ti_trace2.txt  ti_trace3.txt
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$
fkotomat@clone6:~/diplTest/p2p$ ls ti_traces/parserOutput/
communicators.txt  global.txt  round0.txt  round1.txt  round2.txt  round3.txt
```

Σχήμα 4.4: Παράδειγμα εκτέλεσης αναλυτή στην εφαρμογή-1

Στην τελευταία εικόνα, 4.5, ο αναγνώστης θα βρει συγκεντρωτικά τις πληροφορίες που αφορούν τη συγκεκριμένη εφαρμογή. Συγκεκριμένα, φαίνεται ότι χρησιμοποιείται μόνο ένας MPI_COMM_WORLD στον οποίο ανήκουν οι τέσσερις διεργασίες. Επίσης, βλέποντας τα συγκεντρωτικά αποτελέσματα τις πρώτης φάσης εκτέλεσης γίνεται φανερό ότι σε κάθε μία υπάρχουν τέσσερις μεταφορές δεδομένων από σημείο προς σημείο όπου σε κάθε μία μεταφέρονται τέσσερα bytes δεδομένων. Τέλος, παρατηρώντας τα περιεχόμενα της κεντρικής φάσης όσο και επεξηγηματικά μηνύματα στο αρχείο της πρώτης φάσης, ο αναγνώστης μπορεί εύκολα να συμπεράνει την χρονική ακολουθία των φάσεων εκτέλεσης.

4.4.2 Παράδειγμα συλλογικών κλήσεων

Στο δεύτερο παράδειγμα χρήσης, ένας ζυγός αριθμός από διεργασίες MPI χωρίζεται σε δύο σύνολα και σε κάθε σύνολο εκτελείται μία συλλογική κλήση έτσι ώστε τελικά όλες οι διεργασίες ενός συνόλου να έχουν πρόσβαση στο τελικό αποτέλεσμα. Στόχος είναι ο αναγνώστης να κατανοήσει πρακτικά πώς η βιβλιοθήκη και στη συνέχεια ο αναλυτής διαχειρίζονται την κλήση MPI_Comm_split() και τις συλλογικές κλήσεις.

Όπως και προηγουμένως θα συνδέσουμε την εφαρμογή με τη βιβλιοθήκη κατά το στάδιο της μεταγλώττισης και στη συνέχεια αφού δημιουργήσουμε το φάκελο εξόδου ti_traces θα την εκτελέσουμε. Για το συγκεκριμένο παράδειγμα θα εκτελέσουμε την εφαρμογή με 8 διεργασίες και αναμένουμε αντίστοιχο αριθμό από αρχεία εξόδου. Αυτό φαίνεται και στην εικόνα 4.6

Αντίστοιχα τα αποτελέσματα της εκτέλεσης εμφανίζονται στην εικόνα, 4.7

Παρατηρώντας τα αρχεία καταγραφής για δύο διεργασίες, εικόνα 4.8, που ανήκουν σε

```
fkotomat@clone6:~/diplTest/p2p$ cd ti_traces/parserOutput/
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$ ls
communicators.txt  global.txt  round0.txt  round1.txt  round2.txt  round3.txt
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$ cat communicators.txt
MPI_COMM_WORLD {'members': [0, 1, 2, 3], 'size': 4}
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$ cat global.txt
Going to round0.txt
Returned from round0.txt
Going to round1.txt
Returned from round1.txt
Going to round2.txt
Returned from round2.txt
Going to round3.txt
Returned from round3.txt
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/p2p/ti_traces/parserOutput$ cat round2.txt
Coming from global.txt
(0 1 4 contig MPI_COMM_WORLD)
(2 3 4 contig MPI_COMM_WORLD)
(3 0 4 contig MPI_COMM_WORLD)
(1 2 4 contig MPI_COMM_WORLD)
Returning to global.txt
```

Σχήμα 4.5: Δείγμα εξόδου αναλυτή για την εφαρμογή-1

```
fkotomat@clone6:~/diplTest/collective$ ls
collective.c
fkotomat@clone6:~/diplTest/collective$ mpicc collective.c -o collective -L/home/users/fkotomat/mini -lmini
fkotomat@clone6:~/diplTest/collective$ ls
collective  collective.c
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$ mkdir ti_traces
fkotomat@clone6:~/diplTest/collective$ ls
collective  collective.c  ti_traces
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$ mpirun -np 8 ./collective 1>out.txt 2>err.txt
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$ ls ti_traces/
ti_trace0.txt  ti_trace1.txt  ti_trace2.txt  ti_trace3.txt  ti_trace4.txt  ti_trace5.txt  ti_trace6.txt  ti_trace7.txt
```

Σχήμα 4.6: Εκτέλεση εφαρμογής-2 συνδεδεμένη με τη βιβλιοθήκη

Algorithm 2 Collective call example

```
1: ...
2: MPI_Comm_rank(MPI_COMM_WORLD, rank)
3: color = rank % 2
4: MPI_Comm_split(MPI_COMM_WORLD, color, ..., newcomm)
5: MPI_Comm_rank(newcomm, localRank)
6: MPI_Allreduce(rank, result, MPI_INT, MPI_MAX, ..., newcomm)
7: if localRank == 0 then
8:     PRINT("My global rank is rank and max rank in our set is result")
9: end if
10: ...
```

```
fkotomat@clone6:~/diplTest/collective$ cat out.txt
MINI STARTING!(buffersize=420000, buffercount=200, tempsize=100000)
Local rank 0, global rank 1. Max id in set is 7.
Local rank 0, global rank 0. Max id in set is 6.
MINI ENDING!
```

Σχήμα 4.7: Αποτελέσματα εκτέλεσης εφαρμογής-2

διαφορετικούς communicators, γίνεται άμεσα ξεκάθαρη η διαφοροποίηση που έχει κάνει η βιλιοθήκη ανάμεσα στους δύο. Όσον αφορά την καταγραφή της συλλογικής κλήσης, ακολουθεί τα πρότυπα που έχουν περιγραφεί στο κεφάλαιο 3. Αξίζει να παρατηρηθεί η χρήση του αναγνωριστικού στο τέλος της εγγραφής της συλλογικής κλήσης. Αυτή η τιμή θα συσχετιστεί στο επόμενο στάδιο για να συνενώσει τις διαφορετικές εγγραφές σε μία. Είναι μοναδική ανά διαφορετικό communicator και αυξάνεται γραμμικά. Συνεπώς προκύπτει ότι είναι και στα δύο αρχεία 1 διότι στο περιβάλλον κάθε ενός από τους δύο communicators είναι η πρώτη συλλογική κλήση που λαμβάνει χώρα.

```
fkotomat@clone6:~/diplTest/collective$ cat ti_traces/ti_trace0.txt
0 init 1
0 Comm split. color=0, key=0.New comm mini_comm_0_0 of size 4
0 allReduce 1 (of 4 bytes) of type 1 on comm mini_comm_0_0 1
Comm mini_comm_0_0 free.
0 finalize
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$ cat ti_traces/ti_trace1.txt
1 init 1
1 Comm split. color=1, key=1.New comm mini_comm_0_1 of size 4
1 allReduce 1 (of 4 bytes) of type 1 on comm mini_comm_0_1 1
Comm mini_comm_0_1 free.
1 finalize
```

Σχήμα 4.8: Αποτέλεσμα καταγραφής ιχνών εφαρμογής-2

Επόμενο βήμα στη ροή εκτέλεσης είναι να χρησιμοποιήσουμε το εργαλείο MinI Parser έτσι ώστε να παράξουμε το συνολικό σχήμα της επικοινωνίας. Η εφαρμογή του δεύτερου εργαλείου φαίνεται στην εικόνα 4.9, ενώ τα αποτελέσματα φαίνονται στην εικόνα 4.10.

```
fkotomat@clone6:~/diplTest/collective$ ls ti_traces/
ti_trace0.txt  ti_trace1.txt  ti_trace2.txt  ti_trace3.txt  ti_trace4.txt  ti_trace5.txt  ti_trace6.txt  ti_trace7.txt
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$ ../../parser/parser.py ./ti_traces/
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$ ls ti_traces/
parserOutput  ti_trace0.txt  ti_trace1.txt  ti_trace2.txt  ti_trace3.txt  ti_trace4.txt  ti_trace5.txt  ti_trace6.txt  ti_trace7.txt
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$
fkotomat@clone6:~/diplTest/collective$ ls ti_traces/parserOutput/
communicators.txt  global.txt
```

Σχήμα 4.9: Παράδειγμα εκτέλεσης αναλυτή στην εφαρμογή-2

Βλέπουμε ότι σε αυτή τη δεύτερη εφαρμογή το αρχείο με τους communicators περιέχει περισσότερες πληροφορίες. Συγκεκριμένα, βλέπουμε μία εγγραφή για κάθε έναν που έχει χρησιμοποιηθεί. Στις λίστες απεικονίζονται οι παγκόσμιες ταυτότητες των διεργασιών και έτσι είναι εύκολο να δούμε ότι οι διεργασίες έχουν χωριστεί σε ένα περιττό και ένα άρτιο σύνολο

