# ΑΠΕΙΚΟΝΙΣΗ ΜΗ ΕΜΦΩΛΕΥΜΕΝΩΝ ΒΡΟΧΩΝ ΣΕ ΕΝΔΙΑΜΕΣΗ ΑΝΑΠΑΡΑΣΤΑΣΗ ΜΕΤΑΦΡΑΣΤΗ ΠΟΛΥΝΗΜΑΤΙΚΟΥ ΕΠΕΞΕΡΓΑΣΤΗ ΕΙΔΙΚΟΥ ΣΚΟΠΟΥ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τη

Δέσποινα Ευγενίδου

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Μάρτιος 2008

# ΠΕΡΙΕΧΟΜΕΝΑ

| | Σελ |
|---|---|
| ΠΕΡΙΕΧΟΜΕΝΑ | ii |
| ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ | iv |
| ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ | v |
| ΠΕΡΙΛΗΨΗ | vi |
| EXTENDED ABSTRACT IN ENGLISH | vii |
| ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ | 1 |
| 1.1. Στόχοι | 1 |
| 1.2. Δομή της Διατριβής | 1 |
| ΚΕΦΑΛΑΙΟ 2. ΤΟ ΜΟΝΤΕΛΟ SVP | 3 |
| 2.1. Παρουσίαση του SANE Virtual Processor (SVP) | 3 |
| 2.2. Διαμοιραζόμενη μνήμη | 7 |
| 2.3. Μνήμη συγχρονισμού | 8 |
| 2.4. Οικογένεια νημάτων | 9 |
| 2.5. Ένα νήμα | 9 |
| 2.6. Συγχρονισμός μιας οικογένειας νημάτων | 10 |
| 2.7. Διακοπή της εκτέλεσης μιας οικογένειας νημάτων | 10 |
| 2.8. Squeezing μιας οικογένειας νημάτων | 11 |
| ΚΕΦΑΛΑΙΟ 3. Η ΓΛΩΣΣΑ μTC | 13 |
| 3.1. Η γλώσσα μTC | 13 |
| 3.2. Μνήμη και συγχρονισμός στην μTC | 15 |
| 3.3. Προσθήκες στην C | 17 |
| 3.4. Δημιουργώντας οικογένειες νημάτων (`create`) | 18 |
| 3.5. Ονομασμένα νήματα (`thread`) | 20 |
| 3.6. Squeezable νήματα | 21 |
| 3.7. Συγχρονισμός κατά τον τερματισμό μιας οικογένειας νημάτων (`sync`) | 22 |
| 3.8. Διακόπτοντας οριστικά την εκτέλεση μιας οικογένειας (`break`) | 24 |
| 3.9. Κάνοντας `squeeze` μια οικογένεια | 24 |
| 3.10. "Σκοτώνοντας" μια οικογένεια (`kill`) | 25 |
| 3.11. Τοποθεσίες (`place`) | 26 |
| 3.12. Παραδείγματα χρήσης του μοντέλου μνήμης της μTC | 26 |
| 3.13. Επικοινωνία νημάτων μέσω διαμοιραζόμενων μεταβλητών συγχρονισμού | 28 |
| 3.14. Επικοινωνία νημάτων μέσω της ασύγχρονης διαμοιραζόμενης μνήμης | 34 |
| ΚΕΦΑΛΑΙΟ 4. ΣΥΓΓΕΝΕΙΣ ΕΡΓΑΣΙΕΣ | 36 |
| 4.1. Oscar | 36 |
| 4.2. Polaris | 38 |
| 4.3. Cetus | 39 |
| 4.4. SUIF | 40 |

4.5. Parafrase-2 41
4.6. NANOS 43
4.7. PROMIS 44
4.8. RHODOS 45
4.9. PARADIGM 48
4.10. VFC 49
4.11. CoSy 50
ΚΕΦΑΛΑΙΟ 5. ΣΧΕΔΙΑΣΗ 54
5.1. Σχεδίαση 54
5.2. Χρήσιμοι ορισμοί [21] 55
5.3. Καθορισμός του δείκτη ενός βρόχου 56
5.4. Μετασχηματισμοί απλών βρόχων με καθορισμένο δείκτη ([21], [22]) 58
5.5. Μετασχηματισμός βρόχων που περιέχουν περισσότερες εντολές και καθορισμένο βρόχο 65
5.6. Μετασχηματισμοί απλών βρόχων χωρίς καθορισμένο δείκτη 68
5.7. Αλγόριθμος μετασχηματισμού μονοδιάστατων βρόχων 72
5.8. Κανονικοποίηση βρόχων 75
ΚΕΦΑΛΑΙΟ 6. ΥΛΟΠΟΙΗΣΗ 76
6.1. Υλοποίηση 76
6.2. GCC 76
6.3. Σχεδιασμός εξαρχής 77
6.4. Λεκτική και συντακτική ανάλυση 78
6.5. Ενδιάμεση αναπαράσταση και επιπρόσθετες πληροφορίες 79
6.6. Επεξεργασία ενδιάμεσης αναπαράστασης και πληροφοριών 81
6.7. Παραγωγή του κώδικα σε μTC 83
ΚΕΦΑΛΑΙΟ 7. ΜΕΛΛΟΝΤΙΚΑ ΒΗΜΑΤΑ 86
7.1. Μετασχηματισμός εμφωλευμένων βρόχων 86
7.2. Εξαρτήσεις μεταξύ διαφορετικών πινάκων στο σώμα ενός βρόχου 89
7.3. Ονομασμένα νήματα (named threads) 89
7.4. Εξαγωγή παραλληλισμού από συναρτήσεις 90
ΑΝΑΦΟΡΕΣ 91
ΠΑΡΑΡΤΗΜΑ 94
ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ 96

# ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

Πίνακας Σελ
Πίνακας 3.1 Λέξεις-Κλειδιά της μTC 17
Πίνακας 3.2 Ορίσματα της Εντολής create 18
Πίνακας 3.3 Ορίσματα της Εντολής sync 22
Πίνακας 3.4 Όρισμα της Εντολής break 24
Πίνακας 3.5 Όρισμα της Εντολής squeeze 24
Πίνακας 3.6 Όρισμα της Εντολής kill 25

# ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

Σχήμα Σελ
Σχήμα 3.1 Οι δυο τύποι μνήμης της μTC 15
Σχήμα 3.2 Η Σχέση Μεταξύ των Μονοδιάστατων Μεταβλητών Συγχρονισμού σε μια Οικογένεια Νημάτων 28
Σχήμα 3.3 Μη Γειτονικές Εξαρτήσεις στον Πίνακα a[] Μετατρέπονται σε Γειτονικές με τον Μετασχηματισμό του Προβλήματος στις Δυο Διαστάσεις με την Χρήση Εμφωλευμένων Οικογενειών Νημάτων 32
Σχήμα 4.1 Η Αρχιτεκτονική του Oscar 38
Σχήμα 4.2 Συνιστούντα Μέρη και Διεπιφάνειες του Cetus 40
Σχήμα 4.3 Η Αρχιτεκτονική του Parafrase-2 43
Σχήμα 4.4 Περιγραφή του PROMIS 45
Σχήμα 4.5 Οι 6 Φάσεις του Απλού Μεταγλωττιστή Παραλληλοποίησης. 46
Σχήμα 4.6 Αλγόριθμος Παραλληλοποίησης Βρόχων 47
Σχήμα 4.7 Η Δομή του PARADIGM 49
Σχήμα 4.8 Οι λειτουργίες του CoSy 51
Σχήμα 5.1 Εξάρτηση με Απόσταση 1 59
Σχήμα 5.2 Αντί-Εξάρτηση με Απόσταση x 60
Σχήμα 5.3 Εξάρτηση με Απόσταση x 62
Σχήμα 5.4 Εξάρτηση από τα Προηγούμενα x-1 Στοιχεία του Πίνακα 64
Σχήμα 5.5 Εξάρτηση με Απόσταση 1 και Αντί-Εξάρτηση με απόσταση x 65
Σχήμα 5.6 Εξαρτήσεις και αντί-Εξαρτήσεις Βρόχου με Περισσότερες Εντολές 67
Σχήμα 6.1 Λίστα από λίστες 80
Σχήμα 6.2 Λίστες μεταβλητών 80

# ΠΕΡΙΛΗΨΗ


Δέσποινα Ευγενίδου του Αθανασίου και της Ιωσηφίνας. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Μάρτιος, 2008. Απεικόνιση μη-εμφωλευμένων βρόχων σε ενδιάμεση αναπαράσταση μεταφραστή πολύ-νηματικού επεξεργαστή ειδικού σκοπού. Επιβλέποντας: Γιώργος Μανής.

Σκοπός της εργασίας είναι η μελέτη και η αυτόματη μετατροπή σειριακών μη-εμφωλευμένων βρόχων σε παράλληλους, με τη χρήση μικρό-νημάτων, σύμφωνα με το μοντέλο SVP. Το SVP (SANE Virtual Processor) είναι ένας αφηρημένος παράλληλος επεξεργαστής που υποστηρίζει ένα σύνολο παράλληλων λειτουργιών ενώ το SANE (Self-Adaptive Network Entity) είναι ένα μοντέλο επεξεργαστή που υποστηρίζει την αυτό-προσαρμοστικότητα (self-adaptation) στο υλικό. Βασίζεται στα μικρό-νήματα, τα οποία είναι νήματα που μπορούν να είναι τόσο μικρά όσο μια μόνο αριθμητική πράξη. Μια οικογένεια νημάτων είναι ένα διατεταγμένο σύνολο από συνεργαζόμενα νήματα, όπου κάθε νήμα γνωρίζει μόνο τον δείκτη του στην διάταξη του συνόλου. Η κατάσταση ενός επεξεργαστή SANE κάθε στιγμή ορίζεται από δυο αφηρημένες έννοιες, την ασύγχρονη διαμοιραζόμενη μνήμη (shared memory) και τη μνήμη συγχρονισμού (synchronizing memory). Μέσω της μνήμης συγχρονισμού είναι δυνατός ο συγχρονισμός της εκτέλεσης και η ανταλλαγή δεδομένων ανάμεσα σε γειτονικά μικρονήματα. Για να οριστεί καλύτερα το μοντέλο SVP και για περεταίρω πειραματισμό και διερεύνηση, έχει οριστεί η μTC (microthreaded C), η οποία έχει σχεδιαστεί ως μια ενδιάμεση γλώσσα για την υποστήριξη της παραλληλίας σε κατανεμημένα συστήματα. Υλοποιεί την παραλληλία με δυναμικό τρόπο χρησιμοποιώντας την έννοια της οικογένειας νημάτων. Στην εργασία αυτή κατασκευάστηκε ένα εργαλείο που απεικονίζει μη εμφωλευμένους σειριακούς βρόχους προγραμμάτων C σε οικογένειες νημάτων συμβολισμένες σε γλώσσα μTC με τέτοιο τρόπο ώστε να γίνεται εκμετάλλευση (α) της όποιας παραλληλίας μας επιτρέπουν οι βρόχοι και (β) των δυνατοτήτων που μας παρέχει το μοντέλο SVP.

# EXTENDED ABSTRACT IN ENGLISH

Evgenidou, Despoina. MSc, Computer Science Department, University of Ioannina, Greece. March, 2008. Mapping of non-nested loops on an intermediate language representation for a special purpose multi-threaded processor. Thesis Supervisor: Giorgos Manis.

The purpose of this dissertation is the disquisition of serial, non-nested loops and their automatic transformation to concurrent, using micro-threads in accordance with the SVP model. SVP (SANE Virtual Processor) is an abstract concurrent processor that supports a range of concurrency controls and SANE (Self-Adaptive Network Entity) is a processor model which supports self-adaptation in hardware. In this model, self-adaptation is achieved using one of two different mechanisms: delegation, where the place of the execution is defined remotely, and the pre-emption and termination of a concurrent program. It is based on micro-threads, which are threads that can be as small as a single arithmetic operation. A family of threads is an ordered set of identical threads, where each thread has knowledge of an index value defining the position within that order.

The state of a SANE processor at any given instance in time is defined by two abstractions. The first and the most persistent is an asynchronous shared memory, which comprises a number of addressed locations, each of which may be read and written to by a thread but subject to certain constraints. These constraints are imposed by the weakly-consistent nature of this memory, due to its asynchronous nature and concurrent update by multiple threads. No guarantees can be given about the access time to this memory. The second abstraction is a context of synchronising memory associated with each thread in a family, which provides synchronisation of scalar values between threads in the family and between a thread and any input data from the shared memory.

For further defining the SVP model by experimentation and exploration, the language μTC (microthreaded C) is specified, which is an intermediate language to support concurrency in distributed systems. It implements concurrency in a dynamic manner, using the concept of identified families of threads with pre-emption. μTC is defined by some additions to the C language, that capture the SVP abstraction. It is similar to other concurrent languages based on C. However, no other language that we know implements concurrency in such a dynamic manner.

At the transformation of loops of a high-level language to μTC, there are some issues that have to be considered, such as the identification of the index, the limits and the step of the loop. Some loops have more than one index and some have none. The SVP model supports families of threads with undefined number of threads to be created. μTC provides the means for the creation of a family of threads with "infinite" number of threads, which is terminated by a dynamic condition. There are two rules to find the index of a loop: The index is one of the variables updated in the body or the step of the loop and the condition of the loop depends on this variable. If there is only one variable that follows the rules, then this is the index of the loop.

The μTC code derives from the transformation of the C code automatically. Although it seems that there are different types of loops, according to the kinds of dependencies in their bodies, we can manage all of them in the same way. In this dissertation, a tool has been developed, for the mapping of serial, non-nested loops in C and their transformation to families of threads in μTC, so as to exploit (a) all the concurrency of the loops and (b) all the abilities that the SVP model provides.

# ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

---

1.1 Στόχοι

1.2 Δομή της Διατριβής

---

## 1.1. Στόχοι

Ο συνολικός στόχος της διατριβής αυτής είναι η μελέτη και η αυτόματη μετατροπή σειριακών μονοδιάστατων βρόχων σε παράλληλους, με τη χρήση μικρό-νημάτων, σύμφωνα με το μοντέλο SVP, το οποίο περιγράφεται παρακάτω. Στην εργασία αυτή κατασκευάστηκε ένα εργαλείο που απεικονίζει μη εμφωλευμένους σειριακούς βρόχους προγραμμάτων C σε οικογένειες νημάτων συμβολισμένες σε γλώσσα μTC με τέτοιο τρόπο ώστε να γίνεται εκμετάλλευση (α) της όποιας παραλληλίας μας επιτρέπουν οι βρόχοι και (β) των δυνατοτήτων που μας παρέχει το μοντέλο SVP.

Οι επιμέρους στόχοι της διατριβής είναι:

- Να μελετηθούν οι σειριακοί μη-εμφωλευμένοι βρόχοι της γλώσσας C
- Να καθοριστεί η αντιστοιχία τους με παράλληλους, σύμφωνα με το μοντέλο SVP, στη γλώσσα μTC
- Να σχεδιαστεί ένας αλγόριθμος για την αυτόματη μετατροπή τους
- Να αναπτυχθεί ένα εργαλείο για την απεικόνιση τους από C σε μTC

## 1.2. Δομή της Διατριβής

Η διατριβή περιέχει 7 κεφάλαια: Το Κεφάλαιο 1 είναι η Εισαγωγή και περιλαμβάνει τον σκοπό της διατριβής και τη δομή της. Το Κεφάλαιο 2 περιλαμβάνει την περιγραφή του μοντέλου SVP και των χαρακτηριστικών που το κάνουν να διαφέρει

από άλλα μοντέλα παράλληλου υπολογισμού. Το Κεφάλαιο 3 περιλαμβάνει την παρουσίαση της γλώσσας μTC , η οποία παρέχει όλες τις λειτουργίες για την υποστήριξη του μοντέλου SVP. Το Κεφάλαιο 4 είναι μια παρουσίαση συγγενών εργασιών, όπως το Polaris, το Cetus, το NANOS, το Parafrase-2 και άλλα, καθώς και του εργαλείου κατασκευής μεταγλωττιστών CoSy. Το Κεφάλαιο 5 περιλαμβάνει τη μελέτη των μονοδιάστατων βρόχων, την αντιστοιχία τους με παράλληλους σε γλώσσα μTC και τον σχεδιασμό του αλγορίθμου για την αυτόματη μετατροπή τους από C σε μTC. Το Κεφάλαιο 6 περιέχει την περιγραφή της υλοποίησης του εργαλείου για τη μετατροπή των βρόχων από C σε μTC. Το Κεφάλαιο 7 περιλαμβάνει μελλοντικά βήματα που θα μπορούσαν να γίνουν.

# ΚΕΦΑΛΑΙΟ 2. ΤΟ ΜΟΝΤΕΛΟ SVP

2.1. Παρουσίαση του SANE Virtual Processor (SVP)
2.2. Διαμοιραζόμενη μνήμη
2.3. Μνήμη συγχρονισμού
2.4. Οικογένεια νημάτων
2.5. Ένα νήμα
2.6. Συγχρονισμός μιας οικογένειας νημάτων
2.7. Διακοπή της εκτέλεσης μιας οικογένειας νημάτων
2.8. Squeezing μιας οικογένειας νημάτων

## 2.1. Παρουσίαση του SANE Virtual Processor (SVP)

Το SVP (SANE Virtual Processor) ([1], [2]) είναι ένα μοντέλο παράλληλου υπολογισμού. Το SANE (Self-Adaptive Network Entity) είναι ένα μοντέλο επεξεργαστή που υποστηρίζει την αυτό-προσαρμοστικότητα στο υλικό και το SVP είναι ένας αφηρημένος παράλληλος επεξεργαστής που υποστηρίζει ένα σύνολο παράλληλων λειτουργιών. Για την υποστήριξη της αυτό-προσαρμοστικότητας πρέπει να ισχύουν οι παρακάτω ιδιότητες:

- Συγκέντρωση της παραλληλίας ενός προγράμματος
- Διατήρηση της τοπικότητας της επικοινωνίας
- Διατήρηση της δυναμικότητας

Το SVP ασχολείται με την σύνθεση και την διαχείριση δυναμικά δημιουργημένων παράλληλων προγραμμάτων. Ένας από τους βασικούς του στόχους είναι η μελέτη της αυτό-προσαρμοστικότητας σε σύνθετα υπολογιστικά συστήματα και η ύπαρξη ενός

μοντέλου που μπορεί να συντίθεται παράλληλα και με ασφάλεια. Αυτό σημαίνει ότι τα προγράμματα πρέπει να μην περιέχουν αδιέξοδα στον σχεδιασμό τους και να επιτρέπουν ασφαλή σύνθεση, ώστε δυο ή περισσότερα SVP προγράμματα να μπορούν να συνθέσουν ένα τρίτο SVP πρόγραμμα, το οποίο να συμπεριφέρεται καλά, δηλαδή να μην έχει αδιέξοδα και να είναι ντετερμινιστικό αν τα δυο προγράμματα έχουν συντεθεί ντετερμινιστικά. Ντετερμινισμός σημαίνει το πρόγραμμα να δίνει πάντα το ίδιο αποτέλεσμα και είναι μια από τις βασικές ιδιότητες των σειριακών προγραμμάτων.

