ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

# ArrayTracer: Ενα Εργαλείο για την Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών

Τίτος Σαρειδάκης

Μεταπτυχιακή Εργασία

Ηράκλειο, Νοέμβριος 1995

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

# ArrayTracer: Ενα Εργαλείο για την Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών

Εργασία που υποβλήθηκε από τον
Τίτο Σαρειδάκη
ως μερική εκπλήρωση των απαιτήσεων
για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Τίτος Σαρειδάκης
Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Χρήστος Νικολάου, αναπληρωτής καθηγητής, Επόπτης

Ευάγγελος Μαρκάτος, επίκουρος καθηγητής, Μέλος

Δημήτριος Σερπάνος, επίκουρος καθηγητής, Μέλος

Δεκτή:

Πάνος Κωνσταντόπουλος, αναπληρωτής καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, Νοέμβριος 1995

# ArrayTracer: Ενα εργαλείο για την Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών

Τίτος Σαρειδάκης
Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

## Περίληψη

Η ολοένα αυξανόμενη πολυπλοκότητα των παράλληλων εφαρμογών και η ποικιλία των αρχιτεκτονικών και συστημάτων στα οποία εκτελούνται, σε συνδυασμό με τις συνεχώς αυξανόμενες απαιτήσεις των χρηστών, οδήγησαν στην εξέλιξη των *Εργαλείων για την Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών*. Η εξέλιξη αυτή, ανακλάται σε βελτιώσεις στους τομείς της ταχύτητας, της απόδοσης, και της ευχρηστίας αυτών των εργαλείων. Αυτό το γεγονός έχει σαν συνέπεια η κάθε τους λεπτομέρια να σχεδιάζεται με πολλή προσοχή και νέες προσεγγίσεις να υιοθετούνται για την κατασκευή ανταγωνιστικών εργαλείων.

Ο *ArrayTracer* είναι ένα εργαλείο για την *Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών*. Κύρια χαρακτηριστικά του είναι ότι λειτουργεί σε επίπεδο πηγαίου κώδικα της εφαρμογής, προκαλεί μικρή επιβάρυνση (overhead) στον χρόνο εκτέλεσης της εφαρμογής, επηρεάζει ελάχιστα την ροή ελέγχου (control flow) της παράλληλης εφαρμογής και χρησιμοποιεί ελάχιστο χώρο στον δίσκο για την αποθήκευση των ιχνών.

Η λειτουργία του *ArrayTracer* στηρίζεται στην επεξεργασία των ιχνών, η οποία λαμβάνει χώρα μετά το πέρας εκτέλεσης της εφαρμογής (post−mortem processing). Τα ίχνη αυτά έχουν συλλεχθεί είτε κατά την διάρκεια της *Στατικής Ανάλυσης* του πηγαίου κώδικα, είτε κατά την διάρκεια εκτέλεσης της εφαρμογής.

Κατά την διάρκεια της *Στατικής Ανάλυσης*, η πληροφορία που είναι διαθέσιμη σχετικά με την συμπεριφορά που θα παρουσιάσει η εφαρμογή κατά την εκτέλεση της, εξάγεται και αποθηκεύεται στα *αρχεία στατικών ιχνών*. Η υπόλοιπη πληροφορία που χρειάζεται για την ανάλυση της επίδοσης της εφαρμογής, εξαρτάται από τα δεδομένα εισόδου που θα πάρει η εφαρμογή κατά την διάρκεια εκτέλεσης της (run−time input). Για την παραγωγή αυτής της πληροφορίας, κατά την διάρκεια της *Στατικής Ανάλυσης* επιλέγονται κατάλληλα σημεία του πηγαίου κώδικα της εφαρμογής και εισάγεται *κώδικας καθοδήγησης* (instrumentation code). Κατά την εκτέλεση της εμπλουτισμένης εφαρμογή παράγονται ίχνη για τα οποία είχε εισαχθεί ο κώδικας καθοδήγησης και αποθηκεύονται στα *αρχεία δυναμικών ιχνών*.

Η υπάρχουσα υλοποίηση αναλύει την επίδοση παράλληλων εφαρμογών γραμμένων σε γλώσσα προγραμματισμού *Fortran*, οι οποίες χρησιμοποιούν την βιβλιοθήκη *PVM* για την ενδοεπικοινωνία των διεργασιών τους.

Επόπτης: Χρήστος Νικολάου,
αναπληρωτής καθηγητής,
Πανεπιστήμιο Κρήτης.

# ArrayTracer: A Parallel Performance Analysis Tool

Titos Saridakis
Master's Thesis

Computer Science Department
University of Crete

## Abstract

The ever increasing complexity of parallel application along with the variety of architectures and systems on which parallel applications run, combined with the ever increasing demands of the programmers and users, impose the perpetual evolution of *Parallel Performance Analysis Tools* (PPA tools). This evolution must be reflected to improvements in speed, efficiency, and ease of use of PPA tools. Every little detail of those tools should be designed very cautiously and new approaches should be adopted in order for competitive PPA tools to be constructed.

*ArrayTracer* is a PPA tool. Its basic characteristics are that it functions at source code level, introduces minimum overhead to the running applications, causes minimum perturbation to application's control flow and minimizes the volume of the traces. It collects traces either at the *Static Analysis phase* or during the execution of the application and processes them at post−mortem time.

During the Static Analysis, all the available information concerning the behavior that the application will show when it executes, is extracted and stored in *Static trace−files*. The rest of the information depends on the run−time user input and is captured in traces which are produced during application's execution. In order for the application to produce those traces, *instrumentation code* is inserted into application's source code during the Static Analysis phase. Trace produced at run−time are stored in *Dynamic trace−files*.

The current implementation of *ArrayTracer* operates on applications written in *Fortran* and using *PVM* for interprocess communication.

Advisor: Crhistos Nikolaou,
associate professor,
University of Crete.

# Ευχαριστίες

Η παρούσα εργασία δεν θα μπορούσε να ολοκληρωθεί χωρίς την ουσιαστική συμβολή δύο ατόμων, τα οποία και ευχαριστώ θερμά:

- Τον επιβλέποντα καθηγητή μου, κ. Χρήστο Νικολάου, ο οποίος μου έδωσε την δυνατότητα να ασχοληθώ με το θέμα αυτό και με καθοδήγησε σε όλη την διάρκεια της εργασίας μου. Ο σχεδιασμός του *ArrayTracer* βασίστηκε σε δουλειά που είχε ξεκινήσει ο κ. Χρήστος Νικολάου το καλοκαίρι του 1993.
- Τον κ. Ευάγγελο Μαρκάτο, ο οποίος με βοήθησε με τις παρατηρήσεις του τόσο στην σχεδίαση του *ArrayTracer*, όσο και στην συγγραφή αυτής της αναφοράς.

Επίσης, θα ήθελα να ευχαριστήσω τα μέλη της ομάδας των *ΠΛΕΙΑΔΩΝ* για την υποστήριξη τους σε όλη την διάρκεια της εργασίας μου. Ιδιαίτερα ευχαριστώ τους: Π. Κωσταντά–Φανουράκη, Γ. Δραμιτινό, Α. Λαμπρινίδη, Μ. Μαραζάκη και Κ. Παπαχρήστο.

Θα ήταν παράλειψη εκ μέρους μου να μην ευχαριστήσω τους γονείς μου, Αντώνη και Αργυρούλα Σαρειδάκη, για την αμέριστη υποστήριξη που μου έδωσαν καθ' όλη την διάρκεια των σπουδών μου.

Τέλος, θα ήθελα να ευχαριστήσω το Τμήμα Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης καθώς και το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Ερευνας για την υλικοτεχνική και οικονομική υποστήριξη που μου παρείχαν καθ' όλη την διάρκεια των σπουδών μου.

# Περιεχόμενα

# Κατάλογος Πινάκων

# Κατάλογος Σχημάτων

# Κεφάλαιο 1

# Εισαγωγή

Η κατασκευή παράλληλων προγραμμάτων παρακινείται κυρίως από την επιθυμία και την ανάγκη των χρηστών να βελτιώσουν τον χρόνο εκτέλεσης των εφαρμογών τους. Η επιτυχία αυτής της προσπάθειας βελτίωσης μετριέται με το μέγεθος της *επιτάχυνσης* (speedup) της εφαρμογής. Το μέγεθος της *επιτάχυνσης* ορίζεται ως ο λόγος του χρόνου εκτέλεσης της σειριακής υλοποίησης της εφαρμογής, προς τον χρόνο εκτέλεσης της παράλληλης υλοποίησης. Επίσης, ως *βαθμός παραλληλισμού* μιας παράλληλης υλοποίησης συγκεκριμένης εφαρμογής, ορίζεται ο αριθμός των τμημάτων στα οποία διαιρείται η εφαρμογή και τα οποία μπορούν να εκτελεστούν παράλληλα [7]. Το άνω φράγμα της τιμής που μπορεί να λάβει το μέγεθος της επιτάχυνσης μιας εφαρμογής όπως δηλώνει ο *νόμος του Admahl*[20] (Admahl's Law), είναι ίσο με τον βαθμό παραλληλισμού της εφαρμογής όταν:

- η συγκεκριμένη υλοποίηση της διαιρεθεί σε ίσα μεταξύ τους τμήματα [1] και
- όλα τα τμήματα στα οποία έχει διαιρεθεί, μπορούν να εκτελεστούν παράλληλα.

Οι αποκλίσεις από την βέλτιστη τιμή οφείλονται σε διάφορους λόγους. Αλλοτε τα τμήματα στα οποία έχει διαιρεθεί η εφαρμογή είναι άνισα μεταξύ τους και άλλοτε δεν είναι δυνατόν να εκτελεστούν όλα μαζί παράλληλα. Ακόμα, ο συγχρονισμός των τμημάτων και ο ανταγωνισμός τους για πρόσβαση στους πόρους του συστήματος (system resources) εμποδίζουν το μέγεθος της επιτάχυνσης να πλησιάσει το άνω φράγμα του. Επιπλέον, για τις παράλληλες υλοποιήσεις που βασίζονται στο μοντέλο ανταλλαγής μηνυμάτων (message passing), οι μεγάλες αποκλίσεις οφείλονται και στον χρόνο που καταναλώνουν οι διεργασίες στην αποστολή/παραλαβή μηνυμάτων [7]. Ομοίως, για τις παράλληλες υλοποιήσεις που βασίζονται στο μοντέλο κοινόχρηστης μνήμης (shared memory), οι μεγάλες αποκλίσεις οφείλονται και στον ανταγωνισμό των διεργασιών για προσπέλαση στην κοινόχρηστη μνήμη [7]. Εκτός από αυτούς τους παράγοντες που επηρεάζουν αρνητικά την απόδοση μιας παράλληλης εφαρμογής, υπάρχουν και όλοι οι άλλοι παράγοντες που επηρεάζουν αρνητικά και τις σειριακές υλοποιήσεις (π.χ. λανθασμένη σχεδίαση των βρόγχων επανάληψης, χειρισμοί εισόδου/εξόδου που μπορούν να παραληφθούν, κ.λ.π.)

Σε αυτήν την αναφορά παρουσιάζουμε ένα νέο εργαλείο για την ανάλυση της απόδοσης παράλληλων εφαρμογών, το οποίο το ονομάζουμε **ArrayTracer**. Ειδικότερα, επικεντρώνουμε την προσοχή μας στο επίπεδο καθοδήγησης της εφαρμογής για την παραγωγή των ιχνών της, και στην διαδικασία συλλογής των ιχνών. Αυτά συμβαίνουν στο πρώτο επίπεδο του εργαλείου που παρουσιάζουμε. Στο δεύτερο επίπεδο, που περιλαμβάνει την επεξεργασία της πληροφορίας που έχει αποθηκευτεί στα ίχνη και την ανάλυση της απόδοσης της παράλληλης

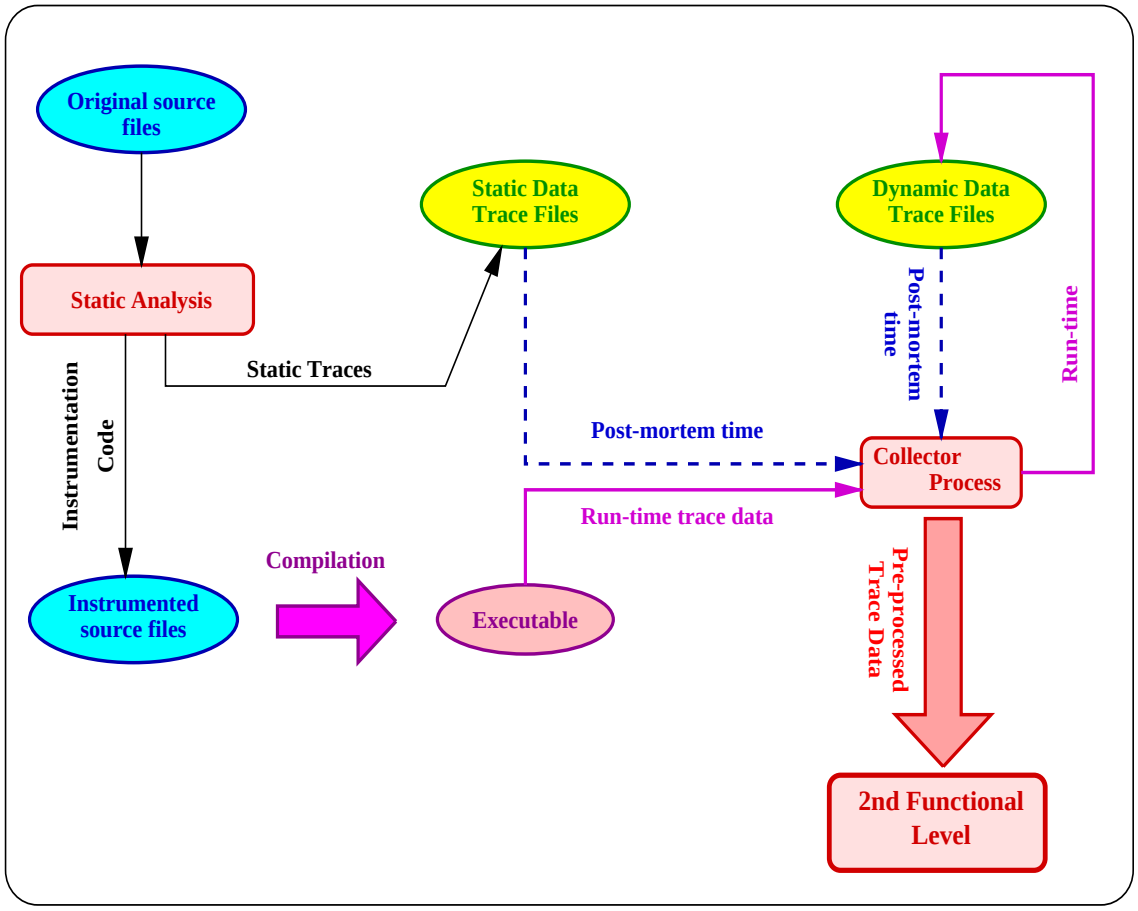
[1] Ισα τμήματα θεωρούμε εκείνα που χρειάζονται τον ίδιο χρόνο για να εκτελεστούν

εφαρμογής, αναφερόμαστε στην προεπεξεργασία που γίνεται στα συλλεγμένα ίχνη ώστε να επιταχυνθεί η διαδικασία εξαγωγής αποτελεσμάτων.

Η τεχνική που χρησιμοποιεί ο *ArrayTracer* για την ιχνοληψία μιας παράλληλης εφαρμογής, υλοποιείται σε δύο φάσεις :

1. **Φάση Στατικής Ανάλυσης.** Σε αυτήν την φάση, γίνεται συντακτική ανάλυση του πηγαίου κώδικα της εφαρμογής και εξάγεται η στατική πληροφορία που περιέχει. Ακόμα, επιλέγονται κατάλληλα σημεία για την εισαγωγή εντολών καθοδήγησης, στα οποία εισάγονται κλήσεις σε ρουτίνες της βιβλιοθήκης ιχνοληψίας του εργαλείου μας. Αυτές οι ρουτίνες έχουν σκοπό την παραγωγή των ιχνών που σχετίζονται με το περιβάλλον εκτέλεσης της εφαρμογής (run−time system) και τα οποία δεν μπορούν να παραχθούν κατά την διάρκεια της στατικής ανάλυσης του πηγαίου κώδικα.

2. **Φάση Συλλογής των Ιχνών.** Κατά την διάρκεια αυτής της φάσης, τα ίχνη που παράγονται από την εφαρμογή που εκτελείται, συλλέγονται από ειδικές διεργασίες του ArrayTracer που ονομάζονται *συλλέκτες* (collector processes). Σε αυτήν την φάση εντάσσεται επίσης και η προεπεξεργασία των ιχνών που γίνεται μετά το πέρας της εκτέλεσης της εφαρμογής, προκειμένου να επιταχυνθεί η διαδικασία επεξεργασίας τους και η εξαγωγή συμπερασμάτων που αφορούν την απόδοση της εφαρμογής.




Σχήμα 1.1: Βασικά δομικά στοιχεία της διαδικασίας καθοδήγησης και ιχνοληψίας της εφαρμογής από το *ArrayTracer* και γραφική αναπαράσταση της ροής δεδομένων ιχνοληψίας.

Στο σχήμα 1.1, ο αναγνώστης μπορεί να βρει μια γενική εικόνα της ροής του κώδικα καθοδήγησης και των ιχνών. Τα βέλη δείχνουν την χρονική ακολουθία κατασκευής των αρχείων που σχετίζονται με την διαδικασία συλλογής ιχνών. Επιστούμε την προσοχή του αναγνώστη στα δύο είδη αρχείων για αποθήκευση ιχνών που υπάρχουν, τα αρχεία για τα *στατικά ίχνη* και τα αρχεία για τα *δυναμικά ίχνη*. Οπως θα αναλύσουμε αργότερα, η συνολική πληροφορία που εξάγεται από την εφαρμογή, συνίσταται από την ένωση των στατικών και δυναμικών ιχνών. Στην φάση προεπεξεργασίας των ιχνών, γίνεται ένας συγκερασμός των περιεχομένων αυτών των αρχείων, και στην συνέχεια αποστέλονται στο δεύτερο επίπεδο λειτουργίας του εργαλείου για την περαιτέρω επεξεργασία τους.

Εκτός από αυτές τις δύο φάσεις που εντάσσονται στην διαδικασία ιχνοληψίας ενός εργαλείου για την ανάλυση της επίδοσης παράλληλων εφαρμογών, υπάρχει μία ακόμα φάση, η *φάση προετοιμασίας*. Κατά την διάρκεια της φάσης αυτής, ο χρήστης του εργαλείου καθορίζει τα *στοιχεία ιχνοληψίας*, δηλαδή τα στοιχεία της εφαρμογής που θα συμμετέχουν στην διαδικασία ιχνοληψίας.

Περιμένουμε ότι αυτή η τεχνική καθοδήγησης μιας παράλληλης εφαρμογής με εισαγωγή κώδικα ιχνοληψίας στον πηγαίο κώδικα της εφαρμογής θα φανεί ιδιαίτερα χρήσιμη σε επιστημονικές παράλληλες εφαρμογές που βασίζονται στο μοντέλο ανταλλαγής μηνυμάτων (message passing scientific application). Στις περιπτώσεις εφαρμογών που βασίζονται στο μοντέλο ανταλλαγής μηνυμάτων, ο άμεσος προσδιορισμός της επικοινωνίας δύο διεργασιών της παράλληλης εφαρμογής μέσα στον πηγαίο κώδικα της, επιτρέπει την εξαγωγή όλης της απαραίτητης πληροφορίας για την ενδοεπικοινωνία των διεργασιών και την προσπέλαση στην μνήμη τους, μέσα από την διαδικασία ιχνοληψίας που αναφέραμε.

Από την άλλη πλευρά, οι περίπτωσεις εφαρμογών που βασίζονται στον μοντέλο κοινόχρηστης μνήμης (shared memory) παρουσιάζουν μεγαλύτερη δυσκολία στην διαδικασία ιχνοληψίας τους. Ο εντοπισμός προβλημάτων συγχρονισμού (synchronization problems) των ινών (threads) καθώς και καταστάσεων ενεργής αναμονής (busy waiting) δεν είναι τόσο άμεσος όσο στις παράλληλες εφαρμογές που ανταλλάσουν μηνύματα για την επικοινωνία τους.

Μερικά από τα χαρακτηριστικά στοιχεία του σχεδιασμού του *ArrayTracer* είναι:

- Η κατασκευή του εργαλείου κατά τέτοιο τρόπο ώστε να εισάγει όσο το δυνατόν λιγότερη επιβάρυνση στον χρόνο εκτέλεσης της εφαρμογής (run time overhead). Αυτό επιτυγχάνεται με την χρήση μιας τεχνικής *επιλεκτικής ιχνοληψίας* (selective tracing) που λειτουργεί σε επίπεδο πηγαίου κώδικα της εφαρμογής.

- Το εργαλείο έχει κατασκευαστεί για να υποστηρίζει παράλληλες εφαρμογές που στηρίζονται στο μοντέλο ανταλλαγής μηνυμάτων (message passing). Ειδικότερα, το εργαλείο μας υποστηρίζει εφαρμογές γραμμένες σε *Fortran* και *PVM*.

- Συνδυάζει την ευχρηστία (μικρό σύνολο εντολών ιχνηλασίας που πλησιάζουν το συντακτικό της Αγγλικής γλώσσας) με την μεγάλη ικανότητα έκφρασης και προσδιορισμού των στοιχείων ιχνηλασίας που καθορίζει ο χρήστης.

Η διάρθρωση αυτής της αναφοράς είναι ως εξής· στο 2ο κεφάλαιο παρουσιάζεται σε συντομία η εργασία που έχει γίνει στον χώρο των *Εργαλείων για την Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών*. Στα κεφάλαια 3, 4 και 5 παρουσιάζονται τα βασικά τμήματα του λειτουργικού επιπέδου ιχνοληψίας του *ArrayTracer*. Ακολουθεί το 6ο κεφάλαιο που συνοψίζει τα συμπεράσματα στα οποία καταλήξαμε από πειραματισμούς στην χρήση του εργαλείου, μετρήσεις της επιβάρυνσης στον χρόνο εκτέλεσης της εφαρμογής που προκαλεί η χρήση του εργαλείου μας. Στην συνέχεια υπάρχει το 7ο κεφάλαιο που περιέχει την κατάσταση της υπάρχουσας υλοποίησης και τις κατευθύνσεις της μελλοντικής εργασίας πάνω στο *ArrayTracer*. Ακολουθούν τέσσερα παραρτήματα με τεχνικές λεπτομέρειες πάνω

στην κατασκευή του *ArrayTracer*. Αυτά αφορούν το σύνολο των εντολών ιχνοληψίας που είναι διαθέσιμο στον χρήστη (παράρτημα Α), την δομή του *πίνακα ιχνοληψίας* (παράρτημα Β), την κατασκευή του κοινόχρηστου ενταμιευτή που χρησιμεύει στην συλλογή των ιχνών (παράρτημα Γ) και την αντιμετώπιση ιδιαιτεροτήτων που παρουσίαζαν οι περιπτώσεις της ιχνοληψίας μέσα στον κορμό υπορουτινών (subroutine body) καθώς και της ιχνοληψίας μεταβλητών που συμμετέχουν σε δήλωση *EQUIVALENCE* (παράρτημα Δ).

Στις ενότητες που ακολουθούν, θα χρησιμοποιθεί καταχρηστικά το ρήμα *ιχνοληπτώ* καθώς και η μετοχή *ιχνοληπτούμενος/η*, σαν αναφορά στην διαδικασία παραγωγής και συλλογής των ιχνών μιας εφαρμογής.

# Κεφάλαιο 2

# Γενική Επισκόπηση

Σε αυτό το κείμενο θα αναφερθούμε πολλές φορές στον όρο *διαδικασία ιχνοληψίας*, γι' αυτό θεωρούμε σκόπιμο να παρουσιάσουμε τι ακριβώς εννοούμε με τον όρο αυτό.

Ενα παράλληλο πρόγραμμα αποτελείται από πολλές διεργασίες, εκ των οποίων μερικές εκτελούνται ταυτόχρονα. Κατά την διάρκεια της εκτέλεσής της, μία διεργασία πραγματοποιεί κάποιες λειτουργίες πάνω στο χώρο μνήμης που της έχει παραχωρηθεί καθώς επίσης και τις βασικές λειτουργίες που υπαγορεύει ο κώδικας της (conditional/unconditional jumps, loops, function/subroutine calls, κ.λ.π). Ακόμα, μπορεί να επικοινωνεί με τις άλλες διεργασίες που εκτελούνται την ίδια χρονική στιγμή στο σύστημα.

Στην *πορεία* της εκτέλεσης της λοιπόν, μια παράλληλη εφαρμογή αφήνει τα *ίχνη* της, τα οποία δεν είναι άλλα από τα αποτελέσματα των λειτουργιών που μόλις περιγράψαμε. Η διαδικασία παρακολούθησης μίας εφαρμογής και συγκέντρωσης των *ιχνών* της με σκοπό την χρησιμοποίηση τους για τον έλεγχο και την ανάλυση της επίδοσής της, ονομάζεται *διαδικασία ιχνοληψίας*.

Η διαδικασία ιχνοληψίας μιας εφαρμογής γενικά, δεν αποτελεί νέο πεδίο έρευνας στον χώρο της Πληροφορικής. Υπάρχουν αρκετοί τομείς της Πληροφορικής, οι οποίοι βασίζουν την λειτουργία τους στην χρησιμοποίηση ιχνών διαφόρων εφαρμογών. Σε αυτούς συμπεριλαμβάνονται οι τομείς διάγνωσης προβλημάτων ορθότητας εφαρμογών (correctness debugging) [42, 10, 1], της διάγνωσης προβλημάτων επίδοσης (performance debugging) [22, 23, 40, 26, 39, 44, 37, 47, 16, 33], των ιχνοκατευθυνόμενων προσομοιώσεων (trace driven simulations) [5, 6, 12], καθώς επίσης και αυτός της αιτιοκρατικής επανεκτέλεσης (deterministic re−execution) παράλληλου προγράμματος.

Ακολουθεί μια συνοπτική παρουσίαση των μεθόδων ιχνοληψίας[1] και μια σύντομη αναφορά στην δουλειά που έχει γίνει σε αυτόν τον τομέα και στην ιστορική εξέλιξη των εργαλείων για την Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών.

## 2.1 Μέθοδοι ιχνοληψίας

Υπάρχουν διάφορες τεχνικές που χρησιμοποιούνται από τις διαδικασίες ιχνοληψίας. Οι τεχνικές αυτές βασίζονται είτε στο υλικό ενός συστήματος (system's hardware) είτε στο λογισμικό (system's software) είτε και στα δύο. Το κριτήριο αξιολόγησής τους είναι, τις περισότερες φορές, το μέγεθος της *διάβρωσης του προγράμματος* (program distortion) που εισάγουν στην εφαρμογή την οποία ιχνοληπτούν. Ενα ευρύτατα χρησιμοποιούμενο μέτρο της διάβρωσης που προκαλεί η διαδικασία ιχνοληψίας πάνω σε μια εφαρμογή, είναι η *διαστολή* (dilation) που παρατηρείται στον χρόνο εκτέλεσης της εφαρμογής. Το μέγεθος της διαστολής

[1] Η ενότητα αυτή έχει γραφεί από τον κ.κ. Ευάγγελο Μαρκάτο.

ορίζεται ως ο λόγος του χρόνου ολοκλήρωσης της εκτέλεσης της εφαρμογής όταν υπόκειται στην διαδικασία της ιχνοληψίας, προς τον χρόνο ολοκλήρωσης της εκτέλεσης της εφαρμογής όταν αυτή δεν ιχνοληπτείται.

Παράγοντες διαστολής από 2 μέχρι 3 θεωρούνται άριστοι, ενώ παράγοντες διαστολής κοντά στο 1000 θεωρούνται κάκιστοι. Οι διάφορες μέθοδοι ιχνοληψίας επιδιώκουν χαμηλές τιμές διαστολής γιατί αυτό τους επιτρέπει την συλλογή μεγάλων ποσοτήτων ιχνών μέσα σε μικρό χρόνο, χωρίς να προκαλούν μεγάλη διάβρωση στην εφαρμογή που ιχνοληπτούν. Οι μέθοδοι ιχνοληψίας ταξινομούνται σε τέσσερεις κατηγορίες:

**Μέθοδοι βασισμένες στο υλικό (hardware−based).** Οι μέθοδοι που ανήκουν σε αυτήν την κατηγορία χρησιμοποιούν ένα εργαλείο παρακολούθησης υλικού (hardware monitor) το οποίο είτε καταγράφει όλες τις αιτήσεις για την κατοχή του διαδρόμου διευθύνσεων (address bus) ενός επεξεργαστή [30, 31], είτε καταγράφει τις προσπελάσεις σε απομακρυσμένη μνήμη (remote memory accesses) [28]. Το βασικό πλεονέκτημα αυτής της ομάδας μεθόδων ιχνοληψίας είναι οι πολύ χαμηλές τιμές διαστολής που επιτυγχάνουν, εξαιτίας του ότι η εκτέλεση της εφαρμογής παραμένει ιδεατά αδιάληπτη (virtually uninterrupted) από το εργαλείο παρακολούθησης υλικού. Το μεγάλο μειονέκτημα αυτής της προσέγγισης είναι το υψηλό κόστος της και η περιορισμένη δυνατότητα εφαρμογής της (applicability) στα σύγχρονα συστήματα τα οποία χρησιμοποιούν κρυφές μνήμες επί του κυκλώματος του επεξεργαστή (on−chip caches) καθώς και κατανεμημένη μνήμη. Επιπλέον, τα εργαλεία για την παρακολούθηση υλικού στις περισότερες περιπτώσεις μπορούν να συγκεντρώσουν ίχνη φυσικών διευθύνσεων μνήμης. Ομως, αυτό που ουσιαστικά περιμένει ο χρήστης από την διαδικασία ιχνοληψίας, είναι να συγκεντρώσει πληροφορίες για το ποιά είναι η βέλτιστη αντιστοίχιση διευθύνσεων της ιδεατής μνήμης σε θέσεις της φυσικής μνήμης.

**Μέθοδοι βασισμένες σε διακοπές (interrupt−based).** Αυτές οι μέθοδοι προκαλούν μια διακοπή του προγράμματος που εκτελείται σε κάθε προσπάθεια προσπέλασης στην μνήμη [4, 40, 39]. Κατά την διάρκεια της διακοπής, καταγράφεται η θέση μνήμης στην οποία επιχειρείται προσπέλαση και αμέσως μετά ο έλεγχος του συστήματος επανέρχεται στην εφαρμογή. Οι μέθοδοι αυτής της ομάδας έχουν απλή υλοποίηση εάν ο υλοποιητής έχει προσπέλαση στον πηγαίο κώδικα (source code) του λειτουργικού συστήματος. Το μειονέκτημα τους είναι ότι εισάγουν μεγάλη διαστολή στην εφαρμογή που ιχνοληπτείται. Το λειτουργικό σύστημα *Unix* παρέχει μια ειδική κλήση συστήματος (system call) η οποία ονομάζεται *ptrace( )* και επιτρέπει σε μια διεργασία να συγκεντρώσει τα ίχνη μιας άλλης διεργασίας. Ο μοναδικός περιορισμός είναι το γεγονός ότι η διεργασία που εκτελεί την διαδικασία της ιχνοληψίας, πρέπει να είναι πατέρας (father−process) της διεργασίας που ιχνοληπτείται. Η κλήση συστήματος *ptrace( )* χρησιμοποιείται ευρύτατα από τα εργαλεία διάγνωσης προβλημάτων (debuggers), αλλά εισάγει μεγάλη καθυστέρηση και γι' αυτό δεν αποτελεί την καλύτερη επιλογή για την ιχνοληψία προσπελάσεων στην μνήμη μιας διεργασίας.

**Μέθοδοι βασισμένες σε μικροκώδικα (microcode−based).** Αυτές οι μέθοδοι μπορούν να εφαρμοστούν μόνο σε συστήματα που έχουν επεξεργαστές βασισμένους σε μικροκώδικα [45]. Τα συστήματα αυτά μπορούν να ιχνοληπτήσουν την εκτέλεση μιας εφαρμογής χρησιμοποιώντας μικροκώδικα ο οποίος έχει τροποποιηθεί κατάλληλα. Παρόλο που η ομάδα αυτή περιλαμβάνει μεθόδους που εισάγουν μικρή διαστολή στις εφαρμογές που ιχνοληπτούνται, η χρησιμοποίηση τους είναι πολύ περιορισμένη στα σύχρονα συστήματα μειωμένου συνόλου εντολών (RISC processors), τα οποία δεν έχουν καθόλου μικροκώδικα.

**Μέθοδοι βασισμένες στην καθοδήγηση του προγράμματος (instrumented program−based).** Ο τρόπος λειτουργίας των μεθόδων αυτής της κατηγορίας είναι ο ακόλουθος. Ειδικές εντολές ιχνοληψίας εισάγονται μέσα στον κώδικα της εφαρμογής. Στην συνέχεια ο εμπλουτισμένος με εντολές ιχνοληψίας κώδικας, μεταφράζεται σε γλώσσα μηχανής (assembly) για το συ− γκεκριμένο σύστημα στο οποίο πρόκειται να εκτελεστεί η εφαρμογή. Κατά την διαδικασία εισαγωγής εντολών ιχνοληψίας στον κώδικα της εφαρμογής, ο κώδικας διαιρείται σε τμήματα εντολών τα οποία δεν περιέχουν εντολές διακλάδωσης. Ετσι, αναλύονται στατικά όλες οι αναφορές σε θέσεις μνήμης που περιέχει κάθε τέτοιο τμήμα του κώδικα. Μερικές αναφορές στην μνήμη όμως, μπορούν να διαπιστωθούν μόνο κατά την διάρκεια εκτέλεσης. Για αυτές τις περιπτώσεις εισάγονται στον κώδικα της εφαρμογής ειδικές εντολές ιχνοληψίας οι οποίες εντοπίζουν αυτές τις αναφορές που εξαρτώνται από το περιβάλλον εκτέλεσης (run−time system). Η εισαγωγή κώδικα ιχνοληψίας στην εφαρμογή μπορεί να γίνει πριν την μετάφραση της εφαρμογής (compilation) [49, 43, 16], κατά την διάρκεια της μετάφρασης [5, 6, 12], κατά την διάρκεια της σύνδεσης (linking) [14, 13], ή τέλος κατά την διάρκεια εκτέλεσης της εφαρμογής [29].

Ο πίνακας 2.1 συνοψίζει τις ιδιότητες των τεσσάρων μεθοδολογιών ιχνοληψίας.

| **Κατηγορίες ιχνοληψίας** | **Διαστολή** |
|---|---|
| Βασισμένες στο υλικό | 1 |
| Βασισμένες σε διακοπές | 100 |
| Βασισμένες σε μικροκώδικα | 10 |
| Βασισμένες σε καθοδήγηση | 3−10 |

Πίνακας 2.1: Χαρακτηριστικές τιμές του μεγέθους της *διαστολής* που εισάγουν οι διάφορες τεχνικές ιχνοληψίας.

## 2.2 Σχετική δουλειά

Τα σημερινά εργαλεία για την ανάλυση της απόδοσης παράλληλων εφαρμογών αποτελούνται από τρία επίπεδα λειτουργίας [24, 27]:

1. Το επίπεδο της ιχνοληψίας.
2. Το επίπεδο επεξεργασίας της πληροφορίας που περιέχουν τα ίχνη.
3. Το επίπεδο γραφικής παρουσίασης των αποτελεσμάτων (results' visualization).

Οπως αναφέρθηκε προηγουμένως, η τεχνική που έχει χρησιμοποιηθεί ευρύτατα για την συλλογή των ιχνών μιας εφαρμογής, είναι αυτή που στηρίζεται στις διακοπές της εκτέλεσης της εφαρμογής. Για την ακρίβεια, η κλήση συστήματος *ptrace( )* του λειτουργικού συστήματος *Unix* έχει χρησιμοποιηθεί σε πολλά εργαλεία της κατηγορίας διάγνωσης προβλημάτων ορθό− τητας (correctness debuggers) αλλά και της κατηγορίας διάγνωσης προβλημάτων παράλληλης επίδοσης (parallel performance debuggers) [44, 10, 1]. Επίσης, έχουν εμφανιστεί επεκτάσεις αυτής της κλήσης συστήματος, οι οποίες πλησιάζουν περισσότερο τις ανάγκες ιχνοληψίας μιας παράλληλης εφαρμογής [22, 40, 26, 39, 44].