```
fkotomat@clone6:~/diplTest/collective$ cd ti_traces/parserOutput/
fkotomat@clone6:~/diplTest/collective/ti_traces/parserOutput$ ls
communicators.txt  global.txt
fkotomat@clone6:~/diplTest/collective/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/collective/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/collective/ti_traces/parserOutput$ cat communicators.txt
MPI_COMM_WORLD {'members': [0, 1, 2, 3, 4, 5, 6, 7], 'size': 8}
mini_comm_0_1 {'members': ['7', '5', '3', '1'], 'size': 4}
mini_comm_0_0 {'members': ['0', '4', '6', '2'], 'size': 4}
fkotomat@clone6:~/diplTest/collective/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/collective/ti_traces/parserOutput$
fkotomat@clone6:~/diplTest/collective/ti_traces/parserOutput$ cat global.txt
(allReduce 4 contig mini_comm_0_0)
(allReduce 4 contig mini_comm_0_1)
```

Σχήμα 4.10: Δείγμα εξόδου αναλυτή για την εφαρμογή-2

όπως ήταν και το αναμενόμενο. Επιπλέον, οι συλλογικές κλήσεις αντί να εμφανίζονται ανά διεργασία όπως στο προηγούμενο στάδιο, έχουν ομαδοποιηθεί και εμφανίζονται συνολικά. Υπενθυμίζουμε ότι το φορτίο της κλήσης είναι ανά μήνυμα, άρα αν θέλουμε να υπολογίσουμε το συνολικό φορτίο που μεταφέρθηκε στο δίκτυο λόγω της allReduce, πρέπει να πολλαπλασιάσουμε με τον αριθμό των διεργασιών του communicator, θεωρώντας ότι οι εσωτερικοί αλγόριθμοι της υλοποίησης του MPI που χρησιμοποιούμε δημιουργούν ένα μήνυμα για κάθε διεργασία. Με βάση αυτές τις υποθέσεις λοιπόν το συνολικό φορτίο θα ήταν 16 bytes.

Κεφάλαιο 5

GetInfo

5.1 Εισαγωγή

Το τρίτο και τελευταίο εργαλείο που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής είναι ο προσομοιωτής GetInfo. Πρόκειται για μία εφαρμογή που υλοποιήθηκε εξ ολοκλήρου στα πλαίσια της διπλωματικής στη γλώσσα Python. Βασική της λειτουργία είναι να παράξει μια αντιστοίχιση των διεργασιών που θα χρησιμοποιηθούν κατά την εκτέλεση της MPI εφαρμογής σε επεξεργαστικούς κόμβους του συστήματος.

5.2 Mpirun

Για να εκτελεστούν διεργασίες οι οποίες κάνουν χρήση του προτύπου MPI και εκτελούνται σε μία ή περισσότερες επεξεργαστικές μονάδες, είναι απαραίτητη η χρήση κάποιου διαχειριστή ο οποίος θα είναι υπεύθυνος για αρχικοποίηση των δομών του συστήματος που απαιτούνται για τη σωστή λειτουργία του προτύπου, τη δέσμευση των αντίστοιχων πόρων του συστήματος, την αρχικοποίηση των συνδέσεων δικτύου που είναι απαραίτητες για την επικοινωνία και τέλος για την αρχικοποίηση της διεργασίας και σε όσους επεξεργαστικούς πυρήνες έχουν ζητηθεί. Ένας από τους ευρέως διαδεδομένους διαχειριστές είναι ο Mpirun. Είναι υπεύθυνος για την αρχικοποίηση, εκτέλεση και παρακολούθηση εφαρμογών που ακολουθούν το μοντέλο SPMD ή MIMD. Η λειτουργία του μπορεί να χωριστεί σε τρία βασικά στάδια. Στο πρώτο στάδιο με βάση τις πληροφορίες που παρέχει το περιβάλλον εργασίας ή τις πληροφορίες που δίνονται ως ορίσματα κατά την κλήση του, καθορίζεται ο αριθμός των διεργασιών που θα δημιουργηθούν και δεσμεύονται οι επεξαργαστικοί πόροι του συστήματος που θα χρησιμοποιηθούν. Αυτό είναι το στάδιο της απεικόνισης (mapping). Στο επόμενο στάδιο, πάλι με βάση εσωτερικούς κανόνες ή με ορίσματα που καθορίζονται κατά την κλήση του Mpirun, ο διαχειριστής θέτει το βαθμό (rank) που θα έχει η εκάστοτε διεργασία στο περιβάλλον του MPI_COMM_WORLD. Αυτό είναι το στάδιο της βαθμονόμησης (ranking). Οι αναθέσεις που εκτελούνται στα πρώτα δύο στάδια είναι λογικές, δηλαδή δεν υπάρχει πραγματική δέσμευση των επεξεργαστικών πυρήνων του συστήματος. Στο τρίτο στάδιο, της δέσμευσης (binding), θα καθοριστεί η τελική ανάθεση με βάση κανόνες που συνήθως προσπαθούν να βελτιστοποιήσουν την τοπικότητα των πυρήνων

σε σχέση με κάποιον πόρο του συστήματος, για παράδειγμα μια κρυφή μνήμη (cache). Τέλος, ο διαχειριστής Mpirun μπορεί να ρυθμίσει και εξωτερικές παραμέτρους ή συστήματα που θα χρησιμοποιηθούν κατά την εκτέλεση της εφαρμογής δίνοντας κατάλληλες οδηγίες σε αυτά. Χαρακτηριστικό παράδειγμα αυτών των οδηγιών είναι ο καθορισμός των πρωτοκόλλων της δικτυακής επικοινωνίας μεταξύ των διεργασιών (π.χ. TCP).

Στη συνέχεια του κεφαλαίου, αφού έγινε μια σύντομη περιγραφή της λειτουργίας του Mpirun, θα αναλυθούν εκτενέστερα τα στάδια που αναφέρθηκαν παραπάνω και θα γίνει σύγκριση με τον προσομοιωτή GetInfo. Σημειώνεται εκ των προτέρων ότι ο προσομοιωτής προσφέρει ένα υποσύνολο των δυνατοτήτων της πραγματικής εφαρμογής και ότι η χρήση του είναι καθαρά απεικονιστική και δεν αλληλεπιδρά με πραγματικούς πόρους του συστήματος.

5.3 Υλοποίηση

Η εφαρμογή που αναπτύχθηκε καλείται από γραμμή εντολών και δέχεται διάφορες παραμέτρους με βάση τις οποίες καθορίζεται το τελικό αποτέλεσμα, σε αντιστοιχία με το Mpirun. Χρησιμοποιήθηκε η βιβλιοθήκη argparse της πψτηον με την οποία διευκολύνεται η εισαγωγή των παραμέτρων, καθώς με σχετικά εύκολη παραμετροποίηση της κλάσης μπορεί να δημιουργηθεί μια αρκετά πολύπλοκη διαδικασία που διαβάζει τις παραμέτρους από τη γραμμή εντολών, εκτελεί τις απαραίτητες μετατροπές, εισάγει προκαθορισμένες τιμές στις παραμέτρους που δεν έχουν οριστεί και δημιουργεί κατάλληλα βοηθητικά μηνύματα στην περίπτωση που ζητηθεί από το χρήστη ή που ο συνδυασμός παραμέτρων που δόθηκε κατά την κλήση δεν είναι σωστός. Στη συνέχεια, η εφαρμογή δημιουργεί μια λογική αναπαράσταση ενός πολυπύρηνου συστήματος με βάση τις προδιαγραφές που έχουν δοθεί και σε αυτό το σύστημα θα γίνει τελικά η απεικόνιση και η βαθμονόμηση των διεργασιών στα πρότυπα του Mpirun. Ο προσομοιωτής δεν υποστηρίζει το τρίτο και τελευταίο βήμα του Mpirun δηλαδή τη δέσμευση των επεξεργαστικών πυρήνων.

5.3.1 Ανάγνωση παραμέτρων

Τα ορίσματα που δέχεται ο προσομοιωτής, όπως φαίνεται και στην παρακάτω εικόνα, ακολουθούν σε ονοματοδοσία αυτά του πραγματικού διαχειριστή. Έχουν προστεθεί μόνο τα απολύτως απαραίτητα για την σωστή εσωτερική λειτουργία της εφαρμογής.

- –core: χρησιμοποιείται εσωτερικά από τον προσομοιωτή. Δέχεται έναν αριθμό που δηλώνει πόσους επεξεργαστικούς πυρήνες περιέχει κάθε επεξεργαστική μονάδα. Αν δεν δοθεί, θα θεωρηθεί ότι σε κάθε μονάδα περιέχεται ένας πυρήνας.

- -o: χρησιμοποιείται εσωτερικά από τον προσομοιωτή. Δηλώνει το αρχείο στο οποίο θα καταγραφεί η έξοδος της εφαρμογής. Σε αντίθετη περίπτωση τα αποτελέσματα τυπώνονται στην οθόνη.

- -v: χρησιμοποιείται εσωτερικά από τον προσομοιωτή. Δηλώνει αν η εφαρμογή κατά την εκτέλεση θα τυπώνει στην οθόνη επεξηγηματικά μηνύματα που θα βοηθούν το χρήστη

```
usage: getInfo.py [-h] [-core CORES] [-host [HOST [HOST ...]]]
                  [-hostfile HOSTFILE] [-c PROCS] [-map-by MAP_BY]
                  [-nooversubscribe] [-bynode] [-rank-by {slot,core,node}]
                  [-rf RANKFILE] [-o [OUTFILE]] [-v]
                  ...

Generating a mapping for a mpi proccess.

positional arguments:
  args

optional arguments:
  -h, --help            show this help message and exit
  -core CORES, --core CORES
                        Number of cores per slot
  -host [HOST [HOST ...]], -H [HOST [HOST ...]], --host [HOST [HOST ...]]
                        Hosts on which to run application
  -hostfile HOSTFILE, --hostfile HOSTFILE, -machinefile HOSTFILE, --machinefile HOSTFILE
                        File containing information on the nodes
  -c PROCS, -n PROCS, --n PROCS, -np PROCS
                        Number of processes to launch
  -map-by MAP_BY, --map-by MAP_BY
                        Mapping of processes.
  -nooversubscribe, --nooversubscribe
                        Do not oversubscribe any nodes
  -bynode, --bynode     assign processes one per node, cycling in a RR fashion
  -rank-by {slot,core,node}, --rank-by {slot,core,node}
                        Ranking of processes.
  -rf RANKFILE, --rankfile RANKFILE
                        Provide a ranking file
  -o [OUTFILE], --outfile [OUTFILE]
                        Where final result will be printed. (default: stdout)
  -v, --verbose         Verbose mode.
```

Σχήμα 5.1: Παράμετροι εισόδου GetInfo

να ακολουθήσει τη ροή εκτέλεσης. Σε περίπτωση που δεν οριστεί, η εφαρμογή θα τυπώσει μόνο τα απολύτως απαραίτητα. Συστήνεται η χρήση της παραμέτρου κυρίως για λόγους αποσφαλμάτωσης.

- –help: εκτυπώνει βοηθητικό μήνυμα σχετικά με τις παραμέτρους με τις οποίες μπορεί να κληθεί το εργαλείο και τερματίζει την εφαρμογή.
- –host: δέχεται μια λίστα από ονόματα κόμβων στους οποίους θα εκτελεστεί η εφαρμογή.
- –hostfile: δέχεται ως είσοδο ένα αρχείο το οποίο περιέχει μια περιγραφή των διαθέσιμων κόμβων του συστήματος από την οποία εξάγονται πληροφορίες σχετικά με την αρχιτεκτονική τους.
- -np: καθορίζει τον αριθμό των MPI διεργασιών που θα δημιουργηθούν κατά την εκτέλεση.
- –nooversubscribe: καθορίζει αν οι κόμβοι του συστήματος μπορούν να υπερφορτωθούν.
- –map-by: υποδεικνύει τον τρόπο με τον οποίο θα δεσμευτούν επεξεργαστικοί πυρήνες του συστήματος. Η μορφές που υποστηρίζονται είναι είτε –map-by {node, slot, core} είτε –map-by ppr:{n}:{node,slot,core} όπου ν ο αριθμός πόρων που θα δεσμευτούν.
- –rank-by: υποδεικνύει τον τρόπο με τον οποίο θα αποδοθούν τα MPI ρανκς ανάμεσα στους πόρους που έχουν επιλεχθεί στο στάδιο του μαππινγ.

- –bynode: συντόμευση η οποία αντιστοιχεί σε –map-by node –rank-by node.
- –rankfile: δέχεται ως είσοδο ένα αρχείο στο οποίο υπάρχει η αντιστοίχιση ανάμεσα σε αναγνωριστικά διεργασιών (MPI ranks) και επεξεργαστικούς πόρους του συστήματος.

5.3.2 Προσομοίωση πολυπύρηνου συστήματος

Μόλις ολοκληρωθεί η ανάγνωση των παραμέτρων εισόδου, ο προσομοιωτής καθορίζει το περιβάλλον εκτέλεσης. Καθώς δεν έχει πρόσβαση στους πόρους του συστήματος, όπως ο πραγματικός διαχειριστής, ο προσομοιωτής θα υλοποιήσει στη μνήμη του μια απεικόνιση ενός πιθανά πολυπύρηνου συστήματος. Η αρχιτεκτονική που έχει επιλεχθεί να υλοποιείται αποτελείται από τέσσερα διαφορετικά δομικά στοιχεία τα οποία συνδέονται μεταξύ τους ιεραρχικά. Συγκεκριμένα, πάντα δημιουργείται μία συστάδα (cluster) στους πόρους της οποία θα εκτελεστεί η εφαρμογή. Η συστάδα αυτή αποτελείται από κόμβους (nodes). Κάθε κόμβος αποτελείται από slots και κάθε slot αποτελείται από επεξεργαστικούς πυρήνες (cores). Στους επεξεργαστικούς πυρήνες θα διανεμηθούν τελικά οι MPI διεργασίες. Για να καθορίσει τον αριθμό και να δημιουργήσει το σύστημα, ο προσομοιωτής χρησιμοποιεί κατά προτεραιότητα τις πληροφορίες που παρέχονται στο hostfile, στην παράμετρο hosts και στην παράμετρο cores. Πιο αναλυτικά, σε ένα hostfile αναφέρεται κάθε κόμβος με το όνομά του και περιγράφεται πόσα slots διαθέτει και πιθανά ποιος είναι και ο μέγιστος αριθμός από slots (max-slots) που μπορεί να διαθέσει. Ο προσομοιωτής έχει σχεδιαστεί έτσι ώστε να δέχεται στην είσοδό του ένα hostfile το οποίο μπορεί να χρησιμοποιηθεί και από πραγματικούς διαχειριστές. Το αρχείο αυτό μπορεί να περιέχει και κενές γραμμές ή σχόλια. Όμως για μεγαλύτερη ευκολία χρήσης αναγνωρίζεται και ένα λεκτικό, το οποίο σε πραγματικούς διαχειριστές θα αναγνωριζόταν ως σχόλιο, με το οποίο υποδεικνύεται ανά κόμβο πόσους επεξεργαστικούς πυρήνες διαθέτει κάθε slot του. Το λεκτικό αυτό είναι το '#cores{N}' το οποίο προστίθεται στο τέλος μιας εγγραφής κόμβου. Αν δεν υπάρχει αυτό το λεκτικό, τότε σε κάθε slot δημιουργούνται τόσοι πυρήνες όσοι και η τιμή της παραμέτρου cores. Σε περίπτωση που δεν δοθεί ούτε αυτή η παράμετρος τότε τα δύο στοιχεία ταυτίζονται και θεωρείται ότι υπάρχει ένας επεξεργαστικός πυρήνας ανά slot. Στο δείγμα από το αρχειό hostfile που ακολουθεί φαίνονται ξεκάθαρα οι τρόποι με τους οποίους μπορεί να συνταχθεί ένα ορθό hostfile.

Η παράμετρος host η οποία δέχεται μια λίστα από ονόματα κόμβων στους οποίους θα διανεμηθούν οι MPI διεργασίες, επίσης μπορεί να καθορίσει τη προσομοίωση του συστήματος. Αν χρησιμοποιηθεί μόνη της τότε θα δημιουργηθούν στη συστάδα τόσοι κόμβοι όσοι και οι διακριτές τιμές της λίστας της παραμέτρου. Για τον καθορισμό των slots ανα κόμβο, η εφαρμογή θα ρωτήσει το χρήστη κατά το χρόνο εκτέλεσης και θα αναθέσει τον ίδιο αριθμό από slots σε κάθε κόμβο. Για τον καθορισμό του αριθμού των επεξεργαστικών πυρήνων ακολουθείται η ίδια διαδικασία με προηγουμένως, δηλαδή όσο η τιμή της παραμέτρου cores ή ένα (1) αν αυτή δεν είναι διαθέσιμη. Σε περίπτωση που η παράμετρος host χρησιμοποιηθεί σε συνδυασμό με την παράμετρο hostfile τότε πρώτα θα δημιουργηθεί το σύστημα με βάση το αρχείο και στη συνέχεια οι κόμβοι θα φιλτραριστούν έτσι ώστε να διατηρηθούν στο σύστημα μόνο όσοι έχουν δηλωθεί στην παράμετρο hosts, διατηρώντας αναλλοίωτα τα χαρακτηριστικά