Ένας υπολογισμός στο SVP θεωρείται ως ένα πακέτο πληροφορίας, το οποίο καθορίζει ένα *μέρος* (place), είτε τοπικά, είτε απομακρυσμένα, όπου το παράλληλο πρόγραμμα θα εκτελεστεί. Η επικοινωνία, όπου χρειάζεται, εμπεριέχει δεδομένα και κάποιον ορισμό της λειτουργικότητας. Αυτό το πακέτο ορίζει μια μονάδα εργασίας, η οποία είναι μια κλήση από ένα νήμα σε όλα τα υφιστάμενα του νήματα, στο μέρος που έχει επιλεγεί για την εκτέλεση του.

Για την επίτευξη της αυτό-προσαρμοστικότητας στο μοντέλο αυτό χρησιμοποιούνται δυο διαφορετικοί μηχανισμοί. Ο πρώτος μηχανισμός είναι η *ανάθεση* (delegation), όπου το μέρος ορίζεται απομακρυσμένα και είναι ανάλογο με την απομακρυσμένη κλήση διαδικασίας (RPC). Η μονάδα εργασίας αποστέλλεται στο νέο μέρος για εκτέλεση. Πρέπει να αναφερθεί ότι λέγοντας μέρος εννοούμε μια αφηρημένη έννοια, και αυτό μπορεί να είναι ένας επεξεργαστής ή ένα σύνολο επεξεργαστών, άλλα και μια υπηρεσία που τρέχει στο μέρος αυτό. Έτσι η έννοια *μέρος* αναπαριστά την ένωση των παρακάτω εννοιών:

- Μια διεύθυνση για επικοινωνία, ορίζοντας το φυσικό μέρος
- Η δυνατότητα να εκτελείται μια μονάδα εργασίας σε αυτό το μέρος
- Μια υπηρεσία σε αυτό το μέρος, η οποία μπορεί να παρέχει αμοιβαίο αποκλεισμό αν αυτός είναι απαραίτητος (π.χ. για την ανάθεση των πόρων)

Ο δεύτερος μηχανισμός είναι η έννοια της *διακοπής της εκτέλεσης* (preemption) και του τερματισμού ενός παράλληλου προγράμματος. Αυτό επιτρέπει στο μέρος όπου

εκτελείται ένα πρόγραμμα να ανακληθεί, και στην περίπτωση της διακοπής, επιτρέπεται η συνέχιση της εκτέλεσης σε ένα άλλο μέρος.

Στο SVP, μια μονάδα εργασίας αναπαρίσταται με μια παραμετροποιημένη οικογένεια νημάτων, η οποία έχει συλλογικές ιδιότητες που ορίζουν τους υπολογιστικούς της στόχους. Αυτοί οι στόχοι ενεργοποιούν την προσαρμοστικότητα σε σχέση με το περιβάλλον της εκτέλεσης. Επειδή απαιτείται από τα νήματα να περιλαμβάνουν μερικά χαρακτηριστικά του υλικού, σε αυτό το μοντέλο το νήμα μπλοκάρεται, δηλαδή θα προχωρήσει στη εκτέλεση μόνο αν υπάρχουν δεδομένα να καταναλωθούν.

Το μοντέλο αυτό βασίζεται στα μικρό-νήματα, τα οποία είναι νήματα που μπορούν να είναι τόσο μικρά όσο μια μόνο αριθμητική πράξη. Οι οικογένειες νημάτων δημιουργούνται δυναμικά, τους ανατίθενται κάποιοι υπολογιστικοί πόροι, κάνουν κάποια δουλειά και επιστρέφουν ένα αποτέλεσμα. Αυτή η ομαδοποίηση των νημάτων, σε παραμετροποιημένες οικογένειες, διαχωρίζει αυτό το μοντέλο από τα άλλα μοντέλα υπολογισμού που βασίζονται σε νήματα και ορίζει και ενεργοποιεί τους μηχανισμούς για αυτό-προσαρμοστικότητα.

Η λειτουργικότητα μιας οικογένειας νημάτων ορίζεται από ένα διατεταγμένο σύνολο πανομοιότυπων ταυτόχρονων νημάτων, όπου κάθε νήμα έχει μια τιμή-δείκτη της θέσης του στην διάταξη αυτή. Χρησιμοποιώντας αυτή την τιμή μπορούμε να ορίσουμε μονάδες εργασίας με δυναμικό εύρος και ομογενείς, όπως και με στατικό εύρος και ετερογενείς (δηλαδή με επιλογή διακλάδωσης πάνω στον δείκτη του νήματος), αλλά και συνδυασμό τους.

Η δημιουργία ενός νήματος ορίζει τα δεδομένα εισόδου και εξόδου της οικογένειας και ο τερματισμός της οικογένειας ορίζει ένα σημείο συγχρονισμού σύμφωνα με τις εξόδους της οικογένειας. Μέσα σε μια οικογένεια, τα νήματα μπορούν να επικοινωνούν και να συγχρονίζονται, χρησιμοποιώντας ένα είδος μονοδιάστατων δεδομένων, επιτρέποντας την ενσωμάτωση της κανονικότητας και της τοπικότητας. Υπάρχει μια αντιστοιχία των οικογενειών νημάτων με το σειριακό μοντέλο υπολογισμών. Οι οικογένειες νημάτων είναι ανάλογες με τους βρόχους και τις κλήσεις συναρτήσεων. Το μοντέλο αυτό επιτρέπει ιεραρχία, δηλαδή επιτρέπεται τα

νήματα να δημιουργούν άλλες οικογένειες νημάτων και να παρέχουν ελέγχους της παραλληλίας ορίζοντας την διακοπή ή τον τερματισμό των οικογενειών.

Τα βήματα για τη δημιουργία μια οικογένειας νημάτων στην γενική περίπτωση είναι τα παρακάτω:

1. Το νήμα ζητάει ένα μέρος (place) από τον διαχειριστή των πόρων. Αυτό το βήμα είναι προαιρετικό, αφού μπορούν να χρησιμοποιηθούν υπάρχοντες πόροι.
2. Αν είναι απαραίτητο, το νήμα αντιγράφει τις εισόδους της οικογένειας στην μνήμη των πόρων που διατέθηκαν αν είναι κατανεμημένη ή ελέγχει την συνέπεια των δεδομένων αν είναι διαμοιραζόμενη.
3. Το νήμα ορίζει την οικογένεια, δηλαδή τον κώδικα, τις παραμέτρους, τα δεδομένα και τον σκοπό της. Συνεχίζει μέχρι τον συγχρονισμό των αποτελεσμάτων της οικογένειας που δημιουργήθηκε.
4. Συγχρόνως, το μέρος (place) δημιουργεί το διατεταγμένο σύνολο νημάτων, υπό τους περιορισμούς που ορίζουν οι πόροι, όπου κάθε νήμα έχει ένα κομμάτι μνήμης συγχρονισμού όπου κάθε νήμα μπορεί να επικοινωνεί και να συγχρονίζει με το γειτονικό του νήμα σε σειρά δημιουργίας.
5. Τέλος, όταν μια οικογένεια ολοκληρώσει την εκτέλεση της αντιγράφει τα αποτελέσματα πίσω στην μνήμη του νήματος που τη δημιούργησε αν είναι κατανεμημένη ή ελέγχει την συνέπεια της αν είναι διαμοιραζόμενη. Τότε ενημερώνει το νήμα που δημιούργησε την μονάδα εργασίας για την ολοκλήρωση της.
6. Το νήμα που δημιούργησε την οικογένεια μπορεί τώρα να χρησιμοποιήσει τα αποτελέσματα και να αποδεσμεύσει τους επιπλέον πόρους που μπορεί να δεσμεύτηκαν.

Όλες οι λειτουργίες πρέπει να συγχρονιστούν σύμφωνα με τους τελεστές τους και ένα νήμα διακόπτει την λειτουργία του μέχρι όλοι οι τελεστές μιας λειτουργίας του να είναι διαθέσιμοι. Αυτό είναι σημαντικό για το μοντέλο παράλληλης εκτέλεσης γιατί οι τελεστές μπορεί να προέρχονται από κάποιο άλλο νήμα ή τα δεδομένα να είναι κατανεμημένα. Αν τα δεδομένα δεν είναι διαθέσιμα, το νήμα που εκτελεί την

λειτουργία θα μπλοκαριστεί και καμιά επόμενη λειτουργία δε θα προχωρήσει μέχρι όλες οι προηγούμενες λειτουργίες να πάρουν τα δεδομένα που χρειάζονται και να ολοκληρωθούν.

Η κατάσταση ενός επεξεργαστή SANE κάθε στιγμή ορίζεται από δυο αφηρημένες έννοιες. Η πρώτη είναι μια *ασύγχρονη διαμοιραζόμενη μνήμη* (shared memory), η οποία αποτελείται από έναν αριθμό διευθυνσιοδοτημένων θέσεων, καθεμιά από τις οποίες μπορεί να διαβαστεί και να γραφτεί από ένα νήμα, υπό συγκεκριμένους περιορισμούς. Αυτοί οι περιορισμοί επιβάλλονται από την έλλειψη συνέπειας της μνήμης, λόγω της ασύγχρονης φύσης της και της ταυτόχρονης ενημέρωσης της από πολλαπλά νήματα. Δεν μπορούν να υπάρξουν εγγυήσεις για τη στιγμή που θα γίνει η πρόσβαση σε αυτή την μνήμη. Η δεύτερη έννοια είναι η *μνήμη συγχρονισμού* (synchronizing memory), που σχετίζεται με κάθε νήμα σε μια οικογένεια, η οποία παρέχει συγχρονισμό μονοδιάστατων τιμών μεταξύ των νημάτων σε μια οικογένεια και μεταξύ ενός νήματος και των δεδομένων εισόδου που προέρχονται από την διαμοιραζόμενη μνήμη.

## 2.2. Διαμοιραζόμενη μνήμη

Όπως περιγράφτηκε προηγουμένως, το μοντέλο αυτό βασίζεται σε μονάδες εργασίας, οι οποίες αποτελούν οικογένειες νημάτων. Μια οικογένεια έχει εισόδους και εξόδους που ορίζονται από το σώμα του νήματος που την δημιουργεί. Αυτές οι είσοδοι και οι έξοδοι είναι οι τοποθεσίες της διαμοιραζόμενης μνήμης όπου γράφει και διαβάζει το νήμα. Αυτή η κατάσταση είναι τελείως ορισμένη στον τερματισμό μιας μονάδας εργασίας. Κάθε άλλη στιγμή, επειδή πρόκειται για ένα ασύγχρονο σύστημα, θα υπάρχει ένα υποσύνολο αυτών των τοποθεσιών που η κατάσταση τους δεν θα είναι καθορισμένη. Η συνέπεια της μνήμης επιβάλλεται με μαζικό συγχρονισμό στην διαμοιραζόμενη μνήμη κατά τον τερματισμό μιας οικογένειας νημάτων. Οι εγγραφόμενες τιμές από τα νήματα μιας οικογένειας είναι εγγυημένα σωστές εκείνη τη στιγμή. Έτσι οι έξοδοι μιας οικογένειας είναι ορισμένες μόνο μετά τον τερματισμό όλων των νημάτων.

**2.3. Μνήμη συγχρονισμού**

Η μνήμη συγχρονισμού παρέχει τον μηχανισμό με τον οποίο τα νήματα μέσα σε μια οικογένεια μπορούν να συγχρονιστούν μεταξύ τους και με το νήμα που τα δημιούργησε. Ακόμα παρέχει έναν μηχανισμό για τον συγχρονισμό της διαμοιραζόμενης μνήμης με τους πράκτορες επεξεργασίας, αφού δεν μπορούμε να κάνουμε καμιά υπόθεση για τους χρόνους πρόσβασης στην διαμοιραζόμενη μνήμη.

Η μνήμη συγχρονισμού είναι δυναμική και παροδική. Περιέχει ένα είδος τοπικών μονοδιάστατων μεταβλητών, οι οποίες χάνονται μόλις ένα νήμα ολοκληρώσει την εκτέλεση του. Η μνήμη αυτή είναι γρήγορη, κατανεμημένη και κοντά στον επεξεργαστή. Η δημιουργία των νημάτων γίνεται με τη σειρά των δεικτών τους και το μέγεθος της μνήμης συγχρονισμού καθορίζει το εύρος της παραλληλίας σε αυτό το μέρος (place).

Όλες οι αριθμητικές και λογικές πράξεις του SVP γίνονται μεταξύ τιμών αποθηκευμένων στην μνήμη συγχρονισμού και οι τιμές που βρίσκονται στη διαμοιραζόμενη μνήμη μεταφέρονται στην μνήμη συγχρονισμού πριν από την εκτέλεση των πράξεων. Κάθε τοποθεσία παρέχει μια εντολή ανάγνωσης που μπλοκάρει ή συγχρονισμό στη ροή των δεδομένων. Οι εξαρτήσεις μεταξύ λειτουργιών σε διαφορετικά νήματα επιβάλλονται από την μνήμη συγχρονισμού, καθώς υπάρχει συγχρονισμός στα δεδομένα από τη διαμοιραζόμενη μνήμη. Υπάρχει ένας περιορισμός που επιβάλλεται στις εξαρτήσεις μεταξύ των νημάτων μιας οικογένειας, ο οποίος προέρχεται από τον συνδυασμό της τοπικότητας και της απουσίας αδιεξόδων. Ένα νήμα μπορεί να εξαρτάται μόνο από τιμές που έχουν παραχθεί από τον προκάτοχο του στον χώρο δεικτών της οικογένειας. Αυτό σιγουρεύει ότι οι εξαρτήσεις μεταξύ των λειτουργιών σε μια οικογένεια νημάτων μπορεί να αναπαρασταθεί από έναν άκυκλο γράφο, ο οποίος με τη σειρά του σιγουρεύει την απουσία αδιεξόδων επικοινωνίας στο μοντέλο.

Αυτοί οι άκυκλοι γράφοι εξαρτήσεων δημιουργούνται και συγχρονίζονται με τις μονοδιάστατες μεταβλητές στο πλαίσιο του νήματος που δημιουργεί την οικογένεια, και οι εξαρτώμενες τιμές που δημιουργήθηκαν από το τελευταίο νήμα είναι διαθέσιμες στο νήμα που δημιουργεί την οικογένεια κατά τον τερματισμό της, μέσω

της ίδιας τοποθεσίας από όπου αρχικοποιήθηκε ο γράφος εξαρτήσεων. Δεν επιτρέπεται ο συγχρονισμός μεταξύ νημάτων σε δυο οικογένειες εκτός από την αρχικοποίηση μιας αλυσίδας εξάρτησης, δηλαδή μεταξύ ενός νήματος και της οικογένειας που αυτό δημιούργησε.

## 2.4. Οικογένεια νημάτων

Μια οικογένεια νημάτων είναι ένα διατεταγμένο σύνολο από ολόιδια νήματα, όπου κάθε νήμα γνωρίζει μόνο τον δείκτη του στην διάταξη του συνόλου. Αυτή η γνώση επιτρέπει στις ομογενείς και τις ετερογενείς οικογένειες να ορίζονται από το ίδιο νήμα, αφού ο δείκτης μπορεί να χρησιμοποιηθεί για τον έλεγχο ενός νήματος. Οι δείκτες ενός διατεταγμένου συνόλου είναι μια αριθμητική ακολουθία που ορίζεται από τον δείκτη αρχής, μια σταθερή διαφορά μεταξύ των διαδοχικών δεικτών και προαιρετικά μια μέγιστη τιμή δείκτη.

Μια οικογένεια μπορεί να θεωρηθεί ως μια εργασία που εκτελείται σε ένα μέρος και διαβάζει και γράφει στην διαμοιραζόμενη μνήμη. Οι επεξεργαστές, που αντιστοιχίζονται στο *μέρος* (place) όπου μια οικογένεια εκτελείται, αναγνωρίζονται από ένα γνώρισμα που αντιστοιχίζεται στην οικογένεια κατά τον ορισμό της. Όταν μια οικογένεια νημάτων δημιουργείται, μια μοναδική τιμή αντιστοιχίζεται σε αυτή την οικογένεια για την αναγνώριση της και την διαχείριση της. Αυτή η τιμή δίνει την δυνατότητα να χρησιμοποιείται για να πιστοποιήσει κάθε ενέργεια που εκτελείται από κάποιο νήμα στην οικογένεια νημάτων, έτσι η ασφάλεια εφαρμόζεται στο χαμηλότερο επίπεδο του συστήματος. Οι ενέργειες που επιτρέπονται σε μια οικογένεια νημάτων από την δυνατότητα αυτή είναι ο αναγκαστικός τερματισμός και ένα είδος διακοπής της εκτέλεσης που ονομάζεται *squeezing* της οικογένειας.