Εκτός από την χρήση του *ptrace( )*, στον χώρο των μεθόδων ιχνοληψίας που στηρίζονται στις διακοπές της εκτέλεσης της εφαρμογής, πολλή δουλειά έχει γίνει και προς την κατεύ− θυνση των *κατ' επιλογήν διακοπών* (selective interrupts) [23, 21, 26, 34, 11, 32, 41], η οποία έχει το ακόλουθο σκεπτικό. Αντί να διακόπτεται η διεργασία που ιχνοληπτείται σε κάθε αναφορά

στην μνήμη της, επινοήθηκαν κάποιοι αλγόριθμοι οι οποίοι καθορίζουν τα διαστήματα που η εκτέλεση της εφαρμογής μπορεί να συνεχίζεται αδιάκοπα χωρίς αυτό να μεταφράζεται σε μείωση της ποιότητας της πληροφορίας που θα περιέχουν τα ίχνη που θα συλλεγούν. Με αυτόν τον τρόπο, επιτυγχάνεται σημαντική μείωση στην καθυστέρηση που εισάγεται στην εκτέλεση της εφαρμογής. Παρόλα αυτά όμως, η διαστολή που επιβάλλει η τεχνική των κατ' επιλογήν διακοπών στην εφαρμογή του χρήστη, παραμένει τάξεις μεγέθους μεγαλύτερη από αυτή που επιβάλλουν οι μέθοδοι που στηρίζονται στην καθοδήγηση της εφαρμογής.

Στην περίπτωση των correctness debuggers, η χρήση της κλήσης συστήματος *ptrace()* είναι απαραίτητη, γιατί επιτρέπει την προσπέλαση στο χώρο μνήμης της εφαρμογής που ιχνοληπτείται. Συνεπώς, παρέχει δυνατότητες πειραματισμού πάνω στην συμπεριφορά της εφαρμογής με αλλαγές που μπορεί να κάνει ο χρήστης στις τιμές των μεταβλητών της. Στις περιπτώσεις που ο χρήστης περιμένει από το εργαλείο να του παρέχει δυνατότητες επέμβασης στο χώρο της μνήμης της εφαρμογής του, είναι αναγκασμένος να υποστεί την διαστολή στον χρόνο εκτέλεσης της εφαρμογής του που εισάγει η χρήση της *ptrace()*.

Στην περίπτωση των εργαλείων τα οποία μετρούν την επίδοση παράλληλων εφαρμογών, ή εκτελούν ιχνοκατευθυνόμενη προσομοίωση ή αιτιοκρατική επανεκτέλεση των παράλληλων εφαρμογών, το παραπάνω πλεονέκτημα που προσφέρει η χρήση της *ptrace()* δεν έχει άμεσο ενδιαφέρον. Εκτός από την μεγάλη διαστολή που εισάγει στον χρόνο εκτέλεσης της εφαρμογής, έχει δύο ακόμα σημαντικά μειονεκτήματα :

1. Η καθυστέρηση που προκαλεί η χρήση της *ptrace()* στην εφαρμογή, μπορεί να προκαλέσει ουσιαστική αλλαγή στην ροή του ελέγχου της εκτέλεσης της παράλληλης εφαρμογής [24] (π.χ. σε συστήματα που χρησιμοποιούν χρονο−όρια (time−outs) για να ελέγχουν ότι όλες οι διεργασίες τους λειτουργούν, η καθυστέρηση που εισάγει η χρήση της *ptrace()* μπορεί να οδηγήσει σε λανθασμένο συμπέρασμα δισλειτουργίας μίας διεργασίας.)

2. Δημιουργεί την ανάγκη κατασκευής πίνακα αντιστοιχίσεων μεταξύ των συμβάντων χαμηλού επιπέδου (low−level events) που έχει καταγράψει μέσα στα ίχνη που έχει συγκεντρώσει, και των υψηλού επιπέδου συμβάντων της εφαρμογής (high−level program concepts), τα οποία μπορεί να αντιληφθεί στην πράξη ο χρήστης [25].

Από την άλλη πλευρά, οι τεχνικές ιχνοληψίας που βασίζονται στην καθοδήγηση της εφαρμογής, εισάγουν μικρότερη διαστολή στον χρόνο εκτέλεσής της. Ετσι, η πιθανότητα να προκαλέσει επιζήμιες επιδράσεις στην ροή ελέγχου της παράλληλης εφαρμογής, ελαχιστοποιούνται. Επιπλέον, η εισαγωγή εντολών ιχνοληψίας σε επίπεδο πηγαίου κώδικα οδηγεί στην συλλογή ιχνών που περιέχουν πληροφορία η οποία αναφέρεται απευθείας στα υψηλού επιπέδου συμβάντα του κώδικα της εφαρμογής. Αυτό με την σειρά του, καθιστά άχρηστη την ύπαρξη των πινάκων αντιστοιχίσεων γεγονότων χαμηλού επιπέδου σε γεγονότα υψηλού επιπέδου.

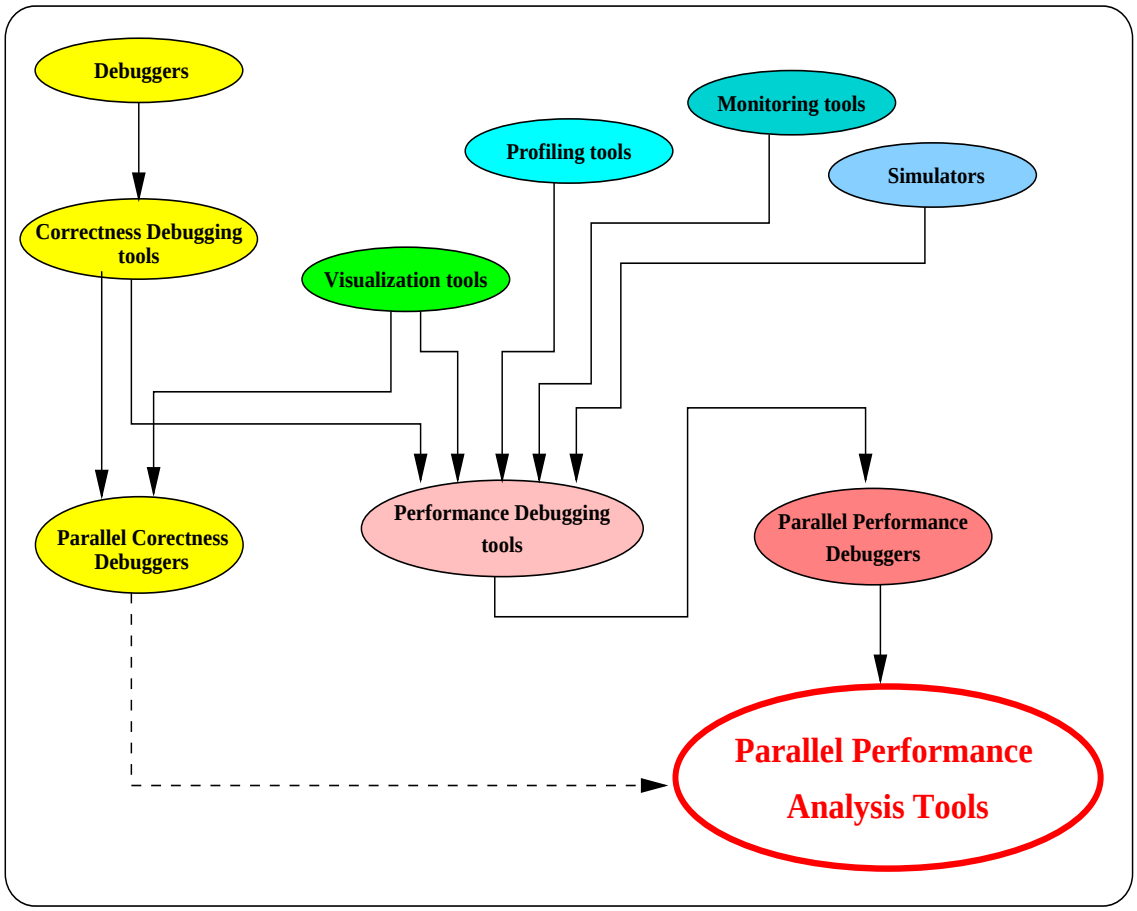
Το εργαλείο που παρουσιάζουμε σε αυτήν την αναφορά, χρησιμοποιεί μια τεχνική ιχνοληψίας που βασίζεται αποκλειστικά στην καθοδήγηση της εφαρμογής με εισαγωγή εντολών ιχνοληψίας στον πηγαίο κώδικα της. Με αυτόν τον τρόπο αντιμετωπίζουμε επιτυχώς τα προβλήματα ελαχιστοποίησης της διαστολής του χρόνου εκτέλεσης της ιχνοληπτούμενης εφαρμογής, καθώς και το ζήτημα του χώρου στον δίσκο που καταλαμβάνουν τα ίχνη που έχουν συλλεχθεί.

Τα επίπεδα επεξεργασίας των ιχνών και παρουσίασης των αποτελεσμάτων της επεξεργασίας είναι μερικές φορές στενά συνδεδεμένα μεταξύ τους [17, 2, 15]. Αλλες φορές το επίπεδο παρουσίασης των αποτελεσμάτων υλοποιείται ανεξάρτητα και δίνεται ιδιαίτερη έμφαση στην αντιμετώπιση προβλημάτων κλιμάκωσης της οπτικοποίησης των αποτελεσμάτων [18, 3]. Σε

αυτήν την αναφορά θα ασχοληθούμε με την παρουσίαση του πρώτου επιπέδου λειτουργίας του *ArrayTracer*.

## 2.3 Ιστορική αναδρομή

Οι συνεχώς αυξανόμενες απαιτήσεις των χρηστών από τις παράλληλες εφαρμογές σε ζητήματα ταχύτητας εκτέλεσης, οδήγησαν στην ανάγκη κατασκευής εφαρμογών υψηλών επιδόσεων. Η ανάπτυξη τέτοιων προγραμμάτων, με την σειρά, της δημιούργησε την ανάγκη εργαλείων που θα έδιναν στον προγραμματιστή πλήρη έλεγχο του προϊόντος του. Τα παραδοσιακά εργελεία που χρησιμοποιούσαν για την ανάπτυξη των σειριακών εφαρμογών, δεν παρείχαν επαρκείς λειτουργίες ώστε να στηρίξουν την ανάπτυξη των παράλληλων εφαρμογών υψηλών επιδόδεων. Ετσι, προκλήθηκε μια διαδικασία εξέλιξης των υπαρχόντων εργαλείων, προσανατολιζόμενη στις απαιτήσεις των παράλληλων εφαρμογών. Επίσης, νέες τεχνικές αναπτύχθηκαν και εργαλεία που βασίζονται σε αυτές παρουσιάστηκαν για να βοηθήσουν τον προγραμματιστή στην ανάλυση της επίδοσης των παράλληλων εφαρμογών και στον εντοπισμό των προβλημάτων τους. Τα εργαλεία αυτά δημιούργησαν την κατηγορία των εργαλείων για την *Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών* (Parallel Performance Analysis).




Σχήμα 2.1: Η ιστορία εξέλιξης των Εργαλείων για την Ανάλυση Επίδοσης Παράλληλων Εφαρμογών.

Μια υποκατηγορία των εργαλείων αυτών, ως επί το πλείστον αυτά που κατασκευάστηκαν παλιότερα, αποτελούν επεκτάσεις πάνω σε υπάρχοντα εργαλεία για την ανάλυση της επίδοση

σειριακών εφαρμογών (debuggers, profilers, κ.λ.π.). Αυτά τα εργαλεία προσφέρουν χαμηλού επιπέδου υπηρεσίες, η χρήση τους σε μια εφαρμογή αυξάνει δραματικά τον χρόνο εκτέλεσής της και πολλές φορές προκαλούν και ανεπιθύμητες παρενέργειες στην ροή εκτέλεσης της εφαρμογής. Επίσης, αντιμετωπίζουν πολλές δυσκολίες στον τομέα της κλιμάκωσης με το μέγεθος (αριθμό διεργασιών) της παράλληλης εφαρμογής, της οποίας αναλύουν την απόδοση.

Μια άλλη υποκατηγορία των εργαλείων αυτών, περιλαμβάνει εκείνα που κατασκευάστηκαν μερικώς από εργαλεία για την διάγνωση προβλημάτων (debuggers) ενώ σχεδιάστηκε από την αρχή η διεπαφή οπτικοποίησης προς τον χρήστη (visualization user interface). Το επίπεδο των υπηρεσιών που προσφέρουν αυτά τα εργαλεία είναι αρκετά υψηλότερο από αυτό της προηγούμενης υποκατηγορίας (γράφοι, γραφικές παραστάσεις, ιστογράμματα, κ.λ.π), αφήνοντας όμως αρκετά περιθώρια για αναβάθμισή του. Τα περισσότερα από αυτά τα εργαλεία λύνουν το πρόβλημα της κλιμάκωσης σε ένα βαθμό. Εξακολουθούν όμως να αντιμετωπίζουν το έντονο πρόβλημα της αύξησης του χρόνου εκτέλεσης της εφαρμογής της οποίας αναλύουν την απόδοση και όλων των συνεπακόλουθων παρενεργειών.

Τέλος, υπάρχει και η υποκατηγορία των εργαλείων *PPA* που συμπεριλαμβάνει εκείνα τα εργαλεία που έχουν κατασκευαστεί εκ νέου. Αυτά τα εργαλεία προσφέρουν πολύ υψηλού επιπέδου υπηρεσίες όπως κινηματογραφικές παραστάσεις (animation) επικοινωνίας των διεργασιών και αυτόματο εντοπισμό σημείων της εφαρμογής που παρουσιάζουν ιδιαίτερα υψηλό φόρτο, ενώ συγχρόνως παρουσιάζουν απροβλημάτιστη κλιμάκωση σε εφαρμογές μαζικού παραλληλισμού (massive parallelism applications).

# Κεφάλαιο 3

# Διεπαφή Εντολών

Ενας από τους πρωτεύοντες στόχους μας στην κατασκευή του *ArrayTracer* είναι η φιλικότητα προς τον χρήστη. Προς αυτήν την κατεύθυνση, πολύ σημαντικό ρόλο παίζει το τμήμα του εργαλείου με το οποίο ο χρήστης έρχεται σε άμεση επαφή. Αναφερόμαστε στην διεπαφή του *ArrayTracer* με τον χρήστη η οποία χωρίζεται σε δύο μέρη, την διεπαφή εισόδου και την διεπαφή εξόδου. Η διεπαφή εισόδου αποτελείται από το τμήμα του εργαλείου μέσω του οποίου ο χρήστης καθορίζει τις παραμέτρους της διαδικασίας ιχνοληψίας (tracing). Η διεπαφή εξόδου περιλαμβάνει το τμήμα του εργαλείου που παρουσιάζει στον χρήστη τα αποτελέσματα της διαδικασίας ιχνοληψίας. Η συνεισφορά της διεπαφής εξόδου στην φιλικότητα του εργαλείου προς τον χρήστη είναι πολύ σημαντική [1, 35, 19, 38] αλλά δεν θα μας απασχολήσει σε αυτήν την εργασία. Στην συνέχεια αυτής της ενότητας θα ασχοληθούμε με την διεπαφή εισόδου.

Η προσφορά της διεπαφής εισόδου στην συνολική φιλικότητα του εργαλείου προς τον χρήστη μπορεί να εκτιμηθεί με βάση τις ακόλουθες παραμέτρους :

- την ευκολία στον προγραμματισμό της διαδικασίας ιχνοληψίας της εφαρμογής, και
- το χρονικό διάστημα που απαιτείται για να εξοικειωθεί ο χρήστης με το εργαλείο (learning curve).

Οπως έχουμε αναφέρει σε προηγούμενη ενότητα, το *ArrayTracer* στηρίζει την λειτουργία του στην εισαγωγή εντολών ιχνοληψίας μέσα στον πηγαίο κώδικα της εφαρμογής. Οι εντολές ιχνοληψίας είναι στην πράξη κλήσεις σε ρουτίνες της βιβλιοθήκης του *ArrayTracer* οι οποίες παράγουν τις εγγραφές ιχνοληψίας (trace records). Η διαδικασία εισαγωγής των εντολών ιχνοληψίας στον πηγαίο κώδικα της εφαρμογής είναι μια πολύ επίπονη εργασία για τον χρήστη ο οποίος θα πρέπει:

- να μάθει τις παραμέτρους των ρουτινών ιχνοληψίας,
- να επιλέξει τα κατάλληλα σημεία του κώδικα που πρέπει να εισαχθούν οι κλήσεις στις ρουτίνες ιχνοληψίας,
- να επέμβει στο αρχείο του πηγαίου κώδικα για να το τροποποιήσει εισάγοντας τις κλήσεις στις ρουτίνες ιχνοληψίας,
- να προσθέσει στο αρχείο πηγαίου κώδικα της εφαρμογής και τον απαραίτητο κώδικα για την αρχικοποίηση του μηχανισμού ιχνοληψίας.

Το *ArrayTracer* αναλαμβάνει να κάνει αυτόματα αυτήν την εργασία. Ο χρήστης πρέπει απλώς να καθορίσει τις *παραμέτρους ιχνοληψίας*. Ο όρος *παράμετροι ιχνοληψίας* περιλαμβάνει

όλα τα στοιχεία της εφαρμογής που ο χρήστης επιθυμεί να ιχνοληπτήσει (προσπελάσεις θέσεων μνήμης, δομικά στοιχεία πηγαίου κώδικα και στοιχεία ενδοεπικοινωνίας διεργασιών). Για τον καθορισμό των παραμέτρων ιχνοληψίας έχει κατασκευαστεί η διεπαφή εισόδου (input interface). Η διεπαφή εισόδου είναι στην πράξη ένας διερμηνέας (interpreter) ο οποίος παρέχει στον χρήστη ένα σύνολο εντολών. Με αυτό το σύνολο εντολών ο χρήστης μπορεί να καθορίσει τις παραμέτρους ιχνοληψίας, τις οποίες αποθηκεύει το *ArrayTracer* σε μια εσωτερική δομή δεδομένων, τον *πίνακα ιχνοληψίας*. Στην συνέχεια, το εργαλείο αναλαμβάνει να επιλέξει τα σημεία εισαγωγής κώδικα καθοδήγησης και να πραγματοποιήσει την εισαγωγή αυτού του κώδικα. Ετσι ο χρήστης δεν έρχεται σε άμεση επαφή με τον πηγαίο κώδικα της εφαρμογής και φυσικά μειώνεται ο αριθμός των αποφάσεων που πρέπει να πάρει και η προσπάθεια που πρέπει να καταβάλει προκειμένου να αρχικοποιήσει και να ενεργοποιήσει την διαδικασία ιχνοληψίας.

## 3.1 Σύνολο εντολών ιχνοληψίας

Το γεγονός ότι το *ArrayTracer* αναλαμβάνει να κάνει την εργασία της εισόδου των εντολών ιχνοληψίας μέσα στον πηγαίο κώδικα της εφαρμογής δεν αρκεί από μόνο του για να θεωρηθεί το εργαλείο φιλικό ως προς την χρήση του. Πρέπει ακόμα, να είναι εύχρηστο το σύνολο των εντολών που προσφέρει στον χρηστη για τον καθορισμό των παραμέτρων ιχνοληψίας. Πιο συγκεκριμένα, το σύνολο αυτό θα πρέπει :

- να μην έχει μεγάλο πλήθος εντολών,
- να περιέχει εντολές των οποίων η σύνταξη να είναι τέτοια ώστε ο χρήστης να εξοικειωθεί σύντομα μαζί τους, και
- να δίνει όσο το δυνατόν μεγαλύτερη ευχέρεια στον καθορισμό των παραμέτρων, ώστε αυτές να προσαρμόζονται ακριβώς στις ανάγκες του χρήστη.

Το *ArrayTracer* πληρεί τις παραπάνω προϋποθέσεις με το να προσφέρει ένα μικρό αλλά πανίσχυρο σύνολο από εντολές ιχνοληψίας των οποίων η σύνταξη είναι πολύ κοντά σε αυτήν της Αγγλικής γλώσσας. Με αυτό το σύνολο εντολών, ο χρήστης μπορεί να καθορίσει την διαδικασία ιχνοληψίας με όση λεπτομέρεια επιθυμεί (π.χ. μπορεί να ζητήσει από την ιχνοληψία όλων των μεταβλητών μια εφαρμογής μέσα σε ολόκληρο τον κώδικα της, μέχρι την ιχνοληψία μιας συγκεκριμένης μεταβλητής σε μία συγκεκριμένη γραμμή του πηγαίου κώδικα της εφαρμογής).

Επίσης ο χρήστης μπορεί να καθορίσει τα στοιχεία μιας εφαρμογής που θα εμπλακούν στην διαδικασία ιχνηλασίας. Τα στοιχεία αυτά μπορεί να είναι τριών ειδών :

1. μεταβλητές του προγράμματος οι οποίες μπορεί να είναι είτε απλές μεταβλητές (scalar variables) είτε ολόκληροι πίνακες είτε υποπεριοχές πινάκων (array sub−blocks),
2. δομικά στοιχεία του κώδικα όπως βρόχοι επανάληψης, κλήσεις ρουτινών και κλήσεις συναρτήσεων,
3. στοιχεία ενδοεπικοινωνίας των διεργασιών (interprocess communication) όπως απο−στολή/παραλαβή μηνυμάτων (message send/receive), είσοδος/έξοδος σε σύνολα διεργα−σιών (join/leave group), αποστολή σημάτων (signalling) μεταξύ διεργασιών, καθώς επίσης και άφιξη σε φράγμα λογισμικού (barrier).

Στο σχήμα 3.1 υπάρχει ένα μικρό παράδειγμα για το πως ο χρήστης μπορεί να καθορίσει παραμέτρους ιχνοληψίας για την εφρμογή που βρίσκεται στο αρχείο */users/Mike/my_appl.f* Το πλήρες συντακτικό των εντολών που παρέχει η διεπαφή εισόδου βρίσκεται στο παράρτημα Α.

| | |
|---|---|
| **trace file** */users/Mike/my_appl.f* | Specify the source−code file |
| **trace variable** *a* **from** *30* **to** *150* | Specify scalar variable |
| **trace variable** *c[1..10]* **from** *100* **to** *300* | Specify sub−array |
| **trace code at loop level from** *50* **to** *125* | Specify tracing of loops |
| **trace code at subroutine level from** *100* **to** *200* | Specify tracing of sub−routine calls |
| **bye** | Quit the tool |

Σχήμα 3.1: Παράδειγμα εντολών του χρήστη στην διεπαφή εισόδου.

## 3.2 Εσωτερικές λειτουργίες της διεπαφής εισόδου

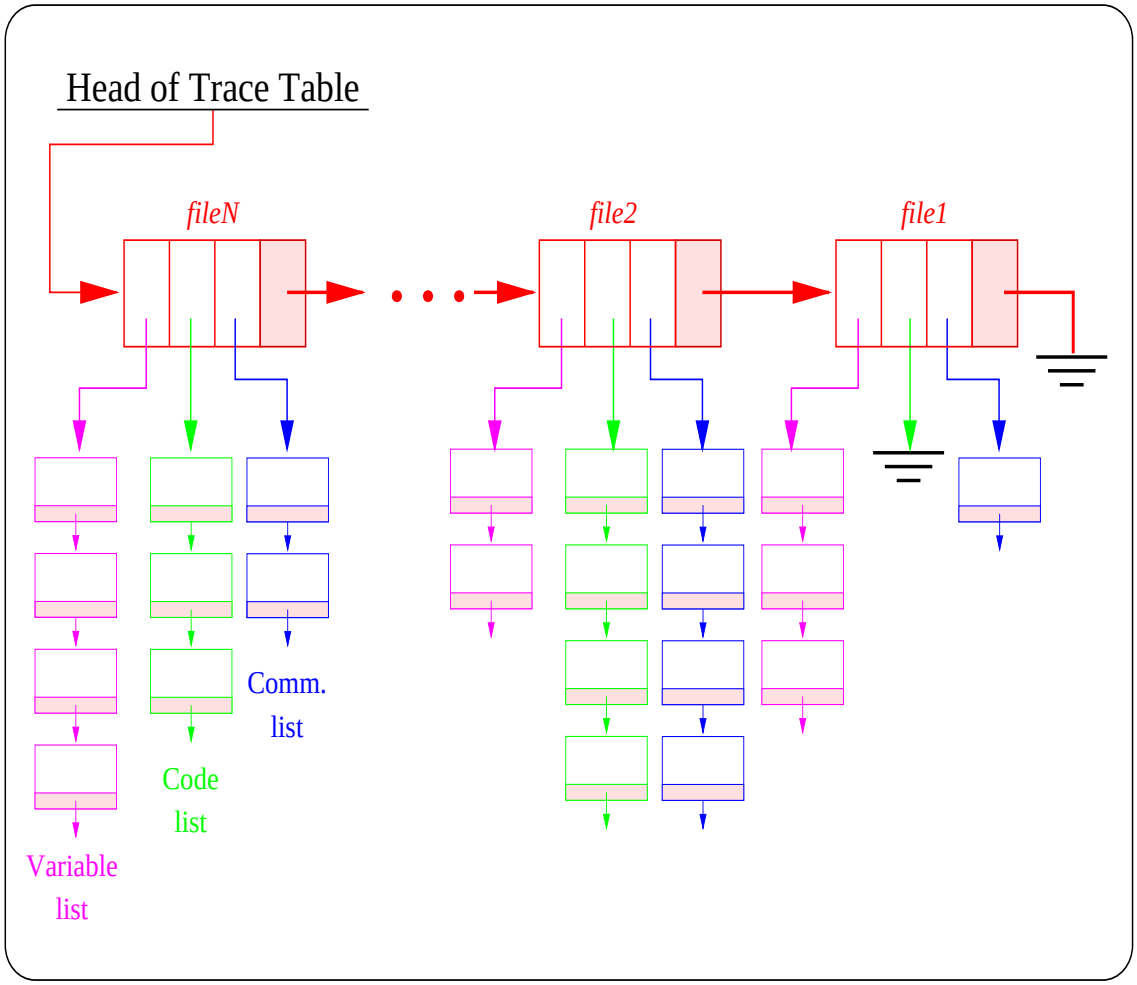
Μέχρι τώρα έχουμε αναφερθεί στην διευκόλυνση του προγραμματιστή μέσα από την χρήση της διεπαφής εισόδου. Ομως, η λειτουργικότητα της διεπαφής εισόδου δεν σταματάει εδώ. Οι εντολές ιχνοληψίας που δίνει ο χρήστης πρέπει να αποθηκευτούν σε κάποια δομή ώστε να χρησημοποιηθούν από το εργαλείο κατά την διαδικασία εισαγωγής των εντολών ιχνοληψίας μέσα στον πηγαίο κώδικα της εφαρμογής. Η δομή που χρησιμοποιείται για την αποθήκευση της πληροφορίας που περιέχουν οι εντολές αυτές, ονομάζεται *πίνακας ιχνοληψίας* (trace table) και περιγράφεται αναλυτικά στο παράρτημα Β.

### 3.2.1 Δομή του πίνακα ιχνηλασίας

Η εν λόγω δομή έχει μορφή απλά συνδεδεμένης λίστας της οποίας κάθε στοιχείο αποτελεί την κεφαλή επίσης μίας λίστας. Κάθε στοιχείο της πρώτης λίστας αντιστοιχεί σε ένα διαφορετικό αρχείο πηγαίου κώδικα της εφαρμογής που πρόκειται να ιχνοληπτηθεί. Η λίστα που αντιστοιχεί σε κάθε στοιχείο της πρώτης λίστας είναι διαιρεμένη σε τρείς υπο−λίστες, μία για τις μεταβλητές της εφαρμογής, μία για τα δομικά στοιχεία του κώδικα και μία για τα στοιχεία ενδοεπικοινωνίας των διεργασιών της εφαρμογής.

Για να βοηθήσουμε τον αναγνώστη να κατανοήσει τον τρόπο λειτουργίας της διεπαφής εισόδου, δίνουμε ένα παράδειγμα καθορισμού των παραμέτρων ιχνοληψίας. Ο πηγαίος κώδικας των διεργασιών της εφαρμογής βρίσκεται στα αρχεία *file1, file2, file3, ..., fileN*. Εστω ότι ο χρήστης αρχίζει τον καθορισμό των παραμέτρων ινχοληψίας από το αρχείο *file1* (με την εντολή **trace file** *file1*). Οι εντολές που εισάγει στην διεπαφή εισόδου, αφορούν την ινχοληψία μεταβλητών του κώδικα που περιέχει το εν λόγω αρχείο, δομικών στοιχείων του κώδικα, καθώς και στοιχείων ενδοεπικοινωνίας διεργασιών. Οταν τελειώσει με τον καθορισμό των παραμέτρων ιχνοληψίας, τότε κατευθύνει την διαδικασία καθορισμού των παραμέτρων ιχνοληψίας σε κάποιο από τα υπόλοιπα αρχεία, έστω το *file2* (με την εντολή ιχνοληψίας **trace file** *file2*). Ο χρήστης συνεχίζει με τον ίδιο τρόπο να καθορίζει τις παραμέτρους ιχνοληψίας για όλα τα αρχεία που τον ενδιαφέρουν.

Η διαδικασία κατασκευής του πίνακα ιχνοληψίας είναι απλή. Η διεπαφή εισόδου λει−τουργεί σαν διερμηνέας (interpreter) των εντολών που δίνει ο χρήστης. Με την εντολή **trace file** *file*, ενεργοποιείται η δημιουργία ενός συνόλου που θα περιέχει εντολές ιχνοληψίας οι οποίες απευθύνονται στο συγκεκριμένο αρχείο *file*. Η δημιουργία αυτού του συνόλου στην πράξη σηματοδοτεί την δημιουργία ενός νέου στοιχείου στην δομή του πίνακα ιχνοληψίας, το οποίο θα περιέχει τις παραμέτρους ιχνοληψίας που αφορούν το συγκεκριμένο αρχείο *file*.




Σχήμα 3.2: Γραφική αναπαράσταση της δομής του πίνακα ιχνοληψίας.

### 3.2.2 Στοιχεία ή παράμετροι ιχνοληψίας

Με τον όρο *στοιχεία ιχνοληψίας* εννοούμε τα στοιχεία τα οποία ο χρήστης καθορίζει να συμμετέχουν στην διαδικασία ιχνοληψίας. Τα στοιχεία αυτά, όπως έχουμε ήδη αναφέρει, μπορούν να ανήκουν σε μία από τρείς συγκεκριμένες κατηγορίες. Σε κάθε κατηγορία μας ενδιαφέρουν διαφορετικά στοιχεία, τα οποία αποθηκεύουμε στα στοιχεία της λίστας που αντιστοιχεί σε κάθε κατηγορία. Οι κατηγορίες είναι οι εξής :

1. **Μεταβλητές της εφαρμογής (απλές, πίνακες, υποπίνακες).** Σε αυτήν την κατηγορία μας ενδιαφέρει το όνομα της μεταβλητής που επιλέγεται για συμμετοχή στην διαδικασία ιχνοληψίας, οι διαστάσεις που οριοθετούν το μέγεθος της (μόνο στην περίπτωση που πρόκειται για υποπίνακα), καθώς και το τμήμα του πηγαίου κώδικα στο οποίο ο χρήστης επιθυμεί να ιχνοληπτηθεί η συγκεκριμένη μεταβλητή [1].

2. **Δομικά στοιχεία κώδικα (βρόχοι επανάληψης, κλήσεις συναρτήσεων/ υπορουτινών).** Σε αυτήν την κατηγορία μας ενδιαφέρουν μόνο δύο στοιχεία, η ταυτότητα του δομικού

[1] Για τις μεταβλητές μας ενδιαφέρουν και άλλα στοιχεία, τα οποία είναι πληροφορίες που θα γίνουν γνωστές κατά την συντακτική ανάλυση του πηγαίου κώδικα της εφαρμογής και στα οποία θα αναφερθούμε στην επόμενη ενότητα.

στοιχείου (βρόχος, συνάρτηση, υπορουτίνα) και το τμήμα του πηγαίου κώδικα στο οποίο ο χρήστης επιθυμεί να ιχνοληπτηθεί το συγκεκριμένο στοιχείο.

3. **Στοιχεία ενδοεπικοινωνίας διεργασιών (αποστολή/λήψη μηνυμάτων, φράγματα λογισμικού, είσοδος/έξοδος σε σύνολα διεργασιών, αποστολή σημάτων).** Και σε αυτήν την κατηγορία, ο χρήστης πρέπει να καθορίσει μόνο την ταυτότητα του στοιχείου ενδοεπικοινωνίας και το τμήμα του πηγαίου κώδικα στο οποίο ο χρήστης επιθυμεί να ιχνοληπτηθεί το συγκεκριμένο στοιχείο.

Στο παράρτημα Β παρουσιάζεται η ακριβής δομή του πίνακα ιχνοληψίας. Επίσης, υπάρχει ένα αναλυτικό παράδειγμα που δείχνει τις πληροφορίες που αποθηκεύονται σε κάθε εντολή ιχνοληψίας που εισάγει ο χρήστης στην διεπαφή εισόδου.

# Κεφάλαιο 4

# Στατική Ανάλυση της Εφαρμογής

Η *Στατική Ανάλυση* του πηγαίου κώδικα της εφαρμογής γίνεται κατά την διάρκεια της πρώτης φάσης της διαδικασίας ιχνοληψίας. Περιλαμβάνει την ανεύρεση σημείων εισαγωγής κώδικα καθοδήγησης και την πραγματοποίηση της εισαγωγής αυτού του κώδικα. Μόλις ο χρήστης ολοκληρώσει τον καθορισμό των παραμέτρων ιχνοληψίας που αφορούν ένα συγκεκριμένο αρχείο πηγαίου κώδικα της εφαρμογής, τότε υπάρχουν όλες οι απαραίτητες πληροφορίες για να αρχίσει το *ArrayTracer* την προαναφερόμενη διαδικασία.

## 4.1 Στατικά ίχνη

Οπως αναφέρθηκε στην ενότητα 3.2.1, ο πίνακας ιχνοληψίας κατασκευάζεται κατά την διάρκεια καθορισμού των παραμέτρων ιχνοληψίας. Μόλις ολοκληρωθεί ο καθορισμός των παραμέτρων ιχνοληψίας, ακολουθεί η συντακτική ανάλυση της εφαρμογής. Με τον εντοπισμό και την ανάλυση των δηλώσεων των μεταβλητών του προγράμματος, συμπληρώνεται η δομή του πίνακα ιχνοληψίας. Στην συνέχεια εντοπίζονται τα δομικά στοιχεία του προγράμματος (κλήσεις συναρτήσεων, κλήσεις υπορουτινών, βρόχοι επανάληψης, αναφορές στην μνήμη, επικοινωνία με άλλες διεργασίας, κ.τ.λ.) και λαμβάνει χώρα η διαδικασία *Στατικής Ανάλυσης*. Κατά την διάρκεια αυτής της διαδικασίας και σύμφωνα πάντα με τις πληροφορίες του πίνακα ιχνοληψίας, αναγνωρίζονται τα σημεία εκείνα στα οποία πρέπει να εισαχθεί κώδικας καθοδήγησης για την παραγωγή των ιχνών που ενδιαφέρουν τον χρήστη.

Υπάρχουν περιπτώσεις στις οποίες η πληροφορία που έχει ζητήσει ο χρήστης (π.χ. προσπέλαση στην θέση μνήμης που καταλαμβάνει συγκεκριμένη μεταβλητή) είναι διαθέσιμη την στιγμή της συντακτικής ανάλυσης του πηγαίου κώδικα. Αυτές οι περιπτώσεις περιλαμβάνουν :

- τα τμήματα κώδικα που δεν περιέχουν εντολές διακλάδωσης, και
- τα τμήματα κώδικα που δεν περιέχουν ετικέτες (labels) οι οποίες ίσως αποτελούν τον προορισμό εντολών διακλάδωσης.

Ο πρώτος βασικός στόχος της *Στατικής Ανάλυσης* του πηγαίου κώδικα της εφαρμογής είναι ο εντοπισμός αυτών των περιπτώσεων και η αποθήκευση σε προσωρινά αρχεία της πληροφορίας που μπορεί να εξαχθεί από τον πηγαίο κώδικα. Συνεπώς, ο αριθμός των αρχείων που θα αποθηκεύσουν την στατική πληροφορία, θα είναι ίσος με τον αριθμό των αρχείων πηγαίου κώδικα της εφαρμογής, στα οποία εφαρμόζεται η διαδικασία της *Στατικής Ανάλυσης*. Τα προσωρινά αρχεία στα οποία αποθηκεύεται η πληροφορία αυτή, τα ονομάζουμε *αρχεία ιχνοληψίας στατικών δεδομένων*.