```
#This is an example hostfile

#comments start with #

#This is a single processor machine
node0

#This is a two-processor machine
node1 slots=2 #cores=3 # this takes a custom cores arg only for getInfo

#This machine can run up to 2 processes

node2 slots=2 maxSlots=2
node3 slots=5 maxSlots=20
```

Σχήμα 5.2: Παράμετροι αρχείου hostfile

τους. Σε περίπτωση που υπάρχει ασυμφωνία μεταξύ των ονομάτων των κόμβων ανάμεσα στις δύο παραμέτρους, τότε όπως και στον πραγματικό διαχειριστή Mpirun η διαδικασία θα τερματίσει με κατάλληλο μήνυμα λάθους. Τέλος αν δεν δοθεί καμία από τις δύο παραμέτρους, ο προσομοιωτής αναγκαστικά κατά το χρόνο εκτέλεσης θα ρωτήσει το χρήστη να δώσει τόσο τον αριθμό των κόμβων της συστάδας όσο και τον αριθμό των slots σε κάθε κόμβο, ο αριθμός των επεξεργαστικών πυρήνων καθορίζεται όπως και παραπάνω. Προφανώς αυτή η επιλογή θα δημιουργήσει ένα σύστημα που θα αποτελείται από N ίδιους μεταξύ τους κόμβους και δεν συστήνεται. Η βέλτιστη εικόνα παράγεται όταν δίνεται ένα hostfile στην είσοδο του προσομοιωτή είτε μόνο του είτε σε συνδυασμό με την παράμετρο hosts και αυτή προτείνεται. Ο πραγματικός διαχειριστής Mpirun έχει στη διάθεσή του μια περισσότερο πολύπλοκη εικόνα του συστήματος η οποία αποτελείται από περισσότερα δομικά μέρη (όπως για παράδειγμα sockets ή hardware threads) αλλά επίσης μπορεί να αναφερθεί και σε μη επεξεργαστικά στοιχεία όπως κρυφές μνήμες. Αντίστοιχα την ίδια δομή έχουν στη διάθεσή τους και διαχειριστές πόρων όπως ο SLURM, οι οποίοι έχουν τη δυνατότητα να αντλήσουν πληροφορίες από το σύστημα.

5.3.3 Απεικόνιση και βαθμονόμηση

Δεδομένου ότι στα προηγούμενα δύο στάδια ο προσομοιωτής έχει συλλέξει όλες τις απαραίτητες πληροφορίες έτσι ώστε να γνωρίζει τις απαιτήσεις σε πόρους της εφαρμογής που θεωρητικά θα κληθεί να ξεκινήσει καθώς και τη δομή του συστήματος, ξεκινάει την εκτέλεση του αλγορίθμου που ακολουθεί και ο πραγματικός διαχειριστής Mpirun. Σε ένα πρώτο κύκλο θα επιλέξει ποιους επεξεργαστικούς πυρήνες, με βάση κανόνες που δίνονται από το χρήστη είτε έχει ενσωματωμένους, θα χρησιμοποιήσει και σε δεύτερο χρόνο θα αποδόσει σε κάθε έναν, ή πιο σωστά θα αποδώσει σε κάθε MPI διεργασία που θα εκτελεστεί στο συγκεκριμένο επεξεργαστικό πυρήνα, ένα βαθμό (rank). Στη πρώτη φάση εκτέλεσης ο προσομοιωτής προσφέρει τρεις δυνατότητες, οι οποίες επιλέγονται κατά κύριο λόγο με βάση την τιμή της παραμέτρου –map-by. Η παράμετρος δέχεται τρεις τιμές node, slot, core οι οποίες προφανώς θα εκτελέσουν τη λειτουργία της απεικόνισης κατά κόμβο, slot και επεξεργαστικό πυρήνα αντίστοιχα. Κάθε επιλογή καθορίζει το βήμα ανάμεσα σε διαδοχικές επιλογές. Για παράδειγμα στην κατά

κόμβο επιλογή πρώτα θα επιλεχθεί ο πρώτος διαθέσιμος επεξεργαστικός πυρήνας ενός slot ενός κόμβου και ο επόμενος που θα επιλεγεί θα είναι ο πρώτος διαθέσιμος επεξεργαστικός πυρήνας του επόμενου κόμβου. Αντίστοιχα σε μια επιλογή κατά slot πρώτα θα επιλεχθεί ένας επεξεργαστικός πυρήνας από κάθε διαθέσιμο slot ενός κόμβου και μετά θα επιλεχθεί πόρος από άλλο κόμβο. Τέλος στην επιλογή κατά επεξεργαστικό πυρήνα θα επιλεχθούν σειριακά όλοι οι πυρήνες ενός κόμβου.

Σε αντιστοιχία με τον πραγματικό διαχειριστή, ο προσομοιωτής θα αναγνωρίσει πότε ένας πόρος του συστήματος θα εξαντληθεί και θα τον αγνοήσει στη συνέχεια της εκτέλεσης. Η προηγούμενη πρόταση δεν ισχύει σε περίπτωση όπου η συστάδα δεν διαθέτει τόσους πόρους όσους έχει ζητήσει ο χρήστης και η υπάρχει η δυνατότητα υπερφόρτωσης. Τότε ο προσομοιωτής θα επανεκκινήσει την ίδια διαδικασία έως ότου διασφαλίσει ότι οι απαιτήσεις του χρήστη ικανοποιούνται. Σημειώνεται ότι ο συγκεκριμένος μηχανισμός είναι πιο απλός από ότι στον πραγματικό διαχειριστή καθώς γίνεται η παραδοχή ότι σε ένα πραγματικό σύστημα και κάποιο πραγματικό πρόβλημα είναι λίγες οι πιθανότητες ένας χρήστης να επιλέξει να μειώσει την απόδοση της εκτέλεσης για να υπερφορτώσει τους πυρήνες. Συνεπώς η συγκεκριμένη λειτουργία έχει προβλεφθεί κυρίως για λόγους πληρότητας.

Όπως ήδη αναφέρθηκε, ο προσομοιωτής θα συνεχίσει να εκτελεί τον αλγόριθμο της δέσμευσης πόρων του συστήματος μέχρι να καλύψει τον αριθμό που ζητήθηκε κατά την κλήση του. Ο χρήστης θα πρέπει να υποδείξει στον προσομοιωτή πόσους επεξεργαστικούς πυρήνες θα χρειαστεί να δεσμεύσει η εφαρμογή κάνοντας χρήση της παραμέτρου -np, ή των συνωνύμων της. Αυτή είναι άλλη μια διαφορά μεταξύ της λειτουργίας του Mpirun και του προσομοιωτή καθώς ο πραγματικός διαχειριστής σε περίπτωση που δεν δοθεί η συγκεκριμένη παράμετρος θα συμπεράνει τον αριθμό των πόρων που ζητούνται με βάση τις τιμές των παραμέτρων –host και –hostfile και εσωτερικούς κανόνες. Αντιθέτως, ο προσομοιωτής θα εκκινήσει μόνο μία MPI διεργασία.

Η φάση της βαθμονόμησης είναι το επόμενο βήμα που ακολουθούν οι διαχειριστές παράλληλων εφαρμογών. Κατα τη διάρκεια της φάσης αυτής, ο εκάστοτε διαχειριστής θα ξαναπεράσει από κάθε δεσμευμένο επεξεργαστικό πυρήνα και θα δώσει στην MPI διεργασία που θα εκτελεστεί σε αυτόν μία ταυτότητα την οποία οι υπόλοιπες διεργασίες θα χρησιμοποιούν κατά το χρόνο εκτέλεσης για να επικοινωνήσουν με αυτήν. Σε αντιστοιχία με την πρώτη φάση, στη δεύτερη ο προσομοιωτής επιτρέπει πάλι τρεις διαφορετικές παραμετροποιήσεις. Με τη χρήση της παραμέτρου –rank-by, η οποία δέχεται τις τιμές host, slot, core, τα οι βαθμοί του προτύπου θα αποδοθούν κατά κόμβο, slot και επεξεργαστικό πυρήνα αντίστοιχα. Επιλέγεται, λοιπόν, όπως και προηγουμένως το βήμα με βάση το οποίο ο προσομοιωτής θα αποδίδει τους διαδοχικούς βαθμούς.

Όλα τα παραπάνω θα γίνουν πιο κατανοητά στην ενότητα των παραδειγμάτων, όπου ο αναγνώστης θα εξετάσει πώς μπορεί να συνδυάσει τις παραπάνω επιλογές για να παράγει διαφορετικές απεικονίσεις των διεργασιών σε ένα σύστημα.

5.3.4 Rankfile

Σε πολλούς διαχειριστές συστήματος είναι εφικτό να ζητηθούν οι ακριβείς πυρήνες που θα χρησιμοποιηθού κατά το χρόνο εκτέλεσης από μία εφαρμογή. Συγκεκριμένα για τον Mpirun αυτός ο τρόπος επιτυγχάνεται δίνοντας ως παράμετρο ένα αρχείο εισόδου που περιέχει την απεικόνιση ανάμεσα σε μία διεργασία MPI και το αναγνωριστικό του πυρήνα. Αυτό το αρχείο φαίνεται και στην εικόνα 5.3, σε δύο διαφορετικές μορφές του.

Αντίστοιχα ο προσομοιωτής υποστηρίζει αυτή τη λειτουργία σε ένα βαθμό. Είναι όμως αναγκασμένος, σε αντίθεση με την περίπτωση του hostfile να υποστηρίξει μια απλοποιημένη μορφή καθώς η σύνταξη του αρχείου δεν μπορεί να αντιστοιχηθεί στην αφαιρετική απεικόνιση του συστήματος που δημιουργεί ο προσομοιωτής. Συνεπώς, η μόνη περίπτωση που χειρίζεται το εργαλείο GetInfo είναι αυτή που απεικονίζεται δεύτερη στην εικόνα 5.3, την οποία μορφή την συναντά κανείς στη βιβλιογραφία ως physical rankfile.