## 2.5. Ένα νήμα

Ένα νήμα είναι μια σειρά από λειτουργίες, συμπεριλαμβανομένων της ανάγνωσης και της εγγραφής στην διαμοιραζόμενη μνήμη, οι οποίες ορίζονται σε ένα σύνολο από τοπικές μονοδιάστατες μεταβλητές, οι οποίες βρίσκονται στην μνήμη συγχρονισμού και είναι διαθέσιμες όταν δημιουργηθεί το νήμα και καταστρέφονται όταν αυτό

ολοκληρώσει την εκτέλεση του. Οι εξαρτήσεις μεταξύ νημάτων της ίδιας οικογένειας πρέπει να οριστούν σε αυτό το πλαίσιο. Ο μοναδικός δείκτης που διαχωρίζει κάθε νήμα μπορεί να χρησιμοποιηθεί για τον ορισμό της λειτουργικότητας του. Ένα νήμα εκτελεί τις λειτουργίες του μόνο όταν οι τιμές που απαιτεί έχουν οριστεί στο πλαίσιο της μνήμης συγχρονισμού. Ένα νήμα μπορεί να δημιουργήσει υφιστάμενες οικογένειες νημάτων, αλλά δεν μπορεί να τερματίσει χωρίς πρώτα να γίνει ο συγχρονισμός και ο τερματισμός της υφιστάμενης οικογένειας.

### 2.6. Συγχρονισμός μιας οικογένειας νημάτων

Ο συγχρονισμός μιας οικογένειας νημάτων γίνεται όταν όλα τα νήματα της οικογένειας έχουν τερματίσει. Τα νήματα τερματίζουν είτε εξαντλώντας τις λειτουργίες τους, είτε μέσω της εκτέλεσης ενός αριθμού σημάτων που έχουν σταλεί στην οικογένεια τους. Κατά τον συγχρονισμό μιας οικογένειας, οι αλλαγές που έχουν γίνει στην διαμοιραζόμενη μνήμη οριστικοποιούνται. Κατά τον συγχρονισμό επιστρέφεται ένας κωδικός και μια τιμή στο νήμα που δημιούργησε την οικογένεια. Ο κωδικός που επιστρέφεται καθορίζει ποιο σήμα, αν υπήρχε, προκάλεσε τον τερματισμό και η τιμή που επιστρέφεται ορίζεται όταν έχει διακοπεί η εκτέλεση μιας οικογένειας ή όταν έχει γίνει squeezing μιας οικογένειας.

### 2.7. Διακοπή της εκτέλεσης μιας οικογένειας νημάτων

Κάθε νήμα μπορεί να εκτελέσει ένα σήμα διακοπής της οικογένειας του. Το αποτέλεσμα για μια οικογένεια που λαμβάνει ένα τέτοιο σήμα είναι η διακοπή της δημιουργίας νέων νημάτων, τερματισμός όλων των ενεργών νημάτων και διακοπή του συγχρονισμού για την οικογένεια. Μια τιμή επιστροφής ορίζεται από το νήμα που εκτελεί το σήμα διακοπής και ο κωδικός επιστροφής καθορίζει ότι η οικογένεια τερματίστηκε από ένα σήμα διακοπής. Αυτή η έννοια επιτρέπει την δημιουργία ενός «άπειρου» αριθμού νημάτων σε μια οικογένεια, η οποία τερματίζεται από μια δυναμική συνθήκη. Αυτή η περίπτωση είναι ανάλογη με τους βρόχους που τερματίζονται δυναμικά στο σειριακό μοντέλο, όπως οι βρόχοι «while».

**2.8. Squeezing μιας οικογένειας νημάτων**

Κάθε νήμα που μπορεί να ορίσει μια οικογένεια και το μέρος όπου εκτελείται και μπορεί να γνωρίζει την ταυτότητα που πήρε κατά την δημιουργία της, μπορεί να στείλει ένα σήμα squeeze σε αυτή την οικογένεια και εξαναγκάσει τον τερματισμό της, ενώ διατηρείται η κατάσταση της οικογένειας. Αυτή είναι μια μορφή διακοπής της μονάδας εργασίας που αναπαριστά η οικογένεια και επιτρέπει στην οικογένεια την επανεκκίνηση, με την επαναδημιουργία της, χρησιμοποιώντας την κατάσταση που διατηρήθηκε κατά το squeeze. Το αποτέλεσμα για μια οικογένεια που λαμβάνει ένα τέτοιο σήμα είναι η διακοπή της δημιουργίας νέων νημάτων, φυσιολογική ολοκλήρωση όλων των ενεργών νημάτων και διακοπή του συγχρονισμού για την οικογένεια. Επιπλέον, η οικογένεια θα στείλει ένα σήμα squeeze σε κάθε οικογένεια που δημιουργήθηκε από τα νήματα της, αλλά μόνο εφόσον το νήμα έχει τον χαρακτηρισμό «squeezable».

Η τιμή που επιστρέφεται εξαρτάται από το αν ένα νήμα είναι squeezable, αν δηλαδή μπορεί να περάσει το σήμα squeeze στα υφιστάμενα νήματα. Αν το νήμα είναι squeezable, τότε η τιμή αυτή ορίζεται ως ο δείκτης του πρώτου νήματος στην σειρά που δημιούργησε έναν χώρο στην μνήμη συγχρονισμού αλλά δεν έχει ακόμα ολοκληρωθεί φυσιολογικά. Αυτός ο δείκτης χωρίζει την οικογένεια σε νήματα που ήδη ολοκληρώθηκαν και σε νήματα που πρέπει να εκτελεστούν ξανά και κάποια που μπορεί να ολοκληρωθούν και στα οποία δίνεται ο χρόνος να ολοκληρωθούν στέλνοντας σήματα squeeze στις υφιστάμενες οικογένειες τους. Με την διάδοση του δείκτη των υφιστάμενων οικογενειών το πρόγραμμα παραμένει ντετερμινιστικό.

Αν μια οικογένεια δεν είναι squeezable, δηλαδή δεν μπορεί να μεταδοθεί ένα σήμα squeeze σε μια υφιστάμενη οικογένεια, τότε όλα τα νήματα της τελευταίας κατηγορίας θα τερματίσουν φυσιολογικά τελικά και η τιμή επιστροφής μπορεί να περιέχει τον δείκτη του πρώτου νήματος που δεν έχει ξεκινήσει να εκτελείται.

Το squeezing μιας οικογένειας νημάτων διακόπτει την εκτέλεση των οικογενειών που ορίστηκαν από αυτή και παρέχει τη δυνατότητα να ξαναρχίσουν την εκτέλεση τους σε ένα διαφορετικό μέρος. Χρησιμοποιώντας την έννοια των squeezable νημάτων παρέχεται μια αυστηρή προσέγγιση στην διακοπή της εκτέλεσης ενός παράλληλου

προγράμματος στο μοντέλο SANE. Αυτή η έννοια είναι βασική για το μοντέλο και παρέχει ένα από τα σημαντικότερα χαρακτηριστικά για να υλοποιηθεί η αυτό-προσαρμοστικότητα.

# ΚΕΦΑΛΑΙΟ 3. Η ΓΛΩΣΣΑ ΜΤC

3.1. Η γλώσσα μTC
3.2. Μνήμη και συγχρονισμός στην μTC
3.3. Προσθήκες στην C
3.4. Δημιουργώντας οικογένειες νημάτων (`create`)
3.5. Ονομασμένα νήματα (`thread`)
3.6. Squeezable νήματα
3.7. Συγχρονισμός κατά τον τερματισμό μιας οικογένειας νημάτων (`sync`)
3.8. Διακόπτοντας οριστικά την εκτέλεση μιας οικογένειας (`break`)
3.9. Κάνοντας `squeeze` μια οικογένεια
3.10. "Σκοτώνοντας" μια οικογένεια (`kill`)
3.11. Τοποθεσίες (`place`)
3.12. Παραδείγματα χρήσης του μοντέλου μνήμης της μTC
3.13. Επικοινωνία νημάτων μέσω διαμοιραζόμενων μεταβλητών συγχρονισμού
3.14. Επικοινωνία νημάτων μέσω της ασύγχρονης διαμοιραζόμενης μνήμης

## 3.1. Η γλώσσα μTC

Για να οριστεί καλύτερα το μοντέλο SVP και για περεταίρω πειραματισμό και διερεύνηση, έχει οριστεί η μTC (microthreaded C) ([1], [2], [3], [4]), η οποία έχει σχεδιαστεί ως μια ενδιάμεση γλώσσα για την υποστήριξη της παραλληλίας σε κατανεμημένα συστήματα. Είναι παρόμοια με άλλες γλώσσες παράλληλου προγραμματισμού που είναι βασισμένες στην C, όμως υλοποιεί την παραλληλία με δυναμικό τρόπο χρησιμοποιώντας τις έννοιες της οικογένειας νημάτων και της διακοπής της εκτέλεσης με την δυνατότητα μεταφοράς της εργασίας σε άλλο μέρος

(σε άλλον επεξεργαστή) και συνέχιση της εκτέλεσης εκεί από το σημείο όπου διακόπηκε.

Το πλεονέκτημα της μTC σε σχέση με άλλες προσεγγίσεις είναι το ότι επιτρέπει μια αφηρημένη περιγραφή της μέγιστης παραλληλίας σε μορφή ανεξάρτητη χρονοδιαγράμματος. Η μTC είναι μια ουσιαστική αλλά απλή επέκταση της C, που επιτρέπει την περιγραφή της παραλληλίας που βασίζεται σε νήματα. Έχει την δυνατότητα να εκφράζει στατική, ομογενή παραλληλία και δυναμική, ετερογενή παραλληλία.

Έχει κάποια κοινά σημεία με το OpenMP, αλλά έχει σημαντικές διαφορές. Μια από τις σημαντικότερες διαφορές είναι ότι η μTC υποθέτει την ύπαρξη μνήμης συγχρονισμού. Αυτό περιγράφει τις εξαρτήσεις μεταξύ νημάτων, επιτρέποντας την μετατροπή σειριακού κώδικα σε παράλληλο, όπου οι εξαρτήσεις ελέγχονται με διαφάνεια μέσω των δεδομένων. Μια άλλη διαφορά είναι ότι η C επεκτείνεται με εκτελούμενες εντολές, παρά με επισημάνσεις όπως στο OpenMP. Αυτή η διαφορά είναι ριζική και παρέχει έναν μηχανισμό για δυναμικό χειρισμό των οικογενειών νημάτων, όπως απαιτείται από τα αυτό-προσαρμοστικά συστήματα.

Η μTC βασίζεται στην έννοια των Μίκρο-νημάτων, η οποία περιέχει την μνήμη συγχρονισμού και το αποδοτικό, χαμηλού επιπέδου χρονοδιάγραμμα. Τα προβλήματα που προκύπτουν από τον προγραμματισμό για παράλληλα συστήματα είναι:

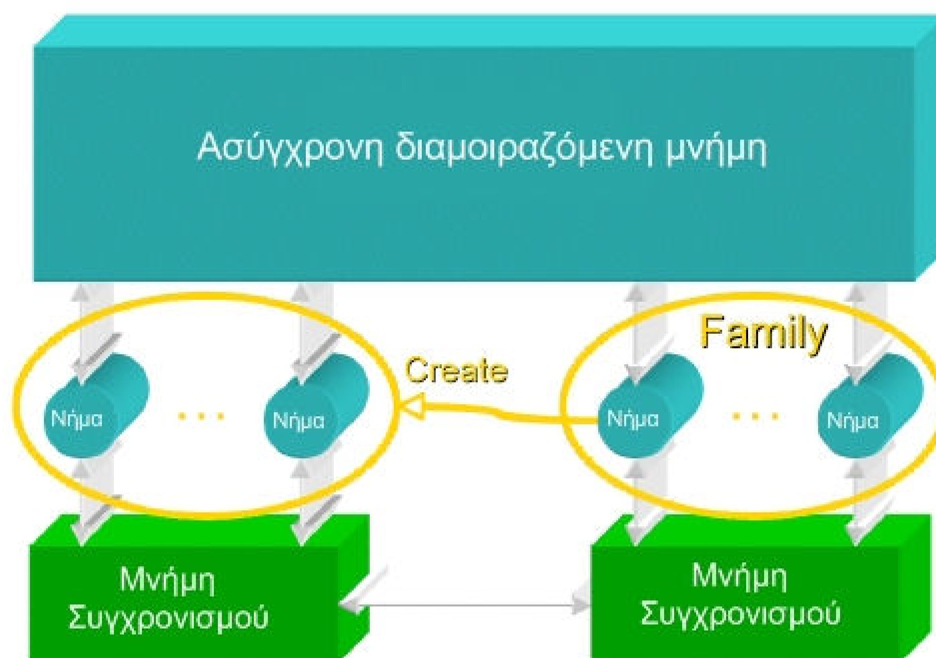
- Ο χρήστης καλείται να παραλληλοποιήσει υπάρχον σειριακό κώδικα
- Προγράμματα με νήματα που χρησιμοποιούν εξωτερικές βιβλιοθήκες δεν είναι φορητά
- Η συγγραφή πολύ-νηματικών προγραμμάτων απαιτεί ιδιαίτερη γνώση της αρχιτεκτονικής του συστήματος

Η μTC έχει λύσεις για αυτά τα προβλήματα. Οι χρήστες δεν παραλληλοποιούν υπάρχουσες εφαρμογές, αλλά δημιουργούν μTC από ντετερμινιστικό κώδικα, όπως κώδικας σε C ή σε SAC (Single-Assignment C). Η παραλληλία στην μTC επιτυγχάνεται με έναν αυθαίρετο τρόπο που δεν απαιτεί αναφορά σε εξωτερικές

βιβλιοθήκες, έτσι μόνο ο μεταγλωττιστής χρειάζεται να έχει γνώση της αρχιτεκτονικής του συστήματος.

### 3.2. Μνήμη και συγχρονισμός στην μTC

Η μTC υποστηρίζει δυο τύπους μνήμης, ανάλογους με τους καταχωρητές και την κυρίως μνήμη στο σειριακό μοντέλο. Αυτοί οι τύποι ονομάζονται *μνήμη συγχρονισμού* και *διαμοιραζόμενη μνήμη*. Στο Σχήμα 3.1 βλέπουμε τα δυο είδη μνήμης και την σχέση τους με τις οικογένειες νημάτων.




Σχήμα 3.1 Οι δυο τύποι μνήμης της μTC

Η διαμοιραζόμενη μνήμη υποστηρίζει συγχρονισμό μεταξύ των οικογενειών νημάτων. Η πρόσβαση γίνεται μέσω ενός κοινού χώρου διευθύνσεων, αν και πρακτικά δεν είναι ακριβώς έτσι. Δεν γίνονται υποθέσεις για την υλοποίηση και την ταχύτητα της μνήμης, και ο μόνος συγχρονισμός που ορίζεται είναι κατά τον τερματισμό μιας οικογένειας νημάτων, όπου όλες οι τοποθεσίες στην μνήμη που ενημερώνονται από την οικογένεια πρέπει να είναι καλά ορισμένες. Αν και μπορούμε να γράψουμε μη ντετερμινιστικά προγράμματα σε μTC, τα οποία να ορίζουν το

περιεχόμενο μιας τοποθεσίας της διαμοιραζόμενης μνήμης σαν ένα σύνολο τιμών, υπάρχουν μερικοί απλοί κανόνες για την συγγραφή ντετερμινιστικών προγραμμάτων. Για παράδειγμα, δυο διαφορετικά νήματα μέσα στην ίδια οικογένεια, ή σε δυο οικογένειες που εκτελούνται ταυτόχρονα, δεν μπορούν να γράψουν ή να διαβάσουν την ίδια τοποθεσία της διαμοιραζόμενης μνήμης, αλλά κάθε νήμα μπορεί να γράψει σε μια θέση μνήμης, η οποία θα διαβαστεί από το γειτονικό του νήμα.

Η μνήμη συγχρονισμού υποστηρίζει την επικοινωνία και τον συγχρονισμό μεταξύ των νημάτων σε μια οικογένεια και μεταξύ του επεξεργαστή που εκτελεί ένα νήμα και της διαμοιραζόμενης μνήμης. Υπάρχουν περιορισμοί στην επικοινωνία, ώστε να υπάρχει τοπικότητα και να διασφαλίζεται η μη ύπαρξη αδιεξόδων σε αυτήν. Οι τύποι ενδονηματικής επικοινωνίας που υποστηρίζονται είναι:

*Καθολικές για ένα νήμα*: ένα υποσύνολο του μονοδιάστατου πλαισίου συγχρονισμού του νήματος που δημιουργεί μια οικογένεια, διαθέσιμο σε μορφή μόνο για ανάγνωση σε κάθε νήμα της οικογένειας ως μέρος του μονοδιάστατου πλαισίου συγχρονισμού.

*Αλυσίδες εξάρτησης μεταξύ των νημάτων μιας οικογένειας*: η αλυσίδα εξάρτησης έχει μια μονοδιάστατη μεταβλητή, ορισμένη ως `shared` στο πλαίσιο κάθε νήματος, η οποία μπορεί να εγγραφεί από αυτό το νήμα και είναι διαθέσιμη μόνο για ανάγνωση στο επόμενο νήμα. Μπορούν να οριστούν πολλές αλυσίδες εξάρτησης χρησιμοποιώντας πολλές `shared` μεταβλητές σε ένα νήμα. Μια αλυσίδα εξάρτησης αρχικοποιείται και συγχρονίζεται με μια μονοδιάστατη μεταβλητή που γράφεται από το νήμα που δημιουργεί την οικογένεια νημάτων. Αυτός ο συγχρονισμός είναι ανεξάρτητος από το γεγονός της δημιουργίας της οικογένειας και μπορεί να συμβεί πριν ή μετά από τη δημιουργία. Η ίδια μεταβλητή θα ενημερωθεί με την έξοδο της αλυσίδας εξάρτησης, χωρίς συγχρονισμό. Η μεταβλητή που αρχικοποιεί την αλυσίδα ενημερώνεται με την τιμή που γράφεται από το τελευταίο νήμα, κατά την πραγματοποίηση του `sync` για την οικογένεια, είτε κατά τον φυσιολογικό τερματισμό της είτε μετά από `squeeze`.