## 4.2 Σημεία εισαγωγής κώδικα καθοδήγησης

Μέχρι τώρα αναφερθήκαμε στα ίχνη τα οποία μπορούμε να συλλέξουμε κατά την διάρκεια της *Στατικής Ανάλυσης* της εφαρμογής. Δυστυχώς όμως, το μεγαλύτερο μέρος του όγκου των ιχνών εξαρτάται άμεσα από το περιβάλλον εκτέλεσης της εφαρμογής. Παράγοντες εξάρτησης αποτελούν τα δεδομένα που εισάγει ο χρήστης κατά την εκτέλεση της εφαρμογής, τα δεδομένα που υπάρχουν στα αρχεία από τα οποία διαβάζει την είσοδο της η εφαρμογή, κ.λ.π. Τα ίχνη αυτά τα ονομάζουμε *δυναμικά ίχνη* και είναι δυνατόν να παραχθούν και να συλλεχθούν μόνο κατά την διάρκεια της εκτέλεσης.

### 4.2.1 Παραγωγή δυναμικών ιχνών

Στις περιπτώσεις ιχνών που εξαρτώνται από το περιβάλλον εκτέλεσης της εφαρμογής, πρέπει να εισαχθεί κώδικας καθοδήγησης, ο οποίος θα τα παράγει όταν αυτά είναι διαθέσιμα. Τα σημεία στα οποία εισάγονται οι εντολές καθοδήγησης μέσα στο αρχείο πηγαίου κώδικα της εφαρμογής τα ονομάζουμε *σημεία εισαγωγής* και επιλέγονται κατά την διάρκεια της *Στατικής Ανάλυσης*. Κατά την εκτέλεση της εφαρμογής, όταν εκτελεστεί ο κώδικας καθοδήγησης, παράγονται τα ίχνη, τα οποία αναλαμβάνει να συλλέξει η διεργασία–*συλλέκτης* που έχει προσκολληθεί στην συγκεκριμένη διεργασία εφαρμογής.

Τα σημεία εισαγωγής στην υπάρχουσα υλοποίηση του *ArrayTracer* επιλέγονται πολύ απλά στις περισσότερες περιπτώσεις. Κώδικας καθοδήγησης εισάγεται μετά από :

- προσπέλαση μνήμης,
- εντοπισμό κάποιου βρόχου επανάληψης,
- κλήση σε συνάρτηση/υπορουτίνα,
- γεγονός που αφορά την ενδοεπικοινωνία των διεργασιών.

Η διαδικασία επιλογής ενός σημείου εισαγωγής γίνεται ελαφρώς πολυπλοκότερη στην περίπτωση εισαγωγής *φόρμας ιχνοληψίας* (trace template) στον πηγαίο κώδικα της εφαρμογής.

### 4.2.2 Φόρμες ιχνοληψίας

Ο κώδικας καθοδήγησης αποτελείται από ένα σύνολο κλήσεων σε συναρτήσεις που ανήκουν στην βιβλιοθήκη ιχνοληψίας του *ArrayTracer*. Κατά συνέπεια, ο χρόνος εκτέλεσης της εφαρμογής επιφορτίζεται με το χρονικό κόστος εκτέλεσης αυτών των κλήσεων. Το γεγονός ότι δεν μπορούμε να αποφύγουμε εντελώς αυτό το κόστος, δεν αποτελεί μεγάλο μειονέκτημα γιατί μπορούμε να το μειώσουμε αρκετά ώστε να μην αποτελέσει ανασταλτικό παράγοντα για την χρήση του *ArrayTracer*. Αυτή η μείωση επιτυγχάνεται με την χρήση των *φορμών ιχνοληψίας*.

Υπάρχουν πολλές περιπτώσεις στις οποίες ένας αριθμός ιχνών μπορεί να συμπιεστεί σε ένα *μοναδικό* στοιχείο μεγέθους λίγων bytes. Αυτό μπορεί να συμβεί σε καταστάσεις στις οποίες υπάρχουν διαδοχικές αναφορές στην ίδια θέση μνήμης (π.χ. μέσα σε βρόχους επανάληψης), ή υπάρχουν αναφορές σε γειτονικές θέσεις μνήμης που όλες ανήκουν στον ίδιο πίνακα (π.χ. σε περιπτώσεις σάρωσης πίνακα). Σε αυτές τις περιπτώσεις, αντί να παράγεται ένα ίχνος για κάθε αναφορά στην συγκεκριμένη θέση μνήμη ή για κάθε ξεχωριστό στοιχείο του υποπίνακα, μπορούμε να συμπιέσουμε όλη αυτήν την πληροφορία σε ένα ειδικό στοιχείο ιχνοληψίας. Αυτό το ειδικό στοιχείο το οποίο θα παράγεται αμέσως μετά το τέλος του βρόχου επανάληψης ή του τμήματος του κώδικα το οποίο περιέχει τις αναφορές σε συνεχόμενες θέσεις μνήμης ενός πίνακα, το ονομάζουμε *φόρμα ιχνοληψίας*.

Στην υπάρχουσα υλοποίηση του *ArrayTracer*, η χρήση των φορμών ιχνοληψίας είναι πολύ περιορισμένη. Τα κριτήρια με βάση τα οποία αποφασίζεται αν ένα σύνολο από εντολές ιχνοληψίας θα αντικατασταθεί με μια φόρμα ιχνοληψίας, είναι πολύ συντηρητικά και απλά, χωρίς να εκμεταλλεύονται όλες τις περιπτώσεις στις οποίες θα μπορούσε να εφαρμοστεί μια τέτοια αντικατάσταση. Η τακτική που ακολουθούμε είναι η ακόλουθη. Σε περιπτώσεις τμημάτων κώδικα που γνωρίζουμε ακριβώς πόσες φορές θα εκτελεστούν, όπως στις περιπτώσεις βρόγχων επανάληψης, θα συγκρατούμε την αντίστοιχη πληροφορία και στο τέλος αυτού του τμήματος κώδικα θα εισάγουμε μία φόρμα ιχνοληψίας με την ακόλουθη μορφή:

**<Tag, Start, End, OpCode, Num>**

- **Tag.** Πρόκειται για την ετικέτα που έχει αποδοθεί στην μεταβλητή στην οποία αναφέρεται η παραχθείσα φόρμα ιχνοληψίας.
- **Start, End.** Τα πεδία αυτά έχουν νόημα μόνο στην περίπτωση που η συγκεκριμένη φόρμα ιχνοληψίας αντιστοιχεί σε κάποιο υποπίνακα. Σε αυτές τις περιπτώσεις, το πεδίο **Start** κρατάει την χαμηλότερη (μικρότερη) θέση μνήμης στην οποία υπήρξε αναφορά μέσα στον κώδικα, ενώ το πεδίο **End** κρατάει αντίστοιχα την υψηλότερη θέση μνήμης στην οποία υπήρξε αναφορά.
- **OpCode.** Αυτό το πεδίο κρατάει τον κωδικό του χειρισμού (ανάγνωση/εγγραφή μνήμης, αναγνωση/εγγραφή δίσκου, αποστολή/παραλαβή μηνύματος, κ.ά.) που έλαβε χώρα κατά την προσπέλαση στην συγκεκριμένη θέση (ή στις συγκεκριμένες θέσεις) μνήμης στις οποίες αναφέρεται η φόρμα ιχνοληψίας.
- **Num.** Αυτό το πεδίο περιέχει τον συνολικό αριθμό αναφορών στην θέση (ή στις θέσεις) μνήμης που περιγράφει η συγκεκριμένη φόρμα ιχνοληψίας.

Η διαδικασία επιλογής των σημείων εισαγωγής μιας φόρμας ιχνοληψίας, συμπεριλαμβάνει τον εντοπισμό του σημείου που τελειώνει το τμήμα του κώδικα που περιέχει τις συνεχείς αναφορές στην ίδια θέση (ή σε γειτονικές θέσεις) μνήμης. Στην πράξη, αυτό σημαίνει τον εντοπισμό είτε του τέλους ενός βρόχου επανάληψης, είτε του σημείου που υπάρχει μία ετικέτα κώδικα (label). Στην δεύτερη περίπτωση, ο εντοπισμός της ετικέτας κώδικα σημαίνει ότι το κομμάτι του κώδικα που ξεκινάει από εκείνο το σημείο, μπορεί να ξαναεκτελεστεί μετά από την εκτέλεση κάποιας εντολής διακλάδωσης (branch command) η οποία θα μεταφέρει την ροή εκτέλεσης του προγράμματος στην συγκεκριμένη ετικέτα κώδικα.

Πιστεύουμε ότι η χρήση των φορμών ιχνοληψίας θα βοηθήσει πάρα πολύ προς την κατεύθυνση επιτάχυνσης της λειτουργίας του εργαλείου μας, για τους λόγους που αναφέρουμε στην επόμενη υποενότητα.

### 4.2.3 Κόστος χρήσης των φορμών ιχνοληψίας

Το άμεσο όφελος που έχουμε από την χρήση των φορμών ιχνοληψίας είναι η μείωση του αριθμού κλήσεων σε ρουτίνες ιχνοληψίας που εισάγονται στον πηγαίο κώδικα της εφαρμογής. Το τίμημα που καλούμαστε να πληρώσουμε για αυτήν την μείωση είναι η αύξηση της πολυπλοκότητας της διαδικασίας *Στατικής Ανάλυσης*. Ομως θεωρούμε ότι τίμημα αυτό δεν δημιουργεί σημαντικά προβλήματα για δύο λόγους :

1. Δεν αυξάνει τον χρόνο εκτέλεσης της ιχνοληπτούμενης εφαρμογής και συνεπώς δεν εισάγει επιπλέον *διαστολή* στον χρόνο εκτέλεσης.

2. Η αύξηση του χρόνου που απαιτείται για την εισαγωγή του κώδικα καθοδήγησης που περιλαμβάνει και φόρμες ιχνοληψίας, είναι μικρή γιατί οι απαραίτητοι υπολογισμοί χρειάζονται μόνο χρόνο επεξεργαστή (CPU time) και δεν περιλαμβάνουν καθόλου χειρισμούς εισόδου/εξόδου στον δίσκο (disk I/O operations).

Εκτός από το γεγονός της μείωσης του χρόνου εκτέλεσης της ιχνοληπτούμενης εφαρμογής, υπάρχει ακόμα ένα πλεονέκτημα που μας προσφέρει η χρήση των φορμών ιχνοληψίας. Η συμπίεση πληροφορίας πολλών εγγραφών ιχνοληψίας σε μία φόρμα ιχνοληψίας έχει σαν αποτέλεσμα την μείωση του όγκου που απαιτείται για την αποθήκευση της και κατά συνέπεια την μείωση του όγκου των *αρχείων αποθήκευσης δυναμικών ιχνών*. Το τίμημα που καλούμαστε να πληρώσουμε για την εξοικονόμηση χώρου στον δίσκο, είναι ο χρόνος που θα χρειαστεί κατά την επεξεργασία των ιχνών μετά το πέρας εκτέλεσης της εφαρμογής (post−mortem processing) για την αποσυμπίεση της πληροφορίας που βρίσκεται μέσα στις φόρμες ιχνοληψίας. Το κόστος όμως αυτό μπορεί να θεωρηθεί αμελητέο για τους εξής δύο λόγους :

1. Ο χρόνος αυτός δεν επηρεάζει τον χρόνο εκτέλεσης της εφαρμογής και επομένως δεν αυξάνει το ποσό της διαστολής που προκαλεί το εργαλείο στον χρόνο εκτέλεσης της εφαρμογής.

2. Σε πολλές περιπτώσεις, η επεξεργασία των ιχνών που γίνεται από το δεύτερο λειτουργικό επίπεδο του *ArrayTracer* περιλαμβάνει χειρισμούς πάνω στο άθροισμα των πληροφοριών που περιέχουν τα ίχνη τα οποία αναφέρονται στο ίδιο στοιχείο ιχνοληψίας. Τέτοιοι χειρισμοί είναι όλοι αυτοί που αποσκοπούν στην εξαγωγή στατιστικών μεγεθών (συνολικός αριθμός αναφορών σε συγκεκριμένη θέση μνήμης, μέσος χρόνος που χρειάζεται να επιστρέψει η κλήση σε μια υπορουτίνα, κ.λ.π.). Σε αυτές τις περιπτώσεις, το άθροισμα (ή τουλάχιστον τα μερικά αθροίσματα) της πληροφορίας πολλών ιχνών βρίσκεται ήδη δημιουργημένο μέσα στις αντίστοιχες φόρμες ιχνοληψίας. Συνεπώς, σε αυτές τις περιπτώσεις η ύπαρξη των φορμών ιχνοληψίας επιταχύνει την διαδικασία επεξεργασίας των ιχνών που λαμβάνει χώρα μετά το πέρας της εκτέλεσης της εφαρμογής.

## 4.3 Επιλεκτική ιχνοληψία

Ενα χαρακτηριστικό της μεθόδου ιχνοληψίας που χρησιμοποιούμε, είναι η παραγωγή *επιλεγμένων ιχνών*. Η εισαγωγή κώδικα καθοδήγησης γίνεται μόνο στα σημεία του πηγαίου κώδικα που περιέχουν πληροφορία σχετική με κάποιο από τα στοιχεία ιχνοληψίας που έχει καθορίσει ο χρήστης. Ετσι αποφεύγουμε την παραγωγή ιχνών που δεν ενδιαφέρουν τον χρήστη και τα οποία θα απορρίπτονταν κατά την διαδικασία επεξεργασίας των ιχνών που λαμβάνει χώρα μετά το πέρας της εκτέλεσης της εφαρμογής. Συνεπώς, μειώνουμε τον αριθμό των κλήσεων σε συναρτήσεις της βιβλιοθήκης ιχνοληψίας, άρα μειώνουμε την διαστολή που θα προκαλούσε το εργαλείο μας στον χρόνο εκτέλεσης της εφαρμογής. Για να γίνει αυτή η *επιλεκτική εισαγωγή κώδικα καθοδήγησης*, κατά την *Στατική Ανάλυση* της εφαρμογής γίνεται μία κατηγοριοποίηση όλων των ιχνών που είναι δυνατόν να παραχθούν και επιλέγονται μόνο αυτά που ενδιαφέρουν τον χρήστη.

**Χρήσιμα ίχνη.** Η κατηγοριοποίηση των ιχνών τα χωρίζει σε *χρήσιμα* (useful) και σε *άχρηστα* (useless). Ο χαρακτηρισμός ενός ίχνους ως *χρήσιμο* ή *άχρηστο* βασίζεται στα δεδομένα που έχει δώσει ο χρήστης στο εργαλείο μας. Πιο συγκεκριμένα, ένα ίχνος το οποίο αφορά ή είναι πιθανό να αφορά κάποιο από τα στοιχεία ιχνοληψίας (όπως αυτά είναι καθορισμένα στον πίνακα ιχνοληψίας), χαρακτηρίζεται ως *χρήσιμο*. Αντίθετα, όλα τα ίχνη εκείνα για τα οποία

**User Input:**
**trace variable** *a[1..10]* **from** *1* **to** *100*
**trace variable** *b* **from** *1* **to** *50*

| **Original Source Code:** | **Instrumented Source Code:** |
|---|---|
| INTEGER i, a(100), b, c | INTEGER i, a(100), b, c |
| b = 1 | b = 1 |
| DO i = M, N | DO i = M, N |
| a[i] = c * b | a[i] = c * b |
| | *CALL tracevar(1, i, OP_WR)* |
| | *CALL tracevar(2, 0, OP_RD)* |
| b = c + 1 | b = c + 1 |
| | *CALL tracevar(2, 0, OP_WR)* |
| c = c + 5 | c = c + 5 |
| ENDDO | ENDDO |

Σχήμα 4.1: Παράδειγμα επιλεκτικής εισαγωγής κώδικα καθοδήγησης.

μπορούμε να είμαστε σίγουροι κατά την διάρκεια της *Στατικής Ανάλυσης* ότι δεν αφορούν κανένα από τα στοιχεία ιχνοληψίας, τα χαρακτηρίζουμε ως *άχρηστα*.

Στο σχήμα 4.1 διασαφηνίζεται η διαδικασία διαχωρισμού των ιχνών. Το τμήμα του πηγαίου κώδικα του παραδείγματος αναφέρεται στις θέσεις μνήμης όπου βρίσκονται οι μεταβλητές *b* και *c*, και ο υποπίνακας *a[M..N]*. Από αυτά που έχει δηλώσει ο χρήστης σαν στοιχεία ιχνοληψίας, οι αναφορές στην μεταβλητή *c* δεν ενδιαφέρουν τον χρήστη, άρα τα ίχνη αυτών των αναφορών χαρακτηρίζονται ως *άχρηστα*. Αντίθετα, τα ίχνη των αναφορών στην μεταβλητή *b*, ενδιαφέρουν σίγουρα τον χρήστη, γι' αυτό και χαρακτηρίζονται ως *χρήσιμα*. Τέλος, τα ίχνη που αφορούν τις αναφορές στον υποπίνακα *a[M..N]*, τα χαρακτηρίζουμε και αυτά ως *χρήσιμα*, γιατί υπάρχει πιθανότητα τα ίχνη τους να ενδιαφέρουν τον χρήστη.

**Ουσιώδη ίχνη.** Ο διαχωρισμός των ιχνών που παρουσιάσαμε παραπάνω, βοηθάει την διαδικασία επιλεκτικής εισαγωγής κώδικα καθοδήγησης. Ετσι, παράγονται μόνο τα ίχνη εκείνα τα οποία είναι πιθανόν να λάβουν μέρος στην επεξεργασία. Ομως, από τα δυναμικά ίχνη που θα συλλεχθούν τελικά, δεν συμμετέχουν όλα στην διαδικασία ανάλυσης της επίδοσης της εφαρμογής. Για την επιλογή των ιχνών που θα συμμετέχουν σε αυτήν την διαδικασία, γίνεται ένας νέος διαχωρισμός.

Από τα χρήσιμα ίχνη που έχουν συλλεχθεί, ξεχωρίζουμε τα *ουσιώδη* (essential) από τα *μή ουσιώδη* με βάση πλέον την συμμετοχή τους στις λειτουργίες του δευτέρου επιπέδου λειτουργίας του εργαλείου μας. Η διαδικασία αυτή του διαχωρισμού των *ουσιωδών* ιχνών γίνεται στο στάδιο προεπεξεργασίας των ιχνών πριν αυτά σταλούν στο δεύτερο λειτουργικό επίπεδο για να συμμετέχουν στην διαδικασία ανάλυσης της επίδοσης της εφαρμογής. Περισσότερες λεπτομέρειες για αυτό το στάδιο υπάρχουν στο επόμενο κεφάλαιο.

Στο σχήμα 4.2 υπάρχει ένα παράδειγμα διαχωρισμού *ουσιωδών* ιχνών από τα *μη ουσιώδη*, τα οποία αφορούν τον κώδικα του σχήματος 4.1. Διακρίνουμε τρεις περιπτώσεις, ανάλογα με τα όρια που σηματοδοτούν την αρχή και το τέλος του βρόχου επανάληψης:

1. Στην πρώτη περίπτωση, στον βρόχο επανάληψης προσπελαύνονται οι θέσεις 2 μέχρι 8 του πίνακα *a*. Ο χρήστης είχε ζητήσει την ιχνοληψία του υποπίνακα *a[1..10]*, ο οποίος περιλαμβάνει αυτές τις θέσεις. Συνεπώς, τα ίχνη που έχουν παραχθεί με

Trace Element **te_1**: *a[1..10]*
Trace Element **te_2**: *b*

**Case 1: (M=2, N=8)**
– 7 accesses in subarray *a[1..8]*, which is included in *te_1*.
– 7 accesses in scalar variable *b*, which is *te_2*.

**Case 2: (M=12, N=30)**
– 19 accesses in subarray *a[12..30]*, which is not included in *te_1*.
– 19 accesses in scalar variable *b*, which is *te_2*.

**Case 3: (M=5, N=15)**
– 11 accesses in subarray *a[5..15]*, which is merely included in *te_1*.
– 11 accesses in scalar variable *b*, which is *te_2*.

Σχήμα 4.2: Παράδειγμα διαχωρισμού των ιχνών σε *ουσιώδη* και *μη ουσιώδη*.

τις εντολές *CALL tracevar(1, i, OP_WR)*, περιέχουν πληροφορία που αντιστοιχεί εξ' ολοκλήρου στο στοιχείο ιχνοληψίας *a[1..10]*. Αυτά τα ίχνη χαρακτηρίζονται *ουσιώδη* και συμπεριλαμβάνονται στο σύνολο των ιχνών που θα συμμετέχουν στην επεξεργασία του δεύτερου λειτουργικού επιπέδου του εργαλείου μας.

2. Στην δεύτερη περίπτωση, στον βρόχο επανάληψης προσπελαύνονται οι θέσεις 12 μέχρι 30 του πίνακα *a*. Καμμία από αυτές τις θέσεις δεν ανήκει σε στοιχείο ιχνοληψίας που έχει καθοριστεί από τον χρήστη. Συνεπώς, τα ίχνη που έχουν παραχθεί με τις εντολές *CALL tracevar(1, i, OP_WR)*, περιέχουν πληροφορία που δεν ενδιαφέρει τον χρήστη. Γι΄αυτό και τα ίχνη αυτά τα χαρακτηρίζουμε ως *μη ουσιώδη* και δεν τα συμπεριλαμβάνουμε στο σύνολο των ιχνών που θα επεξεργαστούν στο επόμενο λειτουργικό επίπεδο του *ArrayTracer*.

3. Τέλος, στην τρίτη περίπτωση, στον βρόχο επανάληψης προσπελαύνονται οι θέσεις 5 μέχρι 15 του πίνακα *a*. Μερικές από αυτές τις θέσεις (από 5 μέχρι 10) ανήκουν στον υποπίνακα *a[1..10]* που έχει επιλέξει για ιχνοληψία, ενώ οι υπόλοιπες (11 μέχρι 15) δεν ανήκουν σε κανένα στοιχείο ιχνοληψίας που έχει καθορίσει ο χρήστης. Συνεπώς, τα ίχνη εκείνα που αφορούν στις θέσεις 5 εως 10 πρέπει να ληφθούν υπ' όψη στο στάδιο επεξεργασίας των ιχνών. Γι' αυτό χαρακτηρίζουμε αυτά τα ίχνη ως *ουσιώδη* και συμμετέχουν στην ανάλυση της επίδοσης της εφαρμογής, ενώ τα υπόλοιπα ίχνη χαρακτηρίζονται ως *μη ουσιώδη* και δεν λαμβάνονται στο εξής υπ' όψη.

Εκτός από τις προσπελάσεις σε διαφορετικούς υποπίνακες του πίνακα *a* σε κάθε μία από τις παραπάνω περιπτώσεις, μέσα στον βρόχο επανάληψης έχουμε προσπελάσεις και των θέσεων μνήμης που καταλαμβάνουν οι μεταβλητές *b* και *c*. Ο χρήστης έχει μαρκάρει την μεταβλητή *b* για συμμετοχή στην διαδικασία ιχνοληψίας. Συνεπώς, όλα τα ίχνη που αναφέρονται σε αυτήν[1] χαρακτηρίζονται ως *ουσιώδη*. Η μεταβλητή *c* δεν έχει μαρκαριστεί από τον χρήστη για συμμετοχή στην διαδικασία ιχνοληψίας. Ετσι, στην διαδικασία διαχωρισμού των ιχνών σε *χρήσιμα* και *άχρηστα* (κατά την διάρκεια της *Στατικής Ανάλυσης*), τα ίχνη που αφορούσαν σε

[1]Πρόκειται για τα ίχνη που παράγονται από τις εντολές *CALL tracevar(2, 0 , OP_WR)* μέσα στον βρόχο επανάληψης.

αυτήν την μεταβλητή χαρακτηρίστηκαν ως *άχρηστα* και δεν εισήχθη κώδικας καθοδήγησης για την παραγωγή τους.

Τέλος, πρέπει να επισημάνουμε την παραγωγή του *στατικού ίχνους* για την προσπέλαση της θέσης μνήμης της *b*, η οποία γίνεται με την εντολή ακριβώς πριν τον βρόχο επανάληψης (*b = 1*). Η συγκεκριμένη εντολή βρίσκεται έξω από τον βρόχο και έξω από τμήματα κώδικα τα οποία μπορεί να εκτελεστούν περισσότερες από μια φορές εξαιτίας εντολών διακλάδωσης. Δηλαδή, όλη η πληροφορία που περιέχει το ίχνος που αφορά σε αυτήν την προσπέλαση μνήμης, είναι γνωστή κατά την διάρκεια της *Στατικής Ανάλυσης* του κώδικα. Επίσης, είναι γνωστό και ότι μόνο ένα τέτοιο ίχνος θα παραχθεί κατά την εκτέλεση του συγκεκριμένου κώδικα. Συνεπώς, το συγκεκριμένο ίχνος μπορεί να παραχθεί κατά την διάρκεια της *Στατικής Ανάλυσης* και να μην επιβαρύνει τον χρόνο εκτέλεσης της εφαρμογής με την παραγωγή του. Πρόκειται λοιπόν για ένα *στατικό ίχνος*, όπως αναφέρθηκε στην ενότητα 4.1.

Με τον διαχωρισμό των ιχνών σε *χρήσιμα* και *άχρηστα*, στον οποίο βασίζεται η *επιλεκτική εισαγωγή κώδικα καθοδήγησης*, επιτυγχάνουμε μερικά αξιοσημείωτα οφέλη. Η παραγωγή μόνο των *χρήσιμων* ιχνών έχει σαν συνέπεια την εισαγωγή λιγότερου κώδικα καθοδήγησης το οποίο συνεπάγεται μικρότερη διαστολή στον χρόνο εκτέλεσης της εφαρμογής. Επιπλέον, η παραγωγή λιγότερων ιχνών απαιτεί λιγότερο χώρο στον δίσκο για την αποθήκευσή τους. Το τίμημα που καλούμαστε να πληρώσουμε για τα οφέλη που αποκομίζουμε, είναι μια αμελητέα αύξηση της πολυπλοκότητας της διαδικασίας της *Στατικής Ανάλυσης*, που θα πρέπει τώρα να διακρίνει τα *χρήσιμα* ίχνη και να εισάγει μόνο για αυτά τον απαραίτητο κώδικα καθοδήγησης.

# Κεφάλαιο 5

# Συλλογή Δυναμικών Ιχνών

Οπως αναφέρθηκε στο προηγούμενο κεφάλαιο, σημαντική προσπάθεια έχει καταβληθεί για την ελάττωση της διαστολής του χρόνου εκτέλεσης της εφαρμογής. Ομως, η προσπάθεια που έχει γίνει για την αποδοτική παραγωγή ιχνών κατά την διάρκεια εκτέλεσης της εφαρμογής, δεν είναι από μόνη της αρκετή για την επίτευξη του στόχου μας. Η διαδικασία συλλογής των παραγόμενων ιχνών πρέπει να έχει την μικρότερη δυνατή αλληλεπίδραση με την εκτελούμενη εφαρμογή.

Η επιβάρυνση στον χρόνο εκτέλεσης της εφαρμογής που προκαλεί η διαδικασία ιχνοληψίας, μετριέται σε δύο επίπεδα:

1. Σε επίπεδο συνολικού χρόνου εκτέλεσης της εφαρμογής (elapsed time). Οι μετρήσεις αυτού του επιπέδου αφορούν την διαστολή που γίνεται αντιληπτή στον χρήστη. Οι τιμές που καταγράφονται δεν μπορεί να είναι πολύ ακριβείς σε ένα χρονο−διαιρετό (time−sharing) σύστημα, γιατί στον συνολικό χρόνο εκτέλεσης της εφαρμογής συμπεριλαμβάνεται και ο χρόνος που ο επεξεργαστής του συστήματος χρησιμοποιήθηκε από διεργασίες που δεν ανήκουν στην συγκεκριμένη εφαρμογή.

2. Σε επίπεδο χρόνου κατοχής του επεξεργαστή από την εφαρμογή (CPU time). Οι μετρήσεις αυτού του επιπέδου αφορούν την πραγματική διαστολή του χρόνου εκτέλεσης της εφαρμογή, που οφείλεται στην διαδικασία ιχνοληψίας. Οι τιμές που καταγράφονται είναι πολύ ακριβείς[1] και αφορούν αποκλειστικά τον χρόνο που χρησιμοποιήθηκε ο επεξεργαστής από τις εκτελέσιμες εκδόσεις της εφαρμογής με και χωρίς κώδικα καθοδήγησης.

Στην συνέχεια αυτού του κεφαλαίου θα αναφερθούμε στην προσπάθειά μας να ελαττώσουμε την διαστολή σε επίπεδο χρόνου κατοχής του επεξεργαστή. Ο λόγος που δώσαμε περισσότερη έμφαση προς αυτήν την κατεύθυνση είναι ο ακόλουθος. Οσο μικρότερη είναι η διαστολή σε αυτό το επίπεδο, τόσο μικρότερη πιθανότητα υπάρχει για την ιχνοληπτούμενη εφαρμογή να παρουσιάσει διαφορετική συμπεριφορά από αυτήν που παρουσιάζει όταν δεν ιχνοληπτείται. Αν λοιπόν κρατηθεί αυτή η διαστολή σε χαμηλές τιμές, τότε η ανάλυση της επίδοσης της εφαρμογής με βάση τα ίχνη που θα έχουν συλλεχθεί, δεν θα απέχει πολύ από αυτό που επιζητεί ο χρήστης.

**Αποθήκευση των δυναμικών ιχνών.** Το βασικό πρόβλημα της φάσης συλλογής των δυναμικών ιχνών είναι το γεγονός ότι, παρά την χρήση των φορμών ιχνοληψίας, ο όγκος των ιχνών που

[1] Τόσο ακριβείς όσο επιτρέπει το ρολόι του συστήματος, μια και οι μετρήσεις χρόνου γίνονται με την χρήστη της *time*.

συλλέγονται είναι πολύ μεγάλος για να επιτραπεί η αποθήκευσή τους στην κύρια μνήμη του παράλληλου συστήματος στο οποίο εκτελείται η εφαρμογή. Γι' αυτό επιλέγουμε την λύση αποθήκευσης των δυναμικών ιχνών σε αρχεία ιχνοληψίας στον δίσκο (αρχεία ιχνοληψίας δυναμικών δεδομένων). Με αυτόν τον τρόπο αποφεύγουμε κινδύνους υπερχείλισης της μνήμης του συστήματος κατά την εκτέλεση της εφαρμογής. Ομως, με την υιοθέτηση αυτής της λύσης, βρισκόμαστε αντιμέτωποι με το πρόβλημα των χρονοβόρων χειρισμών εισόδου/εξόδου στον δίσκο για την αποθήκευση των δυναμικών ιχνών.

Αν οι χειρισμοί που αναφέρθηκαν, ανατεθούν στην υπό εκτέλεση εφαρμογή, τότε θα προκύψει η ακόλουθη ανεπιθύμητη κατάσταση. Κάθε φορά που ένα ίχνος θα παράγεται, η εφαρμογή θα πρέπει να δαπανεί κάποιον χρόνο για να το γράψει στον δίσκο. Στην καλύτερη περίπτωση θα μπορεί να γίνει κάποια ομαδοποίηση των εγγραφών στο δίσκο 'ωστε να μειωθεί ο συνολικός αριθμός των χειρισμών εισόδου/εξόδου που θα πρέπει να εκτελέσει η ιχνοληπτούμενη εφαρμογή. Και στις δύο περιπτώσεις, η διαστολή του χρόνου εκτέλεσης της εφαρμογής είναι τόσο μεγάλη που καθίσταται απαγορευτική για την υιοθέτηση αυτής της λύσης. Αυτό μας οδηγεί στην αναζήτηση μιας πιο αποτελεσματικής λύσης[2] στην οποία η ιχνοληπτούμενη εφαρμογή θα παράγει τα δυναμικά ίχνη και δεν θα επιφορτίζεται με το κόστος αποθήκευσής τους σε αρχείο δίσκου.

## 5.1 Η διεργασία–συλλέκτης

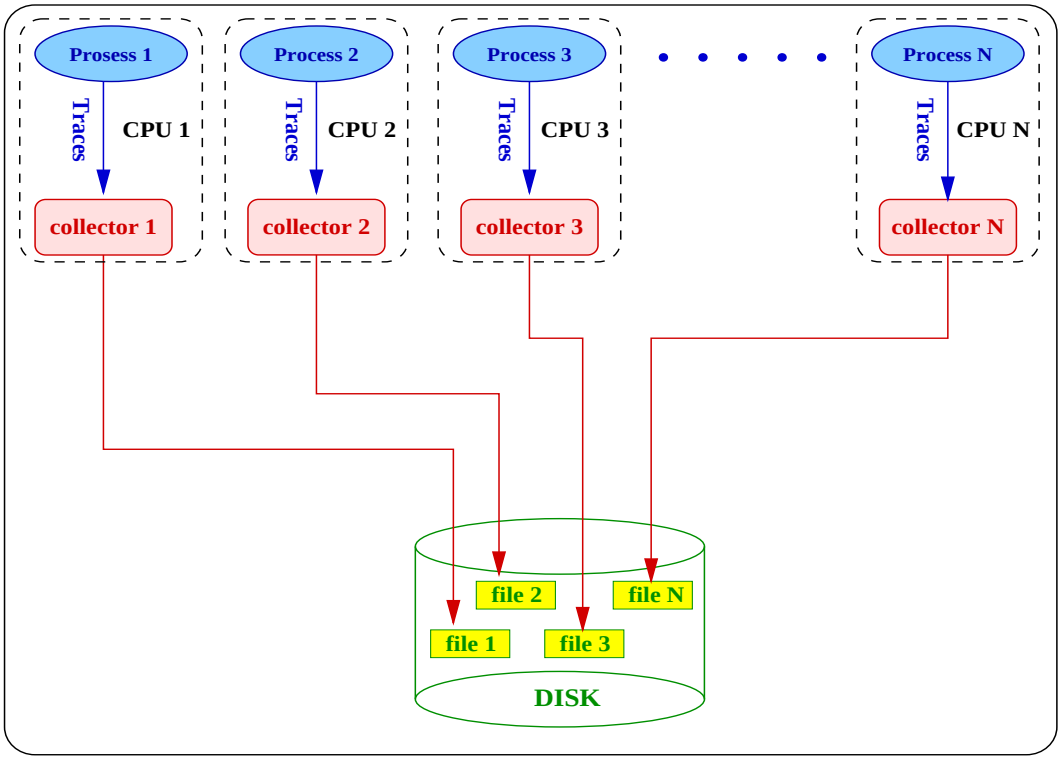
Η λύση του προβλήματος που θίξαμε στην προηγούμενη παράγραφο είναι η χρήση μιας διεργασίας η οποία θα αναλάβει να συλλέγει και να αποθηκεύει τα παραγόμενα ίχνη σε αρχείο δίσκου. Η διεργασία αυτή ανήκει στο *ArrayTracer* και την ονομάζουμε *συλλέκτη*. Η ύπαρξη και λειτουργία αυτής της διεργασίας είναι τελείως ανεξάρτητη από την εκτελούμενη εφαρμογή και συνεπώς δεν επηρεάζει καθόλου τον χρόνο επεξεργαστή (CPU time) της εφαρμογής. Οι αρμοδιότητες της διεργασίας–*συλλέκτη* περιλαμβάνουν :

- την δημιουργία του αρχείου στον δίσκο, όπου θα αποθηκεύονται τα ίχνη που συλλέγονται (*αρχείο ιχνοληψίας δυναμικών δεδομένων*),
- την αποθήκευση των δυναμικών ιχνών που παράγει η εκτελούμενη εφαρμογή στο *αρχείο ιχνοληψίας δυναμικών δεδομένων*,
- την προεπεξεργασία των ιχνών που βρίσκονται στο αρχείο δίσκου μετά το πέρας της εκτέλεσης της εφαρμογής,
- τον συγκερασμό (correlation) των *δυναμικών ιχνών* με τα *στατικά ίχνη* που έχουν αποθη–κευτεί στον δίσκο κατά την διάρκεια της *Στατικής Ανάλυσης* της εφαρμογής,
- την αποστολή των αποτελεσμάτων προεπεξεργασίας και συγκερασμού στο επόμενο λειτουργικό επίπεδο του *ArrayTracer* για την ανάλυση της επίδοσης της παράλληλης εφαρμογής.

Η χρήση αυτής της διεργασίας απαλλάσσει την εφαρμογή από το βάρος της εγγραφής των ιχνών στον δίσκο και έτσι κρατάει σε χαμηλές τιμές την διαστολή σε χρόνο κατοχής επεξεργαστή (CPU time) που προκαλεί η διαδικασία της ιχνοληψίας. Δεν συμβαίνει όμως το ίδιο και με τον συνολικό χρόνο εκτέλεσης της εφαρμογής (elapsed time). Για κάθε διεργασία της παράλληλης εφαρμογής, η διεργασία–*συλλέκτης* που της αντιστοιχεί, εκτελείται στον ίδιο

[2]Μία πιο αποτελεσματική λύση θα ήταν να γίνεται ανάλυση πραγματικού χρόνου (on–the–fly analysis) των παραγόμενων ιχνών, ώστε να εξαληφθεί τελείως το κόστος αποθήκευσης στον δίσκο. Θα αναφερθούμε σε αυτήν στην ενότητα 7.3 που περιγράφει τις μελλοντικές εργασίες μας.