```
fkotomat@scirouter:~/getInfo$ cat rankfile-example.txt
rank 0=aa slot=1:0-2
rank 1=bb slot=0:0,1
rank 2=cc slot=1-2
fkotomat@scirouter:~/getInfo$
fkotomat@scirouter:~/getInfo$
fkotomat@scirouter:~/getInfo$ cat rankfile-physical-example.txt
rank 0=aa slot=1
rank 1=bb slot=8
rank 2=cc slot=6
fkotomat@scirouter:~/getInfo$
```

Σχήμα 5.3: Δείγμα αρχείου rankfile

Ο λόγος που υποστηρίζεται η συγκεκριμένη παράμετρος είναι σε περίπτωση που ο χρήστης θέλει να δοκιμάσει να προβλέψει το χρόνο επικοινωνίας της εφαρμογής του αν δώσει κάποιο ειδικό σχήμα ανάθεσης. Επειδή δεν είναι πολύ διαδεδομένη πρακτική, ο προσομοιωτής κάνει την υπόθεση ότι ο χρήστης γνωρίζει ακριβώς τι θέλει να πετύχει και αγνοεί όλες τις υπόλοιπες παραμέτρους. Αυτό σημαίνει πρακτικά ότι είναι η μόνη περίπτωση κατά την οποία δεν θα δημιουργηθεί το εικονικό υπολογιστικό σύστημα, αλλά κατευθείαν το εργαλείο θα μετατρέψει τα δεδομένα εισόδου στην κατάλληλη μορφή εξόδου.

5.4 Παραδείγματα Χρήσης

Για να γίνει πλήρως κατανοητή η χρήση του προσομοιωτή θα παρουσιάσουμε τέσσερα διαφορετικά παραδείγματα. Στα πρώτα δύο θα χρησιμοποιηθεί τα αρχείο hostfile που παρουσιάστηκε πιο πάνω και θα ζητηθούν για τον ίδιο αριθμό διεργασιών, διαφορετικές απεικονίσεις. Στο τρίτο θα δούμε μια πολύ απλή κλήση του προσομοιωτή και τον τρόπο με τον οποίο αυτός θα ζητήσει πληροφορίες από το χρήστη για να μπορέσει να παράγει τα αποτελέσματα της προσομοίωσης. Τέλος, θα δώσουμε ως είσοδο το αρχείο rankfile και θα παρατηρήσουμε κατά πόσο η έξοδος αντιστοιχεί σε αυτό.

5.4.1 Παράδειγμα-1: Χρήση παραμέτρου bynode

Στο πρώτο παράδειγμα κάνουμε χρήση της παραμέτρου bynode ενώ έχουμε δώσει ένα hostfile το οποίο αντιστοιχεί στο σύστημα το οποίο προσομοιώνουμε. Στην πρώτη εικόνα είναι ενδιαφέρον να δούμε τον τρόπο με τον οποίο το συγκεκριμένο αρχείο εισόδου αναπαρίσταται εσωτερικά από το εργαλείο. Αυτό φαίνεται στην πρώτη εικόνα, 5.4, η οποία προκύπτει από ένα τμήμα της εξόδου της λειτουργίας του εργαλείου, αν αυτό εκτελεστεί σε verbose mode.

```
########SIMULATION ENV##############
Node id=node0
        Slot id=0
                Core id=0:0:node0
Node id=node1
        Slot id=0
                Core id=0:0:node1
                Core id=1:0:node1
                Core id=2:0:node1
        Slot id=1
                Core id=0:1:node1
                Core id=1:1:node1
                Core id=2:1:node1
Node id=node2
        Slot id=0
                Core id=0:0:node2
        Slot id=1
                Core id=0:1:node2
Node id=node3
        Slot id=0
                Core id=0:0:node3
        Slot id=1
                Core id=0:1:node3
        Slot id=2
                Core id=0:2:node3
        Slot id=3
                Core id=0:3:node3
        Slot id=4
                Core id=0:4:node3
####################################
```

Σχήμα 5.4: Εσωτερική αναπαράσταση συστήματος

Στη δεύτερη εικόνα, 5.5φαίνονται τα αποτελέσματα της εκτέλεσης. Υπενθυμίζεται ότι οι κόμβοι επιλέγονται κυκλικά και στη συνέχεια το ίδιο επαναλαμάνεται και για την ανάθεση των βαθμών.

```
foivos@foivos-lpc:~/PycharmProjects/getInfo$ ./getInfo.py --hostfile hostFile-example -np 8 --bynode

##########OUTPUT###############
0 0:0:node0
1 0:0:node1
2 0:0:node2
3 0:0:node3
4 1:0:node1
5 2:0:node1
6 0:1:node2
7 0:1:node3
```

Σχήμα 5.5: Παράδειγμα εκτέλεσης bynode

5.4.2 Παράδειγμα-2: Χρήση πολλαπλών παραμέτρων

Στο δεύτερο παράδειγμα χρήσης του εργαλείου, δίνουμε ως είσοδο το ίδιο αρχείο για την απεικόνιση του συστήματος και ζητάμε τον ίδιο αριθμό διεργασιών για την εκτέλεση της εφαρ-

μογής. Όμως προσδιορίζουμε επιπλέον ότι θέλουμε η απεικόνιση να γίνεται ανά επεξεργαστικό πυρήνα, ενώ η βαθμονόμηση ανά slot. Αυτή η παραμετροποίηση οδηγεί να εξαντλούμε όλους τους επεξεργαστικούς πυρήνες ενός κόμβου πριν προχωρήσουμε στον επόμενο, αλλά η επόμενη φάση εναλλάσεται μεταξύ των κόμβων επιλέγοντας κάθε φορά το πρώτο διαθέσιμο slot. Τα αποτελέσματα αυτής της κλήσης φαίνονται στην εικόνα 5.6.

```
foivos@foivos-lpc:~/PycharmProjects/getInfo$ ./getInfo.py --hostfile hostFile-example -np 8 --map-by core --rank-by slot

##########OUTPUT##############
0 0:0:node0
1 0:0:node1
2 0:1:node1
3 0:0:node2
4 1:0:node1
5 1:1:node1
6 2:0:node1
7 2:1:node1
```

Σχήμα 5.6: Παράδειγμα σύνθετης εκτέλεσης

5.4.3 Παράδειγμα-3: Κλήση προσομοιωτή χωρίς παραμέτρους

Έχοντας αναφερθεί στον τρόπο με τον οποίο ο προσομοιωτής κατασκευάζει την εικόνα του συστήματος προς προσομοίωση, στο συγκεκριμένο παράδειγμα, θέλουμε να δούμε πώς θα συμπεριφερθεί αν δεν έχει τις απαραίτητες πληροφορίες κατά την κλήση του.

Βλέπουμε ότι ρωτά κατά το χρόνο εκτέλεσης τον χρήστη έτσι ώστε να αποκτήσει μια γενική εικόνα και στη συνέχεια λειτουργεί με βάση εσωτερικούς κανόνες οι οποίοι αντιστοιχούν τη λειτουργία του Mpirun σε αντίστοιχες περιπτώσεις. Τα αποτελέσματα φαίνονται στην εικόνα 5.7

```
foivos@foivos-lpc:~/PycharmProjects/getInfo$ ./getInfo.py -np 4
Please specify number of host on the cluster:2
Please specify number of slots on each host:8

##########OUTPUT##############
0 0:0:node0
1 0:1:node0
2 0:0:node1
3 0:1:node1
```

Σχήμα 5.7: Παράδειγμα αλληλεπίδρασης με το χρήστη

5.4.4 Παράδειγμα-4: Χρήση αρχείου rankfile

Στο τελευταίο παράδειγμα χρήσης του εργαλείου, δίνεται ως είσοδος ένα αρχείο rankfile. Θέλουμε να παρατηρήσουμε τον τρόπο με τον οποίο αυτό θα μεταφραστεί σε μορφή σύμφωνη με τα πρότυπα του εργαλείου. Υπενθυμίζεται ότι η συγκεκριμένη παράμετρος υπερκαλύπτει οποιαδήποτε άλλη έχει δωθεί από το χρήστη. Μοναδική εξαίρεση στα παραπάνω είναι η παράμετρος που ανακατευθύνει την έξοδο του προσομοιωτή. Η συγκεκριμένη παράμετρος λαμβάνεται υπόψη αν έχει δωθεί.Τα αποτελέσματα φαίνονται στην εικόνα 5.8.

```
foivos@foivos-lpc:~/PycharmProjects/getInfo$ ./getInfo.py -np 6 --bynode -rf rankfile-physical-example.txt