Η μνήμη συγχρονισμού υλοποιεί την ανάγνωση που μπλοκάρει ή έναν συγχρονισμό στην ροή δεδομένων και αν ένα νήμα έχει οριστεί ως μια ακολουθία λειτουργιών, η εκτέλεση του νήματος δε θα προχωρήσει πέρα από τη λειτουργία που προσπαθεί να διαβάσει έναν τελεστή από την μνήμη συγχρονισμού, ο οποίος δεν έχει ακόμα οριστεί.

### 3.3. Προσθήκες στην C

Οι προσθήκες στην C που ορίζουν την μTC και περιλαμβάνουν τις αφαιρέσεις του μοντέλου SVP περιγράφονται εδώ. Ο βασικός τύπος που υποστηρίζει την δυνατότητα της δημιουργίας και διαχείρισης μιας οικογένειας νημάτων είναι ο τύπος `family`. Οι δομές που περιγράφονται παρακάτω επιτρέπουν τη δημιουργία και τον τερματισμό οικογενειών νημάτων με προσδιορισμένη ταυτότητα, οι οποίες μπορεί να ορίζονται ιεραρχικά. Η μTC προσθέτει τις λέξεις-κλειδιά του πίνακα 3.1 στην C:

Πίνακας 3.1 Λέξεις-Κλειδιά της μTC

| | |
|---|---|
| create | Δομή που χρησιμοποιείται για την δημιουργία μιας οικογένειας παράλληλων Μίκρο-νημάτων |
| thread | Τύπος για τον ορισμό μιας συνάρτησης ως νήμα |
| squeezable | Τύπος συναρτήσεων που ορίζει νήματα που μπορούν να προωθήσουν ένα σήμα squeeze στις υφιστάμενες οικογένειες |
| shared | Τύπος μεταβλητών που είναι κοινές μεταξύ Μίκρο-νημάτων |
| index | Τύπος μεταβλητών που αναπαριστούν τον δείκτη ενός νήματος |
| sync | Δομή που χρησιμοποιείται για τον ορισμό του τερματισμού μιας ορισμένης οικογένειας |
| break | Δομή που χρησιμοποιείται για τον τερματισμό μιας οικογένειας μέσω ενός νήματος της |
| squeeze | Δομή που χρησιμοποιείται για την διακοπή της εκτέλεσης μιας ορισμένης οικογένειας, ώστε να μπορεί να επανεκκινηθεί από το σημείο όπου διακόπηκε |
| kill | Δομή που χρησιμοποιείται για τον τερματισμό μιας ορισμένης οικογένειας |

| | |
|---|---|
| family | Τύπος μεταβλητής που ορίζει μια οικογένεια νημάτων |
| place | Τύπος μεταβλητής που ορίζει ένα μέρος στο οποίο εκτελείται μια οικογένεια νημάτων |

### 3.4. Δημιουργώντας οικογένειες νημάτων (`create`)

Το `create` ορίζει μια εργασία ως μια οικογένεια νημάτων, που ορίζεται πάνω σε μια μεταβλητή δείκτη. Τα νήματα ορίζονται είτε από μια σύνθετη εντολή είτε με τη χρήση ενός ονομασμένου νήματος. Το `create` επιστρέφει ένα μοναδικό αναγνωριστικό για κάθε οικογένεια, για την αναγνώριση και τον έλεγχο της. Είναι μια εντολή ελέγχου ροής και μπορεί να βρίσκεται μέσα στο κώδικα, στο σημείο όπου θα επιτρεπόταν το `for` ή το `if`. Η εντολή `create` έχει την παρακάτω δομή:

```
create (family; place; start; limit; step; block; timer)
<thread> | <compound statement>;
```

Πίνακας 3.2 Ορίσματα της Εντολής create

| | |
|---|---|
| family | Ορίζει το όνομα μιας μεταβλητής τύπου `family` για την αποθήκευση του μοναδικού αναγνωριστικού, που χρησιμοποιείται για τον συγχρονισμό, την επικοινωνία και τον έλεγχο του νήματος |
| place | Ορίζει ένα μέρος όπου θα γίνει η εκτέλεση της οικογένειας νημάτων (προεπιλογή: ορισμός από το σύστημα) |
| start | Ορίζει την αρχή του συνόλου των δεικτών για μια οικογένεια Μίκρο-νημάτων, η έκφραση υπολογίζεται κατά την εκτέλεση του `create` (προεπιλογή: 0) |
| limit | Ορίζει το όριο του συνόλου των δεικτών για μια οικογένεια Μίκρο-νημάτων, η έκφραση υπολογίζεται κατά την εκτέλεση του `create` (προεπιλογή: maxint) |
| step | Ορίζει το βήμα μεταξύ των τιμών που παίρνει ο δείκτης, η έκφραση υπολογίζεται κατά την εκτέλεση του `create` (προεπιλογή: 1) |

| | |
|---|---|
| block | Ορίζει το άνω όριο στον αριθμό νημάτων που μπορούν να ανατεθούν σε έναν επεξεργαστή κάθε στιγμή, η έκφραση υπολογίζεται κατά την εκτέλεση του `create` (προεπιλογή: ορισμός από το σύστημα) |
| timer | Ορίζει ένα ρολόι και περιορίζει την ανάθεση νημάτων σε το πολύ `block` νήματα ανά χτύπο του ρολογιού (προεπιλογή: τα νήματα δημιουργούνται καθώς ελευθερώνονται πόροι, σύμφωνα με τους περιορισμούς που θέτει το `block`) |

Καθένα από αυτά τα μέρη μπορεί να παραληφθεί, εκτός από το `family`, και θα αντικατασταθεί από την αντίστοιχη προεπιλογή για αυτό. Η εντολή `create` αποτιμά τις παραμέτρους και δημιουργεί νήματα σε σειρά block-δεικτών, και δυναμικά αναθέτει ένα πλαίσιο της μνήμης συγχρονισμού σε κάθε νήμα που φτιάχνει. Οι τρεις εκφράσεις για τους δείκτες (`start`, `limit` και `step`) ορίζουν το σύνολο δεκτών πάνω στο οποίο δημιουργούνται τα νήματα και μια μοναδική τιμή από αυτή την ακολουθία είναι διαθέσιμη σε κάθε νήμα σε μια από τις τοποθεσίες στο πλαίσιο της μνήμης συγχρονισμού. Όλες οι άλλες τοποθεσίες αρχικοποιούνται σε κενές, δηλαδή μπλοκάρουν κάθε προσπάθεια ανάγνωσης από αυτές πριν εγγραφούν.

Το `create` μπορεί να κατανέμει νήματα σε έναν αριθμό επεξεργαστών με μια ντετερμινιστική κατανομή ώστε να εκτελεστούν ανεξάρτητα σε καθέναν από τους επεξεργαστές. Ο μόνος συντονισμός που απαιτείται είναι η κατανομή των παραμέτρων για την οικογένεια και η διαχείριση των διαφόρων μορφών τερματισμού. Αν η οικογένεια είναι κατανεμημένη σε `p` επεξεργαστές, η σειρά block-δεικτών ορίζεται ως εξής: Το πρώτο block νημάτων στην ακολουθία των δεικτών ανατίθεται στο πρώτο επεξεργαστή και λοιπά, έτσι ώστε `p*block` δείκτες δημιουργούνται σε έναν κύκλο αναθέσεων σε `p` επεξεργαστές. Η χρήση του `block` υποστηρίζει την δημιουργία άπειρων νημάτων σε μια οικογένεια, αφού μπορούν να δημιουργούνται και να χρησιμοποιούνται σε κύκλους αναθέσεων.

Για να κατανοήσουμε καλύτερα την εντολή `create` ας δούμε τον παρακάτω κώδικα, με τον οποίο παράγεται μια οικογένεια με 10 νήματα. Κάθε νήμα αναθέτει στην αντίστοιχη θέση του πίνακα a[] την τιμή 1.

```
create (fid;1; 10; 1)
{     index i;
      a[i]=1;
}
```

### 3.5. Ονομασμένα νήματα (thread)

Στην μTC όλες οι συναρτήσεις ορίζονται ως νήματα και ένα ονομασμένο νήμα ορίζεται όπως μια συνάρτηση σε C. Χρησιμοποιεί μια λίστα ορισμάτων με τον συνηθισμένο τρόπο και μπορεί να μεταγλωττιστεί χωριστά και να συμπεριληφθεί χρησιμοποιώντας μια κεφαλίδα (header). Τα ονομασμένα νήματα μπορούν να αναφερθούν σε μια δομή `create` με μια λίστα παραμέτρων που παρέχεται από το νήμα που δημιουργεί την οικογένεια νημάτων. Η σύνταξη των ονομασμένων νημάτων είναι η παρακάτω:

```
<break type> thread <name> (<argument list>){…}
```

Υπάρχουν κάποιες διαφορές μεταξύ των νημάτων και των συναρτήσεων. Αρχικά, εκεί που οι συναρτήσεις έχουν την εντολή `return`, μια οικογένεια νημάτων δεν μπορεί να επιστρέψει κάτι. Αντίθετα, ένα νήμα μπορεί να παρέχει μια μονοδιάστατη τιμή στην εντολή `break` και ο τύπος αυτής της τιμής ορίζεται με τον ίδιο τρόπο όπως η τιμή επιστροφής μιας συνάρτησης. Σε πολλές περιπτώσεις, ένα νήμα δε θα εκτελέσει ένα `break` και έτσι ορίζουμε τον τύπο επιστροφής να είναι `void`. Τα νήματα αυτά επιστρέφουν μια τιμή με μια εγγραφή στην διαμοιραζόμενη μνήμη, γράφοντας σε μια μεταβλητή `shared`, η τιμή της οποίας παρέχεται από το τελευταίο νήμα της οικογένειας νημάτων. Η εκτέλεση μιας εντολής squeeze σε μια οικογένεια επίσης επιστρέφει μια τιμή, αλλά αυτή είναι πάντα τύπου `int`. Για τις εντολές `break` και `squeeze`, η τιμή είναι διαθέσιμη στο νήμα που δημιούργησε την οικογένεια μέσω της εντολής `sync`, η οποία επίσης παρέχει πληροφορίες για τις συνθήκες τερματισμού.

Εφόσον όλες οι συναρτήσεις στην μTC ορίζονται ως νήματα, έναν νήμα δεν μπορεί να περιέχει μια κλήση σε συνάρτηση αλλά μπορεί να δημιουργήσει περεταίρω

οικογένειες νημάτων οι οποίες είναι το αντίστοιχο της κλήσης συναρτήσεων, στο παράλληλο μοντέλο. Οι παράμετροι περνάνε με την χρήση μεταβλητών συγχρονισμού τύπου `shared` ή μέσω *globals* και το νήμα που καλεί είναι ελεύθερο να συνεχίσει την εκτέλεση του μέχρι να είναι απαραίτητο στην οικογένεια ένα `sync`. Ένα πρόγραμμα σε C δεν μπορεί να μετατραπεί τελείως από συναρτήσεις σε νήματα με τέτοιο τρόπο. Ας δούμε ένα παράδειγμα για τον ορισμό ενός ονομασμένου νήματος.

```
int a[10];
int g0;
int thread P1 (int * tmp, shared int s0)
{     index i;
      int t0;
      t0=s0+tmp[i];
      s0=t0;
}
create(fid;1;10;1)
      P1(a,g0);
```

### 3.6. Squeezable νήματα

Στο μοντέλο SVP, η διακοπή της εκτέλεσης ορίζεται σε μια μονάδα εργασίας, σε μια οικογένεια νημάτων και στους απογόνους της. Η διάρκεια που μεσολαβεί μέχρι να γίνει η διακοπεί εξαρτάται από το βάθος στο οποίο το σήμα `squeeze` προωθείται. Χρειάζεται μια πειθαρχημένη προσέγγιση για τη γρήγορη διακοπή της εκτέλεσης αλλά, αφού η οικογένεια νημάτων πρέπει να έχει την δυνατότητα να αναδημιουργηθεί, το σήμα squeeze προωθείται προς τα κάτω ιεραρχικά, όπου ο προγραμματιστής μπορεί να εγγυηθεί για αυτή η δυνατότητα. Ο μηχανισμός για την έκφραση αυτής της έννοιας είναι τα `squeezable` νήματα. Ένα `squeezable` νήμα είναι ένα νήμα το οποίο μπορεί να ξαναεκτελεστεί, μετά από ένα `squeeze`, όσες φορές χρειαστεί μέχρι όλες οι οικογένειες νημάτων που δημιουργεί να έχουν ολοκληρωθεί κανονικά. Πρέπει να περιέχει κώδικα που ενεργοποιεί αυτή την δυνατότητα αναδημιουργίας που ακολουθεί την διακοπή της εκτέλεσης του.

Όταν μια οικογένεια που δημιουργείται με ένα squeezable νήμα διακόπτεται, το σήμα squeeze μπορεί να προωθηθεί σε όλες τις υφιστάμενες οικογένειες που δημιουργήθηκαν από αυτό το νήμα. Τα squeezable νήματα μοιάζουν στην ιδέα με τις re-entrant συναρτήσεις. Πρέπει να έχουν σχεδιαστεί ώστε να μην έχουν επίδραση στην διαμοιραζόμενη μνήμη, πέρα από αυτές που απαιτούνται για τον επανορισμό του συνόλου δεικτών πάνω στο οποίο δημιουργούνται οι οικογένειες νημάτων. Στην μTC αυτό σημαίνει την εγγραφή στην διαμοιραζόμενη μνήμη της αρχής του συνόλου των δεικτών των οικογενειών που δημιουργεί το νήμα.

### 3.7. Συγχρονισμός κατά τον τερματισμό μιας οικογένειας νημάτων (sync)

Η εντολή sync χρησιμοποιείται για να εντοπίζεται ο τερματισμός μιας ονομασμένης οικογένειας νημάτων. Πρέπει να εκτελεστεί στο ίδιο νήμα που εκτελέστηκε το αντίστοιχο create και κάθε οικογένεια που δημιουργείται πρέπει να συγχρονιστεί προκειμένου να συνεισφέρει ντετερμινιστικά στην κατάσταση του προγράμματος. Η εντολή ορίζεται ως εξής:

```
sync (fid; return_code; return_value);
```

Τα ορίσματα της εντολής είναι τα παρακάτω:

Πίνακας 3.3 Ορίσματα της Εντολής sync

| | |
|---|---|
| fid | Μεταβλητή τύπου family που ορίζει την οικογένεια νημάτων που συγχρονίζεται |
| return_code | Μεταβλητή που θα λάβει τον κωδικό επιστροφής κατά τον τερματισμό της οικογένειας |
| return_value | Μεταβλητή που θα λάβει την τιμή επιστροφής κατά τον τερματισμό της οικογένειας |

Η εντολή sync κατασκευάζει blocks μέχρι η οικογένεια που ορίζει το fid να ολοκληρωθεί και μετά ολοκληρώνει την εκτέλεση της. Επιστρέφει έναν ακέραιο

κωδικό επιστροφής ο οποίος ορίζει τον τρόπο που η οικογένεια τερματίστηκε και μια τιμή τερματισμού η οποία ορίζεται από την εκτέλεση μιας εντολής break ή squeeze. Τα ορίσματα return_code και return_value είναι προαιρετικά. Ο τύπος που επιστρέφεται από ένα break ορίζεται από τον τύπο διακοπής, ενώ η εντολή squeeze επιστρέφει πάντα μια τιμή τύπου int.

Ο κωδικός επιστροφής καθορίζει τον τρόπο ολοκλήρωσης μιας οικογένειας. Μπορεί να λάβει τέσσερις μοναδικές τιμές: { normal, squeeze, kill, break }. Η εκτέλεση της εντολής sync δίνει το σήμα ότι όλες οι τιμές έχουν εγγραφεί στην διαμοιραζόμενη μνήμη από την συγκεκριμένη οικογένεια και είναι πλήρως ορισμένα για ένα ντετερμινιστικό πρόγραμμα.

Ένα παράδειγμα για την καλύτερη κατανόηση της λειτουργίας του sync μπορεί να δοθεί με το παρακάτω κομμάτι κώδικα:

```
create (fid) {…};
…. /*block a*/
sync (fid);
… /*block b*/
```

Σε αυτό το παράδειγμα, με το create ορίζεται μια υφιστάμενη οικογένεια νημάτων η οποία εκτελείται ταυτόχρονα με το νήμα που τη δημιουργεί, το οποίο συνεχίζει να εκτελεί τις εντολές που περιλαμβάνονται στο block a. Όμως οι εντολές που περιλαμβάνονται στο block b θα εκτελεστούν μόνο αφού όλα τα νήματα που ορίστηκαν από το create έχουν τερματίσει και έχουν ανανεώσει την διαμοιραζόμενη μνήμη. Έτσι ακολουθώντας το sync στο fid, το block b μπορεί να διαβάσει με ασφάλεια οποιαδήποτε μεταβλητή στην διαμοιραζόμενη μνήμη, η οποία έχει εγγραφεί από την οικογένεια fid, ενώ το block a δεν μπορεί.