επεξεργαστή με την πρώτη. Ετσι, η εκτέλεση της διεργασίας–*συλλέκτη* επιβαρύνει το χρονο–διαιρετό σύστημα (time sharing system) εκτέλεσης της εφαρμογής και συνεπώς αυξάνει τον συνολικό χρόνο εκτέλεσης της σε σχέση με αυτόν που θα παρατηρούσαμε αν αναλάμβανε η διεργασία δεν συμμετείχε στην διαδικασία ιχνοληψίας. Οσο περισότερη εργασία αναλαμβάνει να πραγματοποιήσει η διεργασία–*συλλέκτης*, τόσο περισσότερο επιβαρύνεται ο συνολικός χρόνος εκτέλεσης της εφαρμογής. Ομως, περιμένουμε ότι στις περιπτώσεις που το εργαλείο μας χρησιμοποιήσει τις δυνατότητες επιλεκτική ιχνοληψίας που έχει, ο όγκος των ιχνών που θα συλλεχθούν θα είναι μικρό και συνεπώς η εργασία της διεργασία–*συλλέκτη* θα είναι λίγη. Πάντως, η χρήση της διεργασία–*συλλέκτη* σε μελλοντικές υλοποιήσεις, είναι ένα θέμα υπό συζήτηση (βλ. ενότητα 7.3).




Σχήμα 5.1: Στιγμιότυπο της διαδικασίας συλλογής ιχνών κατά την διάρκεια εκτέλεσης της εφαρμογής.

Ο αριθμός των διεργασιών–*συλλεκτών* που θα δημιουργηθούν συνολικά για μια παράλληλη εφαρμογή που ιχνοληπτείται, είναι ίσος με τον αριθμό των διεργασιών της παράλληλης εφαρμογής που θα παράγουν ίχνη κατά την εκτέλεσή τους. Δηλαδή, για κάθε διεργασία της εφαρμογής που παράγει ίχνη κατά την εκτέλεσή της υπάρχει και η αντίστοιχη διεργασία–*συλλέκτης*, η οποία είναι προσκολλημένη (attached) πάνω της, είναι υπεύθυνη για την συλλογή των ιχνών που παράγει η πρώτη, και εκτελείται στον ίδιο επεξεργαστή με την διεργασία της εφαρμογής. Με αυτόν τον τρόπο, η ένα–προς–ένα αντιστοίχιση των αρχείων ιχνοληψίας δυναμικών δεδομένων με τις διεργασίες της εφαρμογής που παράγουν ίχνη γίνεται πιο αποδοτική. Το σχήμα 5.1 αναπαριστά ένα τυχαίο στιγμιότυπο της διαδικασίας συλλογής ιχνών από τις διεργασίες– *συλλέκτες* κατά την διάρκεια εκτέλεσης της παράλληλης εφαρμογής.

## 5.2 Επικοινωνία εφαρμογής – συλλέκτη

Η επικοινωνία της εφαρμογής με την διεργασία–*συλλέκτη* γίνεται μέσω ενός *κοινόχρηστου ενταμιευτή* (shared buffer). Η εφαρμογή γράφει τα ίχνη που παράγει στον κοινόχρηστο ενταμιευτή και η διεργασία *συλλέκτης* αναλαμβάνει να τα μεταφέρει από εκεί στο αρχείο δίσκου. Ο κοινόχρηστος ενταμιευτής εκμεταλλεύεται την δυνατότητα κοινόχρηστης μνήμης που προσφέρει το λειτουργικό σύστημα *Unix*.

Το *Unix* μέσω των κλήσεων συστήματος *shmget( ), shmat( )* και *shmctl( )*, επιτρέπει την κατασκευή ενός τμήματος μνήμης το οποίο μπορούν να το χρησιμοποιήσουν πολλές διεργασίες, αρκεί να έχουν πρόσβαση στην μνήμη του συστήματος όπου δημιουργήθηκε το κοινόχρηστο τμήμα μνήμης. Χρησιμοποιώντας τις δυνατότητες αυτές, υλοποιήσαμε τον κοινόχρηστο ενταμιευτή πάνω σε ένα τέτοιο τμήμα μνήμης. Υπεύθυνη για την κατασκευή του είναι η εφαρμογή, ενώ η διεργασία–*συλλέκτης* απλά τον προσαρτίζει στο χώρο διευθύνσεων μνήμης της.

Τόσο το ξεκίνημα της εκτέλεσης της διεργασίας–*συλλέκτη*, όσο και η κατασκευή του κοινόχρηστου ενταμιευτή έχουν ανατεθεί στην εφαρμογή. Το γεγονός αυτό εισάγει καθυστέρηση στον χρόνο εκτέλεσης της εφαρμογής. Ωστόσο προτιμήθηκε για τους εξής λόγους :

- η εφαρμογή αποφασίζει σε ποιο σύστημα θα ξεκινήσει την εκτέλεση της κάθε διεργασίας της. Το εργαλείο μας, με το να αφήνει την διαδικασία εκκίνησης των διεργασιών–*συλλεκτών* στις αντίστοιχες διεργασίες της εφαρμογής, δεν επηρεάζει καθόλου αυτές τις αποφάσεις.

- απλοποιεί πολύ την διαδικασία εκκίνησης της διεργασίας–*συλλέκτη* μειώνοντας την ποσότητα πληροφορίας που πρέπει να ανταλλαχθεί μεταξύ της ιχνοληπτούμενης εφαρμογής και του *ArrayTracer*.

- το συνολικό κόστος της εκκίνησης της διεργασίας–*συλλέκτη* και της κατασκευής του κοινόχρηστου ενταμιευτή, δεν επιβαρύνει πολύ την εφαρμογή γιατί θεωρείται κόστος αρχικοποίησης της διαδικασίας ιχνοληψίας.

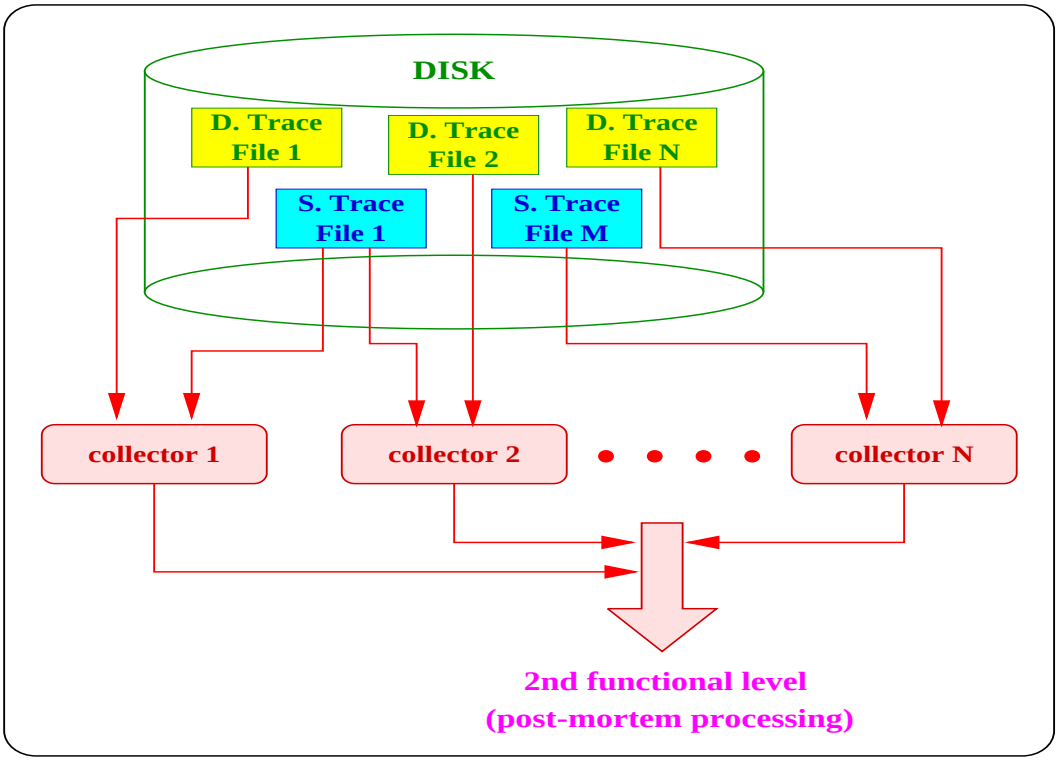
Μέχρι τώρα παρουσιάσαμε την συλλογή των ιχνών από τις διεργασίες–*συλλέκτες* και το σύνολο των ενεργειών τους πάνω σε αυτά. Από αυτήν την παρουσίαση προκύπτει ότι δεν υπάρχει σαφής γραμμή διαχωρισμού του πρώτου λειτουργικού επιπέδου του *ArrayTracer* (που περιλαμβάνει την φάση *Στατικής Ανάλυσης*) από το δεύτερο λειτουργικό επίπεδό του (που περιλαμβάνει την διαδικασία επεξεργασίας των ιχνών μετά το πέρας της εκτέλεσης της εφαρμογής).

Το *ArrayTracer* έχει σχεδιαστεί έτσι ώστε η επεξεργασία των ιχνών να λαμβάνει χώρα μετά το πέρας της εκτέλεσης της ιχνοληπτούμενης εφαρμογής και να γίνεται από μία διεργασία, την διεργασία–*πυρήνα* του εργαλείου. Προκειμένου να επιταχύνουμε την διαδικασία αυτή, οι διεργασίες–*συλλέκτες* εκτελούν μια προεπεξεργασία των ιχνών, ώστε να μειωθεί ο συνολικός όγκος τους, καθώς επίσης και να ελαττωθεί το μέγεθος της εργασίας που θα πρέπει να γίνει στον *πυρήνα* του εργαλείου.

## 5.3 Προεπεξεργασία των ιχνών

Οπως έχει αναφερθεί σε προηγούμενο κεφάλαιο, η χρήση των φορμών ιχνοληψίας δημιουργεί καταστάσεις στις οποίες ένα ίχνος περιέχει πληροφορία που αφορά περισσότερες από μία ενέργειες σε περισότερα από ένα στοιχεία ιχνοληψίας, καθώς επίσης και ενέργειες που δεν αντιστοιχούν σε κανένα στοιχείο ιχνοληψίας. Το ξεκαθάρισμα της πληροφορίας που περιέχουν οι φόρμες ιχνοληψίας γίνεται με την διαδικασία διαχωρισμού των *ουσιωδών* ιχνών

από τα *μη ουσιώδη*. Ακόμα, σε πολλές περιπτώσεις, πολλά από τα συλλεχθέντα ίχνη μπορούν να συνδυαστούν και το συγκεντρωτικό τους αποτέλεσμα να σταλεί στο επόμενο λειτουργικό επίπεδο του εργαλείου. Τέλος, απαιτείται να γίνει ένας συνδυασμός των *στατικών ιχνών* με τα αντίστοιχά τους *δυναμικά ίχνη*, ώστε να σταλεί η συνολική πληροφορία που έχει συλλεχθεί και αφορά την συμπεριφορά της εφαρμογής στο επόμενο λειτουργικό επίπεδο του εργαλείου.




Σχήμα 5.2: Προεπεξεργασία των δυναμικών ιχνών και συγκερασμός του με τα στατικά ίχνη, πριν την αποστολή τους στο δεύτερο λειτουργικό επίπεδο του εργαλείου.

Οι παραπάνω ενέργειες απαιτούν μερικούς απλούς υπολογισμούς και μπορούν εύκολα να γίνουν κατά την διάρκεια ανάκτησης των ιχνών από τα αρχεία δίσκου στα οποία είναι αποθηκευμένα. Η ανάθεσή τους στις διεργασίες–*συλλέκτες*, οι οποίες χειρίζονται τα αρχεία δίσκου όπου έχουν αποθηκευτεί, είναι η ενδεικνυόμενη προσέγγιση. Οι διεργασίες αυτές βοηθάνε στην επιτάχυνση της λειτουργίας της διεργασίας *πυρήνα* του εργαλείου και συγχρόνως μειώνουν τον όγκο της πληροφορίας που θα πρέπει να στείλουν σε αυτήν, ενώ αυξάνουν ελάχιστα την υπολογιστική εργασία των διεργασιών–*συλλεκτών*.

Από την πλευρά της διεργασίας–*πυρήνα* τα οφέλη είναι μεγαλύτερα. Ο όγκος της εργασίας προεπεξεργασίας των ιχνών μπορεί να είναι μικρός από την πλευρά της κάθε διεργασίας–*συλλέκτη*, αλλά αν ανατεθούν συνολικά στην διεργασία–*πυρήνα*, τότε τα πράγματα αλλάζουν τελείως. Ο όγκος της εργασίας αυτής θα είναι ανάλογος του αριθμού των διεργασίων από τις οποίες αποτελείται η παράλληλη εφαρμογή που ιχνοληπτείται. Αυτό σημαίνει ότι για μαζικά παράλληλες εφαρμογές (massively parallel applications) η διεργασία–*πυρήνας* θα επιβαρυνθεί πολύ από αυτές τις απλές ενέργειες και τελικά θα επιμηκυνθεί πολύ ο χρόνος λειτουργίας του *ArrayTracer*.

Η φάση της προεπεξεργασίας στην παρούσα υλοποίση του *ArrayTracer* έχει αναλάβει την επιλογή των *ουσιωδών* ιχνών και τον συγκερασμό των *στατικών* και των *δυναμικών* ιχνών, πριν αυτά σταλούν στο δεύτερο λειτουργικό επίπεδο του εργαλείου. Το σχήμα 5.2 δίνει μια σχηματική αναπαράσταση της φάσης προεπεξεργασίας. Επιστούμε την προσοχή του

αναγνώστη στην ένα–προς–ένα αντιστοιχία *αρχείων ιχνοληψίας δυναμικών δεδομένων* με τις διεργασίες–*συλλέκτες* και κατ' επέκταση με τις ιχνοληπτούμενες διεργασίες της εφαρμογής. Δεν ισχύει η ίδια σχέση ανάμεσα στα *αρχεία ιχνοληψίας στατικών δεδομένων* και τις διεργασίες–*συλλέκτες* αφού είναι δυνατόν το ίδιο *αρχείο ιχνοληψίας στατικών δεδομένων* να αντιστοιχεί σε περισσότερες από μία διεργασίες. Στην πραγματικότητα υπάρχει σχέση ένα–προς–ένα ανάμεσα στα *αρχεία ιχνοληψίας στατικών δεδομένων* και τα αρχεία πηγαίου κώδικα που αντιστοιχούν σε διεργασίες που ιχνοληπτούνται. Ετσι, αν ένα πλήθος διεργασιών εκτελούν τον ίδιο κώδικα και ιχνοληπτούνται, τότε θα έχουν το ίδιο *αρχείο ιχνοληψίας στατικών δεδομένων*, ενώ κάθε μία τους θα έχει ξεχωριστό *αρχείο ιχνοληψίας δυναμικών δεδομένων*.

# Κεφάλαιο 6

# Μετρήσεις και Αξιολόγηση του ArrayTracer

Οπως είδαμε στην αρχή αυτής της αναφοράς, ένα ευρέως αποδεκτό μέτρο αξιολόγησης των διαφόρων τεχνικών ιχνοληψίας, είναι η *διαστολή* που προκαλούν στον χρόνο εκτέλεσης της ιχνοληπτούμενης εφαρμογής. Στην κατηγορία των τεχνικών ιχνοληψίας που βασίζονται στην καθοδήγηση προγράμματος και στην οποία ανήκει αυτή που υιοθετήσαμε, οι συντελεστές *διαστολής* κυμαίνονται σε τιμές από 3 μέχρι 10, όπως είδαμε στον πίνακα 2.1. Στην εισαγωγή του κεφαλαίου 5, αναφέραμε ότι η διαστολή στον χρόνο εκτέλεσης μιας ιχνοληπτούμενης εφαρμογής μπορεί να μετρηθεί σε δύο επίπεδα: σε επίπεδο συνολικού χρόνου εκτέλεσης (elapsed time) και σε επίπεδο χρόνου επεξεργαστή (CPU time). Για να αντιληφθούμε καλύτερα την επιβάρυνση στον χρόνο εκτέλεσης που προκαλεί το *ArrayTracer*, κάναμε μετρήσεις και στα δύο αυτά επίπεδα. Μετρήσαμε τους χρόνους εκτέλεσης τριών εφαρμογών, με και χωρίς ιχνοληψία:

1. Μία σειριακή υλοποίηση ενός αλγορίθμου που πραγματοποιεί εξομάλυνση (smoothing) των τιμών ενός πίνακα ακεραίων. Η εφαρμογή διαβάζει αρχικά τον πίνακα από ένα αρχείο δίσκου, εφαρμόζει τον αλγόριθμο σε όλα τα στοιχεία του και αποθηκεύει τον νέο πίνακα σε αρχείο δίσκου. Η εφαρμογή εκτελέστηκε σε έναν επεξεργαστή *DEC ALPHA 3000*.

2. Μία παράλληλη υλοποίηση του παραπάνω αλγόριθμου εξομάλυνσης, όπου μια διεργασία−αφέντης (master−process) διαβάζει από το αρχείο δίσκου τον πίνακα, στην συνέχεια ξεκινάει έναν προκαθορισμένο αριθμό διεργασιών−σκλάβων (slave−process) και στέλνει ένα τμήμα του πίνακα σε κάθε μία τους. Οι διεργασίες−σκλάβοι πραγ−ματοποιούν τον αλγόριθμο εξομάλυνσης στο τμήμα του πίνακα που τους ανατέθηκε και επιστρέφουν τα αποτελέσματα στην διεργασία−αφέντη, η οποία αναλαμβάνει να τα αποθηκεύσει σε αρχείο δίσκου. Στις μετρήσεις μας χρησιμοποιήσαμε τέσσερεις διεργασίες−σκλάβους και η εφαρμογή εκτελέστηκε σε ένα επεξεργαστή *DEC ALPHA 3000*.

3. Μία παράλληλη υλοποίηση ενός αλγορίθμου για τον υπολογισμό των ενεργειακών τροχιών των ηλεκτρονίων στο μόριο του $H_2O$ σε διάφορες θερμοκρασίες. Η εφαρμογή αποτελείται από μία διεργασία−αφέντη και δύο διεργασίες−σκλάβους και εκτελέστηκε σε έναν επεξεργαστή *DEC ALPHA 3000*.

Για κάθε μία από αυτές τις εφαρμογές δημιουργήσαμε δύο εκτελέσιμες εκδόσεις, μία με και μία χωρίς κώδικα καθοδήγησης. Επιδιώκοντας περισσότερο αξιόπιστα αποτελέσματα,

μετρήσαμε τους χρόνους εκτέλεσης των εφαρμογών σε διαστήματα που τα συστήματα είχαν τον ελάχιστο δυνατό φόρτο[1]. Οι χρόνοι εκτέλεσης των εκδόσεων του αλγορίθμου εξομάλυνσης (σειριακής και παράλληλης) μετρήθηκαν πέντε φορές. Οι χρόνοι εκτέλεσης της εφαρμογής για τον υπολογισμό των ενεργειακών τροχιών των ηλεκτρονίων στο μόριο του $H_2O$ μετρήθηκαν τρεις φορές. Τα συμπεράσματα μας για την επιβάρυνση που προκαλεί το *ArrayTracer* στον χρόνο εκτέλεσης των εφαρμογών βγαίνουν από συγκρίσεις της *μέσης τιμής* των χρόνων που μετρήθηκαν για τις δύο εκδόσεις κάθε εφαρμογής.

## 6.1 Ανάλυση μετρήσεων

Στον πίνακα 6.1 παρουσιάζουμε τα αποτελέσματα των μετρήσεων τα οποία είναι ιδιαίτερα ενθαρρυντικά. Στο σχήμα 6.1 βλέπουμε μία γραφική αναπαράσταση της ποσοστιαίας αύξησης των χρόνων−χρήστη και συνολικού χρόνου εκτέλεσης της εφαρμογής. Αρχικά συγκεντρώνουμε την προσοχή μας στον χρόνο−χρήστη. Οπως φαίνεται από τα αποτελέσματα, ο χρόνος−χρήστη των ιχνοληπτούμενων εφαρμογών είναι δύο με τρεις φορές μεγαλύτερος από τον χρόνο εκτέλεσης των εφαρμογών χωρίς κώδικα ιχνοληψίας. Δηλαδή η διαστολή που προκαλεί το εργαλείο μας στον χρόνο εκτέλεσης μιας εφαρμογής είναι μεταξύ 2 και 3. Οπως προκύπτει από τον πίνακα 2.1, το γεγονός αυτό κατατάσσει το *ArrayTracer* στις κορυφαίες θέσεις αξιολόγησης με βάση το κριτήριο της διαστολής.

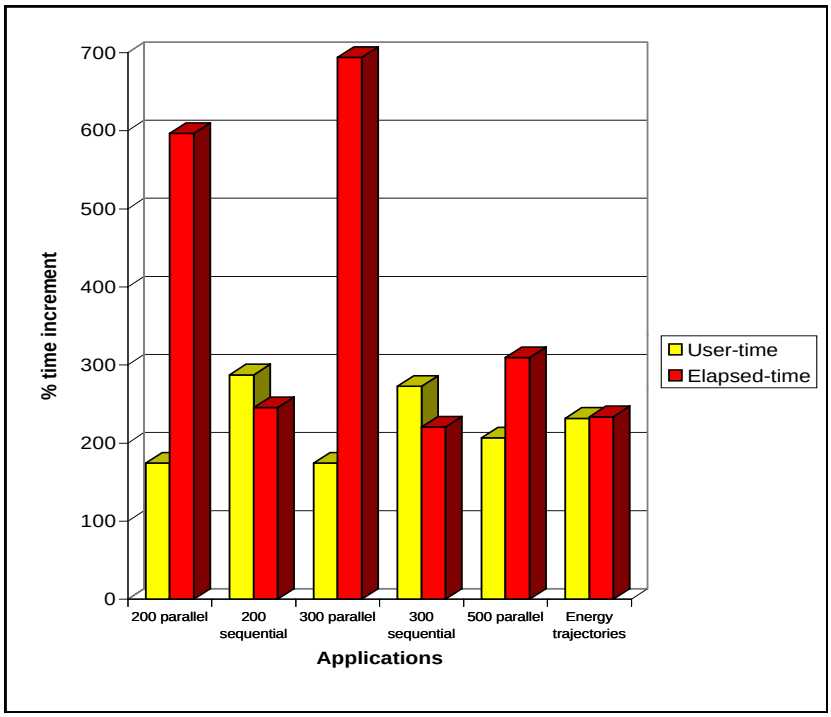
| **APPLICATION** | **USER−TIME** | | **ELAPSED−TIME** | |
|---|---|---|---|---|
| | Original | ArrayTracer | Original | ArrayTracer |
| $200^2$ parallel | 4.572 | 7.976 | 0:0:11.80 | 0:1:10.40 |
| $200^2$ sequential | 5.994 | 17.230 | 0:0:21.20 | 0:0:52.00 |
| $300^2$ parallel | 10.370 | 18.100 | 0:0:23.00 | 0:2:39.60 |
| $300^2$ sequential | 13.780 | 37.610 | 0:0:49.20 | 0:1:48.60 |
| $500^2$ parallel | 51.612 | 106.654 | 1:48.60 | 0:05:36.00 |
| $H_2O$ energy traj. | 38.873 | 90.116 | 14:40:34 | 34:18:17 |

Πίνακας 6.1: Πίνακας αποτελεσμάτων μετρήσεων πάνω στον χρόνο εκτέλεσης των εφαρμογών εξομάλυνσης πινάκων (παράλληλη και σειριακή έκδοση) και υπολογισμού των ενεργειακών τροχιών των ηλεκτρονίων στο μόριο του $H_2O$. Ο χρόνος εμφανίζεται με την μορφή (*ώρες:λεπτά:δευτ/πτα.εκατοστά*).

Θεωρητικά, αν το εργαλείο μας προσέθετε μία κλήση σε ρουτίνα ιχνοληψίας για κάθε εντολή της εφαρμογής, τότε ο χρόνος−χρήστη του προγράμματος θα παρουσίαζε έναν παράγοντα διαστολής ίσο περίπου με 10. Αυτός ο θεωρητικός παράγοντας υπολογίζεται ως εξής. Κάθε ρουτίνα ιχνοληψίας περιέχει κατά μέσο όρο τρείς κλήσεις της συνάρτησης *memcpy( )*. Αν συνυπολογίσουμε και τον χρόνο που απαιτείται για το πέρασμα παραμέτρων σε μια συνάρτηση, τότε ο χρόνος που καταναλώνεται κατά μέσο όρο στην κλήση κάθε ρουτίνας ιχνοληψίας είναι περίπου ίσος με 10. Ακόμα και σε αυτήν την περίπτωση η διαστολή που θα προκαλούσε το εργαλείο μας στον χρόνο εκτέλεσης της εφαρμογής θα ήταν μέσα στο αποδεκτό διάστημα για μεθόδους καθοδήγησης.

Η πειραματική παρατήρηση παράγοντα διαστολής μεταξύ 2 και 3 οφείλεται στο γεγονός ότι στην πράξη δεν εισάγεται μία κλήση ρουτίνας ιχνοληψίας για κάθε εντολή της εφαρμογής. Σημειώνουμε σε αυτό το σημείο ότι στις μετρήσεις αυτές δεν χρησιμοποιήθηκαν φόρμες

[1]Ωστόσο, δεν μπορέσαμε να εξασφαλίσουμε σύστημα στο οποίο να μην εκτελείται καμμία άλλη εφαρμογή. Ετσι, η ακρίβεια των μετρήσεων του συνολικού χρόνου εκτέλεσης των εφαρμογών, αντιμετωπίζει τα προβλήματα που αναφέρθηκαν στο κεφάλαιο 5.




Σχήμα 6.1: Γραφική αναπαράσταση των αποτελεσμάτων από τις μετρήσεις των χρόνων–χρήστη που αφορούν τις εκτελέσεις των εφαρμογών με και χωρίς κώδικα καθοδήγησης.

ιχνοληψίας αλλά κάθε ίχνος που παρήγαγε η εφαρμογή αφορούσε μία μόνο προσπέλαση στην μνήμη. Περιμένουμε λοιπόν, όταν χρησιμοποιηθούν οι φόρμες ιχνοληψίας, να πετύχουμε παράγοντες διαστολής μικρότερους του 2!

Τα παραπάνω αφορούν τους χρόνους–χρήστη που παρατηρήθηκαν. Οσον αφορά τους συνολικούς χρόνους εκτέλεσης των ιχνοληπτούμενων εφαρμογών, παρατηρούμε ότι τα αποτελέσματα είναι ανάλογα με αυτά των χρόνων–χρήστη που μελετήσαμε παραπάνω. Εξαίρεση αποτελεί η περίπτωση της παράλληλης υλοποίηση του αλγορίθμου εξομάλυνσης. Στις μετρήσεις που αφορούν το συγκεκριμένο πρόγραμμα, διαπιστώνουμε μια διαστολή του συνολικού χρόνου εκτέλεσης κοντά στο 700%. Το νούμερο αυτό εξηγείται αν λάβουμε υπόψη μας ότι η εφαρμογή αποτελείται από πέντε διεργασίες συνολικά (μία διεργασία–αφέντης και τέσσερεις διεργασίες–σκλάβοι). Ολες οι διεργασίες ιχνοληπτούνται, άρα για κάθε μία από αυτές έχει δημιουργηθεί μια διεργασία–*συλλέκτης*. Ολες οι διεργασίες της εφαρμογής μαζί με τις διεργασίες–*συλλέκτες* που τους αντιστοιχούν, εκτελούνται στον ίδιο επεξεργαστή. Ετσι, ο επεξεργαστής αυτός έχει ένα μεγάλο αριθμό διεργασιών να εκτελέσει, γεγονός που προκαλεί αυτήν την αύξηση στον συνολικό χρόνο εκτέλεσης που παρατηρείται. Ο ισχυρισμός αυτός επιβεβαιώνεται από τα αποτελέσματα των μετρήσεων της παράλληλης υλοποίησης του αλγορίθμου εξομάλυνσης για πίνακα 500x500. Οι μετρήσεις αυτές πραγματοποιήθηκαν σε ένα σύνολο από 5 μηχανήματα (*DEC ALPHAs*). Κάθε διεργασία της εφαρμογής έτρεξε σε διαφορετικό μηχάνημα και έτσι αποφύγαμε τα προβλήματα της προηγούμενης περίπτωσης. Η διαστολή που παρατηρούμε σε αυτήν την περίπτωση είναι μεταξύ 2 και 3 όπως και στις περιπτώσεις των σειριακών υλοποιήσεων του αλγορίθμου εξομάλυνσης.

Οπως αναφέραμε αρχικά, δίνουμε περισσότερη έμφαση στις μετρήσεις που αφορούν τον χρόνο–χρήστη της εφαρμογής. Υπάρχουν δύο σημαντικοί λόγοι που παρουσιάζουμε αυτόν τον χρόνο και όχι τον συνολικό χρόνο που χρησιμοποιήθηκε ο επεξεργαστής του συστήματος

από την εφαρμογή (cpu−time) :

1. Η διαστολή στον χρόνο εκτέλεσης της εφαρμογής που προκαλεί ο κώδικας καθοδήγησης επηρεάζει κατά κύριο λόγο τον χρόνο του χρήστη (user−time).

2. Οι μετρήσεις χρόνου που αφορούν μόνο τον χρόνο που δαπανήθηκε από τον επεξεργαστή εκτελώντας εντολές του χρήστη (user−time) είναι εξαιρετικά ακριβείς σε σχέση με αντίστοιχες μετρήσεις που περιέχουν και τον χρόνο που δαπάνησε ο επεξεργαστής εκτελώντας εντολές του συστήματος (system time).

Για να αντιληφθούμε καλύτερα τον πρώτο λόγο που αναφέρθηκε, θα αναλύσουμε σύντομα το περιεχόμενο του κώδικα καθοδήγησης και την διαστολή που προκαλεί. Ο κώδικας καθοδήγησης που εισάγεται στην εφαρμογή περιλαμβάνει εντολές για την κατασκευή του κοινόχρηστου ενταμιευτή, για την εκκίνηση της διεργασίας−*συλλέκτη* και ένα σύνολο από κλήσεις σε συναρτήσεις της βιβλιοθήκης ιχνοληψίας του *ArrayTracer*. Από αυτές τις εντολές, όσες αφορούν την κατασκευή του ενταμιευτή και την εκκίνηση της διεργασίας−*συλλέκτη* περιέχουν κλήσεις συστήματος (system calls), οι οποίες προκαλούν διαστολή στον χρόνο συστήματος (system time). Ομως, αυτές οι κλήσεις συστήματος είναι λίγες (μία σε *fork( )*, μία σε *write( )*, μία σε *shmget( )* και μία σε *shmat( )*) καί συμπεριλαμβάνονται στην αρχικοποίηση του συστήματος ιχνοληψίας. Κατά συνέπεια μπορούμε να αγνοήσουμε την επιβάρυνση που προκαλούν στον χρόνο εκτέλεσης της εφαρμογής.

Από την άλλη πλευρά, οι κλήσεις στις συναρτήσεις της βιβλιοθήκης του *ArrayTracer* δεν περιέχουν καμμία κλήση συστήματος (system call) και έτσι δεν προκαλούν καθόλου διαστολή στον χρόνο συστήματος (system time). Αντίθετα, επιβαρύνουν τον χρόνο του χρήστη (user time) της ιχνοληπτούμενης εφαρμογής. Γι' αυτό και μπορούμε να εκτιμήσουμε την διαστολή που προκαλεί το εργαλείο μας στον χρόνο εκτέλεσης της εφαρμογής, μετρώντας μόνο τον χρόνο του χρήστη που χρειάστηκε κατά την εκτέλεση της η εφαρμογή, χωρίς να έχουμε μεγάλο σφάλμα όσον αφορά την πραγματική διαστολή.

Επίσης, η χρήση μόνο του χρόνου−χρήστη, μας παρέχει ακρίβεια στις μετρήσεις μας, την οποία δεν θα πετυχαίναμε μετρώντας τον συνολικό χρόνο του επεξεργαστή που χρειάστηκε η ιχνοληπτούμενη εφαρμογή κατά την εκτέλεση της. Είναι γνωστό ότι οι συναρτήσεις χρονομέτρησης που παρέχει το *Unix* δεν προσφέρουν ακρίβεια στο χρόνο συστήματος. Αυτό συμβαίνει γιατί η ακρίβεια δεν είναι εφικτή (ένα *read( )* μπορεί να πάρει λιγότερο χρόνο αν η κεφαλή του δίσκου βρίσκεται ακριβώς πάνω στο αρχείο από το οποίο διαβάζουμε, από ότι στην περίπτωση που βρίσκεται στο αντιδιαμετρικό σημείο του δίσκου). Συμβαίνει επίσης γιατί οι συναρτήσεις χρονομέτρησης του *Unix* δεν έχουν την δυνατότητα να διακρίνουν τον χρόνο συστήματος που αφορά την εφαρμογή του χρήστη από τον χρόνο που αφορά διακοπές (interrupts) που συμβαίνουν στο σύστημα κατά την διάρκεια εκτέλεσης της εφαρμογής.

## 6.2 Πειράματα

Θα προσπαθήσουμε να παρουσιάσουμε στον αναγνώστη τα πρακτικά οφέλη που αποκομίζουμε από την χρήση της μεθόδου *επιλεκτικής ιχνοληψίας* μέσα από την σύγκριση των χρόνων εκτέ−λεσης μιας εφαρμογής όταν αυτή ιχνοληπτείται από το εργαλείο μας και όταν ιχνοληπτείται από το *ATOM* [13, 14, 46]. Διαλέξαμε το *ATOM* γιατί είναι ένα από τα πιο γρήγορα εργαλεία συλλογής ιχνών που στηρίζουν την διαδικασία ιχνοληψίας στην καθοδήγηση του προγράμ−ματος. Τα συμπεράσματα στα οποία καταλήγουμε από αυτήν την σύγκριση, αποτελούν την βάση για την συζήτηση που ακολουθεί. Επίσης, θα αποτελέσουν το μέτρο αξιολόγησης του *ArrayTracer* σχετικά με τα εργαλεία που χρησιμοποιούνται για την ιχνοληψία εφαρμογών και

| | | 100x100 with disk | 300x300 no disk | 300x300 no disk (arrays only) |
|---|---|---|---|---|
| **User time** | **Original** | 1.894 | 0.134 | 0.134 |
| | **ATOM** | 65.462 | 9.030 | 9.030 |
| | **ArrayTracer** | 5.426 | 43.436 | 3.746 |
| **Elapsed time** | **Original** | 0:03.4 | 0:00.2 | 0:00.2 |
| | **ATOM** | 1:39.4 | 0:25.0 | 0:25.0 |
| | **ArrayTracer** | 0:14.0 | 1:54.6 | 0:10.8 |
| **Trace File size** | **ATOM** | 135,695,136 | 1,943,812 | 1,943,812 |
| | **ArrayTracer** | 12,181,904 | 11,060,504 | 1,383,256 |

Πίνακας 6.2: Χρόνοι εκτέλεσης της εφαρμογής εξομάλυνσης όταν τα δεδομένα εισόδου προέρχονται από αρχείο δίσκου και όταν προέρχονται από την κύρια μνήμη. Ο χρόνος εμφανίζεται με την μορφή (*λεπτά:δευτ/πτα.εκατοστά*) και τα μεγέθη των αρχείων ιχνοληψίας είναι σε *Bytes*.

οποία βασίζονται στην εισαγωγή κώδικα καθοδήγησης στον κώδικα μηχανής της εφαρμογής [36, 13, 14, 16], καθώς και στην προσομοίωση της εκτέλεσης της εφαρμογής [48, 5, 6].

Η σύγκριση που παρουσιάζεται με αριθμητικά δεδομένα στον πίνακα 6.2 , έγινε πάνω στους χρόνους εκτέλεσης της σειριακής υλοποίησης του αλγορίθμου εξομάλυνσης των τιμών ενός πίνακα 100x100. Δημιουργήθηκαν τρεις εκδόσεις: μία έκδοση που δεν συμμετείχε σε διαδικασία ιχνοληψίας, μία έκδοση που διάβαζε τα δεδομένα εισόδου από αρχείο δίσκου και μία τρίτη η οποία είχε τα δεδομένα εισόδου στην κύρια μνήμη της. Ολες οι εκδόσεις εκτελέστηκαν σε ένα επεξεργαστή *DEC ALPHA 3000*. Η έκδοση του *ArrayTracer* που χρησιμοποιήθηκε σε αυτές τις μετρήσεις δεν έκανε χρήση φορμών ιχνοληψίας (trace templates).