##########OUTPUT###############
0 0:1:aa
1 0:8:bb
2 0:6:cc
```

Σχήμα 5.8: Παράδειγμα κλήσης με rankfile

Κεφάλαιο 6

Παραδείγματα και πειραματική αξιολόγηση

6.1 Εισαγωγή

Στο συγκεκριμένο κεφάλαιο θα αξιολογήσουμε την απόδοση της βιβλιοθήκης MinI πάνω σε δύο διαφορετικούς άξονες. Πρώτα θα μελετήσουμε τη επίπτωση που έχει στο χρόνο εκτέλεσης μίας εφαρμογής και στη συνέχεια θα εξετάσουμε την επίπτωση της βιβλιοθήκης στις απαιτήσεις της εφαρμογής για εκχώρηση μνήμης. Για την συγκεκριμένη διαδικασία έχουν επιλεχθεί εφαρμογές του πραγματικού κόσμου, κυρίως από τη σειρά μετροπρογραμμάτων miniapps της Mantevo και αντίστοιχα έργα της ExMatEx, με τα οποία θα μπορέσει να ελεγχθεί η ορθότητα και η αποδοτικότητα της βιβλιοθήκης σε μεγαλύτερα δείγματα εισόδου. Ο λόγος που εξετάζεται μόνο η βιβλιοθήκη ως προς τα παραπάνω χαρακτηριστικά είναι ότι είναι το μοναδικό εργαλείο το οποίο απαιτεί την εκτέλεση της εφαρμογής σε ένα υπολογιστικό σύστημα. Τα άλλα δύο μπορούν να χρησιμοποιηθούν και να εξάγουν τα αποτελέσματά τους στατικά.
Πρέπει να τονιστεί εδώ, η εκτέλεση των μετρήσεων έγινε υπερφορτώντας τους πόρους του συστήματος clones του εργαστηρίου το οποίο είναι ένα σχετικά μικρό σύστημα. Το συγκεκριμένο σύστημα αποτελείται από μια συστάδα κόμβων, όπου κάθε κόμβος διαθέτει οχτώ (8) επεξεργαστικούς πόρους και η μνήμη είναι αρχιτεκτονικής NUMA. Για όλα τα πειράματα χρησιμοποιήθηκαν 9 κόμβοι από τη συστάδα οι οποίοι διέθεταν 8 GB μνήμης ο καθένας. Κάθε εφαρμογή και για τα δύο πειράματα έχει στόχο να κλιμακώσει από μία (1) έως διακόσιες πενηνταέξι (256) διεργασίες, ή μέχρι να εξαντληθούν οι πόροι της συστάδας. Συνεπώς, ταυτόχρονα με την αξιολόγηση της βιβλιοθήκης, θα αξιολογήσουμε και κατά πόσο η προσέγγιση μας για συλλογή ιχνών ανεξάρτητων από το χρόνο είναι εφικτή με βάση αυτή την προσέγγιση. Τέλος θα παρουσιάσουμε και συνοπτικά σε απόλυτα νούμερα τα μεγέθη των αρχείων που παράγονται από τη βιβλιοθήκη και πόσο αυτά συμπυκνώνονται μετά την εφαρμογή του MinI Parser.

6.2 Πειραματική αξιολόγηση

6.2.1 Παρουσίαση μετροπρογραμμάτων

Για την πειραματική αξιολόγηση επιλέχθηκαν μετροπρογράμματα τα οποία είναι ευρέως χρησιμοποιούμενα. Συγκεκριμένα, επιλέχθηκαν τα εξής:

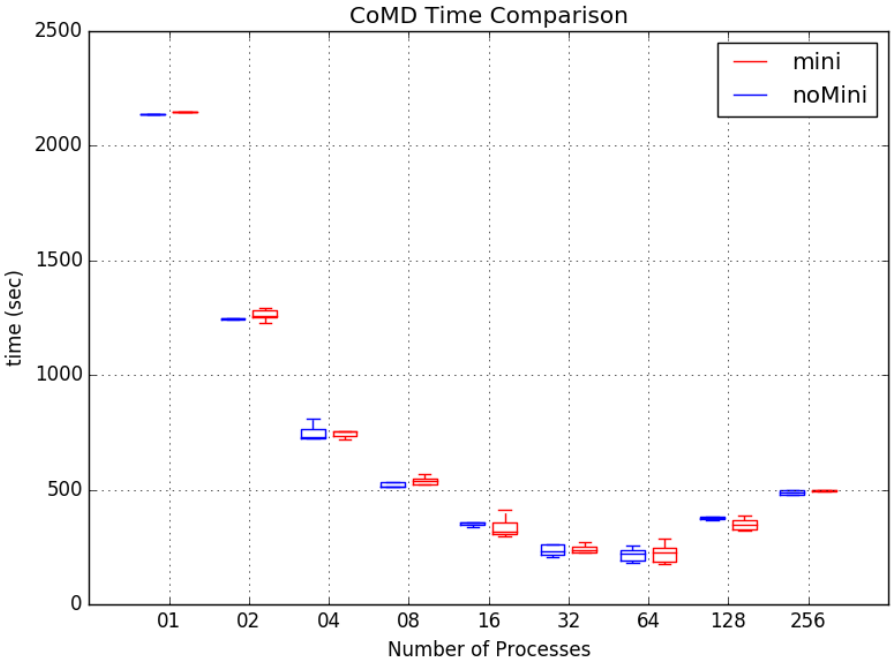
- CoMD: Μία εφαρμογή η οποία προσομοιώνει τη δυναμική των μορίων ενός υλικού. Έχει αναπτυχθεί και συντηρείται από την ExMatEx ως ένα έργο ανοιχτού λογισμικού.
- CoSP2: Μία εφαρμογή η οποία περιέχει στοιχεία αραιής γραμμικής άλγεβρας (sparse linear algebra) και προσομοιώνει τις ηλεκτρικές δυνάμεις μεταξύ των μορίων. Είναι ένα έργο που επίσης δημιουργήθηκε και συντηρείται από την ExMatEx ως ένα έργο ανοιχτού λογισμικού.
- HPCCG: Η συγκεκριμένη εφαρμογή της Mantevo χειρίζεται πίνακες και εκτελεί πράξεις γραμμικής άλγεβρας.
- MiniDFT: Η εφαρμογή της NERSC χρησιμοποιείται για να μοντελοποιήσει υλικά. Λαμβάνει ως είσοδο τη θέση των μορίων του υλικού και προσπαθεί να εξάγει συγκεκριμένα χαρακτηριστικά. Κατά τη διάρκεια εκτέλεσης κάνει χρήση και των αλγορίθμων του Fast Fourier (FFT).
- MiniAMR: Η τελευταία εφαρμογή που χρησιμοποιήθηκε ανήκει και αυτή στις εφαρμογές της Mantevo και εκτελεί stencil υπολογισμούς πάνω σε μία δομή του χώρου η οποία χωρίζεται σε επιμέρους κύβους. Οι κύβοι μοιράζονται στις διεργασίες και βελτιστοποιούνται ανεξάρτητα.

Οι παραπάνω εφαρμογές, όπως φαίνεται εκτελούν μια πληθώρα διαφορετικών μεταξύ τους λειτουργιών. Η επιλογή, όμως, των συγκεκριμένων μετροπρογραμμάτων δεν έγινε με βάση την λειτουργία την οποία είναι σχεδιασμένα να εκτελούν καθώς τα εργαλεία της συγκεκριμένης διπλωματικής δεν εξαρτώνται από το είδος των υπολογισμών που θα εκτελεί μία εφαρμογή. Η επιλογή έγινε πρώτον με βάση τη γλώσσα υλοποίησης. Συγκεκριμένα, επιλέχθηκαν προγράμματα που έχουν υλοποιηθεί στις γλώσσες C όπως το CoMD, C++ όπως το HPCCG και Fortran, το MiniDFT συνδυάζει τη χρήση Fortran και C. Δεύτερον, έγινε προσπάθεια να χρησιμοποιηθούν προγράμματα τα οποία να καλύπτουν όλο των εύρος των MPI κλήσεων που υποστηρίζει η βιβλιοθήκη. Πιο αναλυτικά, όλα τα μετροπρογράμματα χρησιμοποιούν κλήσεις σημείου προς σημείο καθώς και συλλογικές, κάθε ένα κάνοντας μεγαλύτερη χρήση συγκεκριμένων κλήσεων με βάση τις ανάγκες του. Για παράδειγμα η κλήση MPI_Allreduce() είναι η επικρατέστερη και το CoMD είναι το μοναδικό που κάνει χρήση της MPI_Sendrecv(). Τα HPCCG, CoSP2 και MiniAMR κάνουν χρήση ασύγχρονων κλήσεων, τα MiniDFT και MiniAMR χρησιμοποιούν πάνω από ένα communicator ενώ επιπλέον το MiniDFT είναι το μοναδικό μετροπρόγραμμα που κάνει χρήση διανυσματικών συλλογικών κλήσεων.

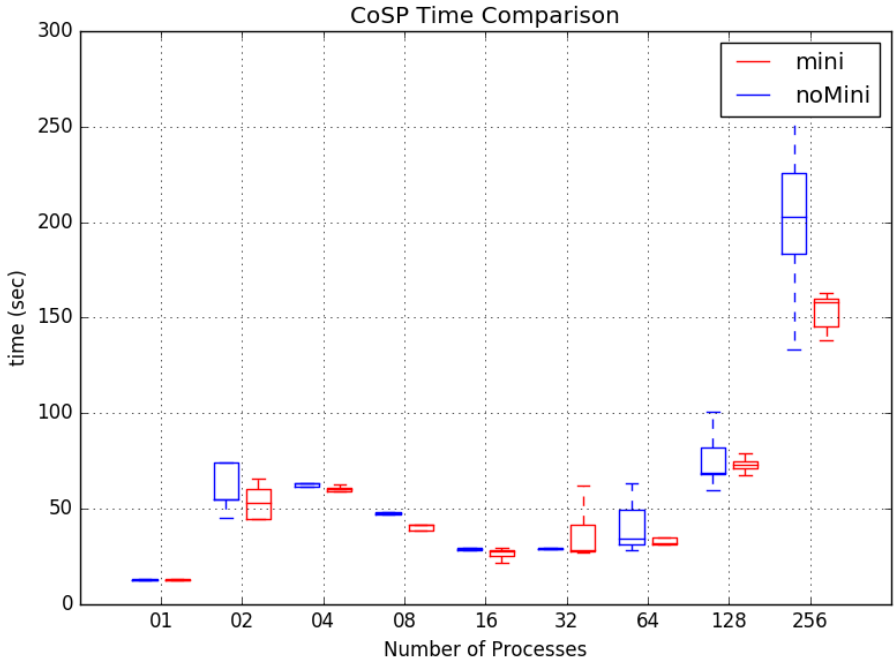
Στα περισσότερα από τα παρακάτω πειραματικά αποτελέσματα, με εξαίρεση το MiniAMR, έχει χρησιμοποιηθεί μια παραμετροποίηση η οποία ακολουθεί τα πρότυπα της ισχυρής κλιμάκωσης, αν και αυτό δεν ήταν αναγκαστικό καθώς στόχος είναι η ίδια συμπεριφορά των μετροπρογραμμάτων τόσο χωρίς τη χρήση της βιβλιοθήκης όσο και με χρήση της, ανεξαρτήτως των παραμέτρων. Όλες οι πειραματικές μετρήσεις έχουν εκτελεστεί πολλαπλό αριθμό φορών έτσι ώστε να περιοριστούν κατά το δυνατό τα τυχαία σφάλματα.