### 3.8. Διακόπτοντας οριστικά την εκτέλεση μιας οικογένειας (**`break`**)

Η εντολή `break` εκτελείται από ένα νήμα και τερματίζει την οικογένεια στην οποία ανήκει αυτό. Αυτό σταματά τη δημιουργία νημάτων και ελευθερώνει όλη τη μνήμη συγχρονισμού, χάνοντας κάθε σημείο συγχρονισμού που μπορεί να έχει η οικογένεια. Επιτρέπει στο νήμα που το εκτέλεσε να επιστρέψει μια μόνο τιμή, με τύπο καθορισμένο από τον ορισμό του νήματος, στο νήμα που το δημιούργησε, η οποία λαμβάνεται από την εντολή `sync` για αυτή την οικογένεια. Η σύνταξη της εντολή είναι:

```
break (return_result);
```

Το όρισμα της εντολής είναι το παρακάτω:

Πίνακας 3.4 Όρισμα της Εντολής break

| | |
|---|---|
| return_result | Μια έκφραση που ορίζει μια τιμή που θα διαβαστεί το την εντολή `sync` για αυτή την οικογένεια νημάτων |

### 3.9. Κάνοντας **`squeeze`** μια οικογένεια

Η εντολή `squeeze` χρησιμοποιείται σε ένα νήμα ελέγχου για να διακόψει μια εργασία που έχει οριστεί ως μια οικογένεια νημάτων. Η σύνταξη της εντολή είναι:

```
squeeze (fid);
```

Το όρισμα της εντολής είναι:

Πίνακας 3.5 Όρισμα της Εντολής squeeze

| | |
|---|---|
| fid | Μια μεταβλητή τύπου `family` για τον ορισμό της οικογένειας που θα διακοπεί |

Αυτή η εντολή φέρνει μια συγκεκριμένη οικογένεια σε έναν καλά ορισμένο τερματισμό, εφόσον η οικογένεια `fid` έχει την δυνατότητα αυτή. Η δημιουργία νέων νημάτων σταματάει και όλα τα νήματα που έχουν ήδη ανατεθεί σε κάποιον επεξεργαστή επιτρέπεται να τερματίσουν φυσιολογικά. Ο τερματισμός ενός νήματος απαιτεί τον τερματισμό όλων των υφιστάμενων οικογενειών νημάτων που μπορεί να έχει δημιουργήσει. Αυτές οι οικογένειες μπορούν να γίνουν `squeeze` μόνο αν τα νήματα που τις δημιούργησαν είναι `squeezable`. Στην πράξη, αυτή η εντολή μπορεί να εκτελεστεί από κάθε νήμα το οποίο εκτελείται παράλληλα με την οικογένεια, αρκεί να έχει λάβει το αναγνωριστικό (`fid`) της οικογένειας. Αυτό προφυλάσσει την οικογένεια από μη εξουσιοδοτημένα σήματα. Απαιτείται από το `squeeze` να αντικαθιστά δυναμικά μια εκτελούμενη εργασία χωρίς απώλεια της κατάστασης συγχρονισμού της.

### 3.10. “Σκοτώνοντας” μια οικογένεια (`kill`)

Η εντολή `kill` είναι παρόμοια με την εντολή `squeeze` και εκτελείται επίσης από ένα παράλληλο νήμα ελέγχου, αλλά χρησιμοποιείται για να υποχρεώσει σε τερματισμό μια συγκεκριμένη οικογένεια, η οποία έχει αυτή την ιδιότητα. Ο συγχρονισμός της οικογένειας χάνεται. Όλα τα νήματα της οικογένειας “σκοτώνονται”, καθώς και αποστέλλεται ένα σήμα `kill` σε όλες τις υφιστάμενες οικογένειες. Η εντολή συντάσσεται έτσι:

```
kill (fid);
```

Το όρισμα της όπως και προηγουμένως:

Πίνακας 3.6 Όρισμα της Εντολής kill

| | |
|---|---|
| fid | Μια μεταβλητή τύπου `family` για τον ορισμό της οικογένειας που θα τερματιστεί |

Αυτή η εντολή οδηγεί σε σταμάτημα μιας παράλληλης περιοχής εκτέλεσης. Αυτό γίνεται με το σταμάτημα της δημιουργίας νημάτων και την υποχρέωση σε τερματισμό

των υπόλοιπων νημάτων τα οποία δημιουργήθηκαν στην ορισμένη οικογένεια και αναδρομικά όλες τις οικογένειες που δημιουργήθηκαν από αυτά τα νήματα. Η μνήμη συγχρονισμού που χρησιμοποιούσαν όλες οι οικογένειες που "σκοτώθηκαν" απελευθερώνεται. Η εντολή αυτή είναι ένας τρόπος να τερματιστεί μια οικογένεια και οι απόγονοι της από περεταίρω εκτέλεση.

### 3.11. Τοποθεσίες (`place`)

Μια υλοποίηση της μTC θα πρέπει να ορίσει έναν μηχανισμό για την ανάθεση των πόρων και η δυναμική παραλληλία θα πρέπει να συνδεθεί με κάποιο τρόπο στους πόρους αυτούς. Θεωρούμε ότι η δυναμική ανάθεση των πόρων υλοποιείται με τη χρήση ενός νήματος του συστήματος το οποίο επιστρέφει μια τιμή τύπου `place`. Αυτό ορίζει το μέρος όπου μια οικογένεια νημάτων θα εκτελεστεί ή θα ανατεθεί. Το `place` είναι ακόμα ο μηχανισμός για την αναγνώριση των υπηρεσιών του συστήματος σε έναν επεξεργαστή SANE, έτσι ώστε το νήμα του συστήματος που αναθέτει τους πόρους να ανατεθεί σε ένα συγκεκριμένο `place` για αυτό το σκοπό. Ακόμα το `place` είναι ο μηχανισμός για τον αμοιβαίο αποκλεισμό, αν αυτός απαιτείται από μια υπηρεσία και υλοποιείται με την ακολουθιακή εκτέλεση των `create` που έχουν ανατεθεί στο `place` όπου απαιτείται ο αποκλεισμός. Στις υπηρεσίες περιλαμβάνονται η ανάθεση πόρων, η ανάθεση μνήμης, η είσοδος και η έξοδος και λοιπά. Δεν απαιτούν όλες οι υπηρεσίες αμοιβαίο αποκλεισμό, αλλά πρέπει να διαχειριστούν την εξάντληση των πόρων τους.

### 3.12. Παραδείγματα χρήσης του μοντέλου μνήμης της μTC

Τα νήματα μοιράζονται την μνήμη, εκτός από το πλαίσιο των μονοδιάστατων μεταβλητών συγχρονισμού. Η διαμοιραζόμενη μνήμη είναι ασύγχρονη, με χαλαρή συνέπεια και χρησιμοποιεί χονδρικό συγχρονισμό κατά τον τερματισμό μιας οικογένειας και χρησιμοποιεί την μνήμη συγχρονισμού για την ασύγχρονη επικοινωνία μεταξύ της μνήμης και του επεξεργαστή. Ο τερματισμός μιας οικογένειας νημάτων (`sync`) στέλνει σήμα ότι όλες τις τοποθεσίες της διαμοιραζόμενης μνήμης που εγγράφονται από την οικογένεια είναι πλέον καλά

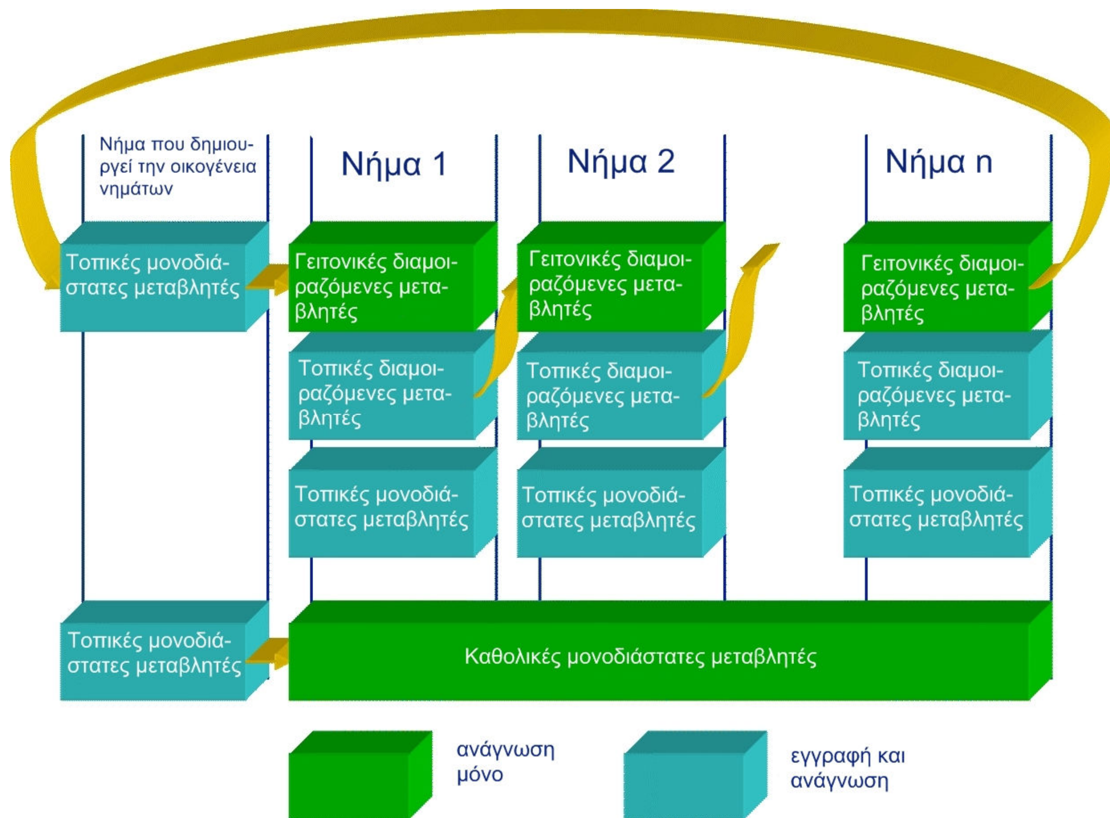
ορισμένες και μια νέα οικογένεια μπορεί πλέον να δημιουργηθεί για να χρησιμοποιήσει αυτές τις τιμές.

Το πλαίσιο ενός νήματος σε μονοδιάστατες μεταβλητές συγχρονισμού είναι είτε:

- Καθολικό σε όλα τα νήματα μιας οικογένειας, αλλά διαθέσιμο μόνο για ανάγνωση
- Τοπικό και μη προσβάσιμο σε κάθε άλλο νήμα
- Διαμοιραζόμενο, το οποίο είναι τοπικό και διαθέσιμο μόνο για ανάγνωση στο επόμενο νήμα στην ακολουθία των δεικτών.

Η μνήμη συγχρονισμού που είναι διαθέσιμη σε μια οικογένεια νημάτων φαίνεται στο Σχήμα 3.2, συμπεριλαμβανομένου του πλαισίου του νήματος που δημιουργεί την νέα οικογένεια, το οποίο παρέχει τις καθολικές τιμές και αρχικοποιεί κάθε αλυσίδα διαμοιραζόμενων μεταβλητών στην οικογένεια. Μια μεταβλητή χαρακτηρισμένη ως `shared` σε ένα νήμα δεν είναι διαθέσιμη σε όλα τα νήματα στην οικογένεια, όπως μπορεί να υποτεθεί, αλλά είναι μια τοπική μεταβλητή που παρέχει την τιμή της στο γειτονικό νήμα στην ακολουθία των δεικτών.

Οι διαμοιραζόμενες μεταβλητές πρέπει να ορίζονται σε κάθε νήμα για να διασφαλίσουν τον ομαλό τερματισμό της οικογένειας, ανεξάρτητα από τον έλεγχο ροής. Μια διαμοιραζόμενη μεταβλητή στο πρώτο νήμα μιας οικογένειας αρχικοποιείται από μια τοπική μονοδιάστατη μεταβλητή του νήματος που το δημιουργεί, είτε κατά την δήλωση της διαμοιραζόμενης μεταβλητής με μια σύνθετη εντολή που ορίζει το νήμα ή ως όρισμα σε ένα ονομασμένο νήμα, δηλωμένο ως διαμοιραζόμενο. Κατά τον τερματισμό της οικογένειας, η μεταβλητή που χρησιμοποιήθηκε για την αρχικοποίηση παίρνει την τιμή που εγγράφεται από το νήμα που εκτελείται τελευταίο σε ακολουθία δεικτών, αλλά αυτός δεν είναι ένας τρόπος συγχρονισμού αφού η μεταβλητή αυτή ήταν ήδη ορισμένη όταν αρχικοποιήθηκε το πρώτο νήμα.




Σχήμα 3.2 Η Σχέση Μεταξύ των Μονοδιάστατων Μεταβλητών Συγχρονισμού σε μια Οικογένεια Νημάτων

### 3.13. Επικοινωνία νημάτων μέσω διαμοιραζόμενων μεταβλητών συγχρονισμού

Το παρακάτω παράδειγμα σχετίζεται με πολλαπλές αναθέσεις σε μια μεταβλητή δηλωμένη ως διαμοιραζόμενη σε ένα νήμα:

```
int *a, n, s0=0;
family fid;
…
create (fid; ; 0; n-1)
{    index i;
     shared int s=s0;
/* αρχικοποίηση του s γίνεται μόνο στο πρώτο νήμα */

     s=s+1;
```

```
      s=s*2;
      a[i]=s;
}
```

Σε αυτό το παράδειγμα, η διαμοιραζόμενη μεταβλητή s ορίζει μια εξάρτηση μεταξύ νημάτων, η οποία αρχικοποιείται από το πρώτο νήμα σε s0. Στον συγχρονισμό της οικογένειας, το s0 θα ανανεωθεί από την τιμή του s, η οποία γράφεται από το νήμα που θα εκτελεστεί τελευταίο στην ακολουθία δεικτών. Όμως σε αυτό το παράδειγμα, το s γράφεται δυο φορές σε κάθε νήμα, το οποίο δημιουργεί ένα θέμα λόγω της ταυτόχρονης εκτέλεσης των νημάτων. Αυτό σημαίνει ότι η τιμή που κάθε νήμα βλέπει από τον γείτονα του είναι μη ντετερμινιστική, δηλαδή μπορεί να είναι η τιμή από την πρώτη ή τη δεύτερη ανάθεση στο s. Για να λυθεί αυτή η κατάσταση, η μTC υποθέτει σημασιολογία ανάλογη με την ακολουθιακή εκτέλεση του κάθε νήματος σε σειρά δεικτών. Αυτό πετυχαίνεται με την διασφάλιση ότι μόνο η τελευταία ανάθεση στο s συγχρονίζει με το επόμενο νήμα και κάθε προηγούμενη ανάθεση στο s γίνεται τοπικά. Έτσι ο κώδικας που θα παράγει ο μεταγλωττιστής της μTC θα είναι αντίστοιχος με αυτόν που θα παράγει για το επόμενο παράδειγμα:

```
int *a, n, s0=0;
family fid;
…
create (fid; ; 0; n-1)
{     index i;
      shared int s=s0;
/* αρχικοποίηση του s γίνεται μόνο στο πρώτο νήμα */

      t=s+1;
      s=t*2;
      a[i]=s;
}
```

Οι τιμές που θα εγγραφούν στο a[i] θα είναι οι {2, 6, 14, 30, …}, το οποίο είναι αντίστοιχο με την ακολουθιακή εκτέλεση αυτών των νημάτων.

Οι εξαρτήσεις μεταξύ νημάτων που ορίζονται σε δεικτοδοτούμενες δομές δεδομένων είναι επίσης μη ντετερμινιστικές, αφού αυτό σπάει τον κανόνα ότι δυο νήματα σε μια οικογένεια γράφουν και διαβάζουν από την ίδια τοποθεσία της διαμοιραζόμενης μνήμης. Η λύση σε αυτό είναι η χρήση σαφών διαμοιραζόμενων μεταβλητών. Αν και θα μπορούσε να γίνει από τον μεταγλωττιστή της μTC αυτός ο μετασχηματισμός, θεωρείται ευθύνη του συγγραφέα του κώδικα μTC, για την αποφυγή επιπλέον ελέγχων σε χρόνο εκτέλεσης. Ένα παράδειγμα:

```
create (fid; ; 1; n-1)
{     index i;
      sum[i]=a[i]+sum[i-1];
}
```

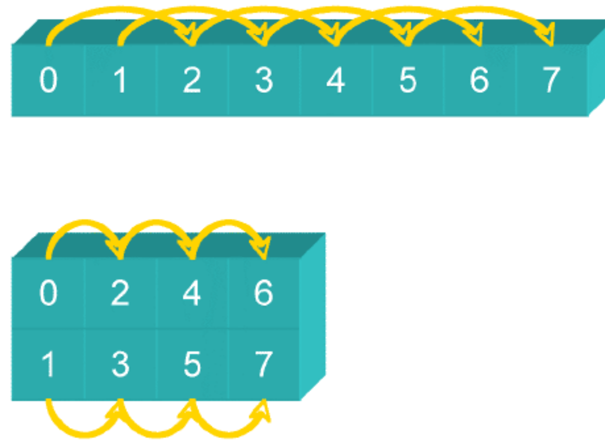
Ο κώδικας αυτός θα πρέπει να ξαναγραφεί για την αποφυγή μη ντετερμινιστικών αποτελεσμάτων:

```
create (fid; ; 1; n-1)
{     index i;
      shared int s=s0;

      s=a[i]+s;
      sum[i]=s;
}
```

Στα παραδείγματα που ακολουθούν οι περιορισμοί της δια-νηματικής επικοινωνίας περιγράφονται. Οι εξαρτήσεις μπορούν να υπάρχουν μόνο ανάμεσα σε γειτονικά νήματα στην ακολουθία των δεικτών. Αυτός ο περιορισμός δεν είναι και τόσο αυστηρός αφού εμφανίζεται λόγω της ιεραρχικής φύσης της παραλληλίας στο μοντέλο. Η μη τοπική επικοινωνία είναι εκτεθειμένη από αυτόν τον περιορισμό και πρέπει να προγραμματιστεί ρητά. Δυο τεχνικές μπορούν να χρησιμοποιηθούν για το μετασχηματισμό του κώδικα ώστε να επιτευχθεί η τοπικότητα της επικοινωνίας. Η πρώτη εφαρμόζεται σε κανονικές, μη τοπικές εξαρτήσεις και μετασχηματίζει μια