Η μεγάλη διαφορά που παρατηρείται στους χρόνους εκτέλεσης, οφείλεται κυρίως σε δύο λόγους :

1. Το *ATOM* για να ιχνοληπτήσει όλες τις προσπελάσεις στην μνήμη της εφαρμογής, καλεί μια ρουτίνα ιχνοληψίας σε *κάθε εντολή* του προγράμματος. Η ρουτίνα αυτή ελέγχει αν πρόκειται για εντολή προσπέλασης της μνήμης (load/store instruction) και τότε μόνο γράφει στο αρχείο ιχνοληψίας στον δίσκο την θέση μνήμης που προσπελάστηκε. Ετσι, η εκτέλεση της εφαρμογής διακόπτεται σε κάθε εκτέλεση μίας εντολής της, ανεξάρτητα αν πρόκειται για εντολή προσπέλασης της μνήμης ή όχι. Στο *ArrayTracer*, διακοπή της εκτέλεση της εφαρμογής συμβαίνει μόνο στα σημεία που έχει εισαχθεί ο κώδικας καθοδήγησης, δηλαδή μόνο στις εντολές που προσπελαύνουν την μνήμη.

2. Στην έκδοση της εφαρμογής που ιχνοληπτείται από το *ATOM*, η εγγραφή των ιχνών στο αρχείο ιχνοληψίας στον δίσκο γίνεται την στιγμή που παράγεται το κάθε ίχνος. Ετσι, κάθε παραγωγή ίχνους επιβαρύνεται με το κόστος των χειρισμών πρόσβασης δίσκου (disk I/O operations). Αντίθετα, στην εφαρμογή που ιχνοληπτείται από το *ArrayTracer*, η εγγραφή των ιχνών στο αρχείο ιχνοληψίας στον δίσκο γίνεται μόνο όταν ένα τμήμα του κοινόχρηστου ενταμιευτή γεμίσει. Ετσι, υπολογίζεται ότι για ενταμιευτή μεγέθους 1KByte, όπως αυτό που χρησιμοποιήθηκε στις μετρήσεις, ένας χειρισμός δίσκου συμβαίνει κάθε δεκατέσσερεις παραγωγές ιχνών[2].

[2]Δεδομένου του περιορισμού που έχουμε θέσει, να χωράει ολόκληρος ο ενταμιευτής σε μία σελίδα μνήμης του συστήματος, μπορούμε να αυξήσουμε το μέγεθος του ενταμιευτή οχτώ φορές γιατί το μέγεθος της σελίδας μνήμης

Από τους παραπάνω λόγους γίνεται φανερό ότι το *ATOM* δαπανάει πολύ χρόνο σε χειρισμούς οι οποίοι δεν είναι απαραίτητοι για την παραγωγή της πληροφορίας που έχει ζητήσει ο χρήστης. Η διαφορά αυτή στους χρόνους που μετρήθηκαν μεταβάλλεται ευνοώντας το *ArrayTracer*, όταν ο χρήστης επιλέξει να ιχνοληπτήσει συγκεκριμένες μεταβλητές της εφαρμογής. Αυτό συμβαίνει γιατί το *ATOM* δεν μπορεί να αποφύγει την διακοπή της εκτέλεσης της εφαρμογής σε κάθε εντολή προσπέλασης της μνήμης για να ελέγξει αν πρέπει να ενεργοποιήσει την διαδικασία ιχνοληψίας. Αντίθετα, το *ArrayTracer*, χρησιμοποιώντας την τεχνική *επιλεκτικής ιχνοληψίας*, καταφέρνει να διακόπτει την εκτέλεση της εφαρμογής ακριβώς μόνο τις φορές που πρέπει να παραχθεί ένα ίχνος.

Για να συγκρίνουμε την απόδοση των δύο εργαλείων, μετρήσαμε τους χρόνους που χρειάζονται για να ιχνοληπτήσουν εφαρμογές που δεν περιέχουν χειρισμούς δίσκου (disk I/O operations). Επειδή το *ATOM* μπορεί να ιχνοληπτήσει μόνο σειριακές εφαρμογές, για την σύγκριση των δύο εργαλείων επιλέξαμε τις ακόλουθες σειριακές εφαρμογές:

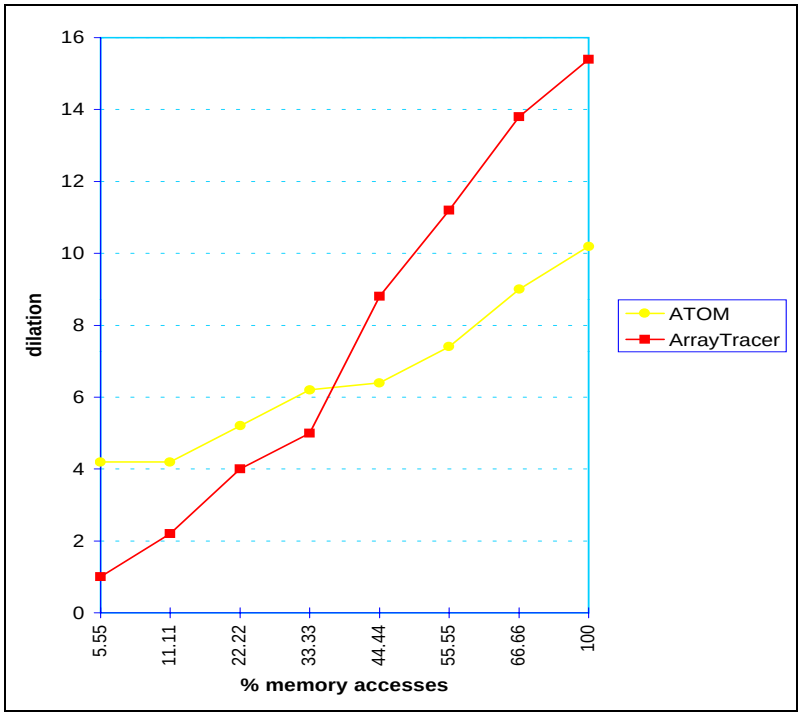
- Την εφαρμογή εξομάλυνσης για πίνακα 300x300. Τα αποτελέσματα των μετρήσεων βρίσκονται στον πίνακα 6.3 και παριστάνονται γραφικά στο σχήμα 6.2.
- Μία εφαρμογή που υπολογίζει την θέση ενός σημείου στον τρισδιάστατο χώρο μετά από διαδοχικές κινήσεις του. Κάθε κίνηση του σημείου περιλαμβάνει μία περιστροφή περί τυχαίου άξονα και μία μεταφορά του προς τυχαία διεύθυνση. Για τον υπολογισμό της θέσης του σημείου χρησιμοποιούνται πίνακες ομογενών συντεταγμένων (πίνακες 4x4). Στην εφαρμογή που ιχνοληπτήθηκε υπολογίστηκε η θέση του σημείου στον τρισδιάστατο χώρο μετά από 10,000 διαδοχικές κινήσεις του. Τα αποτελέσματα των μετρήσεων που αφορούν αυτήν την εφαρμογή, βρίσκονται στον πίνακα 6.4 και παριστάνονται γραφικά στο σχήμα 6.3.
- Μία εφαρμογή πολλαπλασιασμού πέντε πινάκων 100x100. Τα αποτελέσματα βρίσκονται στον πίνακα 6.5 και παριστάνονται γραφικά στο σχήμα 6.4

| | **USER−TIME** | | **ELAPSED−TIME** | | **TRACE−FILE SIZE** | |
|---|---|---|---|---|---|---|
| **%** | **ATOM** | **ArrayTracer** | **ATOM** | **ArrayTracer** | **ATOM** | **ArrayTracer** |
| 5.55% | 4.086 | 0.454 | 0:04.2 | 0:01.0 | 856,089 | 921,970 |
| 11.11% | 4.314 | 0.788 | 0:04.2 | 0:02.2 | 1,731,639 | 1,864,870 |
| 22.22% | 4.762 | 1.492 | 0:05.2 | 0:04.0 | 3,482,739 | 3,750,670 |
| 33.33% | 5.224 | 2.178 | 0:06.2 | 0:05.0 | 5,233,839 | 5,636,470 |
| 44.44% | 5.654 | 2.790 | 0:06.4 | 0:08.8 | 6,984,939 | 7,522,270 |
| 55.55% | 6.960 | 3.464 | 0:07.4 | 0:11.2 | 8,736,039 | 9,408,070 |
| 66.66% | 7.270 | 4.546 | 0:09.0 | 0:13.8 | 10,483,252 | 11,289,684 |
| 100.0% | 7.736 | 4.878 | 0:10.2 | 0:15.4 | 11,653,252 | 12,549,740 |

Πίνακας 6.3: Συγκριτικός πίνακας επιλεκτικής ιχνοληψίας της εφαρμογής *εξομάλυνσης* με το *ATOM* και με το *ArrayTracer*. Η πρώτη στήλη περιέχει το ποσοστό προσπελάσεων πινάκων που ιχνοληπτήθηκαν. Ο χρόνος εμφανίζεται με την μορφή (*λεπτά:δευτ/πτα.εκατοστά*) και το μέγεθος των αρχείων ιχνοληψίας είναι σε *Bytes*.

Από τις μετρήσεις στον χρόνο ιχνοληψίας των δύο πρώτων εφαρμογών διαπιστώνουμε ότι το *ArrayTracer* προκαλεί μικρότερη διαστολή στις περιπτώσεις που ιχνοληπτείται λιγότερο

των συστήματων *DEC ALPHA 3000* είναι 8KBytes. Αυτό θα οδηγούσε στην εμφάνιση χειρισμών δίσκου οχτώ φορές πιο σπάνια, δηλαδή κάθε 112 παραγωγές ιχνών.




Σχήμα 6.2: Γραφική παράσταση της διαστολής που προκαλούν στον χρόνο εκτέλεσης της εφαρμογής *εξομάλυνσης* τα *ATOM* και *ArrayTracer*.

από το 35% των προσπελάσεων σε μεταβλητές της εφαρμογής. Στην περίπτωση της τρίτης εφαρμογής, το όριο αυτό μεταφέρεται στο 65% περίπου. Το φαινόμενο αυτό οφείλεται στο γεγονός ότι το *ATOM* ιχνοληπτεί τις προσπελάσεις θέσεων μνήμης του συστήματος ενώ το *ArrayTracer* ιχνοληπτεί τις προσπελάσεις σε μεταβλητές του προγράμματος. Στις δύο πρώτες εφαρμογές ένα μεγάλο μέρος των μεταβλητών του μπήκε στους καταχωρητές του συστήματος. Ετσι, ενώ το *ArrayTracer* παρήγαγε ίχνη για όλες τις προσπελάσεις στις μεταβλητές που του είχε καθορίσει ο χρήστης, το *ATOM* παρήγαγε ίχνη μόνο για τις πραγματικές προσπελάσεις στην μνήμη. Για αυτόν τον λόγο τα μεγέθη των αρχείων ιχνοληψίας δεν είναι ίσα. Από την άλλη πλευρά, στην εφαρμογή πολλαπλασιασμού πινάκων οι πίνακες της εφαρμογής έχουν διάσταση μεγαλύτερη από αυτή που θα επέτρεπε στον μεταφραστή να τους τοποθετήσει σε καταχωρητές. Αυτό σημαίνει ότι όλες οι προσπελάσεις σε στοιχεία πίνακα αντιστοιχούν σε προσπελάσεις θέσεων της μνήμης άρα ιχνοληπτούνται και από τα δύο εργαλεία.

Αυτό που γίνεται φανερό από τις μετρήσεις είναι ότι υπάρχει μία περιοχή στην οποία το *ArrayTracer* αποδίδει καλύτερα από το *ATOM*. Πιο συγκεκριμένα, όσο μικρότερο[3] είναι το ποσοστό των προσπελάσεων σε μεταβλητές που έχουν επιλεχθεί για ιχνοληψία, τόσο μικρότερη είναι η διαστολή που προκαλεί το εργαλείο μας. Αυτό είναι το σημαντικότερο όφελος από την χρήση της *επιλεκτικής ιχνοληψίας*. Στην πράξη αυτό είναι πολύ χρήσιμο στον χώρο της διάγνωσης προβλημάτων επίδοσης (performance debugging) μίας εφαρμογής γιατί ο προγραμματιστής έχει υπ' όψη του συγκεκριμένα σημεία του κώδικα τα οποία επιθυμεί να αναλύσει. Σε αυτές ακριβώς τις περιπτώσεις περιμένουμε ότι το *ArrayTracer* θα αποδειχθεί ιδιαίτερα χρήσιμο.

Στο σχήμα 6.4 την προσοχή μας τραβάει η καμπύλη του *ATOM* όταν γίνεται ιχνοληψία μεγάλου ποσοστού των προσπελάσεων μνήμης της εφαρμογής. Ειδικότερα παρατηρούμε

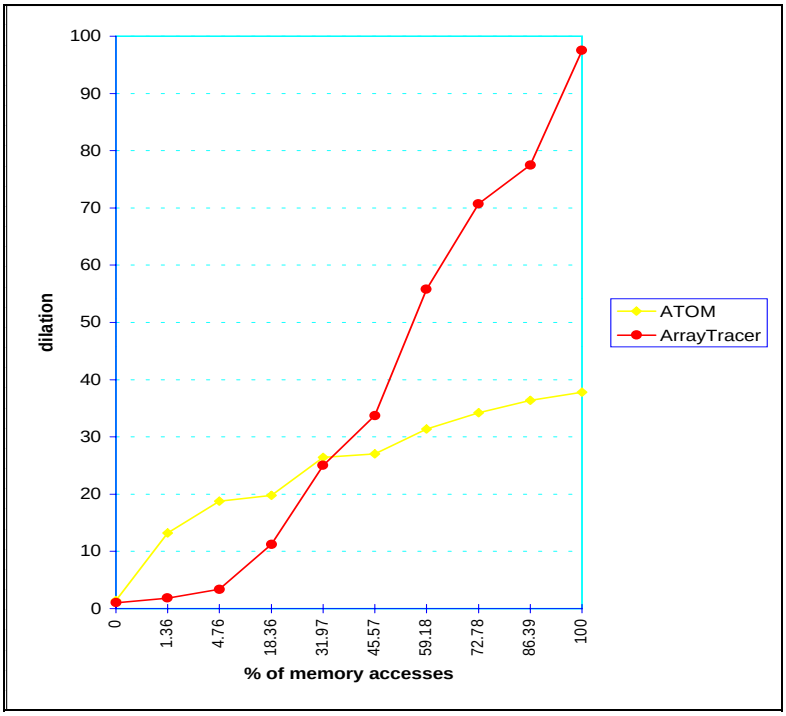
[3]Οι τιμές 35% και 65% που προκύπτουν από τις μετρήσεις, αποτελούν χαρακτηριστικό της φυσιογνωμίας της εφαρμογής και όχι μία οριακή τιμή για την απόδοση του *ArrayTracer* σε σχέση με αυτήν του *ATOM*.

| | USER−TIME | | ELAPSED−TIME | | TRACE−FILE SIZE | |
|---|---|---|---|---|---|---|
| % | ATOM | ArrayTracer | ATOM | ArrayTracer | ATOM | ArrayTracer |
| 1.36% | 12.446 | 0.694 | 0:13.20 | 0:01.80 | 1,040,091 | 1,120,126 |
| 4.76% | 15.064 | 1.638 | 0:18.80 | 0:03.40 | 9,880,091 | 3,920,154 |
| 18.36% | 16.104 | 5.258 | 0:19.80 | 0:11.25 | 12,480,299 | 14,488,969 |
| 31.97% | 18.944 | 8.505 | 0:26.40 | 0:25.00 | 21,320,507 | 22,960,658 |
| 45.57% | 19.852 | 13.090 | 0:27.00 | 0:33.75 | 23,920,715 | 33,529,473 |
| 59.18% | 22.912 | 23.567 | 0:31.40 | 0:55.75 | 32,630,923 | 41,861,162 |
| 72.78% | 23.962 | 31.362 | 0:34.20 | 1:10.75 | 36,790,923 | 53,061,190 |
| 86.39% | 25.460 | 33.434 | 0:36.40 | 1:17.50 | 40,950,923 | 64,261,218 |
| 100.0% | 26.814 | 39.585 | 0:37.80 | 1:37.50 | 45,110,923 | 68,855,637 |

Πίνακας 6.4: Συγκριτικός πίνακας επιλεκτικής ιχνοληψίας της εφαρμογής *κίνησης στον τρισ−διάστατο χώρο* με το *ATOM* και με το *ArrayTracer*. Η πρώτη στήλη περιέχει το ποσοστό προσπελάσεων πινάκων που ιχνοληπτήθηκαν. Ο χρόνος εμφανίζεται με την μορφή (*λεπτά:δευτ/πτα.εκατοστά*) και το μέγεθος των αρχείων ιχνοληψίας είναι σε *Bytes*.

ότι η διαστολή που προκαλεί όταν ιχνοληπτείται το 100% των προσπελάσεων στην μνήμη του συστήματος είναι μικρότερη από αυτή που μετρήθηκε για την ιχνοληψία του 99.88% των προσπελάσεων. Το γεγονός αυτό εξηγείται από τον τρόπο με τον οποίο μπορούμε να επιτύχουμε *επιλεκτική ιχνοληψία* με το *ATOM*. Η τεχνική εισαγωγής κώδικα καθοδήγησης σε αυτό το εργαλείο για την ιχνοληψία των προσπελάσεων μνήμης, είναι η ακόλουθη: ο χρήστη γράφει μία ρουτίνα που περιέχει τον κώδικα που καθορίζει τι ακριβώς θα συμβεί κατά την συλλογή ενός ίχνους. Το *ATOM* εισάγει τον κώδικα αυτής της ρουτίνας πριν την εκτέλεση κάθε εντολή προσπέλασης μνήμης (load/store) που υπάρχει στον εκτελέσιμο κώδικα της εφαρμογής. Για να μπορέσει ο χρήστης να επιλέξει την ιχνοληψία μερικών μόνο θέσεων μνήμης, εισάγει στον κώδικα της ρουτίνας ιχνοληψίας εντολές που ελέγχουν αν η διεύθυνση που προσπελαύνεται κάθε φορά, ανήκει σε αυτές που επιθυμεί να ιχνοληπτήσει. Ετσι, όταν ο χρήστης έχει επιλέξει να ιχνοληπτηθεί το 99.88% των προσπελάσεων μνήμης, οι εντολές ελέγχου που προαναφέραμε είναι αρκετά πολύπλοκες ώστε να καταναλώνουν υπολογίσιμο χρονικό διάστημα. Αντίθετα, στην περίπτωση που ιχνοληπτούνται όλες οι προσπελάσεις μνήμης, τότε αυτές οι εντολές ελέγχου δεν χρειάζονται και συνεπώς ο χρόνος που απαιτείται είναι μικρότερος από αυτόν που αντιστοιχεί στην ιχνοληψία του 99.88% των προσπελάσεων μνήμης.

Από όσα ειπώθηκαν στην προηγούμενη παράγραφο γίνεται φανερό ότι για να μπορέσουμε να ιχνοληπτήσουμε επιλεκτικά ορισμένες μεταβλητές με το *ATOM*, πρέπει να γνωρίζουμε τις θέσεις μνήμης (virtual addresses) που καταλαμβάνουν ώστε να τις χρησιμοποιήσουμε στις εντολές ελέγχου της ρουτίνας ιχνοληψίας που προαναφέραμε. Για να γνωρίσουμε λοιπόν τις δευθύνσεις συγκεκριμένων μεταβλητών χρειάστηκε να εισάγουμε στην εφαρμογή εντολές που τυπώνουν αυτές τις διευθύνσεις και να την εκτελέσουμε μία φορά ώστε να μάθουμε ποιές διευθύνσεις αντιστοιχούν στις μεταβλητές που μας ενδιαφέρουν. Με αυτόν τον τρόπο, η διαδικασία καθορισμού των παραμέτρων ιχνοληψίας για το *ATOM* έγινε πολύπλοκη σε σχέση με αυτήν του *ArrayTracer* όπου απλά χρειαζόταν να καθορίσουμε το όνομα της μεταβλητής που θέλαμε να ιχνοληπτήσουμε. Την πολυπλοκότητα αυτή αυξάνει το γεγονός ότι στην γλώσσα προγραμματισμού *Fortran*, δεν έχουμε άμεση πρόσβαση στις διευθύνσεις των μεταβλητών του κώδικα. Ετσι, χρειάστηκε να κατασκευαστεί μία ρουτίνα σε γλώσσα **C** η οποία επέστρεφε την διεύθυνση του ορίσματος της, να μετατραπεί σε βιβλιοθήκη και να συνδεθεί (linked) με το εκτελέσιμο *Fortran* πρόγραμμα.




Σχήμα 6.3: Γραφική παράσταση της διαστολής που προκαλούν στον χρόνο εκτέλεσης της εφαρμογής *κίνησης στον τρισδιάστατο χώρο* τα *ATOM* και *ArrayTracer*..

Στο τέλος αυτής της ενότητας πάνω στην σύγκριση του *ArrayTracer* με το *ATOM* θα πρέπει να σημειώσουμε την διαφορά στον χρόνο που χρειάζονται τα δύο εργαλεία για να εισάγουν τον κώδικα καθοδήγησης στον κώδικα της εφαρμογής που πρόκειται να ιχνοληπτηθεί. Μετρώντας τον χρόνο που χρειάστηκαν τα δύο εργαλεία για να εισάγουν κώδικα καθοδήγησης και να δημιουργήσουν το εκτελέσιμο που θα παράγει ίχνη πήραμε τα εξής αποτελέσματα: Το *ArrayTracer* χρειάστηκε περίπου 10 δευτερόλεπτα, ενώ το *ATOM* χρειάστηκε 40–60 δευτερόλεπτα αντίστοιχα.

Οι πειραματισμοί και οι παρατηρήσει πάνω στην συμπεριφορά του *ArrayTracer* συνεχίζονται με την σύγκρισή του με το *HeNCE*, ένα εργαλείο που παρακολουθεί την ενδοεπικοινωνία διεργασιών πάνω από *PVM*.

## 6.3 Συμπεράσματα

Οι μετρήσεις και η σύγκριση με το *ATOM* μας οδήγησε σε εξής συμπεράσματα:

- Το εργαλείο μας παρουσιάζει πολύ καλές επιδόσεις όταν γίνεται χρήση της δυνατότητας *επιλεκτικής ιχνοληψίας* που προσφέρει. Αντίθετα, όταν ο χρήστης ζητήσει την ιχνοληψία όλων των μεταβλητών της εφαρμογής, οι επιδόσεις του *ArrayTracer* είναι πολύ φτωχές και ανάλογα με την φυσιογνωμία της εφαρμογής μπορεί να σημειωθεί διαστολή του χρόνου εκτέλεσης της μέχρι και 95! Συνεπώς, το εργαλείο μας είναι κατάλληλο για τον χρήστη που επιθυμεί να επικεντρώσει την ανάλυση της συμπεριφοράς της εφαρμογής του σε λίγες μόνο παραμέτρους τις οποίες έχει επιλέξει. Στις περιπτώσεις συλλογής ιχνών ολόκληρης της εφαρμογής για να χρησιμοποιηθουν αργότερα για ιχνοκατευθυνόμενη επανεκτέλεση της, εργαλεία όπως το *ATOM* είναι αποδοτικότερα.
- Η διαστολή που προκαλεί το εργαλείο μας στον χρόνο εκτέλεσης της ιχνοληπτούμενης εφαρμογής είναι ανάλογη του ποσοστού των προσπελάσεων μεταβλητών του

| | USER−TIME | | ELAPSED−TIME | |
|---|---|---|---|---|
| % | ATOM | ArrayTracer | ATOM | ArrayTracer |
| 0.12 | 384.54 | 10.12 | 6:43.3 | 0:21.4 |
| 12.60 | 404.78 | 68.44 | 6:53.5 | 2:19.8 |
| 25.09 | 490.94 | 125.83 | 8:26.8 | 4:22.6 |
| 37.57 | 552.96 | 193.43 | 9:32.6 | 6:31.5 |
| 50.06 | 585.73 | 262.22 | 10:06.2 | 8:48.6 |
| 62.54 | 628.53 | 308.28 | 10:42.2 | 10:32.3 |
| 75.03 | 679.67 | 370.59 | 11:40.2 | 12:35.3 |
| 87.51 | 724.50 | 434.38 | 12:21.4 | 14:25.0 |
| 100.0 | 771.04 | 477.84 | 13:10.5 | 16:12.5 |

Πίνακας 6.5: Συγκριτικός πίνακας επιλεκτικής ιχνοληψίας της εφαρμογής *πολλαπλασιασμού πινάκων* με το *ATOM* και με το *ArrayTracer*. Η πρώτη στήλη περιέχει το ποσοστό προσπελάσεων πινάκων που ιχνοληπτήθηκαν. Ο χρόνος εμφανίζεται με την μορφή (*λεπτά:δευτ/πτα.εκατοστά*).

προγράμματος. Συνεπώς, η συμπεριφορά του είναι πάντα προβλέψιμη, ανεξάρτητα από το σύστημα στο οποίο θα εκτελεστεί η εφαρμογή. Αυτό δεν ισχύει για εργαλεία όπως το *ATOM* των οποίων η συμπεριφορά εξαρτάται από τον μεταφραστή και από το επίπεδο βελτιστοποίησης που χρησιμοποιούνται.

- Η παραγωγή των ιχνών ανταποκρίνεται σε αυτά ακριβώς που ορίζει ο χρήστης. Σε εργαλεία όμως που εισάγουν κώδικα καθοδήγησης σε επίπεδο εκτελέσιμου κώδικα της εφαρμογής τα πράγματα είναι διαφορετικά. Σε αυτό το επίπεδο έχει εισαχθεί ο κώδικας των διαφόρων βιβλιοθηκών που χρησιμοποιεί ο χρήστης. Ετσι είναι δυνατόν να παρατηρηθούν φαινόμενα όπως αυτά που παρουσιάζονται στον πίνακα 6.2, όπου το μέγεθος του αρχείο ιχνοληψίας που δημιουργήθηκε δεν ήταν το αναμενόμενο.

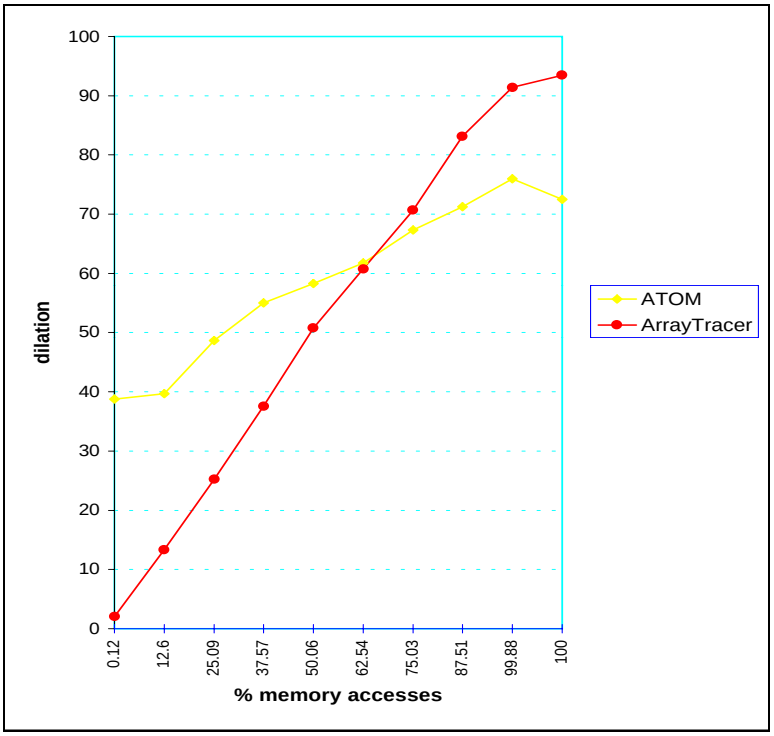
- Η ευκολία καθορισμού των παραμέτρων ιχνοληψίας που προσφέρει το *ArrayTracer* το κάνει πολύ φιλικό προς τον χρήστη που ενδιαφέρεται για την *επιλεκτική ιχνοληψία* της εφαρμογής του. Σε εργαλεία προσομοίωσης της εκτέλεσης εφαρμογών όπως τα *ATOM*, *MemSpy* [36], *Mtool* [16] αλλά και προσομοιωτές αρχιτεκτονικών όπως τα *Proteus* [5, 6] και *MINT* [48], είναι πολύ δύσκολο να καθορίσει ο χρήστης την συλλογή ιχνών που να αφορούν μόνο ορισμένες μεταβλητές της εφαρμογής του.

Συνοψίζοντας, μπορούμε να πούμε ότι το *ArrayTracer* είναι το κατάλληλο εργαλείο για τον χρήστη που επιθυμεί να αναλύσει την συμπεριφορά της εφαρμογής του σαν παράμετρο μερικών[4] μόνο μεταβλητών της.

## 6.4 Αξιολόγηση του ArrayTracer

Η τεχνική ιχνοληψίας που χρησιμοποιεί το *ArrayTracer*, παρουσιάζει δύο βασικά χαρακτηριστικά τα οποία την διαχωρίζουν από τις τεχνικές ιχνοληψίας που χρησιμοποιούνται στον χώρο των Εργαλείων για την Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών:

[4]Για ποσοστό μικρότερο του 35% οι πειραματικές μετρήσεις έδειξαν ότι συμπεριφέρεται καλύτερα από το *ATOM*.




Σχήμα 6.4: Γραφική παράσταση της διαστολής που προκαλούν στον χρόνο εκτέλεσης της εφαρμογής *πολλαπλασιασμού πινάκων* τα *ATOM* και *ArrayTracer*.

1. Η διαδικασία ιχνοληψίας είναι *επιλεκτική* τόσο ως προς τις μεταβλητές που θα ιχνο–ληπτηθούν όσο και ως προς τα τμήματα πηγαίου κώδικα στα οποία λαμβάνει χώρα η διαδικασία αυτή. Η διαστολή που θα προκληθεί στην εκτέλεση της εφαρμογής καθώς και ο όγκος των ιχνών που θα παραχθούν είναι ανάλογα του ποσοστού των προσπελάσεων σε μεταβλητών που ο χρήστης έχει επιλέξει για ιχνοληψία.

2. Εισάγει κώδικα καθοδήγησης σε επίπεδο πηγαίου κώδικα της εφαρμογής. Τα ίχνη που παράγονται από αυτόν τον κώδικα δεν αφορούν άμεσα στις προσπελάσεις θέσεων μνήμης, αλλά στις προσπελάσεις μεταβλητών της εφαρμογής. Το γεγονός αυτό επιτρέπει στον χρήστη να καθορίσει με ευκολία τις μεταβλητές που επιθυμεί να ιχνοληπτήσει. Επίσης δεν χρειάζεται να γίνει καμία αντιστοίχιση της πληροφορίας που περιέχουν τα ίχνη με στοιχεία του πηγαίου κώδικα που είναι αντιληπτά στον χρήστη γιατί η πληροφορία που περιέχουν δεν αφορά γεγονότα χαμηλού επιπέδου (π.χ. προσπελάσεις συγκεκριμένων διευθύνσεων μνήμης).

Βασιζόμενο σε αυτά τα χαρακτηριστικά, το *ArrayTracer* επιτρέπει στον χρήστη να κα–τευθύνει με μεγάλη ευκολία την διαδικασία ιχνοληψίας σε συγκεκριμένα στοιχεία και σε συγκεκριμένα τμήματα της εφαρμογής του. Επίσης, επιτρέπει στον χρήστη να αναλύσει την συμπεριφορά της εφαρμογής του σαν συνάρτηση της εσωτερικής δομής της και όχι σαν συνάρτηση με παραμέτρους τα χαρακτηριστικά του συγκεκριμένου συστήματος στο οποίο εκτελέστηκε όταν ιχνοληπτήθηκε. Ετσι, ο χρήστης έχει την δυνατότητα να βετλιώσει τις επιδόσεις της εφαρμογής του ανεξάρτητα από το υλικό (hardware) πάνω στο οποίο εκτελείτε και συνεπώς τον φέρνει αντιμέτωπο με τα ουσιαστηκά προβλήματα στην σχεδίαση της και όχι με εκείνα που φέρνει στην επιφάνεια το συγκεκριμένο περιβάλλον εκτέλεσης.

Το πεδίο χρήσης του *ArrayTracer* φαίνεται να καθορίζεται από αυτά τα χαρακτηριστικά του. Οι χρόνοι και οι τιμές του μεγέθους της *διαστολής* που μετρήσαμε καθιστούν σαφές ότι το εργαλείο μας βοηθάει τον χρήστη να επικεντρώσει την διαδικασία ανάλυσης της

συμπεριφοράς της εφαρμογής σε επιλεγμένα σημεία. Στο επόμενο κεφάλαιο και ειδικότερα στην ενότητα 7.3 αναφέρουμε ορισμένες ιδέες που πιστεύουμε ότι θα διεύρυναν το πεδίο χρήσης του *ArrayTracer*.

# Κεφάλαιο 7

# Θέματα υλοποίησης

Το εργαλείο *ArrayTracer* και ειδικότερα η τεχνική που χρησιμοποιεί για την παραγωγή ιχνών, αποτελούν μια ελκυστική πρόταση στον χώρο των *Εργαλείων για την Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών*. Στην περιγραφή του πρώτου λειτουργικού επιπέδου του εργαλείου μας, αναφέραμε τέσσερα βασικά δομικά στοιχεία του:

1. την διεπαφή εισόδου εντολών ιχνοληψίας,
2. το στοιχείο που αναλαμβάνει την *Στατική Ανάλυση* της εφαρμογής και την εισαγωγή κώδικα καθοδήγησης,
3. την διεργασία−*συλλέκτη*,
4. την βιβλιοθήκη ρουτινών ιχνοληψίας.

Από τα τέσσερα αυτά στοιχεία, ιδιαίτερο ενδιαφέρον παρουσιάζει η υλοποίηση του δευτέρου. Υπάρχουν τρεις εναλλακτικές προτάσεις για την υλοποίηση της φάσης *Στατικής Ανάλυσης* του *ArrayTracer*, τις οποίες παρουσιάζουμε στην επόμενη ενότητα.

## 7.1 Προσεγγίσεις υλοποίησης.

Η διεπαφή εισόδου είναι ένας απλός διερμηνέας (interpreter) των εντολών που δίνει ο χρήστης και έχει υλοποιηθεί σε *yacc* και *lex*. Στο μέλλον προβλέπεται να επεκταθεί ώστε να περιλαμβάνει και γραφικό τμήμα για να γίνει πιο φιλική προς τον χρήστη. Η διεργασία−*συλλέκτης* μαζί με τις ρουτίνες διαχείρησης του κοινόχρηστου ενταμιευτή έχουν υλοποιηθεί σε γλώσσα προγραμματισμού **C** και παρουσιάζουν ενδιαφέρον κυρίως όσον αφορά θέματα αποτελεσματικότητα και ταχύτητας της λειτουργίας του ενταμιευτή. Η βιβλιοθήκη ρουτινών ιχνοληψίας παρουσιάζει ενδιαφέρον σε ότι αφορά την κατασκευή ρουτινών σε γλώσσα προγραμματισμού **C** οι οποίες να μπορούν να κληθούν μέσα από προγράμματα γραμμένα σε γλώσσα προγραμματισμού *Fortran*. Τέλος, η υλοποίηση του στοιχείου που πραγματοποιεί την *Στατική Ανάλυση* είναι αυτή που παρουσιάζει το μεγαλύτερο ενδιαφέρον από πλευράς σχεδίασης, αφού υπήρχαν πολλές εναλλακτικές δυνατότητες προσέγγισης.

**Ενσωμάτωση σε μεταφραστή.** Μια προσέγγιση υλοποίησης είναι η ενσωμάτωση της φάσης *Στατικής Ανάλυσης* του *ArrayTracer* σε κάποιον υπάρχοντα μεταφραστή (compiler) της γλώσσας *Fortran*. Αν ο χρήστης καλέσει τον μεταφραστή με την κατάλληλη επιλογή (π.χ. $-AT$) τότε, κατά την διάρκεια της συντακτικής ανάλυσης που πραγματοποιεί ο ίδιος ο μεταφραστής, θα εκτελούσε και την *Στατική Ανάλυση*. Ετσι, το εκτελέσιμο που θα προέκυπτε θα περιείχε και τις

κατάλληλες εντολές καθοδήγησης. Τίποτε άλλο δεν χρειάζεται ιδιαίτερο σχεδιασμό, αφού η διαδικασία συλλογής των ιχνών αρχικοποιείται από τις ίδιες τις διεργασίες της παράλληλης εφαρμογής. Αυτός ο τρόπος προσέγγισης είναι ο πιο άμεσος. Σημαντικό πλεονέκτημα του αποτελεί το γεγονός ότι ο μηχανισμός που θα εκτελέσει την διαδικασία συντακτικής ανάλυσης του κώδικα της εφαρμογής βρίσκεται έτοιμος στον μεταφραστή.