6.2.2 Αξιολόγηση απόδοσης στο χρόνο εκτέλεσης

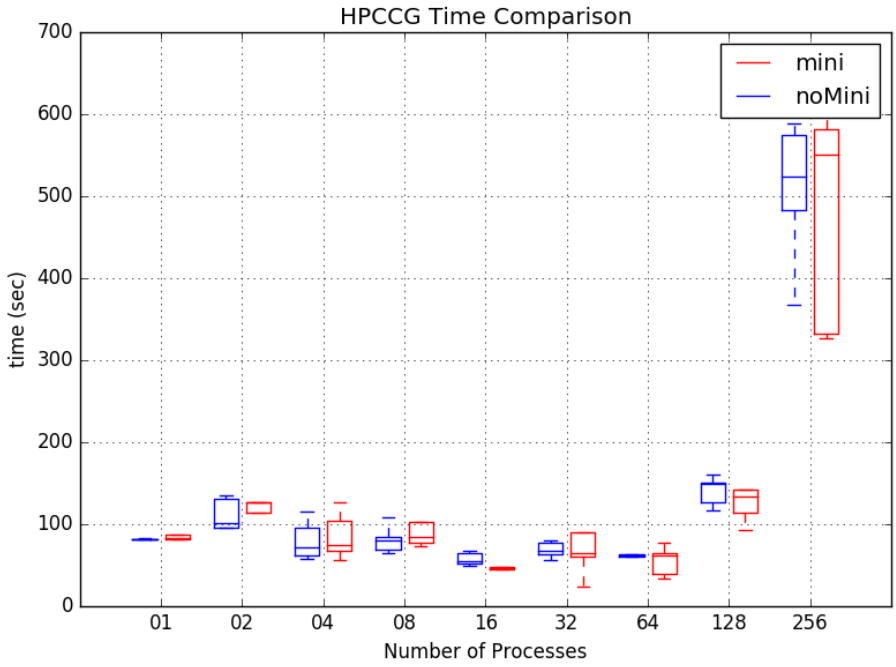
Η πρώτη μέτρηση που εκτελέστηκε έχει σχέση με τη χρονική καθυστέρηση που εισάγει η βιβλιοθήκη στο χρόνο εκτέλεσης μίας εφαρμογής. Τα μετροπρογράμματα που έχουν παρουσιαστεί εκτελέστηκαν στα συστήματα του εργαστηρίου και μετρήθηκε ο συνολικός χρόνος εκτέλεσης με βάση την εντολή του UNIX time, πρώτα χωρίς τη χρήση της βιβλιοθήκης και στη συνέχεια αφού η βιβλιοθήκη ενσωματώθηκε σε αυτά. Τα αποτελέσματα απεικονίστηκαν σε γραφήματα με χρήση boxplots.



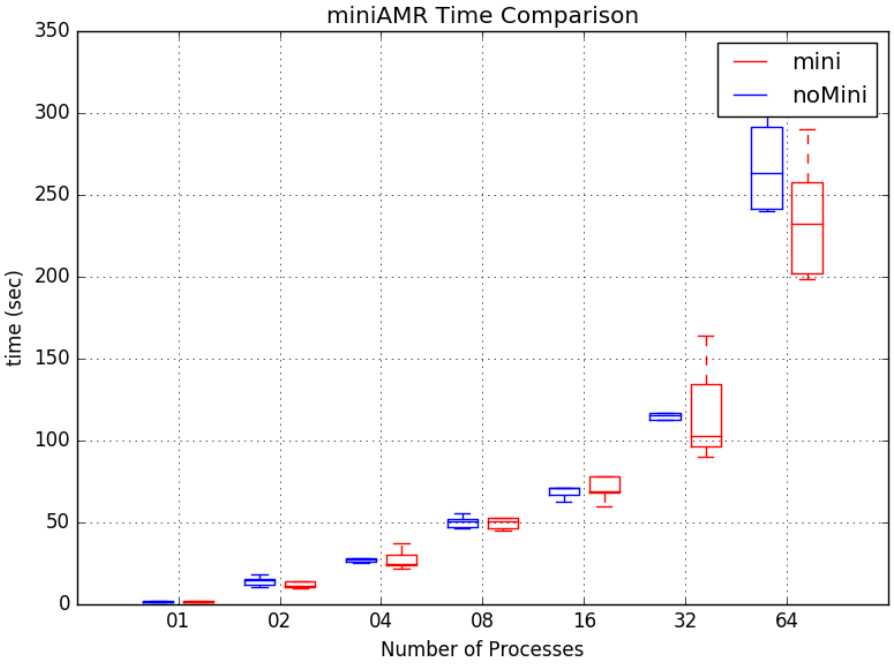
Σχήμα 6.1: Συσχετισμός χρόνου εκτέλεσης για CoMD



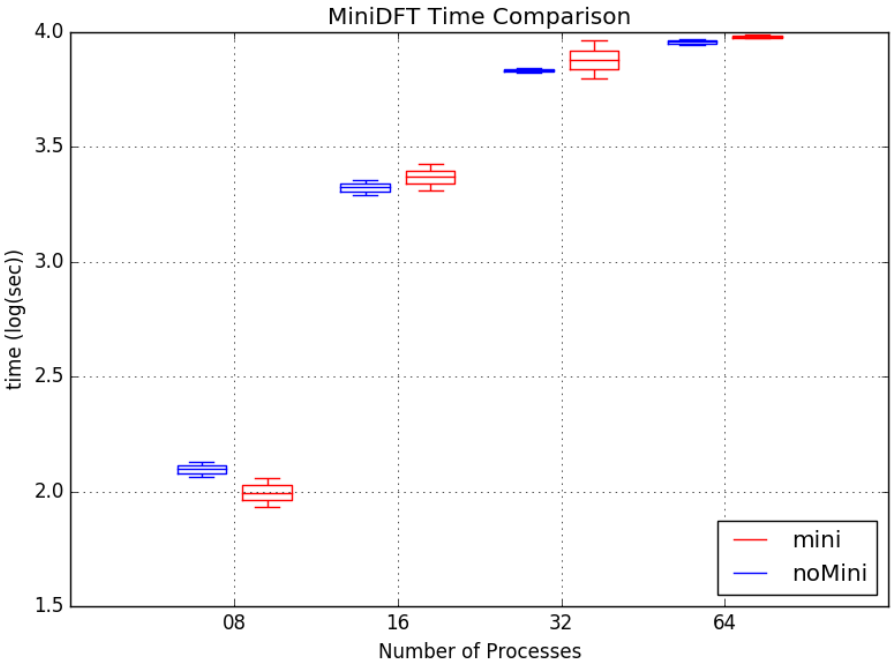
Σχήμα 6.2: Συσχετισμός χρόνου εκτέλεσης για CoSP



Σχήμα 6.3: Συσχετισμός χρόνου εκτέλεσης για HPCCG



Σχήμα 6.4: Συσχετισμός χρόνου εκτέλεσης για miniAMR



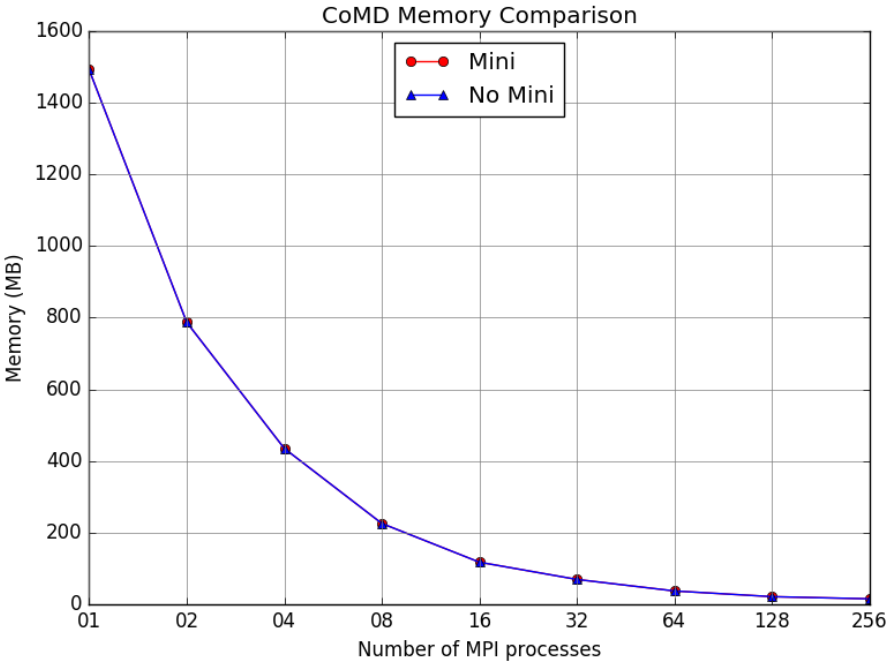
Σχήμα 6.5: Συσχετισμός χρόνου εκτέλεσης για MiniDFT

Από τα παραπάνω πειραματικά αποτελέσματα γίνεται φανερό ότι η βιβλιοθήκη δεν εισάγει χρονική καθυστέρηση στην εκτέλεση. Είναι εμφανές σε όλα τα διαγράμματα ότι και οι δύο εφαρμογές εμφανίζουν την ίδια συμπεριφορά όσο αλλάζει ο αριθμός των πυρήνων εκτέλεσης. Είναι γεγονός, βέβαια ότι παρατηρείται μια μεταβλητότητα τόσο ως προς τη μέση τιμή όσο και ως προς τη διακύμανση για την εκάστοτε παραμετροποίηση. Όμως, αυτή η μεταβλητότητα επίσης παρουσιάζει μια τυχαιότητα καθώς σε άλλες περιπτώσεις η εκτέλεση χωρίς τη χρήση της βιβλιοθήκης εμφανίζεται ταχύτερη και σε άλλες παρατηρείται το ανάποδο. Λόγω αυτής της τυχαιότητας, που παρατηρείται σε όλα τα γραφήματα, θεωρείται ότι οφείλεται περισσότερο σε εξωτερικούς παράγοντες παρά στην ίδια τη βιβλιοθήκη. Τέτοιοι παράγοντες είναι κατά κύριο λόγο η μνήμη, όπου πολλές διεργασίες συγκρούονται για συγκεκριμένες προσβάσεις, και το δίκτυο διασύνδεσης μεταξύ των κόμβων.

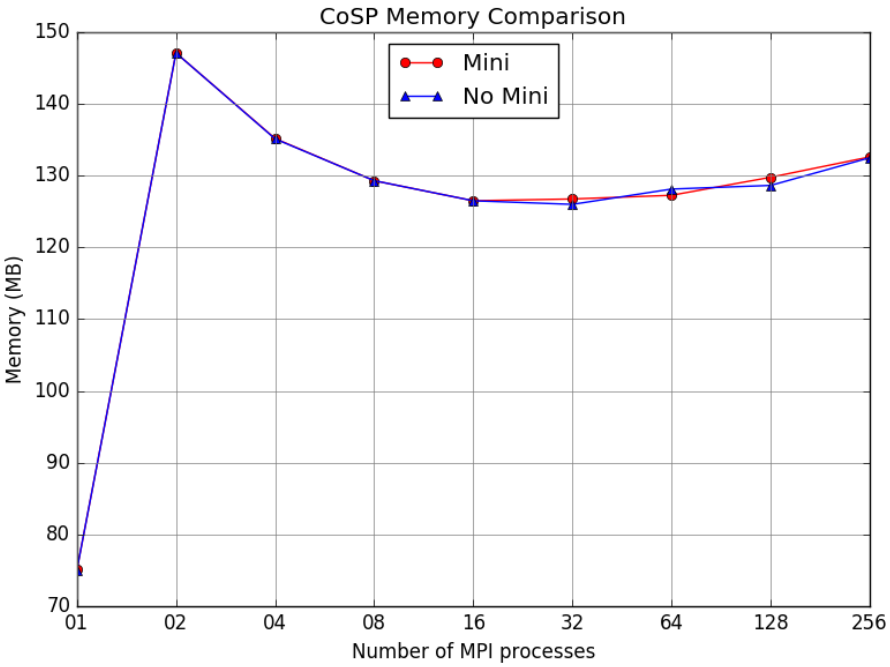
Σημειώνεται εδώ ότι το MiniDFT είναι μια εφαρμογή υπολογιστικά αρκετά βαρύτερη από όλες τις υπόλοιπες. Συνεπώς, οι απαιτήσεις της ως προς τους πόρους του συστήματος είναι πολύ πιο υψηλές. Συνεπώς δεν μπορούσε να εξυπηρετηθεί από τα μηχανήματα clones στις περιπτώσεις που θα χρειαζόταν υπερφόρτωση των πυρήνων. Αποδεικνύεται εδώ, όπως είχε αναφερθεί στην αρχή ότι οι δυνατότητες να εξαχθούν ίχνη από μια εφαρμογή, με βάση την προσέγγιση που έχουμε κάνει, περιορίζεται στις δυνατότητες κλιμάκωσης του συστήματος.

6.2.3 Αξιολόγηση απόδοσης στις απαιτήσεις μνήμης

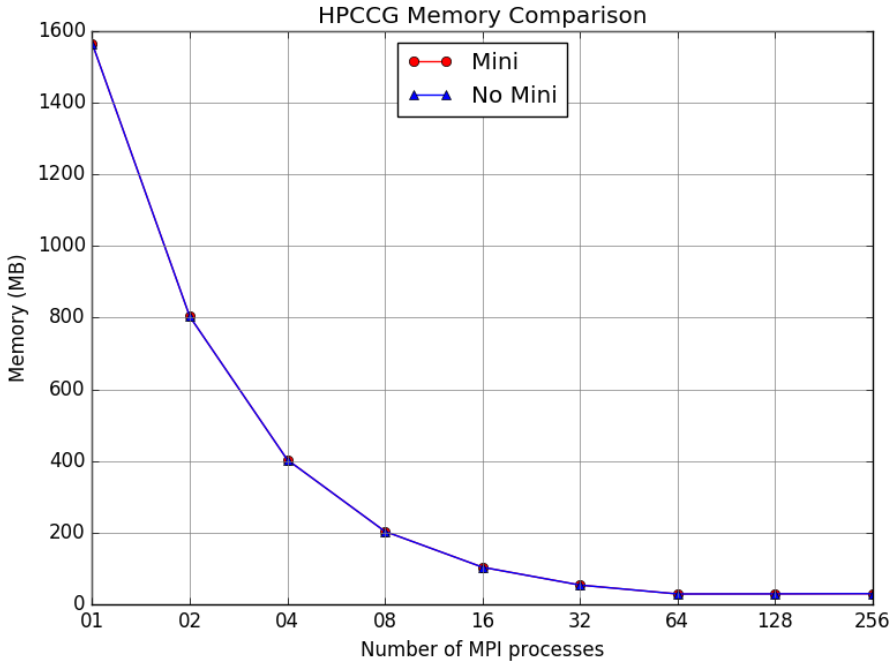
Η δεύτερη μέτρική που πρέπει να μελετηθεί σχετικά με την αποδοτικότητα του εργαλείου που αναπτύχθηκε είναι η επίδραση του στις απαιτήσεις μνήμης μιας εφαρμογής. Για την εκτέλεση του συγκεκριμένου πειράματος χρησιμοποιήθηκε το εργαλείο massif του valgrind το οποίο μέτρησε τη χρησιμοποιούμενη μνήμη από κάθε MPI διεργασία στο χρόνο εκτέλεσης. Στη συνέχεια, από κάθε διεργασία συλλέχθηκε η μέγιστη τιμή και υπολογίστηκε ο μέσος όρος. Όπως και προηγουμένως, η μέτρηση πραγματοποιήθηκε σε όλα τα μετροπρογράμματα, στα συστήματα του εργαστηρίου, χωρίς τη χρήση της βιβλιοθήκης και στη συνέχεια επαναλήφθηκαν αφού η βιβλιοθήκη ενσωματώθηκε στις εφαρμογές. Τα αποτελέσματα παρουσιάζονται στα παρακάτω γραφήματα.



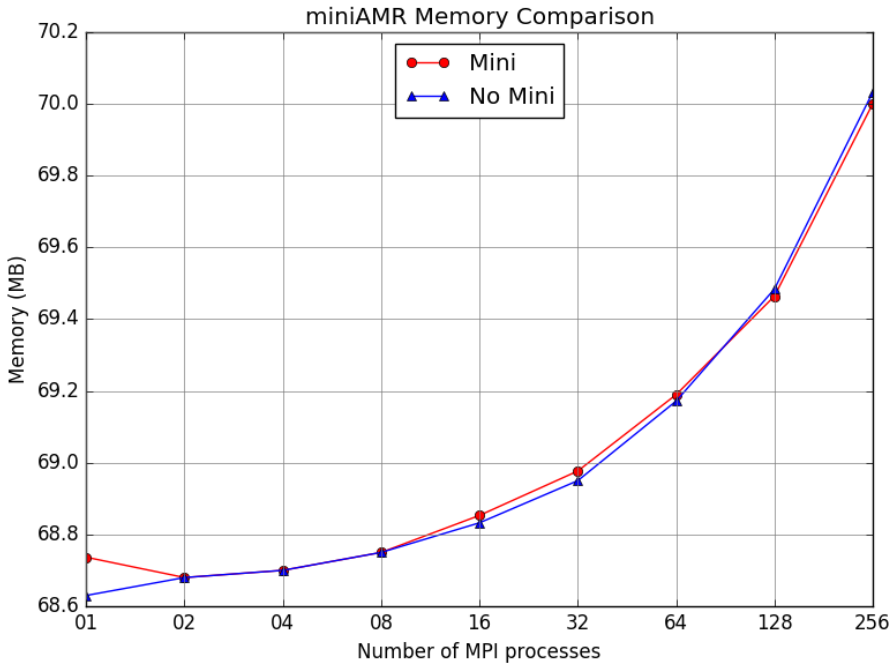
Σχήμα 6.6: Συσχετισμός μέγιστης χρήσης μνήμης CoMD



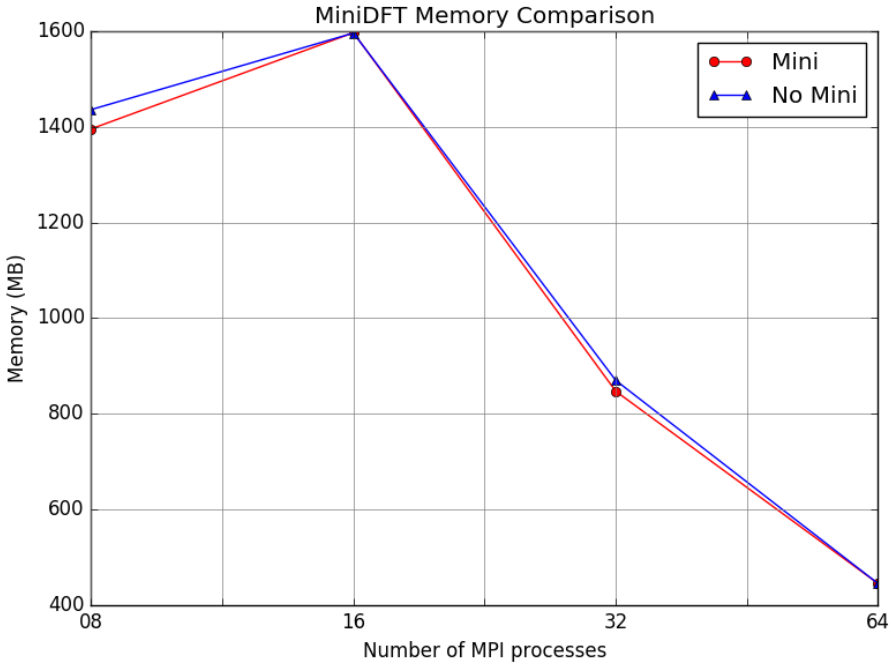
Σχήμα 6.7: Συσχετισμός μέγιστης χρήσης μνήμης CoSP



Σχήμα 6.8: Συσχετισμός μέγιστης χρήσης μνήμης HPCCG



Σχήμα 6.9: Συσχετισμός μέγιστης χρήσης μνήμης miniAMR



Σχήμα 6.10: Συσχετισμός μέγιστης χρήσης μνήμης MiniDFT

Τα παραπάνω γραφήματα δείχνουν ότι οι εκτελέσεις πρακτικά ταυτίζονται. Αυτό σημαίνει ότι ο φόρτος που προσθέτει η χρήση της MinI στις απαιτήσεις μνήμης μίας εφαρμογής είναι ουσιαστικά αμελητέος.

6.3 Μέγεθος Παραγόμενων ιχνών

Ένα δεύτερο στοιχείο που μας ενδιαφέρει είναι να εξετάσουμε το μέγεθος των αρχείων καταγραφής που προέκυψαν από τη διαδικασία συλλογής ιχνών καθώς και το μέγεθος των παραγόμενων αρχείων στην έξοδο του δεύτερου εργαλείου. Ακόμα θα θέλουμε να δούμε το βαθμό στον οποίο η πληροφορία που παράγεται από την βιβλιοθήκη μπορεί να συμπυκνωθεί στην έξοδο του αναλυτή. Στον πίνακα 6.1 εμφανίζονται αυτά τα αποτελέσματα καθώς και το ποσοστό της συμπίεσης για την εκτέλεση όλων των μετροπρογραμμάτων. Τα στοιχεία αυτά προέκυψαν από την εκτέλεση των εφαρμογών με 256 διεργασίες, με εξαίρεση το MiniDFT στο οποίο παρουσιάζονται τα στοιχεία για εκτέλεση με 8 για τους ίδιους λόγους που έχουν εξηγηθεί και στην προηγούμενη ενότητα. Σημειώνεται ότι τα στοιχεία αυτά έχουν προκύψει για ένα συγκεκριμένο δείγμα εισόδου και δεν μπορεί να εξάγει κανείς κάπως τη μεταβολή του μεγέθους τους με βάση τη μεταβολή του μεγέθους της εισόδου.

Μετροπρόγραμμα	**minI out size (KB)**	**parser out size (KB)**	**Συμπίεση %**
CoMD	21484	5752	73.2
CoSP	11900	3236	72.8
HPCCG	8180	1460	82.15