γραμμική ακολουθία νημάτων που περιέχει μη γειτονική επικοινωνία σε ένα πρόβλημα δυο διαστάσεων, με την χρήση εμφωλευμένων οικογενειών νημάτων, όπου η ιδιότητα της γειτονικής επικοινωνίας επιτυγχάνεται. Αυτή η τεχνική μπορεί να εφαρμοστεί σε αυθαίρετα μεγάλους πίνακες και περιγράφεται στο Σχήμα 3.2 και τον κώδικας που φαίνεται παρακάτω:

```
create (fid; ; 0; 7)
{     index i;
      a[i]=(a[i]+a[i-2])/2;

/* Αυτό δεν μπορεί να αναπαρασταθεί με μια διαμοιραζόμενη
μεταβλητή */

}
sync(fid);


create (fid1; ; 0; 1)
{     index k;
      family fid2;
      create (fid2; ; 0; 3)
      {     index j;
            shared float s;
            i=2*j+k;
            s=(a[i]+s)/2;
            a[i]=s;

/*  Αυτή  η  οικογένεια  μπορεί  να  χρησιμοποιήσει  μια
διαμοιραζόμενη μεταβλητή για να πάρει το a[i-2] */
      }
}sync(fid);
```




Σχήμα 3.3 Μη Γειτονικές Εξαρτήσεις στον Πίνακα a[] Μετατρέπονται σε Γειτονικές με τον Μετασχηματισμό του Προβλήματος στις Δυο Διαστάσεις με την Χρήση Εμφωλευμένων Οικογενειών Νημάτων

Η δεύτερη τεχνική για να αποφύγουμε την δια-νηματική επικοινωνία μεταξύ μη γειτονικών νημάτων είναι η χρήση στατικής δια-νηματικής δρομολόγησης. Το παρακάτω παράδειγμα υλοποιεί εξαρτήσεις στο προηγούμενο νήμα και στο προ-προηγούμενο. Υπολογίζει τους πρώτους δέκα αριθμούς Fibonacci. Σε αυτόν τον κώδικα, η ανάθεση `temp1=temp2` υλοποιεί τη δια-νηματική δρομολόγηση στην μνήμη συγχρονισμού:

```
void thread fib (int *fibonacci, n)
{     family fid;
      t1=0;
      t2=1;
      fibonacci[0]=t1;
      fibonacci[1]=t2;
      create (fid; ; 2; n-1)
      {    index i;
           shared int temp1=t1, temp2=t2;
           fibonacci[i]=temp1+temp2;
           temp1=temp2;
           temp2=fibonacci[i];
```

```
    }
    sync(fid);
…
}
```

Τέλος, παρακάτω δίνεται ένα πιο περίπλοκο παράδειγμα, το οποίο υλοποιεί τον πολλαπλασιασμό ενός πίνακα με ένα διάνυσμα. Αυτό είναι ένα ακόμα παράδειγμα της χρήσης των εμφωλευμένων οικογενειών νημάτων, με δεικτοδότηση δυο διαστάσεων. Χρησιμοποιεί έναν δείκτη `i` για την εξωτερική οικογένεια, η οποία είναι τοπική για το νήμα που τη δημιουργεί. Στην εσωτερική οικογένεια νημάτων ο πίνακας `a` δεικτοδοτείται μαζί από το `i` το οποίο είναι καθολικό για όλα τα εσωτερικά νήματα και από το `j`, το οποίο είναι τοπικό στο εσωτερικό νήμα. Αυτό ορίζει $n^2$ νήματα, όπου τα `n` εξωτερικά νήματα (`family fido`) είναι ανεξάρτητα και τα `n` εσωτερικά (`family fidi`) περιέχουν μια αλυσίδα εξάρτησης στο `s`. Παρατηρούμε τη χρήση της παραμέτρου block στην εξωτερική οικογένεια για τον έλεγχο της ανάθεσης των πόρων.

```
void thread matvec (int *a, *x, *y, n)
{    family fido;
     create (fido; ; 0; n-1; 1; 4)
     {    index i;
          family fidi;
          int s0=0;
         create (fidi; ; 0; n-1)
         {    index j;
              shared int s=s0;
              s=s+a[i][j]*x[j];
         }
         sync (fidi);
         y[i]=s;
     }
     sync(fido);
}
```

### 3.14. Επικοινωνία νημάτων μέσω της ασύγχρονης διαμοιραζόμενης μνήμης

Η διαμοιραζόμενη μνήμη χρησιμοποιεί τους συνηθισμένους κανόνες εμβέλειας της C. Παρόλα αυτά, υπάρχουν κάποιοι περιορισμοί στην χρήση της διαμοιραζόμενης μνήμης για να αποφύγουμε τα μη ντετερμινιστικά προγράμματα. Οι αναγνώσεις και οι εγγραφές στην διαμοιραζόμενη μνήμη πρέπει να ρυθμίζονται, ώστε δυο ταυτόχρονα νήματα δεν θα διαβάσουν και γράψουν στην ίδια τοποθεσία στην μνήμη και επίσης η κάθε τοποθεσία δεν θα εγγραφεί από περισσότερα από ένα νήματα. Αυτό αφορά νήματα στην ίδια οικογένεια η σε διαφορετικές οικογένειες που εκτελούνται παράλληλα.

Στο παρακάτω μTC πρόγραμμα δυο ταυτόχρονα νήματα διαβάζουν και γράφουν στην ίδια θέση στη διαμοιραζόμενη μνήμη:

```
void thread main( )
{     int *a, n;
      family fid;
      …
      create (fid; ; 0; n-1)
      {     index i;
            a[i] = a[i] + a[i+1];
      }
}
```

Το πρόγραμμα αυτό δεν δίνει με συνέπεια το αναμενόμενο αποτέλεσμα από την ανάγνωση του a[i+1] (όπως ορίζεται από την ακολουθιακή εκτέλεση σύμφωνα με τους δείκτες του βρόχου), αφού ένα νήμα μπορεί να διαβάσει οποιαδήποτε από τις δυο τιμές, είτε την τιμή του a[i+1] που ορίστηκε πριν αρχίσει η εκτέλεση της οικογένειας, είτε τη νέα τιμή που γράφτηκε στο a[i] από το διάδοχο του νήματος στην ακολουθία των δεικτών. Για να δοθεί μια ντετερμινιστική λύση στο πρόγραμμα αυτό μπορούμε να μετονομάσουμε τον πίνακα όπου γίνεται η ανάθεση:

```
void thread main( )
{     int *a, *ca, n;
```

```
      family fid;
      …
      create (fid; ; 0; n-1)
      {     index i;
            ca[i] = a[i] + a[i+1];
      }
}
```

Στη γενική περίπτωση δεν είναι εύκολο να εντοπιστεί ο μη ντετερμινισμός, όπως δείχνει το παράδειγμα που ακολουθεί. Εδώ, το πρόγραμμα είναι καλά ορισμένο, αφού το σύνολο δεικτών περιέχει μόνο άρτιους δείκτες, άρα δεν υπάρχουν δυο ταυτόχρονα νήματα που να διαβάζουν και να γράφουν στην ίδια θέση με αυτή την αλλαγή στο παραπάνω πρόγραμμα:

```
void thread main( )
{     int *a, n;
      family fid;
      …
      create (fid; ; 0; n-1; 2)
      {     index i;
            a[i] = a[i] + a[i+1];
      }
}
```

# ΚΕΦΑΛΑΙΟ 4. ΣΥΓΓΕΝΕΙΣ ΕΡΓΑΣΙΕΣ

4.1. Oscar
4.2. Polaris
4.3. Cetus
4.4. SUIF
4.5. Parafrase-2
4.6. NANOS
4.7. PROMIS
4.8. RHODOS
4.9. PARADIGM
4.10. VFC
4.11. CoSy

## 4.1. Oscar

Ο Oscar ([13]) είναι ένας μεταγλωττιστής μετατροπής σειριακού κώδικα σε παράλληλο, ο οποίος αναπτύχθηκε στο Ιαπωνικό Πρόγραμμα IT21 με τίτλο «Advanced Parallelizing Compiler». Ο μεταγλωττιστής αυτός υλοποιεί μετατροπή κώδικα σε παράλληλο για διάφορα συστήματα, από chip-πολυεπεξεργαστές ως και εξυπηρετητές. Ιεραρχικά, εκμεταλλεύεται χοντρικά τον παραλληλισμό μεταξύ των βρόχων, των υπό-ρουτινών και των βασικών blocks και σχεδόν με λεπτομέρεια μεταξύ των εντολών ενός βασικού block, επιπλέον του συμβατικού παραλληλισμού μεταξύ των επαναλήψεων των βρόχων. Ακόμα, βελτιστοποιεί καθολικά την χρήση της μνήμης cache μεταξύ διαφορετικών βρόχων, βασισμένος στην τεχνική του εντοπισμού των δεδομένων για την μείωση του επιπλέον κόστους προσπέλασης της μνήμης.

Λόγοι της δημιουργίας του Oscar ήταν η βελτίωση της απόδοσης, η ευκολία της χρήσης και το κόστος της απόδοσης των συστημάτων πολλών επεξεργαστών με κατανεμημένη μνήμη. Το Πρόγραμμα IT21 περιλαμβάνει την δημιουργία υλικού και λογισμικού, ουσιαστικού για τη επόμενη γενιά υπολογιστών και δικτύων, για την προώθηση της συνεργασίας μεταξύ της βιομηχανίας, της εκπαίδευσης και της κυβέρνησης.

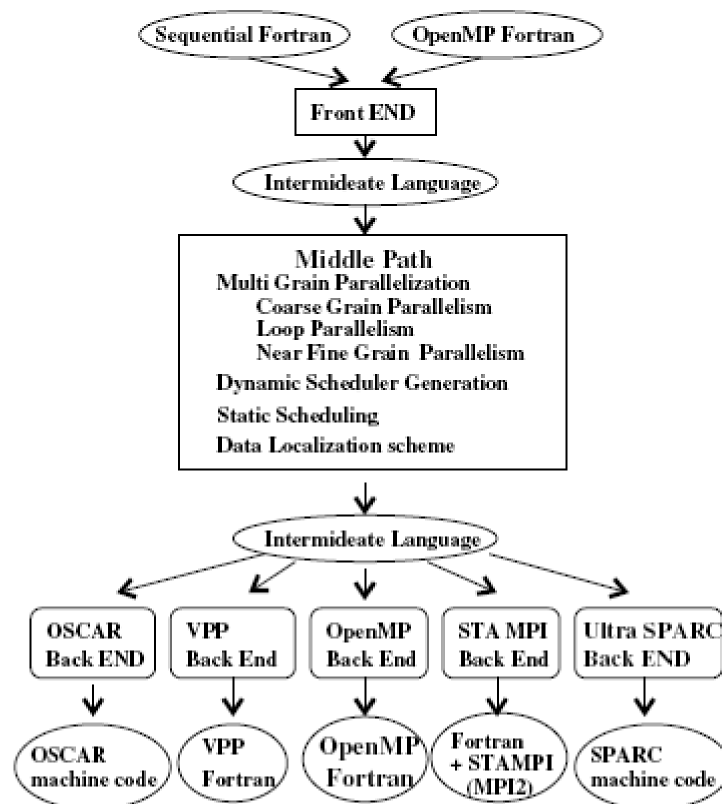
Σκοπός του Προγράμματος αυτού είναι η δημιουργία μιας τεχνολογίας μεταγλώττισης αυτόματης μετατροπής κώδικα σε παράλληλο, ανεξάρτητη από πλατφόρμα, η οποία εξάγει παραλληλισμό ιεραρχικά από ολόκληρο το πρόγραμμα. Ο μεταγλωττιστής αυτός βελτιώνει πολύ την δυνατότητα χρήσης και το κόστος της απόδοσης των συστημάτων πολυεπεξεργαστών που χρησιμοποιούνται εκτεταμένα στους Υπολογισμούς Υψηλών Επιδόσεων.

Όπως φαίνεται στο Σχήμα 4.1, ο Oscar αποτελείται από ένα front-end για Fortran, ένα ενδιάμεσο μονοπάτι για την μετατροπή του κώδικα σε παράλληλο και πολλά διαφορετικά back-ends για διαφορετικά μηχανήματα, όπως UltraSPARC chip-multiprocessor, OSCAR chip-multiprocessor, SMP (Symmetric multiprocessing – πολλοί επεξεργαστές συνδεδεμένοι σε μια διαμοιραζόμενη μνήμη) μηχανήματα που υποστηρίζουν OpenMP και cluster συστήματα που υποστηρίζουν MPI.

Ο μεταγλωττιστής αυτός παράγει χοντρικά εργασίες που ονομάζονται «macro-tasks», όπως βρόχους, υπό-ρουτίνες και βασικά blocks, αναλύει την παραλληλία μεταξύ των εργασιών βασισμένος στην ανάλυση ελέγχου και εξάρτησης δεδομένων, αποσυνθέτει τις εργασίες και τα δεδομένα για βελτιστοποίηση στην cache ή την κοινή κατανεμημένη μνήμη με αποσύνθεση ευθυγραμμισμένη στον βρόχο (loop aligned decomposition), αναθέτει τις εργασίες στους επεξεργαστές ή σε ομάδες επεξεργαστών, στατικά ή δυναμικά υπολογίζοντας την τοπικότητα των δεδομένων.

Αντίθετα με τον Oscar, στο μοντέλο που υποστηρίζει ο δικός μας μεταγλωττιστής ορίζονται οικογένειες νημάτων για κάθε εργασία και υποστηρίζεται η επικοινωνία μεταξύ των νημάτων της ίδιας οικογένειας μέσω της μνήμης συγχρονισμού, ενώ τα

δεδομένα βρίσκονται αποθηκευμένα στην ασύγχρονη διαμοιραζόμενη μνήμη. Μια ακόμα διαφορά είναι η γλώσσες για τις οποίες προορίζονται ο Oscar και ο μεταγλωττιστής μας. Ο Oscar προορίζεται για μετατροπή από Fortran σε κώδικα μηχανής για κάποια συγκεκριμένα παράλληλα συστήματα ή σε Fortran με μια παράλληλη επέκταση όπως VPP, OpenMP ή STA MPI. Αντίθετα ο μεταγλωττιστής μας προορίζεται για μετατροπή από κώδικα σε C στην ενδιάμεση αναπαράσταση που υποστηρίζει ο πολύ-νηματικός επεξεργαστής μας, δηλαδή σε μTC.




Σχήμα 4.1 Η Αρχιτεκτονική του Oscar

### 4.2. Polaris

Ο Polaris ([5], [6]) είναι ένα εργαλείο για μετατροπές από κώδικα σε κώδικα και πολύπλοκη ανάλυση κώδικα Fortran 77. Η εσωτερική του αναπαράσταση αποτελείται από ένα βασικό αφηρημένο συντακτικό δέντρο, στην κορυφή του οποίου υπάρχουν πολλά επίπεδα λειτουργικότητας. Αυτή η λειτουργικότητα επιτρέπει περίπλοκες λειτουργίες στις δομές δεδομένων, καθώς και του επιτρέπει να μιμηθεί άλλες εσωτερικές αναπαραστάσεις. Ακόμα, η εσωτερική αναπαράσταση είναι σχεδιασμένη

ώστε να ενισχύει την συγκρότηση της κατάστασης της εσωτερικής δομής, και ως προς την ορθότητα των δομών δεδομένων και του κώδικα Fortran που χειρίζεται. Επιπλέον, οι λειτουργίες πάνω στην εσωτερική αναπαράσταση έχουν ως αποτέλεσμα την αυτόματη ανανέωση των δομών δεδομένων που επηρεάζονται, όπως οι πληροφορίες ροής.

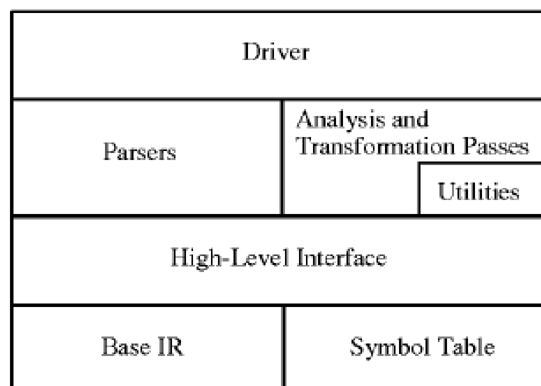
Σκοπός του είναι να παρέχει έναν μεταγλωττιστή παραλληλοποίησης που είναι ικανός να παραλληλίζει ικανοποιητικά προγράμματα γραμμένα σε Fortran 77 για έναν αριθμό μηχανημάτων, περιλαμβανομένων μαζικά παράλληλων συστημάτων και παράλληλων σταθμών εργασίας. Βασίζεται στο παλαιότερο «Cedar Fortran project» και στο σύστημα χειρισμού προγραμμάτων «Delta».

Γενικά, ο Polaris είναι αντιπροσωπευτικός για τους μεταγλωττιστές που είναι σχεδιασμένοι για μια συγκεκριμένη γλώσσα, επιτελούν καλά τον σκοπό, αλλά είναι δύσκολο να επεκταθούν. Αξιοποιεί τον παραλληλισμό των βρόχων χρησιμοποιώντας την inline επέκταση των υπό-ρουτινών, την συμβολική διάδοση, την ιδιωτικοποίηση των πινάκων και την ανάλυση εξάρτησης δεδομένων σε χρόνο εκτέλεσης.

### 4.3. Cetus

Διάδοχος του Polaris είναι ο Cetus ([10]), ο οποίος βρίσκεται ακόμα υπό ανάπτυξη και υποστηρίζει C, C++ και Java. Η μετατροπή ενός προγράμματος Fortran 77 σε παράλληλο είναι πιο εύκολη, αφού η γλώσσα αυτή είναι πιο απλή και δεν έχει δείκτες και τύπους ορισμένους από τον χρήστη, και η τεχνολογία αυτή είναι πιο ώριμη από ότι για τις μοντέρνες γλώσσες, όπως η C++, η Java και η C. Ο σχεδιασμός του Cetus, όπως περιγράφεται στο Σχήμα 4.2, έγινε γιατί κρίθηκε χρήσιμο ένα εργαλείο που βοηθάει στη μετατροπή κώδικα σε παράλληλο, το οποίο να υποστηρίζει ανάλυση μεταξύ των διαδικασιών, να παρέχει ένα εξελιγμένο περιβάλλον τεχνολογίας λογισμικού και να επιτρέπει την μεταγλώττιση μεγάλων, ρεαλιστικών εφαρμογών. Ο σχεδιασμός του Cetus έγινε από την αρχή, αλλά είναι επηρεασμένος από τον Polaris και έχει διατηρήσει πολλά στοιχεία του.