**Προεπεξεργαστής.** Μια άλλη προσέγγιση είναι η πραγματοποίηση της *Στατικής Ανάλυσης* από έναν προεπεξεργαστή του πηγαίου κώδικα της εφαρμογής. Αυτός ο προεπεξεργαστής θα εκτελεί μια μετάφραση από πηγαίο σε πηγαίο κώδικα (source−to−source translation). Θα δέχεται σαν είσοδο τα αρχεία πηγαίου κώδικα της εφαρμογής και θα παράγει σαν έξοδο αρχεία που θα περιέχουν πηγαίο κώδικα *Fortran* εμπλουτισμένο με τις εντολές ιχνοληψίας που θα οδηγήσουν στην παραγωγή ιχνών κατά την διάρκεια εκτέλεσης της εφαρμογής. Τα εμπλουτισμένα αρχεία θα πρέπει στην συνέχεια να μεταφραστούν από κάποιον υπάρχοντα μεταφραστή της γλώσσας *Fortran* και στην συνέχεια να συνδεθούν (linked) με την βιβλιοθήκη του *ArrayTracer*.

Αυτή η προσέγγιση αν και δεν είναι τόσο άμεση όσο η προηγούμενη, ωστόσο έχει το πλεονέκτημα ότι το εργαλείο καθίσταται αυθύπαρκτο και ουσιαστικά φορητό (portable), μια και δεν εξαρτάται από άλλες εφαρμογές (π.χ. μεταφραστές) οι οποίες με την σειρά τους εξαρτώνται από τα μηχανήματα στα οποία μπορούν να εκτελεστούν (hardware depended).

**Συνεργασία με εργαλείο Παροχής Πληροφοριών Λογισμικού.** Τέλος, μια ακόμα προσέγγιση είναι η κατασκευή του εργαλείου σε στενή συνεργασία με ένα εργαλείο Παροχής Πληροφοριών Λογισμικού (Software Information Base Tool), όπως το *SIS* [8, 9]. Αυτό το εργαλείο θα αναλάβει να εκτελεί την φάση της *Στατικής Ανάλυσης* της εφαρμογής και θα επιστρέφει στον πυρήνα του *ArrayTracer* πληροφορίες σχετικά με τα δομικά στοιχεία του κώδικά της. Ο πυρήνας του εργαλείου θα αναλαμβάνει να εισάγει τον κώδικα καθοδήγησης και τελικά θα παράγονται αρχεία πηγαίου κώδικα τα οποία θα περιέχουν τον κώδικα της εφαρμογής εμπλουτισμένο με τις εντολές ιχνοληψίας. Από αυτό το σημείο και μετά ακολουθείται η ίδια διαδικασία όπως στην προηγούμενη προσέγγιση.

Αυτή η πρόταση υλοποίησης, αν και υστερεί σε θέματα ανεξαρτησίας του εργαλείου μας από άλλες εφαρμογές, προσφέρει ευκολία στην υλοποίηση μια και ένα πολύ βασικό τμήμα του δεν θα χρειαστεί να υλοποιηθεί από την αρχή, αλλά θα χρησιμοποιηθεί το εργαλείο Παροχής Πληροφοριών Λογισμικού για να προσφέρει τις αντίστοιχες υπηρεσίες.

**Η πρώτη υλοποίηση** Ξεκινώντας να υλοποιήσουμε το *ArrayTracer*, προτιμήσαμε την δεύτερη προσέγγιση. Η φάση της *Στατικής Ανάλυσης* πραγματοποιείται από έναν προεπεξεργαστή πηγαίου κώδικα γλώσσας *Fortran* αποκομίζοντας έτσι τα οφέλη της ανεξαρτησίας και της φορητότητας που αναφέραμε προηγουμένως. Ομως, στα μελλοντικά μας σχέδια ανήκει και η ολοκλήρωση του εργαλείου (tool integration) μας μέσα σε κάποιο *Περιβάλλον Ανάλυσης Παράλληλων Επιδόσεων*. Ετσι, το *ArrayTracer* θα μπορεί να συνεργάζεται με *SIB* εργαλεία για βελτίωση της φάσης της *Στατικής Ανάλυσης* καθώς και με εργαλεία γραφικής παράστασης (visualization tools) των αποτελεσμάτων της ανάλυσης της συμπεριφοράς της παράλληλης εφαρμογής για βελτίωση της ποιότητας πληροφορίας που παρουσιάζεται στον χρήστη.

## 7.2 Κατάσταση Υλοποίησης

Στην υπάρχουσα σχεδίαση του *ArrayTracer*, το εργαλείο αποτελείται από τρία δομικά στοιχεία. Κάθε δομικό στοιχείο του αποτελεί υλοποίηση ενός από τα λειτουργικά επίπεδα ενός

εργαλείου για την *Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών*. Από αυτά τα τρία επίπεδα, στην αναφορά αυτή παρουσιάσαμε αναλυτικά την σχεδίαση και υλοποίηση του πρώτου επιπέδου, το οποίο ασχολείται με την παραγωγή και συλλογή των ιχνών της εφαρμογής. Τα υλοποιημένα τμήματα του επιπέδου αυτού είναι τα εξής :

- Η διεπαφή εισόδου εντολών του χρήστη. Με βάση αυτές τις εντολές κατασκευάζεται ο πίνακας ιχνοληψίας στον οποίο στηρίζεται η διαδικασία της *Στατικής Ανάλυσης* της εφαρμογής.
- Ο προεπεξεργαστής πηγαίου κώδικα *Fortran*, ο οποίος πραγματοποιεί την *Στατική Ανάλυση* της εφαρμογής και την εισαγωγή κώδικα καθοδήγησης σε επιλεγμένα σημεία του αρχικού πηγαίου κώδικα.
- Ο μηχανισμός του κοινόχρηστου ενταμιευτή που χρησιμοποιείται για την επικοινωνία των διεργασιών της εφαρμογής με τις αντίστοιχες διεργασίες–*συλλέκτες*. Η βιβλιοθήκη ιχνοληψίας του *ArrayTracer* αποτελείται από ρουτίνες που γράφουν τις απαραίτητες πληροφορίες στον ενταμιευτή αυτόν.
- Ο μηχανισμός της διεργασίας–*συλλέκτη* που αναλαμβάνει την μεταφορά των ιχνών από τον κοινόχρηστο ενταμιευτή στο αντίστοιχο αρχείο δίσκου, κατά την διάρκεια εκτέλεσης της εφαρμογής. Επίσης, ο μηχανισμός που αναλαμβάνει την προεπεξεργασία των ιχνών, πριν αυτά σταλούν στην διεργασία–*πυρήνα* του εργαλείου για την περαιτέρω επεξεργασία τους. Αυτή η προεπεξεργασία λαμβάνει χώρα μετά το πέρας εκτέλεσης της εφαρμογής.

Η υπάρχουσα υλοποίηση έχει πραγματοποιηθεί σε μηχανήματα *DEC ALPHAs* με λειτουργικό σύστημα *DEC OSF/1 V3.0 Worksystem Software*. Ο κώδικας έχει γραφτεί σε γλώσσα προγραμματισμού *C* και ο μεταφραστής που έχει χρησιμοποιηθεί είναι *gcc version 2.6.3*. Η επικοινωνία των διεργασιών του εργαλείου μεταξύ τους πραγματοποιείται μέσω της βιβλιοθήκης *PVM version 3.3.2*. Για την μετάφραση των πηγαίων αρχείων που είναι εμπλουτισμένα με τον κώδικα καθοδήγησης χρησιμοποιήθηκε ο μεταφραστής *DEC f77 version 3.4* ενώ και σε αυτήν την περίπτωση, η ενδοεπικοινωνία των διεργασιών της εφαρμογής γίνεται με την χρήση της βιβλιοθήκης *PVM version 3.3.2*.

## 7.3 Μελλοντική Εργασία

Στον χώρο της μελλοντικής εργασίας εντάσσουμε πολλά ενδιαφέροντα θέματα τα οποία αφορούν σύνθετους χειρισμούς ιχνοληψίας. Η ενασχόληση μας με τον χώρο των εργαλείων για την *Ανάλυση της Επίδοσης Παράλληλων Εφαρμογών* και ειδικότερα με τον τομέα της εισαγωγής κώδικα καθοδήγησης σε επίπεδο πηγαίου κώδικα, μας δημιούργησε νέα ερωτήματα τα οποία απαιτούν δυναμικές λύσεις. Το ενδιαφέρον μας, σε πρώτο τουλάχιστον στάδιο, επικεντρώνεται στα εξής σημεία :

- Επέκταση της χρήσης των *φορμών ιχνοληψίας* για αποδοτικότερη συνοπτική παραγωγή δυναμικών ιχνών. Στην υπάρχουσα υλοποίηση, οι φόρμες ιχνοληψίας έχουν χρησιμοποιηθεί ελάχιστα και στις περιπτώσεις στις οποίες χρησιμοποιήθηκαν, αυτό έγινε με πολυ συντηρητικά κριτήρια. Πιστεύουμε ότι αυτός ο τομέας έχει να προσφέρει στο εργαλείο μας πολλά οφέλη.
- Ολοκλήρωση της διαδικασίας συλλογής ιχνών της εφαρμογής με την προσθήκη πληροφοριών σχετικά με το μέγεθος των μηνυμάτων που ανταλλάσουν οι διεργασίες της εφαρμογής.

- Μελέτη επέκτασης της λειτουργίας του *ArrayTracer* με αποδοτικό τρόπο και σε εφαρμογές που χρησιμοποιούν το μοντέλο *κοινόχρηστης μνήμης*. Αυτή η κατεύθυνση φαίνεται να περιέχει αρκετά ανοιχτά ζητήματα, και ίσως να χρειαστεί να συζητηθεί η συνεργασία του εργαλείου μας με κάποιο εργαλείο παρακολούθησης της παράλληλης μνήμης (parallel memory monitor), ώστε να μπορέσουμε να πάρουμε αρκετές πληροφορίες για τις αναφορές της εφαρμογής στην κοινόχρηστη μνήμη.

- Υλοποίηση των δύο άλλων λειτουργικών επιπέδων του εργαλείου, δηλαδή του λειτουργικού επιπέδου το οποίο είναι υπεύθυνο για την επεξεργασία των ιχνών που λαμβάνει χώρα μετά το πέρας της εκτέλεσης της εφαρμογής και το λειτουργικού επιπέδου που είναι υπεύθυνο για την παρουσίαση των αποτελεσμάτων στον χρήστη.

- Επέκταση του εργαλείου ώστε να μπορεί να λειτουργήσει και για εφαρμογές γραμμένες σε άλλες γλώσσες προγραμματισμού, κυρίως σε *C* και *C++*. Αυτό θα βοηθούσε στην χρήση του εργαλείου από ευρύτερο φάσμα χρηστών.

- Την συνεργασία του εργαλείου μας με αλγορίθμους για την τοποθέτηση δεδομένων και διεργασιών (data/thread placement algorithms). Η συνεργασία αυτή θα μπορεί να γίνει με την παροχή πληροφορίας υπό την μορφή ανάδρασης (feedback) από τα αποτελέσματα της επεξεργασίας των ιχνών.

- Συνδυασμό της τεχνικής *επιλεκτικής ιχνοληψίας* με τεχνικές παραγωγής ιχνών μέσα από την διαδικασία προσομοίωσης της εκτέλεσης εφαρμογών. Ετσι, θα γίνει εφικτό να συνδυάσουμε τον χαμηλό παράγοντα διαστολής της *επιλεκτικής ιχνοληψίας* με τα πλεονεκτήματα που προσφέρει η ιχνοληψία των εργαλείων προσομοίωσης· δηλαδή, με την παραγωγή ιχνών τα οποία περιέχουν λεπτομέρειες για το σύστημα στο οποίο εκτελείται η εφαρμογή.

# Παράρτημα Α

# Σύνολο Εντολών της Διεπαφής Εισόδου

Αυτό το παράρτημα περιέχει αναλυτική παρουσίαση των εντολών που παρέχει στον χρήστη η διεπαφή εισόδου τόσο από πλευράς συντακτικού, όσο και από πλευράς λειτουργικότητας.

- **trace file** <*file*>
  - <*file*> := Το απόλυτο μονοπάτι (absolutl path) του αρχείου με όνομα *file*

  Αυτή η εντολή επιτρέπει στον χρήστη να καθορίσει το αρχείο πηγαίου κώδικα της εφαρμογής στο οποίο θα απευθύνονται οι εντολές ιχνοληψίας που θα ακολουθήσουν.Η εντολή αυτή καθιστά το αρχείο *file* ενεργό. Οταν ο χρήστης χρησιμοποιήσει ξανά την εντολή αυτή για να καθορίσει κάποιο άλλο αρχείο *file2* ως ενεργό, τότε αυτόματα το προηγούμενο αρχείο *file*, παύει να θεωρείται ενεργό. Συνεπώς, η εισαγωγή αυτής της εντολής στην διεπαφή εισόδου απενεργοποιεί το αρχείο που μέχρι εκείνη την στιγμή ήταν ενεργό και ενεργοποιεί το αρχείο που ορίζεται από την παράμετρο της.

  Παράδειγμα χρήσης : **trace file** */home/users/abcd/myprog.f*

- **trace variable** <*var_spec*> **from** <*lineNo*> **to** <*lineNo*>
  - <*var_spec*> := **id** [<*vol_spec*>]*
  - <*vol_spec*> := **[number .. number]**
  - <*lineNo*> := **number**

  Αυτή η εντολή επιτρέπει στον χρήστη να καθορίσει μία μεταβλητή της εφαρμογής η οποία θα συμμετέχει στη ιχνοληψία. Τα στοιχεία που καλείται να δώσει ο χρήστης για τον καθορισμό της μεταβλητής είναι το όνομα της, οι διαστάσεις που καθορίζουν τα όρια ενός υποπίνακα (μόνο σε περίπτωση που ο χρήστης επιθυμεί την ιχνοληψία κάποιου υποπίνακα) και οι αριθμοί των γραμμών του πηγαίου κώδικα οι οποίες οριοθετούν το διάστημα στο οποίο η συγκεκριμένη μεταβλητή θα συμμετέχει στην ιχνοληψία.

  Παράδειγμα χρήσης : **trace variable** *A[1..5][10..25]* **from** *120* **to** *345*

- **trace code at** <*lev_spec*> **level from** <*lineNo*> **to** <*lineNo*>

  - <*lev_spec*> := **loop** | **function** | **subroutine**
  - <*lineNo*> := **number**

  Με αυτήν την εντολή ο χρήστης μπορεί να επιλέξει για ιχνοληψία μια από τις εξής τρείς κατηγορίες δομικών στοιχείων του προγράμματος: βρόχους επανάληψης, κλήσεις συναρτήσεων και κλήσεις υπορουτινών. Οι παράμετροι που καθορίζει ο χρήστης είναι το είδος των δομικών στοιχείων που θέλει να συμμετέχουν στην διαδικασία ιχνοληψίας καθώς, και η περιοχή του πηγαίου κώδικα στην οποία η διαδικασία ιχνοληψίας θα λάβει χώρα για το συγκεκριμένο στοιχείο.

  Παράδειγμα χρήσης : **trace code at loop level from** *34* **to** *355*

- **trace communication at** <*lev_spec*> **level from** <*lineNo*> **to** <*lineNo*>

  - <*lev_spec*> := **send** | **receive** | **barrier** | **group** | **task**
  - <*lineNo*> := **number**

  Με αυτήν την εντολή ο χρήστης καθορίζει ένα στοιχείο ενδοεπικοινωνίας των διεργασιών της παράλληλης εφαρμογής για να συμμετέχει στην διαδικασία ιχνοληψίας. Για τον πλήρη καθορισμό αυτού του στοιχείου, ο χρήστης περνάει στις παραμέτρους της εντολής το είδος του στοιχείου ενδοεπικοινωνίας που θέλει, καθώς και την περιοχή του πηγαίου κώδικα της εφαρμογής στην οποία η διαδικασία ιχνοληψίας θα λάβει χώρα για το συγκεκριμένο στοιχείο.

  Παράδειγμα χρήσης : **trace communication at barrier level from** *145* **to** *512*

- **show**

  Με την εισαγωγή αυτής της εντολής στην διεπαφή εισόδου από τον χρήστη παρουσιάζονται τα περιεχόμενα του τρέχοντος πίνακα ιχνοληψίας στην οθόνη. Ως τρέχοντα πίνακα ιχνοληψίας ορίζουμε τον πίνακα που περιέχει στοιχεία που αφορούν το ενεργό αρχείο στην συγκεκριμένη χρονική στιγμή. Τα περιεχόμενα του πίνακα ιχνοληψίας δεν παρουσιάζονται με την σειρά εισαγωγής τους στον πίνακα, αλλά ομαδοποιημένα ανά κατηγορία ιχνοληψίας. Πρώτα εμφανίζονται τα στοιχεία που αφορούν την ιχνοληψία μεταβλητών, στην συνέχεια τα στοιχεία που αφορούν την ιχνοληψία δομικών στοιχείων του πηγαίου κώδικα της εφαρμογής, και τέλος τα στοιχεία που αφορούν την ενδοεπικοινωνία διεργασιών της παράλληλης εφαρμογής. Το σημαντικό στοιχείο με αυτήν την εντολή είναι ότι παρουσιάζει στον χρήστη και τις ετικέτες που έχουν αποδοθεί αυτόματα σε κάθε στοιχείο του πίνακα ιχνοληψίας. Αυτές είναι χρήσιμες στον χρήστη κατά την διαδικασία διαγραφής κάποιου στοιχείου του εν λόγω πίνακα, όπως θα δούμε παρακάτω.

  Παράδειγμα χρήσης : **show**

- **erase** <*entry_spec*> <*tag*>

- <*entry_spec*> := **variable** | **code** | **communication**
- <*tag*> := **number**

Με αυτήν την εντολή ο χρήστης μπορεί να επιλέξει ένα συγκεκριμένο στοιχείο του πίνακα ιχνοληψίας και να το διαγράψει. Οπως φαίνεται από το συντακτικό της εντολής, το στοιχείο του πίνακα ιχνοληψίας καθορίζεται με την χρήση δύο συνιστωσών: της κατηγορίας στοιχείων ιχνοληψίας στην οποία ανήκει (μεταβλητές, δομικά στοιχεία κώδικα ή στοιχεία ενδοεπικοινωνίας) και της ετικέτας που του έχει αποδοθεί αυτόματα από το εργαλείο. Ο χρήστης μπορεί να μάθει την ετικέτα που έχει κάθε μεταβλητή με την χρήση της εντολής **show** που παρουσιάστηκε παραπάνω.

Παράδειγμα χρήσης : **erase code** *437*

- **quit | exit | bye**

  Αυτή είναι η εντολή εξόδου από την διεπαφή εισόδου του *ArrayTracer*. Εισάγοντας αυτήν την εντολή, ο χρήστης δηλώνει ότι τελείωσε την διαδικασία καθορισμού των παραμέτρων ιχνοληψίας Μετά από αυτήν την εντολή, ο χρήστης θα πρέπει να περάσει από τον μεταφραστή (compiler) τα αρχεία που έχουν παραχθεί από το *ArrayTracer*. Τα αρχεία αυτά περιέχουν τον πηγαίο κώδικα εμπλουτισμένο με τις κλήσεις σε ρουτίνες ιχνοληψίας του *ArrayTracer*. Στην συνέχεια, ο χρήστης πρέπει να τα συνδέσει (link) με την βιβλιοθήκη του *ArrayTracer*.

  Παράδειγμα χρήσης : **quit**

Σε αυτό το σημείο αναφέρουμε ότι η αυτόματη απόδοση ετικετών στα στοιχεία ιχνοληψίας που καθορίζει ο χρήστης γίνεται ανεξάρτητα για κάθε κατηγορία στοιχείων ιχνοληψίας. Ετσι, η ετικέτα *3* μπορεί να αποδοθεί και σε μεταβλητή που έχει επιλεχθεί για να συμμετέχει στην διαδικασία ιχνοληψίας, και σε δομικό στοιχείο του πηγαίου κώδικα αλλά και σε στοιχείο ενδοεπικοινωνίας διεργασιών που επίσης έχουν επιλεχθεί για συμμετοχή στην διαδικασία ιχνοληψίας.

Ακόμα ένα σημείο που αξίζει προσοχής είναι το γεγονός ότι αν το ίδιο στοιχείο ιχνοληψίας[1] (μεταβλητή ή δομικό στοιχείο πηγαίου κώδικα ή στοιχείο ενδοεπικοινωνίας διεργασιών) επιλεχθεί για συμμετοχή στην διαδικασία ιχνοληψίας σε περισσότερες από μία περιοχές του πηγαίου κώδικα, τότε :

1. Ο αριθμός των εντολών ιχνοληψίας που έχει εισάγει ο χρήστης στην διεπαφή εισόδου για τον προσδιορισμό της διαδικασίας ιχνοληψίας σε αυτές τις περιοχές του πηγαίου κώδικα, είναι ίσος με τον αριθμό των διαφορετικών περιοχών πηγαίου κώδικα που προσδιορίστηκαν.

2. Με κάθε εντολή ιχνοληψίας, από αυτές που προαναφέρθηκαν, θα δημιουργηθεί και μία διαφορετική είσοδος στον πίνακα ιχνοληψίας και φυσικά θα της αποδοθεί διαφορετική ετικέτα.

[1] Το ίδιο ισχύει και για διαφορετικούς υποπίνακες του ίδιου πίνακα. Κάθε διαφορετικός υποπίνακας αντιστοιχίζεται σε διαφορετική είσοδο στον πίνακα ιχνοληψίας και προφανώς του αποδίδεται μία ετικέτα μοναδική.

# Παράρτημα Β

# Η Δομή Δεδομένων του Πίνακα Ιχνοληψίας

Σε αυτό το παράρτημα παρουσιάζεται αναλυτικά η δομή δεδομένων που χρησιμοποιείται για την αποθήκευση των στοιχείων του πίνακα ιχνοληψίας. Μια γραφική αναπαράσταση της δομής αυτής δίνεται στο σχήμα 3.2. Θυμίζουμε στον αναγνώστη ότι ο πίνακας ιχνοληψίας αρχίζει να κατασκευάζεται από την διεπαφή εισόδου κατά την διάρκεια εισαγωγής εντολών ιχνοληψίας από τον χρήστη και ολοκληρώνεται κατά την διάρκεια της *Στατικής Ανάλυσης* του πηγαίου κώδικα της εφαρμογής, όταν γίνουν γνωστές πληροφορίες που αφορούν τις διαστάσεις των πινάκων και την ομαδοποίηση των μεταβλητών με βάση τις εντολές *EQUIVALENCE* και *COMMON*.

Ο πίνακας ιχνοληψίας υλοποιείται σαν μια απλά συνδεδεμένη λίστα της οποίας κάθε στοιχείο αντιστοιχεί σε ένα διαφορετικό αρχείο πηγαίου κώδικα της εφαρμογής που πρόκειται να ιχνοληπτηθεί. Το κάθε στοιχείο της λίστας αυτής είναι η κεφαλή μίας λίστας η οποία περιέχει τα στοιχεία που αφορούν κάθε κατηγορία ιχνοληψίας (μεταβλητές, δομικά στοιχεία πηγαίου κώδικα, και στοιχεία ενδοεπικοινωνίας διεργασιών) ομαδοποιημένα σε τρεις υπολίστες. Οι υπολίστες αυτές ονομάζονται *υποπίνακας μεταβλητών*, *υποπίνακας κώδικα* και *υποπίνακας ενδοεπικοινωνίας* αντίστοιχα.

Η διαίρεση του πίνακα ιχνοληψίας σε τρείς υποπίνακες προσφέρει τα εξής πλεονεκτήματα :

- Η υλοποίηση του πίνακα ιχνοληψίας έρχεται πολύ κοντά στην διαισθητική εικόνα που έχει ο προγραμματιστής για αυτόν. Ετσι, η κατανόηση της δομής του και συνεπώς οι όποιες επεκτάσεις και τροποποιήσεις πάνω σε αυτήν, θα γίνονται πολύ εύκολα.

- Κατά την διάρκεια της στατικής ανάλυσης του πηγαίου κώδικα της εφαρμογής και της εισόδου του κώδικα ιχνοληψίας είναι αναγκαίο να γίνουν πολλές σαρώσεις αυτού του πίνακα. Με την διαίρεση του σε τρεις υποπίνακες, επιταχύνεται η διαδικασία της σάρωσης. Αν σε αυτό προστεθεί και ο μηχανισμός που εκτελεί την σάρωση, ο οποίος υλοποιεί μια αυτο−ταξινομούμενη λίστα (self−ordered list), τότε καταλήγουμε σε ένα πολύ γρήγορο μηχανισμό προγραμματισμού του εργαλείου.

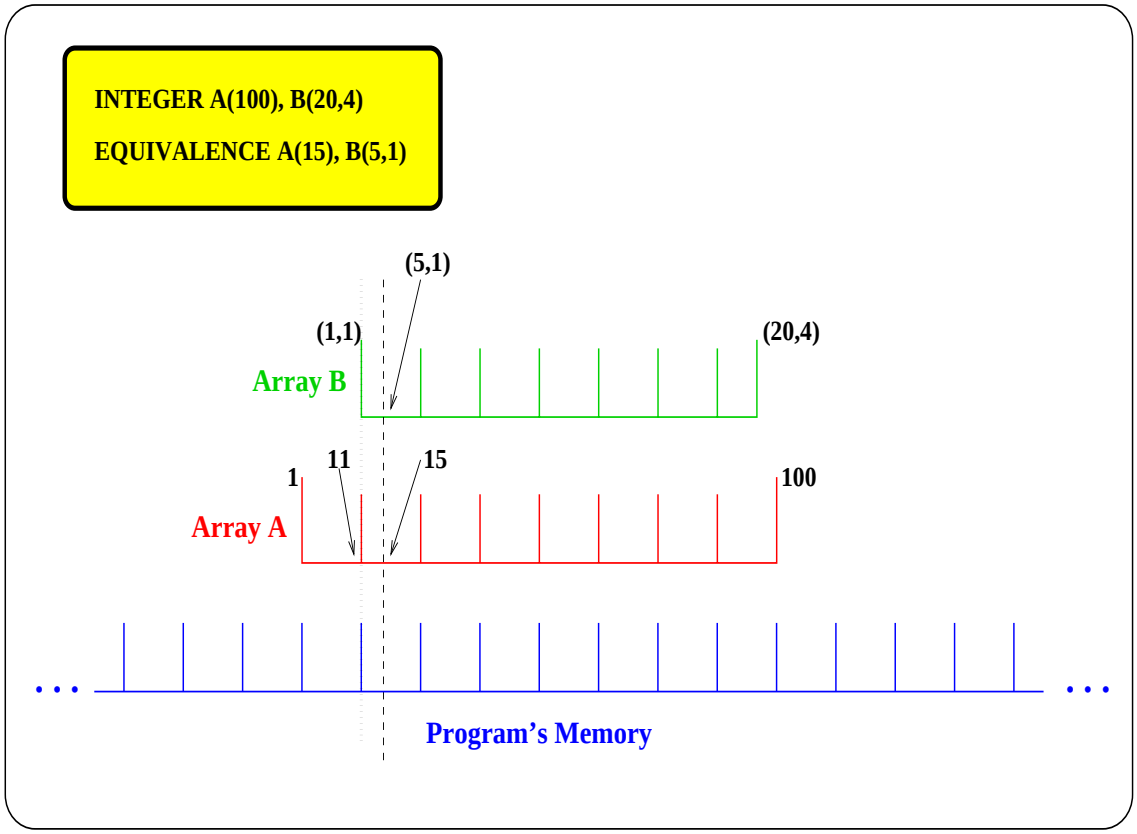
Η ακριβής δομή του πίνακα ιχνοληψίας, είναι η ακόλουθη. Τα στοιχεία της λίστας που υλοποιεί τον πίνακα ιχνοληψίας έχουν τύπο *Table*. Τα στοιχεία του υποπίνακα που χρησιμοποιείται για την αποθήκευση των πληροφοριών σχετικά με τις μεταβλητές που έχουν επιλεχθεί για ιχνοληψία, έχουν τύπο *VEntry*. Τα στοιχεία των υποπινάκων που αποθηκεύουν πληροφορίες σχετικά με τα δομικά στοιχεία του πηγαίου κώδικα και με τα στοιχεία της ενδοεπικοινωνίας διεργασιών, έχουν τύπο *CEntry*. Τέλος, οι λίστες που χρησιμοποιούνται

τόσο για την αποθήκευση των διαστημάτων που καθορίζουν τις διαστάσεις ενός υποπίνακα (σε περίπτωση που ο χρήστης επιλέξει υποπίνακες για να συμμετέχουν στην διαδικασία ιχνοληψίας), όσο και για την αποθήκευση του διαστήματος του πηγαίου κώδικα στο οποίο κάθε στοιχείο ινχοληψίας συμμετέχει στην διαδικασία ιχνοληψίας, έχουν τύπο *Volum*.

Οι παραπάνω δομές παρουσιάζονται αναλυτικά στο σχήμα Β.3 εκφρασμένες με βάση το συντακτικό της γλώσσας προγραμματισμού **C**, στην οποία έχει υλοποιηθεί η υπάρχουσα έκδοση του *ArrayTracer*.

Το όνομα των περισσοτέρων πεδίων δηλώνει και το είδος των περιεχομένων τους. Δι–ευκρινήσεις χρειάζεται μόνο για ορισμένα πεδία της δομής *VEntry*, τα οποία είναι τα εξής :

- Τα πεδία *widN* και *width*. Σε αυτά τα πεδία αποθηκεύονται πληροφορίες σχετικά με τις διαστάσεις που καθορίζουν τα όρια ενός υποπίνακα, στις περιπτώσεις που ο χρήστης επιλέξει κάποιον υποπίνακα για να συμμετέχει στην διαδικασία ιχνοληψίας. Η χρησιμότητά τους είναι ίδια με αυτήν των πεδίων *dimN* και *dim*. Η διαφορά τους από αυτά είναι ότι τα πεδία *dimN* και *dim* καθορίζονται κατά την συντακτική ανάλυση του πηγαίου κώδικα, όταν γίνουν γνωστές οι διαστάσεις ενός πίνακα και ο αριθμός τους, ενώ τα πεδία *widN* και *width* καθορίζονται από τον χρήστη κατά την διαδικασία εισαγωγής εντολών στην διεπαφή εισόδου. Οι διαφορές αυτές διασαφηνίζονται στο παράδειγμα Β.2.




Σχήμα Β.1: Παράδειγμα στοίχισης στην μνήμη δύο πινάκων, των οποίων δύο στοιχεία του δηλώθηκαν ισοδύναμα με την εντολή *EQUIVALENCE*.

- Το πεδίο *offs*. Αυτό το πεδίο χρησιμοποιείται μόνο σε περιπτώσεις που, κατά την συντακτική ανάλυση του πηγαίου κώδικα της εφαρμογής, βρεθεί κάποια μεταβλητή *ισοδύναμη* (αυτό γίνεται με την εντολή *EQUIVALENCE* της *Fortran*) με κάποιο από

τα στοιχεία ιχνοληψίας που έχει καθορίσει ο χρήστης. Οπως έχουμε ήδη αναφέρει, σε τέτοιες περιπτώσεις, όλες οι μεταβλητές που έχουν δηλωθεί ισοδύναμες με την μεταβλητή που έχει επιλέξει ο χρήστης για να συμμετέχει στην διαδικασία ιχνοληψίας, εισάγονται αυτόματα από το εργαλείο στον πίνακα ιχνοληψίας. Για να είναι δυνατή η διάκριση των στοιχείων του πίνακα ιχνοληψίας που έχει άμεσα εισάγει ο χρήστης, από αυτά που εισάγονται έμμεσα εξαιτίας της ισοδυναμίας των μεταβλητών, σε όσα στοιχεία ιχνοληψίας αντιστοιχούν σε *ισοδύναμες* μεταβλητές αποδίδεται η ίδια ετικέτα με το ισοδύναμο στοιχείο ιχνοληψίας, αλλά με αρνητικό πρόσημο (παράρτημα Δ). Το σχήμα Β.2 διασαφηνίζει την χρήση του πεδίου *offs*.

**User Input:**
**trace file** */tmp/my_file_1.f*
**trace variable** *a[1..10]* **from** *1* **to** *200*
**trace variable** *a[35..50]* **from** *25* **to** *250*
**trace variable** *l* **from** *1* **to** *50*
**quit**

**Trace Table Entries (before source code parsing):**
<*tag, name, dimN, dim, offs, widN, width, range*>
<1, a, 0, (nil), 0, 1, <1, 10>, <1, 200>>
<2, a, 0, (nil), 0, 1, <35, 50>, <25, 250>>
<3, l, 0, (nil), 0, 0, (nil), <1, 50>>

**Program Declarations:**
**INTEGER** *A, B, C, L*
**DIMENSION** *A(100), B(10,4)*
**EQUIVALENCE** *A(15), B(5,1), C*

**Trace Table Entries (after source code parsing):**
<*tag, name, dimN, dim, offs, widN, width, range*>
<1, a, 1, <1, 100>, 0, 1, <1, 10>, <1, 200>>
<2, a, 1, <1, 100>, 0, 1, <35, 50>, <25, 250>>
<−2, b, 2, (<1, 10>, <1, 4>), 10, 2, (<5, 10>, <3, 3>), <25, 250>>
<−2, b, 2, (<1, 10>, <1, 4>), 10, 2, (<1, 10>, <4, 4>), <25, 250>>
<−2, c, 0, (nil), 15, 0, (nil), <25, 250>>
<3, l, 0, (nil), 0, 0, (nil), <1, 50>>

Σχήμα Β.2: Παράδειγμα έμμεσης εισαγωγής στοιχείων στον πίνακα ιχνοληψίας εξαιτίας της δήλωσης *EQUIVALENCE* που υπάρχει στον πηγαίο κώδικα της εφαρμογής.

Από το παράδειγμα του σχήματος Β.2 γίνεται φανερό το είδος της πληροφορίας που αποθηκεύεται στο πεδίο *offs*. Στην *Fortran* υπάρχει η δυνατότητα να δηλωθούν δύο μεταβλητές ισοδύναμες. Αυτό σημαίνει ότι θα καταλαμβάνουν τον ίδιο χώρο στην μνήμη του συστήματος κατά την εκτέλεση του προγράμματος. Αν πρόκειται για δύο πίνακες του οποίους ο προγραμματιστής επιθυμεί να δηλώσει ως ισοδύναμους, αυτό μπορεί να το κάνει δηλώνοντας ως

ισοδύναμα ένα στοιχείο από τον κάθε πίνακα. Στην μνήμη του συστήματος εκτέλεσης του προγράμματος, τα στοιχεία των δύο πινάκων θα αντιστοιχιθούν στην ίδια θέση μνήμης και τα υπόλοιπα στοιχεία των πινάκων θα στοιχιθούν με γνώμονα αυτήν την αντιστοίχιση (σχήμα Β.1).

Ανάμεσα σε ένα σύνολο ισοδυνάμων στοιχείων, θεωρούμε ως στοιχείο αναφοράς αυτό το οποίο έχει επιλέξει ο χρήστης για να συμμετέχει στην διαδικασία ιχνοληψίας. Στο πεδίο *offs* όλων των στοιχείων του πίνακα ιχνοληψίας που έχουν εισαχθεί έμμεσα εξαιτίας της ισοδυναμίας με ένα στοιχείο αναφοράς, θα βάλουμε την *Ευκλείδια* απόσταση της πρώτης θέσης μνήμης που καταλαμβάνουν, από την πρώτη θέση μνήμης που καταλαμβάνει το στοιχείο αναφοράς τους. Αυτό χρησιμεύει σε περιπτώσεις που κάποιο από τα στοιχεία του συνόλου ισοδυναμίας είναι στοιχείο πίνακα (βλ. σχ. Β.1). Σε περίπτωση που τα στοιχεία του συνόλου ισοδυναμίας είναι όλα απλές μεταβλητές (scalar variables) τότε αυτό το πεδίο δεν εξυπηρετεί κανένα σκοπό.

```
typedef struct Table {
   char              fname[128];   /* File name */
   int               VarSize;      /* Size of Variable sub-table */
   int               CodeSize;     /* Size of Code sub-table */
   int               CommSize;     /* Size of Communication sub-table */
   VEntry            *var;         /* Variable's sub-table */
   CEntry            *code;        /* Code's sub-table */
   CEntry            *comm;        /* Communication's sub-table */
   struct Table      *next;        /* Pointer to next list-element */
};


typedef struct VEntry {
   int               tag;          /* Variable's trace-tag */
   char              *name;        /* Variable's name */
   int               dimN;         /* Number of variable's dimensions */
   Volum             *dim;         /* Dimensions of variable */
   int               offs;         /* Used with EQUIVALENCE variables */
   int               widN;         /* Number of sub-array dimensions */
   Volum             *width;       /* Sub-array dimensions */
   Volum             *range;       /* Region of trace in source code */
   struct VEntry     *next;        /* Pointer to next list-element */
};


typedef struct CEntry {
   int               tag;          /* Code/Communication trace tag */
   int               level;        /* Level of tracing */
   Volum             *range;       /* Region of trace in source code */
   struct CEntry     *next;        /* Pointer to next list-element */
};


typedef struct Volum {
   int               up;           /* Upper bound of the interval */
   int               low;          /* Lower bound of the interval */
   struct Volum      *next;        /* Pointer to next list-element */
};
```

Σχήμα Β.3: Οι δομές δεδομένων που χρησιμοποιήθηκαν για την κατασκευή του πίνακα ιχνοληψίας.