MiniAMR	4796	696	85.48
MiniDFT	23748	4332	81.75

Πίνακας 6.1: Μέγεθος αρχείων εξόδου

Παρατηρώντας τον πίνακα προκύπτει ότι η συμπίεση είναι αρκετά υψηλή, κυμαίνεται σε όλες τις περιπτώσεις μεταξύ από 72 έως 86 %. Το χαρακτηριστικό που μπορεί να εξαχθεί όμως από τα δείγματα αυτά είναι ότι η συμπίεση εξαρτάται, όπως και ήταν αναμενόμενο, από τα χαρακτηριστικά των κλήσεων του προτύπου. Για παράδειγμα το CoMD το οποίο χρησιμοποιεί πολλές κλήσεις σημείου προς σημείο, διατηρεί περισσότερες εγγραφές μετά το δεύτερο στάδιο. Αντίθετα, τα μετροπρογράμματα τα οποία εκτελούν κατά κύριο λόγο συλλογικές κλήσεις επιτυγχάνουν μεγαλύτερο βαθμό συμπίεσης.

Κεφάλαιο 7

Επίλογος

Προτού ολοκληρωθεί η παρούσα εργασία κρίνεται σκόπιμο στο συγκεκριμένο κεφάλαιο να γίνει μια σύντομη αναφορά στο τελικό αποτέλεσμα, να συζητηθεί κατά πόσο οι στόχοι που τέθηκαν στην αρχή επιτεύχθηκαν και να προταθούν πιθανές επεκτάσεις και βελτιστοποιήσεις των εργαλειών που αναπτύχθηκαν με στόχο να δώσουν μια ώθηση στον αναγνώστη να κάνει τα πρώτα του βήματα στην περαιτέρω εξέλιξη της συγκεκριμένης δουλειάς.

7.1 Σύνοψη και συμπεράσματα

Η υλοποίηση παράλληλων εφαρμογών οι οποίες να είναι αποδοτικές είναι μια αρκετά απαιτητική διαδικασία, η οποία δυσκολεύει περισσότερο καθώς οι υπολογιστές, και συγκεκριμένα οι υπερυπολογιστές, εξελίσονται με μεγάλη ταχύτητα. Στα πλαίσια της παρούσας εργασίας, όπως αυτή αναλύθηκε σε όλα τα προηγούμενα κεφάλαια, έγινε προσπάθεια να δημιουργηθούν εργαλεία τα οποία θα διευκολύνουν αυτή τη διαδικασία προσφέροντας εποπτικά πληροφορίες οι οποίες θα διευκολύνουν τον εκάστοτε χρήστη να κατανοήσει καλύτερα συγκεκριμένα τμήματα της λειτουργίας των εφαρμογών του. Αναπτύχθηκαν λοιπόν εργαλεία τα οποία σε πρώτη φάση προσφέρουν μια εικόνα της συνολικής ανταλλαγής δεδομένων μεταξύ των διεργασιών κατά το χρόνο εκτέλεσης και σε δεύτερη φάση δίνουν μία προσομοίωση της εικόνας του συστήματος κατά το χρόνο εκτέλεσης όσον αφορά τη δέσμευση επεξεργαστικών πυρήνων. Συνδυάζοντας τα εργαλεία αυτά ένας χρήστης έχει τη δυνατότητα μετά να εξάγει μια συνολική εικόνα της κίνησης στο σύστημα και έτσι να ανιχνεύσει πιθανά προβλήματα τα οποία μπορεί να επιφέρουν καθυστερήσεις.

Για την πρώτη φάση εκτέλεσης των εργαλείων, έχουν τεθεί στόχοι που αφορούν κυρίως την ελαχιστοποίηση των απαιτήσεων εκτέλεσης της διαδικασίας συλλογής ιχνών, την επεκτασιμότητα και την πλήρη αντικατάσταση των κλήσεων του MPI. Όσον αφορά τον πρώτο στόχο θεωρείται ότι η MinI είναι πλήρως επιτυχημένη καθώς στα πειράματα που περιγράφηκαν στο 7 απέδειχθηκε ότι έχει από ελάχιστη ως μηδενική επιβάρυνση τόσο στο χρόνο εκτέλεσης όσο και στις απαιτήσεις μνήμης. Επίσης, δεν υπάρχουν και απαιτήσεις όσον αφορά το σύστημα εκτέλεσης της προσομοίωσης καθώς έχει δειχθεί ότι η βιβλιοθήκη μπορεί να λειτουργήσει και σε περιβάλλοντα που προσφέρουν πολύ λιγότερους πυρήνες από όσους εκτελείται η προσο-

μοίωση. Όσον αφορά την επεκτασιμότητα, θεωρείται ότι ο κώδικας είναι εύκολα επεκτάσιμος καθώς οι γενικότερες αρχιτεκτονικές επιλογές του έχουν μοντελοποιηθεί και μπορούν να ενσωματωθούν εύκολα σε περίπτωση ανάγκης προσθήκης νέων κλήσεων. Το ίδιο ισχύει και για τη μορφή των παραγόμενων ιχνών, στα οποία έχει προβλεφθεί συγκεκριμένο τμήμα στο οποίο μπορεί να προσθέσει ο χρήστης επιπλέον εξαγόμενη πληροφορία χωρίς να επηρεάσει τη σωστή ροή της εργαλειοθήκης. Όσον αφορά τον τελευταίο στόχο, παρόλο που η βιβλιοθήκη παρέχει την δυνατότητα να αποκοπούν οι πραγματικές κλήσεις MPI, η λειτουργία αυτή αποδείχθηκε προβληματική σε όλα τα πειράματα. Ο λόγος που συμβαίνει αυτό βέβαια δεν έχει σχέση με το ίδιο το εργαλείο όσο με το γεγονός ότι θα μπορούσε να εφαρμοστεί μόνο σε μία πολύ μικρή κατηγορία προβλημάτων, σε αυτά στα οποία ο κύκλος λειτουργίας τους είναι ανεξάρτητος από τα δεδομένα που μεταφέρονται.

Για τη δεύτερη φάση εκτέλεσης, στόχος ήταν να δημιουργηθεί ένα εργαλείο του οποίου η λειτουργία να μιμείται κατά το δυνατό έναν πραγματικό διαχειριστή συστήματος. Η τελική έκδοση του εργαλείου υλοποιεί ένα απλοποιημένο μοντέλου του διαχειριστή Mpirun. Οι απλοποιήσεις που έγιναν ήταν υποχρεωτικές για να αντισταθμίσουν τη διαφορά πληροφοριών που έχει στη διάθεσή του το εργαλείο που αναπτύχθηκε σε σχέση με το πραγματικό. Επιπλέον, ένας άλλος περιορισμός του εργαλείου είναι ότι ακολουθεί τη μορφή συγκεκριμένου διαχειριστή, συνεπώς σε περίπτωση που σε κάποιο σύστημα χρησιμοποιείται κάποιος άλλος διαχειριστής πόρων, όπως για παράδειγμα ο srun ο οποίος είναι αρκετά διαδεδομένος σε μεγάλα υπολογιστικά συστήματα, δεν θα μπορεί να λειτουργήσει με την ίδια μορφή της εισόδου. Όμως θεωρείται αρκετά βοηθητική η ικανότητά του να παράγει πολλές διαφορετικές απεικονίσεις οι οποίες μπορούν να χρησιμοποιηθούν στη πορεία για να εκτελεστούν πειράματα σχετικά με την εύρεση της καταλληλότερης απεικόνισης.

7.2 Μελλοντικές επεκτάσεις