Ο Cetus χρησιμοποιεί μια ιεραρχική δομή αναπαράστασης του προγράμματος, σε αντίθεση με τον Polaris που έχει έναν επίπεδο τρόπο αναπαράστασης. Δημιουργεί ένα συντακτικό δέντρο, το οποίο μετατρέπεται στην εσωτερική αναπαράσταση, ώστε να είναι διαθέσιμη για τα επόμενα περάσματα. Δεν περιλαμβάνει κάποιο back-end και προορίζεται για μετατροπή από κώδικα σε κάποια κανονικοποιημένη μορφή μετά από ανάλυση, η οποία μπορεί να χρησιμοποιηθεί από άλλα προγράμματα ή scripts. Έχει χρησιμοποιηθεί σε συστήματα για την ανάλυση βρόχων, μετατροπή κώδικα από OpenMP σε κώδικα για συστήματα διαμοιραζόμενης μνήμης με χρήση του POSIX API και για συστήματα κατανεμημένης διαμοιραζόμενης μνήμης, μετατροπή κώδικα από OpenMP σε MPI, ανάλυση δεικτών και μετατροπή κώδικα Java bytecode σε bytecode μετά από εφαρμογή βελτιστοποιήσεων.




Σχήμα 4.2 Συνιστούντα Μέρη και Διεπιφάνειες του Cetus

Αν και υποστηρίζει τη γλώσσα C, είναι περισσότερο ένα εργαλείο ανάλυσης και βελτιστοποίησης των προγραμμάτων που προορίζονται για παραλληλοποίηση, παρά ένα εργαλείο παραλληλοποίησης. Είναι κατάλληλο για μοντέλα που περιλαμβάνουν διαμοιραζόμενη μνήμη, κατανεμημένη ή όχι, αλλά δεν υποστηρίζει αναλύσεις και βελτιστοποιήσεις για τη μνήμη συγχρονισμού και την επικοινωνία μεταξύ νημάτων.

## 4.4. SUIF

Ο μεταγλωττιστής SUIF (Stanford University Intermediate Format) ([7]) σχεδιάστηκε για να μετατρέπει επιστημονικά προγράμματα σε παράλληλα για μονοδιάστατους παράλληλους υπολογιστές. Μερικές από τις τεχνικές που χρησιμοποιεί παράγουν κώδικα σωστό και αποδοτικό είτε η μνήμη του συστήματος είναι διαμοιραζόμενη,

είτε κατανεμημένη. Δεν υποστηρίζει όμως την ύπαρξη μνήμης συγχρονισμού για την επικοινωνία μεταξύ των παράλληλων διεργασιών. Μετατρέπει βρόχους σε παράλληλους χρησιμοποιώντας ανάλυση μεταξύ των διαδικασιών, μετασχηματισμούς και βελτιστοποίηση της τοπικότητας των δεδομένων.

Ένας από τους βασικότερους σκοπούς του μεταγλωττιστή αυτού είναι να μεταφράζει αυτόματα σειριακούς υπολογισμούς πυκνών πινάκων σε αποδοτικό παράλληλο κώδικα για μεγάλης κλίμακας παράλληλα μηχανήματα. Δέχεται είσοδο σε Fortran-77 και C, η οποία πρώτα μετατρέπεται στην ενδιάμεση αναπαράσταση του SUIF, όπου γίνονται βελτιστοποιήσεις και ανάλυση του προγράμματος. Η λειτουργία του χωρίζεται σε 4 φάσεις, την συμβολική ανάλυση, την ανάλυση παραλληλισμού και τοπικότητας, την ανάλυση επικοινωνίας και παραλληλισμού και την δημιουργία του κώδικα. Οι 3 πρώτες φάσεις είναι ανάλυση του ενδιάμεσου κώδικα και δεν κάνουν αλλαγές αλλά περνάνε τις πληροφορίες που χρειάζεται η τελευταία φάση μέσω σχολίων στον ενδιάμεσο κώδικα.

Ο SUIF εκτελεί πρώτα μια τοπική ανάλυση, όπου χρησιμοποιεί τους μετασχηματισμούς (π.χ. ανταλλαγή βρόχων, skewing και αντιστροφή), ώστε να πετύχει την λεπτότερη δυνατή παραλληλοποίηση. Στη συνέχεια γίνεται μια συνολική ανάλυση των βρόχων όπου αποφασίζεται η καλύτερη ανάθεση των δεδομένων και των υπολογισμών στους διαθέσιμους επεξεργαστές. Ο μεταγλωττιστής προσπαθεί να βρει αναθέσεις με την ελάχιστη επικοινωνία μεταξύ των επεξεργαστών, ενώ ο παραλληλισμός είναι επαρκής ώστε να διατηρεί τους επεξεργαστές απασχολημένους.

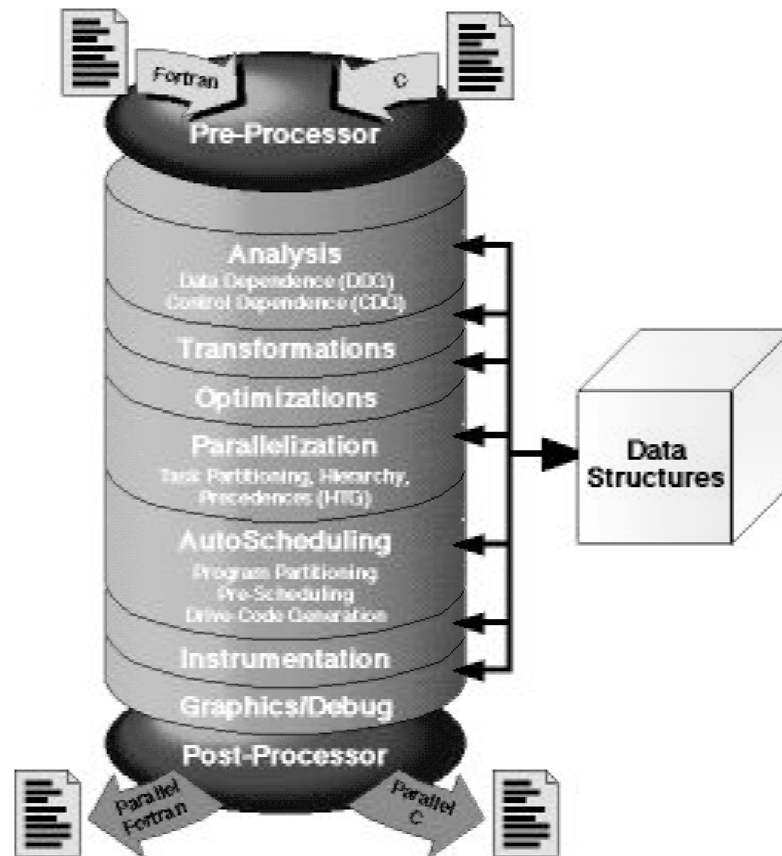
### 4.5. Parafrase-2

Ο μεταγλωττιστής Parafrase-2 ([15], [20]) παίρνει ως είσοδο κώδικα σε C ή σε Fortran (Fortran 77 ή Cedar Fortran) και η έξοδος του είναι αντίστοιχα σε C ή Cedar Fortran. Περιλαμβάνει αναλύσεις και βελτιστοποιήσεις βρόχων, με σκοπό την κανονικοποίηση τους. Συλλέγει πληροφορίες για τους βρόχους, τα όρια τους και τις μεταβλητές επαγωγής. Χρησιμοποιείται από πολλά συστήματα για μια προ-επεξεργασία της εισόδου, στο frontend. Τα βήματα που ακολουθεί είναι η αναδόμηση του προγράμματος σε εργασίες και μετά η ανάλυση ελέγχου και δεδομένων, ώστε να

καθοριστούν οι εργασίες αυτές που μπορούν να εκτελεστούν παράλληλα. Η πρωτοτυπία του είναι η εστίαση τόσο στην ανάλυση ελέγχου όσο και στην ανάλυση των δεδομένων.

Ο Parafrase-2 χρησιμοποιεί την αναπαράσταση της ιεραρχίας του προγράμματος με το HTG (Γράφημα Ιεραρχικών Εργασιών), το οποίο είναι η ενδιάμεση αναπαράσταση που έχει σχεδιαστεί για τον Parafrase-2. Περιλαμβάνει τα γραφήματα ελέγχου ροής και εξάρτησης δεδομένων, ως υπό-γραφήματα. Η αναπαράσταση του προγράμματος σε HTG κρύβει τη σύνταξη του προγράμματος, αλλά τονίζει τις εξαρτήσεις μεταξύ περιοχών του προγράμματος. Η ιεραρχική οργάνωση του HTG διευκολύνει την επιλογή εργασιών κατάλληλων για παραλληλισμό στην αρχιτεκτονική-στόχο.

Απευθύνεται σε συστήματα με διαμοιραζόμενη μνήμη, κατανεμημένη ή όχι, και όχι σε αυτά που περιλαμβάνουν επίσης μνήμη συγχρονισμού μεταξύ των εργασιών. Επιτρέπει διαφορετικές γλώσσες εισόδου και αυτό είναι εφικτό λόγω της κοινής ενδιάμεσης αναπαράστασης που υποστηρίζεται. Η έξοδος του Parafrase-2 είναι μια παραλλαγή του κώδικα που δόθηκε ως είσοδο. Η αρχιτεκτονική του παρουσιάζεται στο Σχήμα 4.3.




Σχήμα 4.3 Η Αρχιτεκτονική του Parafrase-2

**4.6. NANOS**

Ο μεταγλωττιστής NANOS ([16], [17], [18], [19]), που είναι βασισμένος στον Parafrase-2, προσπαθεί να επωφεληθεί από τον παραλληλισμό πολλών επιπέδων, συμπεριλαμβάνοντας τον χοντρικό παραλληλισμό χρησιμοποιώντας το εκτεταμένο API του OpenMP και τη γλώσσα Fortran 77. Ο μεταγλωττιστής αυτός είναι μέρος ενός ευρύτερου συστήματος που περιλαμβάνει:

1. ένα περιβάλλον ανάπτυξης εφαρμογών που αποτελείται από ένα εργαλείο απεικόνισης και μετασχηματισμού δομών εφαρμογών
2. ένας εκτεταμένος μεταγλωττιστής παραλληλισμού OpenMP
3. μια βιβλιοθήκη νημάτων χρόνου εκτέλεσης επιπέδου χρήστη
4. ένα εργαλείο απεικόνισης και ανάλυσης της απόδοσης μιας εφαρμογής
5. έναν διαχειριστή του επεξεργαστή
6. ένα εργαλείο απεικόνισης της δραστηριότητας του συστήματος.

Εκμεταλλεύεται τον παραλληλισμό που ορίζεται από τον χρήστη μέσω των ρυθμιστικών εντολών (directives) του OpenMP, καθώς και ανάλυση εξαρτήσεων δεδομένων και ελέγχου, ώστε να εξάγει αυτόματα παραλληλισμό. Ο μεταγλωττιστής είναι επίσης υπεύθυνος για την ανάθεση των εργασιών σε νήματα, τον ορισμό προτεραιοτήτων στην εκτέλεση και την επιλογή των μηχανισμών για την παράλληλη εκτέλεση τους. Ο κώδικας που δημιουργείται από τον μεταγλωττιστή είναι κώδικας σε C ή Fortran και περιέχει κλήσεις στη βιβλιοθήκη νημάτων NthLib, αξιοποιώντας τις πληροφορίες που αντλεί από το HTG (Γράφημα Ιεραρχικών Εργασιών).

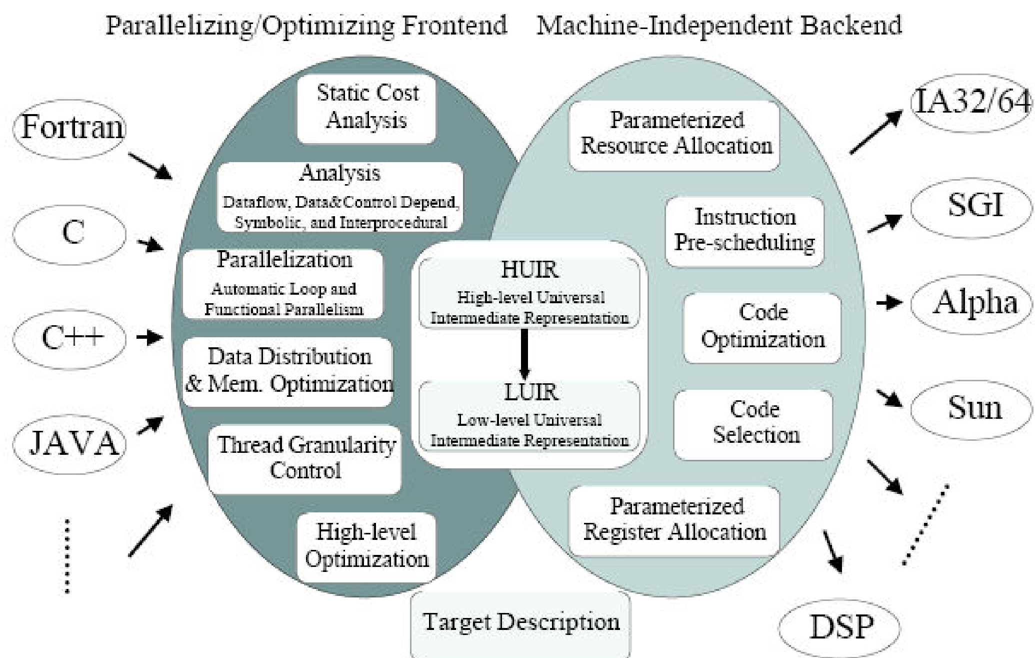
Το περιβάλλον NANOS περιλαμβάνει τεχνικές μεταγλώττισης, σύστημα εκτέλεσης και λειτουργικό σύστημα για την επίτευξη καλής απόδοσης για τις παράλληλες εφαρμογές σε συστήματα πολύ-επεξεργαστών με διαμοιραζόμενη μνήμη και δεν υποστηρίζει την ύπαρξη μνήμης συγχρονισμού μεταξύ των νημάτων.

## 4.7. PROMIS

Ο PROMIS ([14]) ιεραρχικά συνδυάζει τον μεταγλωττιστή Parafrase-2, χρησιμοποιώντας τεχνικές HTG (Γράφημα Ιεραρχικών Εργασιών) και συμβολικής ανάλυσης, και τον μεταγλωττιστή EVE για εκτεταμένη μετατροπή του κώδικα σε παράλληλο. Υποστηρίζει τη μετατροπή κώδικα από διάφορες γλώσσες, συμπεριλαμβανομένων της Fortran, της C και της Java, σε παράλληλο για κάποιες μηχανές-στόχους, όπως RISC, CISC και DSP. Περιλαμβάνει έναν αριθμό από αναλύσεις και βελτιστοποιήσεις του κώδικα, οι οποίος εξαρτώνται από την αρχιτεκτονική της μηχανής-στόχου. Η δομή του PROMIS παρουσιάζεται στο Σχήμα 4.4.

Ο PROMIS μετατρέπει την είσοδο του σε μια κοινή ενδιάμεση αναπαράσταση. Αυτή η ενδιάμεση αναπαράσταση περιλαμβάνει ένα υποσύνολο της ένωσης των γλωσσών C, Fortran και Java. Οι πληροφορίες προωθούνται από το frontend στο backend μέσω της ενδιάμεσης αναπαράστασης χωρίς να χάνεται η σημασιολογία.

Οι αναλύσεις και οι βελτιστοποιήσεις που περιλαμβάνει ο PROMIS δεν περιλαμβάνουν το μοντέλο μας, δηλαδή τη διαμοιραζόμενη μνήμη σε συνδυασμό με τη μνήμη συγχρονισμού για την επικοινωνία των νημάτων. Παράγει κώδικα μηχανής για κάποια από τις αρχιτεκτονικές που υποστηρίζει και όχι κώδικα σε μια ενδιάμεση αναπαράσταση υψηλού επιπέδου, όπως η μTC.



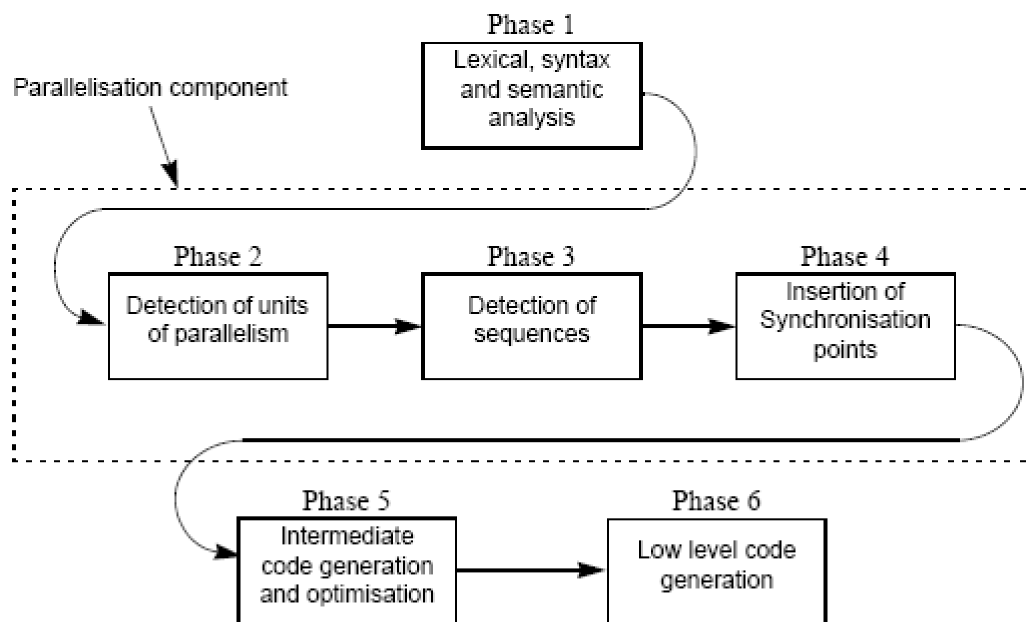

Σχήμα 4.4 Περιγραφή του PROMIS

## 4.8. RHODOS

Το RHODOS ([11], [12]) περιλαμβάνει ένα κατανεμημένο λειτουργικό σύστημα με έλεγχο παράλληλων διεργασιών, μαζί με ένα σύνολο κλήσεων βιβλιοθηκών παράλληλης εκτέλεσης, που μπορούν να χρησιμοποιηθούν στον χειρισμό των διάφορων συστατικών των παράλληλων προγραμμάτων. Πάνω σε αυτό το σύστημα έχει σχεδιαστεί έναν απλός μεταγλωττιστής ο οποίος μετατρέπει βρόχους σε παράλληλους.