# Παράρτημα Γ

# Κατασκευή κοινόχρηστου ενταμιευτή

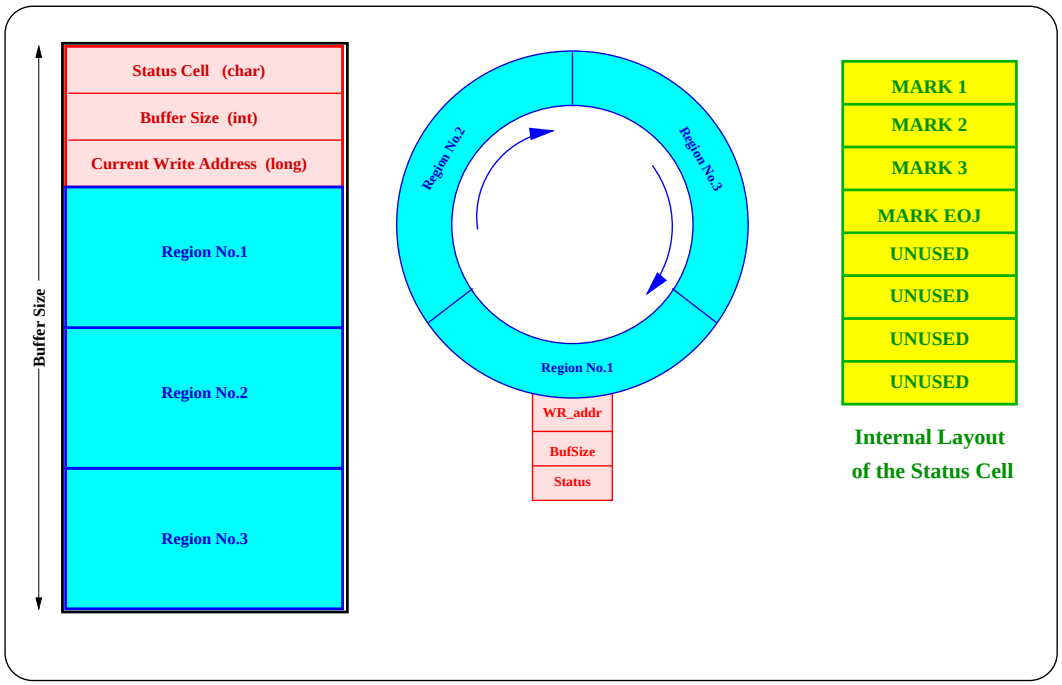
Ο κοινόχρηστος ενταμιευτής που χρησιμοποιείται για την επικοινωνία μεταξύ μιας διεργασίας της εφαρμογής και της αντίστοιχης διεργασίας–*συλλέκτη*, έχει υλοποιηθεί με βάση τις δυνατότητες που παρέχει το *Unix* για ύπαρξη κοινόχρηστης μνήμης μεταξύ δύο διεργασιών που εκτελούνται στο ίδιο μηχάνημα (*shmget( ), shmatt( ),* και *shmctl( )*). Ο ενταμιευτής αυτός χωρίζεται σε τέσσερα τμήματα, στο ιδιωτικό του τμήμα όπου αποθηκεύονται στοιχεία που αφορούν την κατάσταση του και σε τρία άλλα τμήματα στα οποία μπορούν να αποθηκευτούν δεδομένα.

Στην περίπτωση του *ArrayTracer*, ο ενταμιευτής αυτός χρησιμοποιείται για να υπάρχει όσο το δυνατόν γρηγορότερη παραγωγή ιχνών από την ιχνοληπτούμενη εφαρμογή και να μην επηρεάζεται πολύ ο χρόνος εκτέλεσής της. Κάθε κλήση σε ρουτίνα ιχνοληψίας της βιβλιοθήκης του *ArrayTracer* που αντιστοιχεί στην παραγωγή ενός ίχνους, προκαλεί και μια εγγραφή στον κοινόχρηστο ενταμιευτή. Το κόστος αυτής της εγγραφής είναι όσο το κόστος προσπέλασης στην τοπική μνήμη της διεργασίας της εφαρμογής, αφού ο κοινόχρηστος ενταμιευτής προσαρτίζεται τόσο στον χώρο μνήμης της διεργασίας της εφαρμογής όσο και στον χώρο μνήμης της διεργασίας–*συλλέκτη*.

Για να επιτευχθεί όμως η μικρότερη δυνατή καθυστέρηση, πρέπει να εξασφαλιστούν δύο παράγοντες.

1. Να μην μπλοκάρεται η διαδικασία εγγραφής. Αυτό δεν εξασφαλίζεται πάντα, όπως θα φανεί στην συνέχεια, γιατί η ρουτίνα εγγραφής δεδομένων μπλοκάρεται στην περίπτωση που δεν βρει ελεύθερο χώρο στον ενταμιευτή για να γράψει τα δεδομένα. Ομως στην πράξη αυτό δεν έχει παρατηρηθεί καμμία φορά γιατί η διεργασία–*συλλέκτης* που αναλαμβάνει την μεταφορά των δεδομένων από τον ενταμιευτή σε αρχείο δίσκου, δεν έχει επιφορτιστεί με άλλη εργασία κατά την διάρκεια εκτέλεσης της εφαρμογής. Ετσι, μόλις διαπιστώσει ότι κάποιο τμήμα εγγραφής δεδομένων του ενταμιευτή γέμισε, το αδειάζει αμέσως.

2. Να μην μεταφερθεί η σελίδα μνήμης του ενταμιευτή στην δευτερεύουσα μνήμη (swapped out). Αν συμβεί αυτό, η επόμενη εγγραφή δεδομένων θα επιβαρυνθεί με την καθυστέρηση επαναφοράς της σελίδας μνήμης στην οποία βρίσκεται ο ενταμιευτής από την δευτερεύουσα μνήμη (swap in). Στην περίπτωση της υλοποίησης του *ArrayTracer*, αυτό το φαινόμενο αποφεύγεται γιατί το μέγεθος του ενταμιευτή είναι μικρότερο από την σελίδα μνήμης του συστήματος στο οποίο τρέχει, και ο ενταμιευτής έχει ευθυγραμμιστεί ώστε να περιέχεται ολόκληρος στην ίδια σελίδα μνήμης. Αυτή η σελίδα δέχεται συνεχείς προσπελάσεις και έτσι δεν μεταφέρεται ποτέ στην δευτερεύουσα μνήμη του συστήματος.




Σχήμα Γ.1: Διαφορετικές απόψεις το κοινόχρηστου ενταμιευτή. Αριστερά παρουσιάζεται όπως ακριβώς βρίσκεται πάνω στην κοινόχρηστη μνήμη, στο κέντρο υπάρχει μια γραφική παράσταση της εικόνας κυκλικού ενταμιευτή που δίνει στην διεργασία που τον χρησιμοποιεί, ενώ στα δεξιά βρίσκεται η εσωτερική δομή του Status Cell.

### Γ.0.1 Τμήματα αποθήκευσης δεδομένων.

Τα τμήματα αυτά φέρουν τις ετικέτες *Region 1, Region 2,* και *Region 3*, όπως φαίνεται στο σχήμα Γ.1. Η λειτουργία τους είναι να αποθηκεύσουν προσωρινά τα δεδομένα που γράφει μία διεργασία, μέχρι μία άλλη διεργασία να τα μεταφέρει σε κάποιο αρχείο δίσκου. Δεν διαφέρουν στο μέγεθος τους το οποίο ισούται με το $^1/_3$ του συνολικού μεγέθους του ενταμιευτή μείον το χώρο που καταλαμβάνει το ιδιωτικό τμήμα.

Ο τρόπος με τον οποίο γεμίζουν με δεδομένα αυτά τα τμήματα ακολουθεί τον αλγόριθμο *round–robin*. Πρώτα γεμίζει το πρώτο τμήμα και μόνο όταν αυτό γεμίσει πλήρως αρχίζουν να γράφονται δεδομένα στο δεύτερο τμήμα. Το ίδιο συμβαίνει με το δεύτερο τμήμα που μόνο όταν γεμίσει μπορούν να γραφτούν δεδομένα στο τρίτο τμήμα. Οταν γεμίσει και το τρίτο τμήμα και μόνο αν η διεργασία που μεταφέρει τα δεδομένα από τον ενταμιευτή στο αρχείο δίσκου έχει προλάβει να αδειάσει το πρώτο τμήμα, τότε μπορεί να συνεχιστεί η εγγραφή δεδομένων στο πρώτο τμήμα κ.ο.κ.

Με ίδιο τρόπο ακριβώς γίνεται και η μεταφορά των δεδομένων που περιέχει ένα τμήμα του ενταμιευτή σε αρχείο δίσκου. Μόλις γεμίσει το πρώτο τμήμα του ενταμιευτή, μια διεργασία αναλαμβάνει να μεταφέρει τα δεδομένα που βρίσκονται μέσα σε αυτό το τμήμα στον δίσκο. Στην συνέχεια περιμένει μέχρι να γεμίσει το αμέσως επόμενο τμήμα του ενταμιευτή (όπως αυτό ορίζεται από τον αλγόριθμο *round–robin*) και τότε μόνο μεταφέρει σε αρχείο δίσκου τα δεδομένα που βρισκονται σε αυτό.

Στον παραπάνω μηχανισμό εγγραφής δεδομένων στον κοινόχρηστο ενταμιευτή, παρα–τηρούμε ότι είναι πιθανόν η διεργασία που γράφει δεδομένα να μπλοκάρει. Η κατάσταση αυτή θα προκύψει όταν γεμίσει το τμήμα εγγραφής δεδομένων στο οποίο γράφει η διεργασία και το αμέσως επόμενο τμήμα (όπως αυτό ορίζεται από τον αλγόριθμο *round–robin*) δεν είναι

ελεύθερο. Με το να εμποδίζουμε την διεργασία να γράψει στο κατελημένο τμήμα του ενταμιευτή, προστατεύουμε τα δεδομένα του τμήματος αυτού, τα οποία δεν έχουν γραφεί ακόμα στον δίσκο. Ομως, στην περίπτωση του *ArrayTracer*, η διεργασία που γράφει δεδομένα στον ενταμιευτή είναι η διεργασία της εφαρμογής. Κατά συνέπεια, δεν θα θέλαμε να μπλοκάρει αυτή η διεργασία γιατί αυτό θα προκαλούσε αύξηση του χρόνου εκτέλεσης της. Πράγματι, η διεργασία της εφαρμογής δεν βρίσκει ποτέ γεμάτο τον ενταμιευτή. Αυτό συμβαίνει γιατί η διεργασία–*συλλέκτης*, η οποία είναι υπεύθυνη για την μεταφορά των δεδομένων από τον ενταμιευτή στο αρχείο δίσκου, ελέγχει συνεχώς την κατάσταση του ενταμιευτή και ελευθερώνει κάθε τμήμα δεδομένων μόλις γεμίσει. Το γεγονός αυτό συνδυασμένο με το μέγεθος του ενταμιευτή, δεν επιτρέπει ποτέ στον δεύτερο να γεμίσει και να προκαλέσει το μπλοκάρισμα της διεργασίας εγγραφής.

### Γ.0.2 Το ιδιωτικό τμήμα.

Το τμήμα αυτό χρησιμοποιείται για να αποθηκεύσει στοιχεία που αφορούν την δομή του ενταμιευτή καθώς και μία περιγραφή της κατάστασης στην οποία βρίσκεται κάθε στιγμή. Οπως φαίνεται στο σχήμα Γ.1, καταλαμβάνει συνολικό χώρο 13 bytes (char = 1 byte, int = 4 bytes, long = 8 bytes). Ο χώρος που φέρει την ετικέτα *Buffer Size* χρησιμοποιείται για να αποθηκεύσει το συνολικό μέγεθος του ενταμιευτή (συμπεριλαμβανομένου και του χώρου που καταλαμβάνει αυτό το ιδιωτικό τμήμα). Η πληροφορία αυτή χρησιμεύει στον καθορισμό των ορίων των τμημάτων που αποθηκεύουν δεδομένα.

Ο χώρος που φέρει την ετικέτα *Current Write Address* αποθηκεύει την πρώτη ελεύθερη διεύθυνση του ενταμιευτή στην οποία μπορούν να γραφούν δεδομένα. Την διεύθυνση αυτήν την χρειάζεται κάθε διεργασία που επιχειρεί να γράψει δεδομένα στον ενταμιευτή. Αποθηκεύοντας την διεύθυνση αυτή στον ίδιο τον ενταμιευτή, επιτρέπουμε σε όλες τις διεργασίες που έχουν προσαρτίσει (attach) τον ενταμιευτή στο χώρο μνήμης τους, να μπορούν να την διαβάσουν. Ετσι, μπορούν όλες να γράφουν στο σωστό σημείο του, χωρίς να αφήνουν κενά και χωρίς να επικαλύπτουν δεδομένα που έχουν γραφεί από άλλη διεργασία. Επίσης, αυτό χρησιμεύει και κατά την διαδικασία εκκένωσης του ενταμιευτή, πριν την καταστροφή του. Η διεργασία που εκτελεί την μεταφορά των δεδομένων στο αρχείο δίσκου, την χρησιμοποιεί για να μπορέσει να μεταφέρει όλα τα δεδομένα που έχουν γραφεί, χωρίς να μεταφέρει *σκουπίδια*.

Τέλος, ο χώρος που φέρει την ετικέτα *Status Cell* περιέχει ένα σύνολο από σημαίες (flags) που δηλώνουν την κατάσταση του κάθε τμήματος για την εγγραφή δεδομένων, καθώς και αν έχει δοθεί εντολή για την καταστροφή του ενταμιευτή. Από τις οχτώ θέσεις που έχει το *Status Cell*, οι τέσσερις μόνο χρησιμοποιούνται στην υπάρχουσα υλοποίηση. Οι υπόλοιπες τέσσερις μπορούν να χρησιμοποιηθούν για να εξυπηρετήσουν κάποια νέα λειτουργία σε μελλοντική επέκταση του υπάρχοντος μηχανισμού (π.χ. στην δημιουργία μηχανισμού κλειδώματος για να επιτρέπεται σε πολλές διεργασίες να επιχειρούν ταυτόχρονα την εγγραφή δεδομένων στον ενταμιευτή). Με την δημιουργία του ενταμιευτή, αυτός ο χώρος αρχικοποιείται με την τιμή μηδέν. Δηλαδή όλες οι σημαίες είναι απενεργοποιημένες.

Στα δεξιά του σχήματος Γ.1, βλέπουμε τις σημαίες που περιέχει αυτός ο χώρος. Οι σημαίες με την ετικέτα *MARK 1, MARK 2* και *MARK 3* ενεργοποιούνται όταν το αντίστοιχο τμήμα εγγραφής δεδομένων γεμίσει. Ετσι, η διεργασία που έχει αναλάβει την μεταφορά των δεδομένων από τον ενταμιευτή στο αρχείο δίσκου, μπορεί να γνωρίζει την κατάσταση του τμήματος που περιμένει να γεμίσει για να μεταφέρει τα περιεχόμενά του στο αρχείο δίσκου. Οταν η αυτή η διεργασία μεταφέρει τα δεδομένα ενός συγκεκριμένου τμήματος στο αρχείο δίσκου, τότε απενεργοποιεί την αντίστοιχη σημαία. Ετσι η διεργασία που επιχειρεί να γράψει νέα δεδομένα στον ενταμιευτή, μπορεί να είναι σίγουρη ότι δεν θα επικαλύψει τα υπάρχοντα δεδομένα πριν αυτά έχουν αποθηκευτεί στον δίσκο.

Η σημαία με την ετικέτα *MARK EOJ* ενεργοποιείται όταν μια διεργασία που γράφει στον ενταμιευτή, αποφασίσει την καταστροφή του. Ουσιαστικά, αυτή η σημαία χρησιμοποιείται για να ειδοποιηθεί η διεργασία που μεταφέρει τα δεδομένα από τον ενταμιευτή στο αρχείο δίσκου ότι η διεργασία που γράφει δεδομένα στον ενταμιευτή, δεν πρόκειται να γράψει άλλα δεδομένα σε αυτόν. Ετσι, η διεργασία που μεταφέρει τα δεδομένα από τον ενταμιευτή στο αρχείο δίσκου, όταν διαπιστώσει ότι η σημαία αυτή έχει ενεργοποιηθεί, αδειάζει όλα τα γεμάτα τμήματα εγγραφής δεδομένων στο αρχείο δίσκου καθώς και τα έγγυρα περιεχόμενα του τμήματος στο οποίο γράφονταν δεδομένα αλλά δεν πρόλαβε να γεμίσει πριν ενεργοποιηθεί η σημαία με ετικέτα *MARK EOJ*. Για να εντοπίσει μέχρι πιο ακριβώς σημείο αυτού του τελευταίου τμήματος φτάνουν τα έγκυρα δεδομένα, χρησιμοποιεί την πρώτη ελεύθερη διεύθυνση προς εγγραφή, η οποία βρίσκεται αποθηκευμένη στο τρίτο μέρος του ιδιωτικού τμήματος του ενταμιευτή, το οποίο στο σχήμα Γ.1 φέρει την ετικέτα *Current Write Address*.

### Γ.0.3 Ρουτίνες διαχείρησης κοινόχρηστου ενταμιευτή

Για την διαχείρηση του κοινόχρηστου ενταμιευτή έχει υλοποιηθεί ένα σύνολο ρουτινών το οποίο συμπεριλαμβάνει τις εξής :

- **ShBufCreat().** Αυτή η ρουτίνα είναι υπεύθυνη για την κατασκευή του κοινόχρηστου ενταμιευτή μέσω της δυνατότητας κοινόχρηστης μνήμης που παρέχει το *Unix*. Καλείται μία φορά μόνο από την διεργασία που αναλαμβάνει την κατασκευή του ενταμιευτή και η οποία στην περίπτωση του *ArrayTracer* είναι η διεργασία της εφαρμογής.

- **ShBufDel().** Πολύ απλή η λειτουργία αυτής της ρουτίνας η οποία καταστρέφει το τμήμα κοινόχρηστης μνήμης που δημιούργησε η προηγούμενη ρουτίνα και πάνω στο οποίο υλοποιείται ο κοινόχρηστος ενταμιευτής. Καλείται μόνο μία φορά από την διεργασία που μεταφέρει τα δεδομένα στο αρχείο δίσκου και η οποία στην περίπτωση του *ArrayTracer* είναι η διεργασία−συλλέκτης.

- **ShBufAtt().** Η ρουτίνα αυτή προσαρτίζει (attaches) το δημιουργημένο τμήμα κοινόχρη−στης μνήμης στο χώρο μνήμης της διεργασίας που την καλεί. Καλείται από όλες τις διεργασίες που θέλουν να έχουν προσπέλαση στον κοινόχρηστο ενταμιευτή. Για το σχήμα ιχνοληψίας του *ArrayTracer*, αυτό σημαίνει ότι θα την καλέσουν και η διεργασία της εφαρμογής και η διεργασία−*συλλέκτης*. Πρώτα την καλεί η διεργασία της εφαρμο−γής η οποία και αρχικοποιεί το ιδιωτικό τμήμα του ενταμιευτή. Μετά την καλεί και η διεργασία−*συλλέκτης* και μπορεί πλέον να μπει σε λειτουργία ο μηχανισμός μεταφοράς των ιχνών που παράγει η εφαρμογή στα *αρχεία ιχνοληψίας δυναμικών δεδομένων*.

- **ShBufWrt().** Αυτή η ρουτίνα εκτελεί την λειτουργία εγγραφής δεδομένων στον κοι−νόχρηστο ενταμιευτή. Διαβάζει από το ιδιωτικό τμήμα του ενταμιευτή την πρώτη ελεύθερη διεύθυνση προς εγγραφή. Στην συνέχεια, αν η σημαία του τμήματος στο οποίο μέσα βρίσκεται αυτή η διεύθυνση είναι απενεργοποιημένη, τότε γράφει σε αυτήν την διεύθυνση τόσα δεδομένα όσα χωράνε στο συγκεκριμένο τμήμα εγγραφής δεδομένων του ενταμιευτή. Αν ο ελεύθερος χώρος σε αυτό το τμήμα είναι μεγαλύτερος από τον συνολικό όγκο των δεδομένων που προσπαθεί να γράψει αυτή η ρουτίνα, τότε τα δεδο−μένα γράφονται, ενημερώνεται το ιδιωτικό τμήμα του ενταμιευτή για την νέα ελεύθερη προς εγγραφή διεύθυνση και η ρουτίνα επιστρέφει. Αν όμως τα δεδομένα καταλαμ−βάνουν όγκο μεγαλύτερο από τον ελεύθερο χώρο του τμήματος εγγραφής δεδομένων μέσα στο οποίο βρίσκεται η ελεύθερη προς εγγραφή θέση μνήμης, τότε ακολουθείται η εξής διαδικασία: γράφονται όσα ακριβώς δεδομένα χωράνε σε αυτό το τμήμα εγγραφής και ελέγχεται αν η σημαία του επόμενου τμήματος είναι απενεργοποιημένη ώστε να

επιτρέπεται η εγγραφή. Η ίδια διαδικασία επαναλαμβάνεται μέχρι να γραφούν όλα τα δεδομένα στον ενταμιευτή. Αν κατά την διάρκεια αυτής της διαδικασίας, η σημαία του τμήματος στο οποίο πρόκειται να πραγματοποιηθεί το επόμενο βήμα της εγγραφής βρεθεί ενεργοποιημένη, τότε η ρουτίνα μπλοκάρεται περιμένοντας να απενεργοποιηθεί η συγκεκριμένη σημαία. Αυτό μας εξασφαλίζει την εγγραφή δεδομένων με όγκο μεγαλύτερο από αυτόν που καταλαμβάνει συνολικά ο ενταμιευτής, χωρίς να υπάρχει κίνδυνος επικάλυψης έγκυρων δεδομένων πριν αυτά μεταφερθούν στο αρχείο δίσκου.

- **ShBufDump().** Η λειτουργία αυτής της ρουτίνας είναι διπλή, να μεταφέρει τα δεδομένα από τον ενταμιευτή στο αρχείο δίσκου και να καταστρέψει τον ενταμιευτή στο τέλος της διαδικασίας εγγραφής. Στα πλαίσια αυτού του διπλού ρόλου, ελέγχει συνεχώς τις σημαίες του ενταμιευτή μέχρι να διαπιστώσει ότι έχει γεμίσει το επόμενο τμήμα, όπως αυτό καθορίζεται από τον αλγόριθμο *round–robin*. Τότε, μεταφέρει τα δεδομένα του τμήματος αυτού στο αρχείο δίσκου το οποίο καθορίζεται από το όρισμα της, και απενεργοποιεί την αντίστοιχη σημαία ώστε να μπορούν να γραφούν νέα δεδομένα στο συγκεκριμένο τμήμα. Αν κατά την διάρκεια που περιμένει να ενεργοποιηθεί μια συγκεκριμένη σημαία τμήματος εγγραφής διαπιστωθεί ότι έχει ενεργοποιηθεί η σημαία που δηλώνει το τέλος της διαδικασίας εγγραφών (*MARK EOJ*), τότε μεταφέρει όσα έγκυρα δεδομένα υπάρχουν στο συγκεκριμένο τμήμα (δηλαδή όσα δεδομένα έχουν γραφτεί σε αυτό χωρίς όμως να το έχουν γεμίσει) στο αρχείο δίσκου και στην συνέχεια καλεί την ρουτίνα *ShBufWrt( )* για να καταστρέψει τον ενταμιευτή, ο οποίος δεν χρειάζεται πλέον.

# Παράρτημα Δ

# Λεπτομέρειες Στατικής Ανάλυσης

Η φάση της *Στατικής Ανάλυσης* υλοποιείται στον προεπεξεργαστή πηγαίου κώδικα γλώσσας *Fortran*. Υπενθυμίζουμε σε αυτό το σημείο, ότι στις λειτουργίες της φάσης αυτής συμπεριλαμβάνονται η διαλογή των *στατικών ιχνών*, η επιλογή των *σημείων εισαγωγής* και η εισαγωγή *κώδικα καθοδήγησης* σε αυτά τα σημεία. Οι περισσότερες από αυτές τις ενέργειες είναι άμεσα συνδεδεμένες με την συντακτική ανάλυση της εφαρμογής. Ο εντοπισμός των κλήσεων συναρτήσεων, των βρόγχων επανάληψης, των προσβάσεων στην μνήμη και των υπόλοιπων δομικών στοιχείων του κώδικα της εφαρμογής, όπως αυτά καθορίζονται από την γραμματική της γλώσσας *Fortran*, γίνεται κατά την συντακτική ανάλυση του κώδικα. Με τον εντοπισμό κάθε τέτοιου στοιχείου, ελέγχεται ο *πίνακας ιχνοληψίας* για να διαπιστωθεί αν το συγκεκριμένο στοιχείο έχει επιλεχθεί για ιχνοληψία, και σε αυτήν την περίπτωση εισάγεται ο κατάλληλος κώδικας καθοδήγησης.

Εκτός όμως από τις ενέργειες της φάσης *Στατικής Ανάλυσης* που είναι άμεσα συνδεδεμένες με την συντακτική ανάλυση της εφαρμογής, υπάρχουν και μερικές ενέργειες οι οποίες απαιτούν ανεξάρτητη σχεδίαση. Για την συγκεκριμένη υλοποίηση του *ArrayTracer*, τέτοιες περιπτώσεις είναι δύο :

1. Η περίπτωση αντιμετώπισης των δηλώσεων *EQUIVALENCE* και *COMMON* της γλώσσας *Fortran*.

2. Η περίπτωση αντιμετώπισης της ιχνοληψίας μέσα στο σώμα συναρτήσεων που καλούνται στον κώδικα της εφαρμογής.

## Δ.1 EQUIVALENCE και COMMON

Η δήλωση *EQUIVALENCE* της *Fortran* αναθέτει δύο ή περισσότερες μεταβλητές του κυρίως προγράμματος στην ίδια διεύθυνση τοπικής μνήμης. Αυτό σημαίνει ότι επηρεάζοντας μία μεταβλητή, όλες οι *ισοδύναμές* της μεταβλητές θα υποστούν την ίδια μεταβολή. Συνεπώς, αν μία μεταβλητή από ένα σύνολο ισοδύναμων μεταβλητών επιλεχθεί για ιχνοληψία, τότε θα πρέπει κατά την φάση της *Στατικής Ανάλυσης* να εισάγεται κώδικας καθοδήγησης για κάθε αναφορά σε οποιοδήποτε στοιχείο του συγκεκριμένου συνόλου ισοδυναμίας. Επίσης, για την καλύτερη λειτουργία του εργαλείου μας, θα πρέπει να εξαλειφθούν περιπτώσεις που ο χρήστης επιλέγξει δύο στοιχεία από το ίδιο σύνολο ισοδυναμίας να συμμετέχουν στην διαδικασία ιχνοληψίας στην ίδια περιοχή του πηγαίου κώδικα.

### Δ.1.1 Αυτόματη επιλογή όλων των στοιχείων ενός συνόλου ισοδυναμίας

Οπως αναφέραμε προηγουμένως, στόχος μας είναι να εισάγουμε κώδικα για όλα τα στοιχεία ενός συνόλου ισοδυναμίας (όπως αυτό καθορίζεται από τις δηλώσεις *EQUIVALENCE* του προγράμματος) αν ο χρήστης έχει επιλέξει ένα από τα στοιχεία του για συμμετοχή στην διαδικασία ιχνοληψίας. Η μέθοδος που ακολουθούμε για να πετύχουμε τον στόχο μας είναι η εξής. Μόλις συναντήσουμε μια δήλωση *EQUIVALENCE* κατά την συντακτική ανάλυση του κώδικα της εφαρμογής ελέγχεται ο *πίνακας ιχνοληψίας* για να διαπιστωθεί αν κάποιο από τα στοιχεία του συνόλου ισοδυναμίας που ορίζεται με αυτήν την δήλωση έχει επιλεχθεί από τον χρήστη για συμμετοχή στην διαδικασία ιχνοληψίας. Αν βρεθεί ένα τέτοιο στοιχείο, τότε για όλα τα υπόλοιπα στοιχεία του συγκεκριμένου συνόλου ισοδυναμίας δημιουργείται ένα νέο στοιχείο στον πίνακα ιχνοληψίας.

Με αυτόν τον τρόπο, στην συνέχεια της φάσης της *Στατικής Ανάλυσης* του κώδικα, όλα τα στοιχεία του συνόλου ισοδυναμίας εμφανίζονται επιλεγμένα για ιχνοληψία, και δεν χρειάζεται καμμία άλλη αλλαγή σε αυτήν την φάση. Ομως, δεν αρκεί μόνο να εισαχθεί κώδικας καθοδήγησης για τα ίχνη όλων των στοιχείων του συγκεκριμένου συνόλου. Πρέπει επίσης να υπάρξει και ένα τρόπος αλληλοσχετισμού αυτών των ιχνών, ώστε να είναι δυνατόν να διαπιστωθεί τελικά σε ποιό στοιχείο ιχνοληψίας που έχει καθορίσει ο χρήστης αντιστοιχούν τα ίχνη που θα συλλεχθούν.

Ο τρόπος με τον οποίο γίνεται η διάκριση των ιχνών είναι μέσω των ετικετών (tags) που αποδίδονται στα αντίστοιχα στοιχεία ιχνοληψίας κατά την εισαγωγή τους στον πίνακα ιχνοληψίας. Συνεπώς, τα ίχνη των στοιχείων που ανήκουν στο ίδιο σύνολο ισοδυναμίας θα πρέπει να έχουν την ίδια ετικέτα. Αυτή ακριβώς είναι η δικιά μας προσέγγιση. Τα στοιχεία του πίνακα ιχνοληψίας που έχουν εισαχθεί κατά την διάρκεια της *Στατικής Ανάλυσης* εξαιτίας μιας δήλωσης *EQUIVALENCE* έχουν την ίδια ετικέτα που έχει και το στοιχείο του συνόλου ισοδυναμίας το οποίο είχε δηλώσει ο χρήστης για να συμμετέχει στην διαδικασία ιχνοληψίας.

Με αυτόν τον τρόπο όμως, στην επεξεργασία των δεδομένων που λαμβάνει χώρα μετά το πέρας εκτέλεσης της εφαρμογής, δεν είναι δυνατός ο διαχωρισμός των ιχνών που προέρχονται από το στοιχείο ιχνοληψίας που καθόρισε ο χρήστης, από εκείνα που προέρχονται από αναφορές στις υπόλοιπες μεταβλητές του συνόλου ισοδυναμίας στο οποίο ανήκει αυτό το στοιχείο. Αυτό βέβαια δεν είναι σημαντικό πρόβλημα, αφού ουσιαστικά όλα τα ίχνη αναφέρονται στην ίδια θέση μνήμης. Ομως περιορίζουν σε κάποιο βαθμό την αναλυτική παρουσίαση των αποτελεσμάτων της ιχνοληψίας.

Για να αποφύγουμε αυτόν τον περιορισμό, στηριζόμαστε στο γεγονός ότι οι ετικέτες που αποδίδονται στα στοιχεία του πίνακα ιχνοληψίας που εισάγει ο χρήστης, είναι θετικοί ακέραιοι αριθμοί. Ετσι, για να ξεχωρίσουμε αυτά τα στοιχεία του πίνακα ιχνοληψίας από εκείνα τα οποία εισάγονται εξαιτίας μιας δήλωσης *EQUIVALENCE*, δίνουμε στα τελευταία ετικέτα με την ίδια απόλυτη τιμή αλλά με *αρνητικό* πρόσημο. Αυτό καθιστά τον συσχετισμό των ιχνών ισοδυνάμων στοιχείων άμεσο χωρίς να μειώνει καθόλου τον βαθμό αναλυτικής παρουσίασης των αποτελεσμάτων της ιχνοληψίας.

**Η περίπτωση COMMON.** Η δήλωση *COMMON* έχει τα ίδια αποτελέσματα με την *EQUIVALENCE*, με την διαφορά ότι ενώ η δεύτερη χρησιμοποιείται για να δηλώσει την ισο–δυναμία μεταξύ μεταβλητών που ανήκουν όλες στο κύριο πρόγραμμα ή στην ίδια υπορουτίνα, η πρώτη χρησιμοποιείται για να δημιουργήσει σύνολα από μεταβλητές που καταλαμβάνουν την ίδια θέση μνήμης με μεταβλητές από άλλες υπορουτίνες ή από το κύριο πρόγραμμα. Πρακτικά, με την δήλωση *COMMON*, δημιουργούνται περιοχές στην μνήμη, στις οποίες μπορούν να ανατεθούν μεταβλητές από το κύριο πρόγραμμα και από τις υπορουτίνες της εφαρμογής.

Οι λόγοι για τους οποίους θέλουμε να παράγονται ίχνη από όλες τις μεταβλητές που

ανήκουν σε μια τέτοια περιοχή μνήμης αν μία από τις μεταβλητές αυτής της περιοχής έχει επιλεχθεί από τον χρήστη για να συμμετάσχει στην διαδικασία ιχνοληψίας, είναι οι ίδιοι με αυτούς που εκθέσαμε προηγουμένως για την περίπτωση των δηλώσεων *EQUIVALENCE*.

Οπως θα δουμε παρακάτω, όταν αρχίσει η συντακτική ανάλυση του κορμού μιας υπο−ρουτίνας (subroutine's body), κατασκευάζεται ένας νέος πίνακας ιχνοληψίας που ισχύει μόνο μέσα στην εμβέλεια του κώδικα της υπορουτίνας. Αυτός ο πίνακας περιλαμβάνει τόσο τα τυπικά ορίσματα της (formal parameters) που αντιστοιχούν σε πραγματικά ορίσματα (actual parameters) τα οποία έχουν επιλεχθεί για συμμετοχή στην διαδικασία ιχνοληψίας, όσο και τις μεταβλητές που έχουν δηλωθεί με την δήλωση *COMMON*, να ανήκουν στην ίδια πε−ριοχή μνήμης με κάποια μεταβλητή την οποία έχει επιλέξει ο χρήστης για συμμετοχή στην διαδικασία ιχνοληψίας.

### Δ.1.2 Απαλοιφή πολλαπλών ισοδύναμων στοιχείων ιχνοληψίας

Η παραγωγή ιχνών από όλα τα στοιχεία ενός συνόλου ισοδυναμίας, όπως αυτό ορίζεται με τις δηλώσεις *EQUIVALENCE* και *COMMON* ενός προγράμματος, είναι απαραίτητη προκειμένου να συγκεντρωθεί όλη η απαραίτητη πληροφορία που χρειάζεται για την ανάλυση της απόδοσης της εφαρμογής. Από την άλλη πλευρά, σε περίπτωση που ο χρήστης έχει επιλέξει για ιχνοληψία, περισσότερα από ένα στοιχεία ενός συνόλου ισοδυναμίας, είναι απαραίτητο να συσχετίσουμε αυτά τα στοιχεία προκειμένου να παραχθούν ακριβή αποτελέσματα κατά την ανάλυση της απόδοσης της εφαρμογής. Αν δεν συσχετίσουμε τα επιλεγμένα ισοδύναμα στοιχεία μεταξύ τους, τότε υπάρχει κίνδυνος να παραχθούν συμπεράσματα για αναφορές σε περισσότερες θέσεις μνήμης από αυτές που πραγματικά αναφέρθηκαν κατά την εκτέλεση της εφαρμογής. Γι' αυτόν τον λόγο, κατά την διάρκεια της διαδικασίας αυτόματης εισαγωγής στοιχείων στον πίνακα ιχνοληψίας, τα οποία αντιστοιχούν στις μεταβλητές ενός συνόλου ισοδυναμίας που ορίζει μια δήλωση *EQUIVALENCE*, γίνεται μια σάρωση ολόκληρου του υποπίνακα ιχνοληψίας που περιέχει τις μεταβλητές που έχουν επιλεχθεί για ιχνοληψία. Αν κατά την σάρωση αυτή βρεθεί το τρέχον στοιχείο του συνόλου ισοδυναμίας να είναι επιλεγμένο για ιχνοληψία, τότε αλλάζει η ετικέτα του παίρνοντας την απόλυτη τιμή της ετικέτας του στοιχείου−εκπροσώπου του συνόλου ισοδυναμίας και το αρνητικό πρόσημο.