Κλείνοντας, παρατίθονται συνοπτικά μερικά θέματα τα οποία πιστεύεται ότι θα είναι τα φυσικά επακόλουθα πρώτα βήματα σε μια πιθανή επέκταση των συγκεκριμένων εργαλείων. Θα αναφερθούμε ξεχωριστά στις πιθανές επεκτάσεις του εργαλείου συλλογής ιχνών και στον προσομοιωτή του συστήματος ανάθεσης. Προφανώς, το εργαλείο επεξεργασίας των ιχνών είναι υλοποιημένο συγκεκριμένα πάνω στη MinI, συνεπώς σε κάθε βασική αλλαγή της βιβλιοθήκης θα πρέπει να γίνονται οι απαραίτητες αλλαγές για να ενσωματωθούν οι νέες λειτουργίες ή για να επεξεργάζεται σωστά η νέα μορφή εξόδου του προηγούμενου σταδίου. Όσον αφορά τη MinI, δεν θα αναφερθούμε σε επεκτάσεις της μορφής νέων κλήσεων αποστολής μηνυμάτων σημείου προς σημείο ή συλλογικής επικοινωνίας, καθώς θεωρείται ότι έχουν καλυφθεί σε μεγάλο βαθμό και ότι η εισαγωγή μερικών ακόμα είναι τετριμμένη. Θα είχε νόημα όμως η εισαγωγή λειτουργικότητας για τη διαχείριση τριών νέων οικογενειών κλήσεων. Η πρώτη είναι οι κλήσεις μονόπλευρης (one-sided) επικοινωνίας. Ο λόγος που δεν έχουν προβλεφθεί στα πλαίσια της συγκεκριμένης εργαλειοθήκης είναι ότι η χρήση τους μέχρι και την έκδοση MPI-3 του προτύπου ήταν πολύ περιορισμένη στις εφαρμογές. Η δεύτερη είναι το σύνολο των λειτουργιών που υλοποιούν communicators σε μία ή πολλές διαστάσεις καθώς και οι κλήσεις

για το χειρισμό των Comm_Group. Αυτή η επέκταση θα ωθήσει τη βιβλιοθήκη να μπορεί να χειριστεί πλήρως έναν πολύ μεγαλύτερο αριθμό από εφαρμογές. Η τελευταία κατηγορία κλήσεων στην οποία θα μπορούσε να επεκταθεί η εφαρμογή είναι οι κλήσεις που αφορούν την αλληλεπίδραση του MPI με το σύστημα αρχείων. Αυτές οι κλήσεις θα ανοίξουν πολλούς νέους δρόμους για το συγκεκριμένο εργαλείο, αφού στο μέλλον θα έχει τη δυνατότητα να προσφέρει στα συστήματα πρόβλεψης τα απαραίτητα χαρακτηριστικά για να δημιουργήσουν μοντέλα τα οποία θα λαμβάνουν υπόψη τους τη σχέση των διεργασιών με τους πόρους του παράλληλου συστήματος αρχείων.

Ο απώτερος στόχος της παρούσας εργαλειοθήκης είναι να εκτελείται αυτόματα και σε συνδυασμό με την εκτέλεση της πρόβλεψης του χρόνου επικοινωνίας από κάποιο εξωτερικό σύστημα ανάμεσα στις χρονικές στιγμές της υποβολής του αιτήματος για την εκτέλεση μίας εφαρμογής και στην έναρξη της εκτέλεσης. Αυτό είναι εφικτό καθώς ο χρόνος εκτέλεσης των εργαλείων (πέραν της βιβλιοθήκης) και της διαδικασίας μοντελοποίησης και πρόβλεψης είναι αρκετά μικρός και η διάρκεια μεταξύ των δύο χρονικών στιγμών μπορεί να είναι αρκετά μεγάλη σε ένα πραγματικό υπολογιστικό σύστημα. Αυτός λοιπόν είναι ο τελικός στόχος του προσομοιωτή, ο οποίος θα ενεργοποιείται την ώρα της υποβολής του αιτήματος για εκτέλεση και αφού συλλέξει όλα τα απαραίτητα δεδομένα και λάβει πίσω τα αποτελέσματα της εκτέλεσης χρόνου από το σύστημα πρόβλεψης, θα είναι σε θέση είτε αυτόματα είτε με κατάλληλα διαγνωστικά μηνύματα προς το χρήστη να επηρεάσει τον διαχειριστή του συστήματος έτσι ώστε να δημιουργήσει ευνοϊκότερες συνθήκες για την εκτέλεση της εφαρμογής.

Βιβλιογραφία

[1] Bhatele, Abhinav and Titus, Andrew R and Thiagarajan, Jayaraman J and Jain, Nikhil and Gamblin, Todd and Bremer, Peer-Timo and Schulz, Martin and Kale, Laxmikant V. Identifying the culprits behind network congestion. Iν *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, παγες 113–122. IEEE, 2015.

[2] Blaise Barney. *Message Passing Interface (MPI).* `https://computing.llnl.gov/tutorials/mpi/`.

[3] Casanova, Henri and Desprez, Frédéric and Markomanolis, George S. and Suter, Frédéric. Simulation of MPI applications with time-independent traces. *Concurrency and Computation: Practice and Experience*, 27(5):1145–1168, 2014.

[4] Geimer, Markus and Wolf, Felix and Wylie, Brian JN and Ábrahám, Erika and Becker, Daniel and Mohr, Bernd. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, 2010.

[5] Hoefler, Torsten and Snir, Marc. Generic topology mapping strategies for large-scale parallel architectures. Iν *Proceedings of the international conference on Supercomputing*, παγες 75–84. ACM, 2011.

[6] Majed Al Saeed, Patrick Maier, Phil Trinder, Lilia Georgieva. A Critical Analysis of Parallel Functional Profilers. 2013. `http://www.cs.ru.nl/P.Achten/IFL2013/symposium_proceedings_IFL2013/ifl2013_submission_18.pdf`.

[7] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, 2015. `https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`.

[8] Papadopoulou, Nikela and Goumas, Georgios and Koziris, Nectarios. Predictive communication modeling for HPC applications. *Cluster Computing*, 20(3):2725–2747, 2017.

[9] Wilke J.J., Kenny J.P., Knight S., Rumley S. Compiler-Assisted Source-to-Source Skeletonization of Application Models for System Simulation. *Lecture Notes in Computer Science*, 10876, 2018. Yokota R., Weiland M., Keyes D., Trinitis C. (eds) High Performance Computing. ISC High Performance 2018.

Γλωσσάριο

Ελληνικός όρος	Αγγλικός όρος
αναλυτής	parser
διαχειριστής πόρων	resource manager
εργαλειοθήκη	toolchain
ίχνος	trace
κόμβος	node
μεταγλώττιση	compiling
μετροπρόγραμμα	benchmark
προσωμοιωτής	simulator
πυρήνας	core
συλλογή ιχνών	tracing
συστάδα	cluster
χρονοσφραγίδα	timestamp