Ο τρόπος επιλογής του στοιχείου−εκπροσώπου ενός συνόλου ισοδυναμίας είναι πολύ απλός. Ως τέτοιο στοιχείο επιλέγεται το πρώτο στοιχείο στην σειρά της *EQUIVALENCE* δήλωσης τους, το οποίο έχει επιλεχθεί για συμμετοχή στην διαδικασία ιχνοληψίας. Η επιλογή αυτή δεν προκαλεί κανένα πρόβλημα στην διαδικασία ανάλυσης της πληροφορίας που περιέχουν τα ίχνη.

### Δ.1.3 Ισοδύναμοι πίνακες

Μέχρι τώρα αναφερθήκαμε στην περίπτωση απλών μεταβλητών που ανήκουν στο ίδιο σύ−νολο ισοδυναμίας που παράγεται από μια δήλωση *EQUIVALENCE*. Σε σύνολα ισοδυναμίας όμως είναι δυνατόν να ανήκουν και στοιχεία πινάκων. Και επειδή τα στοιχεία πινάκων καταλαμβάνουν γειτονικές θέσεις στην τοπική μνήμη ενός προγράμματος, όταν δύο στοιχεία πινάκων δηλωθούν ισοδύναμα, τότε οι σειριακή αναπαράσταση των πινάκων αυτών στην μνήμη του συστήματος, ευθυγραμμίζεται με βάση την ισοδυναμία των δύο συγκεκριμένων στοιχείων του που συμμετέχουν στην δήλωση *EQUIVALENCE*. Στο σχήμα Β.2 υπάρχει μια γραφική περιγραφή της διαδικασίας μετατροπής των πινάκων στα σειριακά αντίστοιχα τους και ευθυγράμμιση των δευτέρων. Στην μετατροπή αυτή, μεταφέρεται και ο υποπίνακας του πίνακα της εφαρμογής που έχει επιλεχθεί για ιχνοληψία, στα τμήματα του μονοδιάστατου πίνακα που του αντιστοιχούν. Στην συνέχεια, τα τμήματα αυτά μεταφράζονται σε υποπίνακες

ενός δισδιάστατου πίνακα ο οποίος έχει δηλωθεί ισοδύναμος με τον αρχικό τρισδιάστατο. Τα διακεκομμένα περιγράμματα οριοθετούν αυτούς ακριβώς τους υποπίνακες.

Οι χειρισμοί που χρειάζονται για να αντιμετωπιστεί ένα σύνολο ισοδυναμίας το οποίο περιέχει περισσότερους από έναν πίνακες, είναι πιο πολύπλοκοι από τους αντίστοιχους στην περίπτωση που το σύνολο ισοδυναμίας αποτελείται από απλές μεταβλήτες. Στην περίπτωση συνόλου ισοδυναμίας που περιέχει στοιχεία πινάκων, και που συμβαίνει τα στοιχεία αυτά να ανήκουν σε υποπίνακα επιλεγμένο για συμμετοχή στην διαδιακασία ιχνοληψίας, ακολουθούνται τα εξής βήματα :

- Με βάση τις πληροφορίες που υπάρχουν στον πίνακα ιχνοληψίας για τις διαστάσεις του πίνακα της εφαρμογής του οποίου ένα στοιχείο ανήκει σε σύνολο ισοδυναμίας, γίνεται μετατροπή του σε μονοδιάστατο πίνακα. Ετσι δημιουργείται ο *πρώτος μονοδιάστατος υποπίνακας*. Στο σχήμα Δ.1, αυτό το βήμα περιλαμβάνει την μετατροπή του τρισδιάστατου πίνακα της πάνω αριστερής γωνίας, στην μονοδιάστατη αναπαράσταση του πάνω σε θέσεις μνήμης.

- Ανάλογη διαδικασία ακολουθείται και για τον δεύτερο πίνακα της εφαρμογής του οποίου ένα στοιχείο ανήκει στο ίδιο σύνολο ισοδυναμίας. Αποτέλεσμα είναι η δημιουργία του *δεύτερου μονοδιάστατου υποπίνακα*.

- Γίνεται η αντιστοίχιση του αρχικού υποπίνακα που έχει επιλεχθεί για ιχνοληψία, σε υποπίνακες του πρώτου μονοδιάστατου.

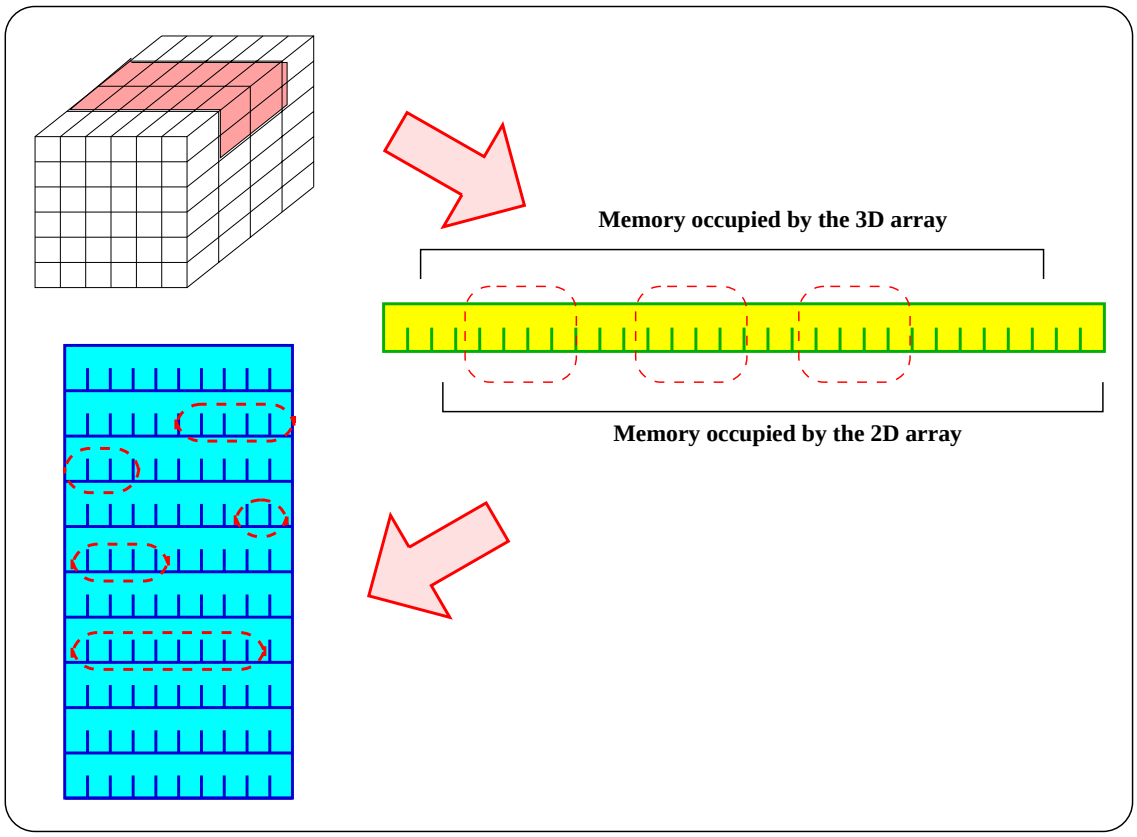
- Εκτελείται η ευθυγράμμιση των δύο μονοδιάστατων πινάκων με βάση την ισοδυναμία των στοιχείων του που ανήκουν στο σύνολο ισοδυναμίας της δήλωσης *EQUIVALENCE* (βλ. παράρτημα Β). Στην συνέχεια υπολογίζεται η *Ευκλείδια* απόσταση της θέσης μνήμης του αρχικού στοιχείου του πρώτου μονοδιάστατου πίνακα από την θέση μνήμης του αρχικού στοιχείου του δεύτερου μονοδιάστατου πίνακα. Η απόσταση αυτή αποθηκεύεται στο πεδίο *offs* του στοιχείου του πίνακα ιχνοληψίας που αντιστοιχεί στον δεύτερο πίνακα. Στο ίδιο βήμα, γίνεται και η αντιστοίχιση των υποπινάκων του αρχικού τρισδιάστατου πίνακα που θα συμμετέχουν στην διαδικασία ιχνοληψίας, σε υποπίνακες του πρώτου μονοδιάστατου πίνακα.

- Με βάση την απόσταση των αρχικών σημείων των δύο μονοδιάστατων πινάκων γίνεται η αντιστοίχιση των υποπινάκων του πρώτου μονοδιάστατου πίνακα που έχουν επιλεχθεί για συμμετοχή στην διαδικασία ιχνοληψίας, με υποπίνακες του δευτέρου μονοδιάστατου πίνακα.

- Γίνεται η αντιστοίχιση των υποπινάκων του δεύτερου μονοδιάστατου πίνακα που έχουν επιλεχθεί για ιχνοληψία, με υποπίνακες του δεύτερου πίνακα που ανήκει στο σύνολο ισοδυναμίας. Στο σχήμα Δ.1 ο πίνακας αυτός είναι ο δισδιάστατος πίνακας που βρίσκεται στην κάτω αριστερή γωνία της εικόνας.

## Δ.2 Ιχνοληψία ρουτινών

Ενα άλλο πολύ σημαντικό σημείο της υλοποίησης της *Στατικής Ανάλυσης* του *ArrayTracer* είναι η διαδικασία ιχνοληψίας μέσα στον κορμό των υπορουτινών της εφαρμογής. Η δυσκολία στις περιπτώσεις των υπορουτινών είναι η εξής: τις περισσότερες φορές μέσα στον κώδικα της εφαρμογής η ίδια συνάρτηση καλείται με διαφορετικές πραγματικές παραμέτρους (actual parameters). Στις περιπτώσεις αυτές, όταν οι πραγματικές παράμετροι της κάθε κλήσης είναι




Σχήμα Δ.1: Στο σχήμα αυτό φαίνεται η μετατροπή ενός τρισδιάστατου πίνακα (πάνω αριστερά) στον αντίστοιχο μονοδιάστατο που αναπαριστά την φυσική τοποθέτηση του πίνακα του προγράμματος σε θέσεις τοπικής μνήμης.

στοιχεία ιχνοληψίας, πρέπει να εισαχθεί κατάλληλος κώδικας ιχνοληψίας στον κορμό της υπορουτίνας ώστε να παραχθούν τα αντίστοιχα ίχνη.

Ομως, οι πραγματικές παράμετροι της κάθε κλήσης της ίδιας υπορουτίνας μπορεί να είναι διαφορετικά στοιχεία ιχνοληψίας (διαφορετικές μεταβλητές του προγράμματος που έχουν επιλεχθεί για συμμετοχή στην διαδικασία ιχνοληψίας). Συνεπώς, το πρόβλημα που δημιουργείται είναι πώς θα εισαχθεί ο κατάλληλος κώδικας καθοδήγησης στον κορμό των υπορουτινών ώστε και να παραχθούν όλα τα ίχνη των παραμέτρων που θα δοθούν στις διάφορες κλήσεις της συγκεκριμένης υπορουτίνας μέσα στο πρόγραμμα, αλλά και να αποφύγουμε την παραγωγή *άχρηστων* ιχνών. Δηλαδή να μην παράγονται ίχνη για όλες τις πραγματικές παραμέτρους που θα δεχθεί η υπορουτίνα κατά την εκτέλεση της εφαρμογής, παρά μόνο για αυτές που έχουν επιλεχθεί για συμμετοχή στην διαδικασία ιχνοληψίας.

Οι εναλλακτικές λύσεις που ερευνήσαμε ήταν δύο :

1. Κατασκευή αντιγράφων των υπορουτινών. Για κάθε υπορουτίνα που καλείται με πραγ–ματικές παραμέτρους οι οποίες έχουν επιλεχθεί για συμμετοχή στην διαδικασία της ιχνοληψίας, θα κατασκευάζονται τόσα αντίγραφα όσες και οι κλήσεις της συγκε–κριμένης υπορουτίνας που περιέχουν πραγματικές παραμέτρους οι οποίες αποτελούν διαφορετικά στοιχεία ιχνοληψίας. Κάθε υπορουτίνα–αντίγραφο θα διακρίνεται από το όνομα της το οποίο θα είναι το ίδιο με το όνομα της πρωτότυπης υπορουτίνας επαυ–ξημένο με ένα νούμερο το οποίο αντιπροσωπεύει τον αριθμό έκδοσης του αντιγράφου. Επίσης, κάθε αντίγραφο θα περιέχει κώδικα καθοδήγησης που θα αφορά μόνο στις

συγκεκριμένες πραγματικές παραμέτρους με τις οποίες θα καλείται μέσα στον κώδικα της εφαρμογής που θα έχει επαυξηθεί με εντολές ιχνοληψίας. Για τις περιπτώσεις που η υπορουτίνα θα καλείται με πραγματικές παραμέτρους που δεν έχουν επιλεχθεί για συμμετοχή στην διαδικασία ιχνοληψίας, θα χρησιμοποιείται η πρωτότυπη υπορουτίνα χωρίς καμία αλλαγή/προσθήκη.

Η προσέγγιση αυτή αυξάνει τον αριθμό των υπορουτινών που περιέχει ο κώδικας της εφαρμογής που έχει εμπλουτιστεί με κλήσεις σε ρουτίνες ιχνοληψίας. Επίσης, αυξάνει την πολυπλοκότητα της διαδικασίας συντακτικής ανάλυσης του κώδικα της εφαρμογής, αλλά δεν επηρεάζει την πολυπλοκότητα της διαδικασίας *Στατικής Ανάλυσης* της εφαρμογής.

2. Προσθήκη κώδικα ιχνοληψίας στον κορμό της υπορουτίνας, που να παίρνει αποφάσεις κατά την διάρκεια εκτέλεσης της εφαρμογής για την παραγωγή ιχνών. Δεν θα υπάρχουν αντίγραφα της κάθε υπορουτίνας, αλλά σε κάθε εντολή του κορμού της που θα μπορούσε να παράγει ίχνος σε τουλάχιστον μία κλήση της υπορουτίνας, θα προστίθεται κώδικας καθοδήγησης ο οποίος θα είναι υπεύθυνος να αποφασίσει ποιο ακριβώς ίχνος θα πρέπει να παραχθεί.

   Η προσέγγιση αυτή δεν επηρεάζει τον αριθμό των υπορουτινών του προγράμματος, ούτε την πολυπλοκότητα της διαδικασίας συντακτικής ανάλυσης του κώδικας της εφαρμογής. Ομως, αυξάνει την πολυπλοκότητα της διαδικασίας *Στατικής Ανάλυσης* σε μεγάλο βαθμό. Επίσης, αυξάνει πολύ την πολυπλοκότητα του εκτελέσιμου κώδικα των υπορουτινών, αφού θα πρέπει σε κάθε κλήση του να γίνονται όλοι οι έλεγχοι που αντιστοιχούν σε κάθε εντολή της υπορουτίνας η οποία μπορεί να παράγει ίχνος σε κάποια κλήση της.

Η προσέγγιση που επιλέξαμε εμείς για την υπάρχουσα υλοποίηση του *ArrayTracer* είναι η πρώτη, δηλαδή η κατασκευή πολλαπλών αντιγράφων της κάθε υπορουτίνας του προγράμματος.

**Πολλαπλά αντίγραφα υπορουτινών** Για την κατασκευή πολλαπλών αντιγράφων του κώδικα μιας υπορουτίνας που καλείται σε ένα πρόγραμμα, θα πρέπει να είναι διαθέσιμος ο πηγαίος κώδικας της. Αυτό το γεγονός σημαίνει ότι δεν θα πραγματοποιείται εισαγωγή κώδικα καθοδήγησης σε υπορουτίνες που δεν έχουμε τον πηγαίο κώδικα τους, όπως οι υπορουτίνες των διαφόρων βιβλιοθηκών λογισμικού. Η κατασκευή των αντιγράφων των υπορουτινών στις οποίες έχουμε προσπέλαση στον πηγαίο κώδικα τους, γίνεται με την ακόλουθη διαδικασία :

- Κατά την συντακτική ανάλυση του κώδικα (code parsing), όταν εντοπιστεί ο κορμός μιας υπορουτίνας, αυτός αποθηκεύεται σε ένα προσωρινό αρχείο. Ετσι, για κάθε αρχείο που περιέχει ένα κύριο πρόγραμμα που πρόκειται να εκτελεστεί από μία διεργασία της παράλληλης εφαρμογής, δημιουργείται ένα σύνολο προσωρινών αρχείων, του οποίου κάθε στοιχείο περιέχει τον πηγαίο κώδικα μίας υπορουτίνας του συγκεκριμένου κυρίου προγράμματος. Το σχήμα Δ.2 διασαφηνίζει την κατάσταση που επικρατεί κατά την διαδικασία κατασκευής αντιγράφων υπορουτινών που λαμβάνει χώρα κατά την διάρκεια της *Στατικής Ανάλυσης* της εφαρμογής.

- Κατά την διαδικασία της *Στατικής Ανάλυσης*, όταν η πραγματική παράμετρος μιας υπορουτίνας είναι επιλεγμένη για συμμετοχή στην διαδικασία ιχνοληψίας, και αν στο συγκεκριμένο τμήμα του πηγαίου κώδικα ο χρήστης έχει ζητήσει ιχνοληψία δομικών στοιχείων του προγράμματος σε επίπεδο κλήσης υπορουτίνας, τότε συμβαίνει το εξής: σταματάει η συντακτική ανάλυση του πηγαίου κώδικα για να αρχίσει η συντακτική ανάλυση του προσωρινού αρχείου που περιέχει τον πηγαίο κώδικα της συγκεκριμένης υπορουτίνας.

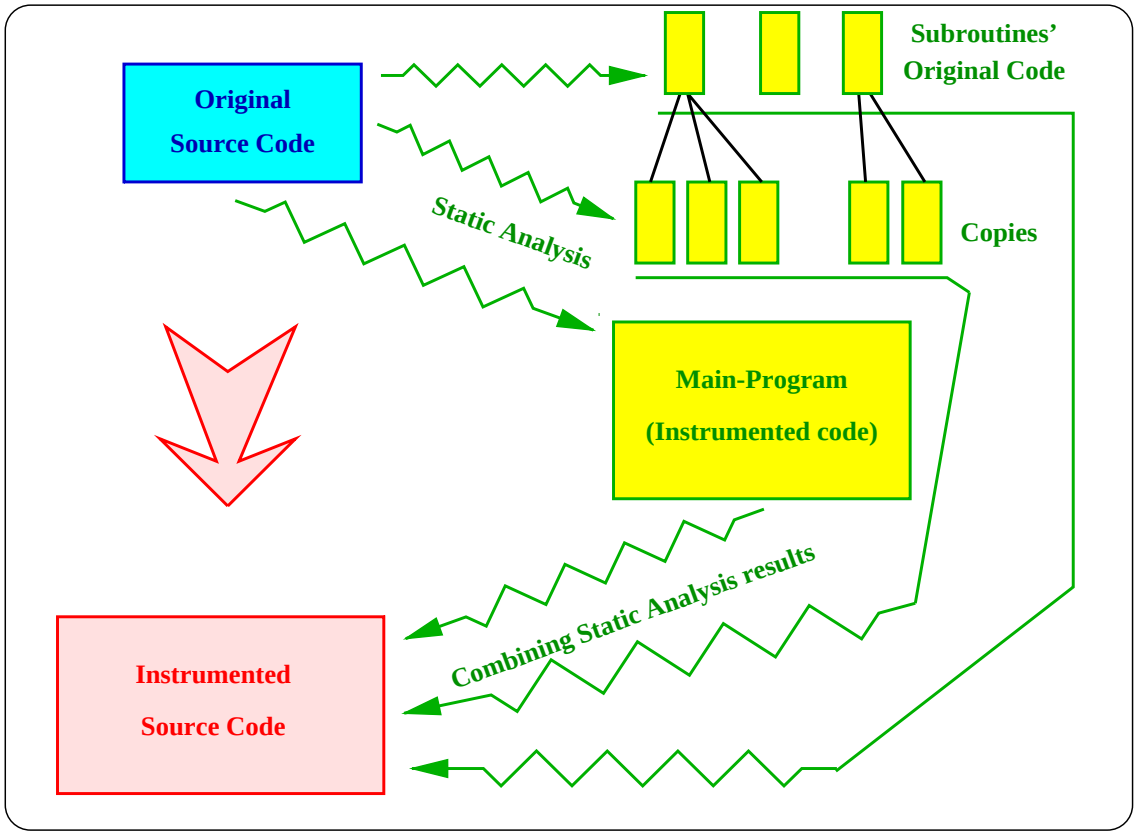
- Πριν αρχίσει η συντακτική ανάλυση του πηγαίου κώδικα της υπορουτίνας, φτιάχνεται ένας νέος πίνακας ιχνοληψίας που περιέχει σαν στοιχεία του τις τυπικές παραμέτρους της υπορουτίνας οι οποίες αντιστοιχούν σε πραγματικές παραμέτρους που πρέπει να συμμετέχουν στην διαδικασία ιχνοληψίας. Ο πίνακας αυτός συμπληρώνεται με τα στοιχεία που ορίζουν οι διάφορες δηλώσεις *COMMON* που πιθανόν να υπάρχουν στον κώδικα της υπορουτίνας, όπως αναφέραμε σε προηγούμενη παράγραφο.

- Τέλος, αρχίζει η συντακτική ανάλυση και συγχρόνως η *Στατική Ανάλυση* του πρωτότυπου πηγαίου κώδικα της συγκεκριμένης υπορουτίνας. Το αποτέλεσμα αυτής της διαδικασίας, δηλαδή ο πηγαίος κώδικας του αντιγράφου της υπορουτίνας που περιέχει και εντολές ιχνοληψίας, αποθηκεύεται σε ένα άλλο προσωρινό αρχείο, το οποίο περιέχει όλα τα αντίγραφα μιας συγκεκριμένης υπορουτίνας. Στο σχήμα Δ.2 φαίνονται τρία προσωρινά αρχεία με τους πρωτότυπους πηγαίους κώδικες των υπορουτινών (πάνω δεξιά γωνία του σχήματος). Από αυτές τις υπορουτίνες, κατασκευάζονται τρία αντίγραφα για την πρώτη, κανένα για την δεύτερη και δύο για την τρίτη.

- Οταν ολοκληρωθεί η συντακτική ανάλυση του πρωτότυπου πηγαίου κώδικα της υπορουτίνας, ο έλεγχος της διαδικασίας *Στατικής Ανάλυσης* επιστρέφει στο σημείο του κώδικα του κυρίου προγράμματος στο οποίο είχε εντοπίσει την συγκεκριμένη υπορουτίνα, αντικαθιστά το πρωτότυπο όνομα της με το όνομα του αντιγράφου που μόλις κατασκεύασε και συνεχίζει την διαδικασία της *Στατικής Ανάλυσης* του προγράμματος μέχρι να εντοπίσει μία άλλη κλήση υπορουτίνας, οπότε επαναλαμβάνει την ίδια διαδικασία.

Εδώ θα πρέπει να αναφέρουμε ότι σε περίπτωση που η υπορουτίνα η οποία εντοπίστηκε κατά την *Στατική Ανάλυση* του πηγαίου κώδικα του κυρίως προγράμματος δεν αντιστοιχεί σε μια υπορουτίνα από αυτές των οποίων ο πηγαίος κώδικας έχει ήδη αποθηκευτεί σε προσωρινά αρχεία, τότε το εργαλείο μας τις θεωρεί υπορουτίνες βιβλιοθήκης και δεν εκτελεί την παραπάνω διαδικασία.

Ενα άλλο σημείο που αξίζει να τονιστεί είναι το γεγονός ότι η διαδικασία που παρουσιάσαμε προηγουμένως έχει υλοποιηθεί ώστε να μπορεί να εκτελεστεί αναδρομικά. Δηλαδή, αν σε κάποιο σημείο την *Στατικής Ανάλυσης* του πηγαίου κώδικα μιας υπορουτίνας, εντοπιστεί κλήση σε μια άλλη (ή ακόμα και στην ίδια) υπορουτίνα, τότε σταματάει η *Στατική Ανάλυση* της αρχικής υπορουτίνας και ξεκινάει η ίδια διαδικασία για την υπορουτίνα που μόλις εντοπίστηκε. Το βάθος αναδρομικότητας δεν περιορίζεται από κανένα παράγοντα. Συνεπώς, η διαδικασία της ιχνοληψίας μπορεί να λάβει χώρα σε οποιοδήποτε βάθος φωλιασμένων κλήσεων υπορουτινών.




Σχήμα Δ.2: Αναπαράσταση της διαδικασίας κατασκευής αντιγράφων των υπορουτινών ενός προγράμματος κατά της διαδικασία της *Στατικής Ανάλυσης* της εφαρμογής.

# Παράρτημα Ε

# Ευρετήριο Ορων

| | |
|---|---|
| absolutl path | *απόλυτο μονοπάτι* |
| address bus | *διάδρομος διευθύνσεων* |
| alignment | *στοίχιση, ευθυγράμμιση* |
| animation | *κινηματογραφική παράσταση* |
| applicability | *δυνατότητα εφαρμογής* |
| assembly | *γλώσσα μηχανής* |
| barrier | *φράγμα λογισμικού* |
| buffer | *ενταμιευτής* |
| busy waiting | *ενεργή αναμονή* |
| compiler | *μεταφραστής* |
| correctness debugger | *εργαλείο διάγνωσης προβλημάτων ορθότητας* |
| correlation | *συγκερασμός* |
| deadlock | *αδιέξοδο* |
| debugger | *εργαλείο διάγνωσης προβλημάτων* |
| dilation | *διαστολή* |
| distortion | *διάβρωση* |
| feedback | *ανάδραση* |
| hardware | *υλικό* |
| instrumentation | *καθοδήγηση* |
| interface | *διεπαφή* |
| interpreter | *διερμηνέας* |
| interprocess communication | *ενδοεπικοινωνία διεργασιών* |
| linking | *σύνδεση* |
| loops | *βρόχοι επανάληψης* |
| massive parallelism | *μαζικός παραλληλισμός* |
| master−process | *διεργασία−αφέντης* |
| message passing | *ανταλλαγή μηνυμάτων* |
| monitor | *εργαλείο παρακολούθησης* |
| on−chip cache | *κρυφή μνήμη επί του κυκλώματος* |
| on−the−fly analysis | *ανάλυση πραγματικού χρόνου* |
| overhead | *επιβάρυνση* |
| page−fault | *λάθος σελίδας* |
| parsing | *συντακτική ανάλυση* |
| performance debugger | *εργαλείο διάγνωσης προβλημάτων επίδοσης* |
| profiler | *εργαλείο ανάλυσης φυσιογνωμίας* |
| remote memory | *απομακρυσμένη μνήμη* |

| | |
|---|---|
| run−time overhead | *επιβάρυνση χρόνου εκτέλεσης* |
| run−time system | *περιβάλλον εκτέλεσης* |
| scalar variable | *απλή μεταβλητή* |
| section | *ενότητα* |
| selective tracing | *επιλεκτική ιχνοληψία* |
| self−ordered list | *αυτο−ταξινομούμενη λίστα* |
| shared memory | *κοινόχρηστη μνήμη* |
| signaling | *αποστολή σημάτων* |
| slave−process | *διεργασία−σκλάβος* |
| software | *λογισμικό* |
| source code | *πηγαίος κώδικας* |
| subroutine body | *κορμός υπορουτίνας* |
| system call | *κλήση συστήματος* |
| system resources | *πόροι συστήματος* |
| thread | *ίνα* |
| time−out | *χρονο−όριο* |
| time−sharing system | *χρονο−διαιρετό σύστημα* |
| trace | *ίχνος* |
| trace driven simulation | *ιχνοκατευθυνόμενη προσομοίωση* |
| trace records | *εγγραφές ιχνοληψίας* |
| trace template | *φόρμα ιχνοληψίας* |
| tracing | *διαδικασία ιχνοληψίας* |
| valid | *έγκυρος* |
| virtual memory | *ιδεατή μνήμη* |
| visualization | *οπτικοποίηση* |

# Βιβλιογραφία


[1] BBN Systems and Technologies. *TotalView: User's Guide*, April 1994.

[2] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, K. Moore, and V. Sunderam. Pvm and hence: Tools for heterogeneous network programming. In *Environments and Tools for Parallel Scientific Computing*, 1993.

[3] T. Bemmerl and P. Braun. Visualization of message passing parallel programs with the topsys parallel programming environment. *Journal of Parallel and Distributed Computing*, 18, 1993.

[4] W. J. Bolosky, R. P. Fitzgerald, and M. L. Scott. Simple but effective techniques for numa memory managment. In *Proceedings of the 12th Symposium on Operating Systems Principles*, December 1989.

[5] E. A. Brewer and C. N. Dellarocas. Proteus user documentation. Technical Report MA02139, MIT, 545 Technology Square, Cambridge, 1991.

[6] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl. Proteus: A high–performance parallel architecture simulator. In *Proceedings of the 12th ACM SIGMETRICS and PERFORMANCE '92 Conference*, June 1992.

[7] N. Carriero and D. Gelernter. *How to Write Parallel Programs. A First Course*. The MIT Press, Cambridge, Massachusetts, London, England, 1992.

[8] Panos Constantopoulos and Martin Doerr. The semantic index system: A brief presentation. ICS – FORTH, Working Paper No.6, 1993.

[9] Panos Constantopoulos and Martin Doerr. Software classification and static analysis in the software information base. *ERCIM News*, 14, July 1993.

[10] CONVEX Press, Richardson – Texas, USA. *CXdb Reference: Concepts and Messages*, 1 edition, November 1992.

[11] Mark E. Corvella and Thomas J. LeBlanc. Performance debugging using parallel performance predicates. Technical Report WPDD, University of Rochester, 1993.

[12] H. Davis, S. R. Goldschmidt, and J. Hennessy. Multiprocessor simulation and tracing using tango. In *Proceedings of the 1991 International Conference on Parallel Processing*, 1991.

[13] Digital Equipment Corporation, Maynard, Massachusetts. *Atom: Reference Manual*, December 1994.

[14] Digital Equipment Corporation, Maynard, Massachusetts. *Atom: User Manual*, March 1994.

[15] I. Glendinning, V. S. Getov, A. Hellberg, R. W. Hockney, and D. J. Pritchard. Performance visualization in a portable parallel programming environment. In *Proceedings of Workshop on Monitoring and Visualization of Parallel processing Systems, Moravany*, October 1992.

[16] Aaron H. Goldberg and John L. Hennessy. Mtool: An integrated system for performance debugging shared memory multiprocessor applications. *IEEE Transactions on Parallel and Distributed Systems*, 4(1), January 1993.

[17] Graphics, Visualization and Usability Center. *Visualization, Animation in Cluster Environments – PVaniM*, 1995.

[18] W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, and D. J. Vetter. Falcon: On−line monitoring and steering of large scale parallel programs. Technical Report GA 30332, Georgia Institute of Technology, Atlanta, 1994.

[19] M. T. Heath and J. A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, September 1991.

[20] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[21] Jeffrey K. Hollingsworth, R. Bruce Irvin, and Barton P. Miller. The integration of application and system based metrics in a parallel program performance tool. In *Proceedings of th 3rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, April 1991.

[22] Jeffrey K. Hollingsworth and Barton P. Miller. Dynamic control of performance monitoring on large scale parallel systems. Technical Report tr1133, University of Madison−Wisconsin, 1993.

[23] Jeffrey K. Hollingsworth, Barton P. Miller, and Jon Cargille. Dynamic program instrumentation for scalable performance tools. Technical Report tr1207, University of Madison−Wisconsin, 1993.

[24] Anna Hondroudakis. Performance analysis tools for parallel programs. EPCC, The University of Edinburgh, July 1995.

[25] R. Bruce Irvin and Barton P. Miller. A performance tool for high−level parallel programming languages. Technical Report tr1204, University of Madison−Wisconsin, 1993.

[26] R. Bruce Irvin and Barton P. Miller. Multiapplication support in a parallel−program performance tool. *IEEE Parallel & Distributed Technology*, Spring 1994.

[27] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. New York: Addison−Wiley, 1991.

[28] M. Katevenis. Telegraphos: High−speed communication architecture for parallel and distributed computer systems. Technical Report tr123, Institute of Computer Science – FORTH, May 1994.

[29] R. E. Kessler, A. Borg, and D. W. Wall. Generation and analysis of very long address traces. In *Proceedings of the 17th International Conference on Computer Architectures*, 1990.

[30] D. Kimelman and T. Ngo. Program visualization for rp3: An overview. Technical report, IBM Research Division, T. J. Watson Research Center, 1990.

[31] R. Klar. Event−driven monitoring of parallel programs. In *Proceedings of Monitoring and Visualizing of Parallel Processing Systems, Moravany, CSFR*, October 1992.

[32] James R. Larus. Abstract execution: A technique for efficiently tracing programs. *Software Practice and Experience*, 20(12), December 1990.

[33] Thomas J. LeBlanc and John M. Mellor−Crummey. Debugging parallel programs with instant replay. *IEEE TRansactions on Computers*, C−36(4), April 1987.

[34] Thomas J. LeBlanc, John M. Mellor−Crummey, and Robert J. Flower. Analyzing parallel program executions using multiple views. *Journal of Parallel and Distributed Computing*, 9, 1990.

[35] T. Lehr, Z. Segall, D. F. Vrsalovic, E. Caplan, A. L. Chung, and C. E. Fineman. Visualizing performance debugging. *Computer*, October 1989.

[36] M. Martonosi, A. Goopta, and T. Anderson. Memspy: Analyzing memory system bottlenecks in programs. In *ACM SIGMETRICS, Performance Evaluation Review*, volume 20, June 1992.

[37] John May and Francine Berman. Panorama: A portable, extensible parallel debugger. *ACM SIGPLAN*, 28(12), December 1993.

[38] B. P. Miller. What to draw? when to draw? an essay on parallel program visualization. Technical Report tr1024, University of Madison−Wisconsin, June 1992.

[39] Barton P. Miller, Morgan Clark, Jeff Hollingsworth, Steven Kierstead, Sek−See Lim, and Timothy Torzewski. Ips−2: The second generation of a parallel program measurment system. *IEEE Transactions on Parallel and Distributed Systems*, 1(2), April 1990.

[40] Barton P. Miller, Jeffrey K. Hollingsworth, and Mark D. Callaghan. The paradyn parallel performance tools and pvm. Technical Report tr1240, University of Madison−Wisconsin, 1994.

[41] R. H. B. Netzer and B. P. Miller. Optimal tracing and replay for debugging message−passing parallel programs. In *Proceedings in Supercomputing '92, Minneapolis*, November 1992.

[42] Barton P.Miller and Jong−Deok Choi. A mechanism for efficient debugging of parallel programs. Technical Report tr754, University of Madison−Wisconsin, 1988.

[43] D. A. Reed, R. A. Aydt, T. M. Madhyastha, R. G. Noe, K. A. Shields, and B. W. Schwartz. An overview of the pablo performance analysis environment. Technical report, University of Illinois, Computer Science Department, 1992.

[44] Steve Sistare, Don Allen, Rich Bowker, Karen Jourdenais, Josh Simons, and Rich Title. A scalable debugger for massively parallel message−passing programs. *IEEE Parallel & Distributed Technology*, Summer 1994.

[45] R. L. Sites and A. Agarwal. Multiprocessor cache analysis using atum. In *Proceedings of the 15th International Conference on Computer Architectures*, 1988.

[46] Amitabh Srivastava and Alan Eustace. Atom: A system for building customized program analysis tools. *Proceedings of the SIGPLAN '94 PLDI*, June 1994.

[47] M. Timmerman, F. Gielen, and P. Lambrix. High level tools for the debugging of real−time multiprocessor systems. *ACM SIGPLAN*, 28(12), December 1993.

[48] J. E. Veenstra and R. J. Flower. Mint tutorial and user manual. Technical Report tr452, The University of Rochester, Computer Science Department, June 1993.

[49] W. Williams, T. Hoel, and D. Pase. The mmp apprentice performance tool: Delivering the performance of the cray t3d. In *Programming Environments for Massivelly Parallel Distributed Systems*, 1994.