Ο μεταγλωττιστής αυτός είναι ικανός να παραλληλοποιήσει χοντρικά ένα σειριακό πρόγραμμα σε επίπεδο διαδικασιών. Ο παραλληλισμός υλοποιείται με τη εξέταση των μεταβλητών μέσα στο πρόγραμμα. Ανάλογα με την χρήση μιας μεταβλητής (αν

δηλαδή διαβάζεται ή γράφεται) γίνεται και ο παραλληλισμός του κομματιού κώδικα που εξετάζεται. Ο μεταγλωττιστής αυτός είναι ανεξάρτητος από το περιβάλλον εκτέλεσης του, αλλά η διαδικασία παραλληλισμού είναι πιο αποδοτική σε ένα περιβάλλον όπως το RHODOS. Η γλώσσα η οποία παίρνει ως είσοδο είναι η Pascal και η έξοδος του είναι ένα παράλληλο πρόγραμμα σε C, που περιέχει τις κατάλληλες κλήσεις του συστήματος RHODOS για παραλληλοποίηση. Στο Σχήμα 4.5 περιγράφονται οι φάσεις που περιλαμβάνει ο μεταγλωττιστής ώστε να παραλληλοποιήσει τον κώδικα που παίρνει ως είσοδο.




Σχήμα 4.5 Οι 6 Φάσεις του Απλού Μεταγλωττιστή Παραλληλοποίησης.

Η παραλληλοποίηση των βρόχων γίνεται με την εξέταση του σώματος του βρόχου για να δούμε αν αυτό εξαρτάται από τον δείκτη του βρόχου. Η πολυπλοκότητα της εξάρτησης αυτής καθορίζει τον τρόπο με τον οποίο θα γίνει ο χωρισμός. Ο μεταγλωττιστής αυτός υποστηρίζει μόνο απλούς ορισμένους μονοδιάστατους βρόχους, οι οποίοι είναι γραμμικοί, με απλούς δείκτες και με καλά ορισμένη συνθήκη εξόδου. Αν υπάρχει πίνακας μέσα στο σώμα του βρόχου, αυτός θα πρέπει να είναι μιας διάστασης.

Αν η πρόσβαση στον δείκτη του βρόχου είναι σύνθετη τότε ο βρόχος δεν παραλληλοποιείται. Η πολυπλοκότητα της πρόσβασης ορίζεται από τον τρόπο

πρόσβασης του δείκτη. Αν απλώς διαβάζεται η τιμή του μέσα στο σώμα του βρόχου η πρόσβαση θεωρείται απλή, αλλιώς θεωρείται σύνθετη. Στο Σχήμα 4.6 παρουσιάζεται ο αλγόριθμος με τον οποίο γίνεται η παραλληλοποίηση του κώδικα.

```
/* blocks are identified */
…
while blocks exist
do
   while statements in block
   do
        /* examine block */
        check each statement within the loop block
        if there is a dependency amongst each statement
             - on any of the indices of the loop
             - on any variable that is not an index of the loop
        mark each statement affected
   end
   while marked statements
   do
        /* examine each statement references */
        if the statement is reading index then omit this block
             as parallelisable[1]
        else
             insert tags for insertion of primitives in final code
                 generation phase
   end
end
```

1. Note that this could be further parallelised with loop parallelisation methods that this simple parallelising compiler is not yet capable of.

Σχήμα 4.6 Αλγόριθμος Παραλληλοποίησης Βρόχων

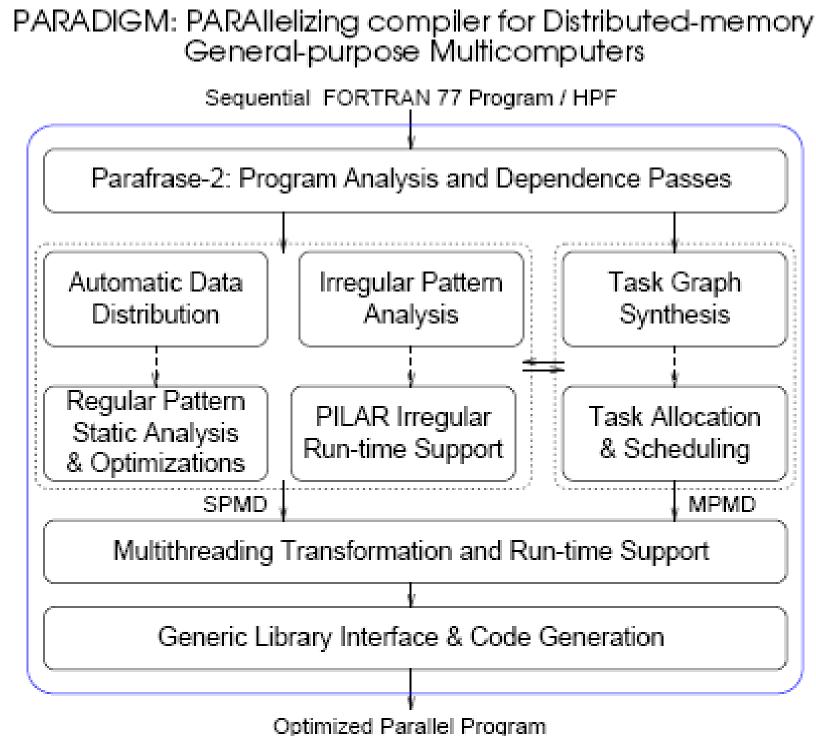
Ο μεταγλωττιστής αυτός παίρνει ως είσοδο ένα πρόγραμμα στη γλώσσα Pascal, η οποία, όπως και η Fortran, έχουν καλά δομημένους βρόχους, σε αντίθεση με τη C. Η έξοδος του είναι ένα πρόγραμμα σε C που περιλαμβάνει κλήσεις του συστήματος RHODOS, έτσι είναι εστιασμένος στις ανάγκες και τις απαιτήσεις του συγκεκριμένου συστήματος και οι επιδόσεις του έξω από αυτό δεν είναι τόσο καλές. Απευθύνεται στο μοντέλο κατανεμημένης μνήμης και δεν υποστηρίζει την ύπαρξη οικογενειών νημάτων και μνήμης συγχρονισμού για την επικοινωνία μεταξύ νημάτων της ίδιας οικογένειας.

**4.9. PARADIGM**

Ο μεταγλωττιστής PARADIGM (PARAllelizing compiler for Distributed memory General-purpose Multicomputers) ([9]) είναι ένα αυτοματοποιημένο μέσο για την παραλληλοποίηση και βελτιστοποίηση σειριακών προγραμμάτων, ώστε να εκτελούνται ικανοποιητικά σε πολύ-υπολογιστές κατανεμημένης μνήμης.

Όπως και άλλοι μεταγλωττιστές, εκτελεί τις παραδοσιακές βελτιστοποιήσεις στην κατανομή των υπολογισμών και στην μείωση την επιβάρυνσης της επικοινωνίας μεταξύ των επεξεργαστών. Η διαφορά του από άλλους παρόμοιους μεταγλωττιστές είναι ότι στα πλαίσια μιας ενοποιημένης πλατφόρμας περιλαμβάνει αυτόματη κατανομή των δεδομένων για τους κανονικούς υπολογισμούς, βελτιστοποιήσεις στον τρόπο επικοινωνίας των επεξεργαστών για τους κανονικούς υπολογισμούς, υποστήριξη μη κανονικών υπολογισμών με έναν συνδυασμό ανάλυσης κατά τον χρόνο μεταγλώττισης και υποστήριξης κατά τον χρόνο εκτέλεσης, εκμετάλλευση του παραλληλισμού λειτουργιών και δεδομένων συγχρόνως και πολύ-νηματική εκτέλεση.

Η είσοδος του PARADIGM είναι είτε FORTRAN 77 είτε HPF (High Performance Fortran) και η έξοδος του είναι ένα παράλληλο πρόγραμμα, βελτιστοποιημένο για το μηχάνημα στόχο. Στο Σχήμα 4.7 περιγράφεται η δομή του μεταγλωττιστή PARADIGM. Για την ανάλυση του προγράμματος στο πρώτο στάδιο χρησιμοποιείται ο Parafrase-2, για να μετατραπεί ο κώδικας σε μια ενδιάμεση αναπαράσταση και να εφαρμοστούν κάποιοι μετασχηματισμοί.




Σχήμα 4.7 Η Δομή του PARADIGM

Απευθύνεται σε συστήματα που ακολουθούν το μοντέλο κατανεμημένης μνήμης και όχι το μοντέλο μας, που περιλαμβάνει ομαδοποίηση των νημάτων σε οικογένειες, ύπαρξη διαμοιραζόμενης μνήμης, αλλά και μνήμης συγχρονισμού για την επικοινωνία μεταξύ των νημάτων που ανήκουν στην ίδια οικογένεια. Η είσοδος του είναι σε Fortran ή HPF και όχι σε C, η οποία δεν έχει αυστηρά δομημένους βρόχους.

### 4.10. VFC

Ο VFC (Vienna Fortran Compiler) ([8]) είναι ένα σύστημα παραλληλοποίησης που δέχεται ως είσοδο προγράμματα σε Fortran 95 με την επέκταση HPF+, μια βελτιωμένη έκδοση της HPF, η οποία εκτός από μηχανισμούς εκτεταμένης κατανομής δεδομένων και εργασίας, παρέχει στον προγραμματιστή στοιχεία για τον ορισμό συγκεκριμένων πληροφοριών που επηρεάζουν αποφασιστικά την απόδοση ενός προγράμματος, ώστε να συγκρίνεται με αυτή ενός προγράμματος γραμμένου από έναν προγραμματιστή, δηλαδή πρόκειται για ένα ημιαυτόματο σύστημα, σε αντίθεση με το δικό μας.

Η έξοδος του VFC είναι προγράμματα γραμμένα σε Fortran 90 και MPI για πέρασμα μηνυμάτων. Εκτός από την κατανομή βασικού block και την κυκλική κατανομή δεδομένων που προσφέρονται και από άλλους μεταγλωττιστές για HPF, ο VFC προσφέρει νέους τύπους κατανομών δεδομένων για μη-ομαλούς κώδικες, οι οποίες υλοποιούν πλήρως την δυναμική ανακατανομή των δεδομένων.

Ο VFC παρέχει στρατηγικές παραλληλοποίησης για ένα μεγάλο αριθμό μη-τέλεια εμφωλευμένων βρόχων με ανώμαλο τρόπο πρόσβασης που καθορίζεται κατά τον χρόνο εκτέλεσης. Η λειτουργία του είναι η μετατροπή της εισόδου σε ένα πρόγραμμα με πέρασμα μηνυμάτων, το οποίο είναι παραμετροποιημένο έτσι ώστε να μπορεί να εκτελεστεί σε έναν αυθαίρετο αριθμό επεξεργαστών. Σε αντίθεση, το δικό μας σύστημα υποστηρίζει το μοντέλο διαμοιραζόμενης μνήμης για επικοινωνία μεταξύ των οικογενειών νημάτων και την ύπαρξη μνήμης συγχρονισμού για την επικοινωνία μεταξύ νημάτων που ανήκουν στην ίδια οικογένεια.

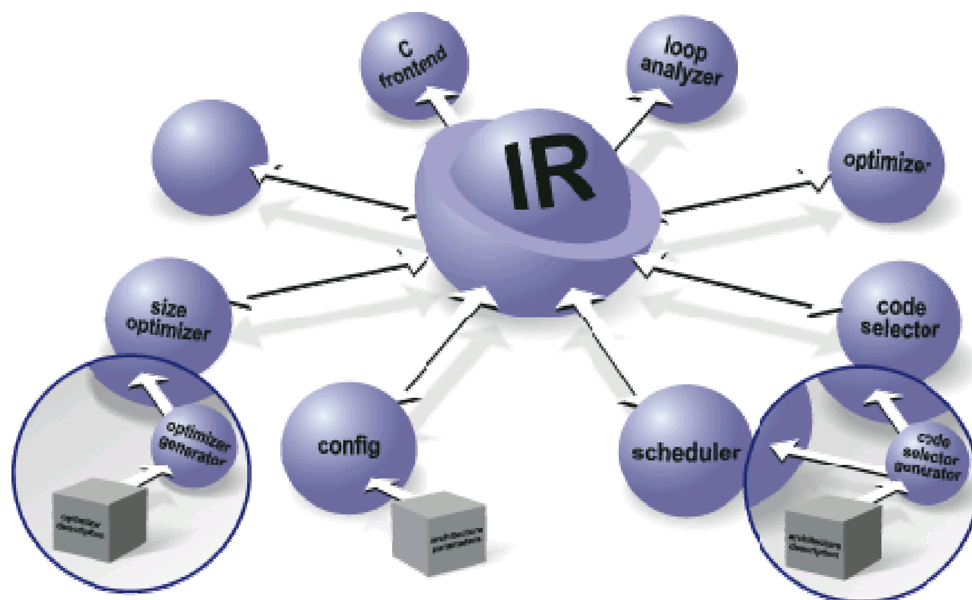
Κατά τον χρόνο εκτέλεσης της εξόδου του VFC, κάθε μηχάνημα εκτελεί τον ίδιο κώδικα αλλά με διαφορετικά δεδομένα, ενώ στο δικό μας μοντέλο κάθε νήμα εκτελεί τον ίδιο κώδικα, με διαφορετική τιμή δείκτη και με τα κατάλληλα δεδομένα που λαμβάνονται από την διαμοιραζόμενη μνήμη. Οι αλλαγές σε αυτά τα δεδομένα περνάνε από το ένα νήμα στο άλλο και μετά την ολοκλήρωση της εκτέλεσης της οικογένειας νημάτων γίνεται η ενημέρωση της διαμοιραζόμενης μνήμης.

### 4.11. CoSy

Το CoSy ([23]) είναι ένα εμπορικό εργαλείο που έχει υλοποιηθεί από την εταιρία ACE για τη δημιουργία ολοκληρωμένων μεταγλωττιστών. Οι μεταγλωττιστές αυτοί κατασκευάζονται με τις ίδιες μεθόδους και είναι μοναδικοί για την προσαρμοστικότητά τους στις απαιτήσεις και τις καινοτόμες τεχνικές που χρησιμοποιούνται για τη δημιουργία τους. Το CoSy δημιουργεί στρατηγικές βελτιστοποίησης από τις περιγραφές των χαρακτηριστικών, του παραλληλισμού και του χρονισμού μιας αρχιτεκτονικής. Και καθώς η προσοχή στην κατασκευή μεταγλωττιστών έχει μεταφερθεί στη δημιουργία της περιγραφής του επεξεργαστή, οι

μεταγλωττιστές μπορούν να είναι διαθέσιμοι μόλις ολοκληρωθεί η περιγραφή της αρχιτεκτονικής.

Οι μεταγλωττιστές που κατασκευάζονται με το σύστημα CoSy έχουν υψηλή ποιότητα και επιδόσεις. Για την επίτευξη αυτών των επιδόσεων, τα front-ends, οι αλγόριθμοι ανάλυσης, οι αλγόριθμοι βελτιστοποίησης και οι μέθοδοι ανάθεσης καταχωρητών είναι όλες αυτό-οριζόμενες «μηχανές», οι οποίες εκτελούν τις λειτουργίες τους στο IR σε συνεργασία με όλες τις άλλες «μηχανές που έχουν οριστεί στο μεταγλωττιστή.




Σχήμα 4.8 Οι λειτουργίες του CoSy

Ο γενικός σχεδιασμός του CoSy το κάνει ιδανικό για τη δημιουργία μεταγλωττιστών για κάθε είδος αρχιτεκτονικής επεξεργαστών, όπως 8/16/32/64-bit CISC, RISC, DSP, NoC και VLIW επεξεργαστές. Ειδικές «μηχανές» ανάλυσης του CoSy φροντίζουν ότι ο παραλληλισμός σε επίπεδο εντολών που εμπεριέχεται στις DSP και VLIW αρχιτεκτονικές λαμβάνεται πλήρως υπ’ όψιν. Τα front-ends τα οποία υπάρχουν διαθέσιμα αυτή τη στιγμή είναι για τις γλώσσες ISO C, K&R C, Fortran 77, Fortran 90, HPF, και Modula-2, καθώς και η επέκταση της ISO C για DSP (Digital Signal Processors).

Η τεχνική που χρησιμοποιεί το CoSy για την ανάλυση βρόχων είναι η σήμανση των γνωρισμάτων τους. Τα σημάδια που χρησιμοποιούνται ονομάζονται *loop markers*. Με τη χρήση αυτής της σήμανσης διευκολύνεται η συγγραφή «μηχανών» για τη βελτιστοποίηση των βρόχων. Δημιουργείται από τα front-ends και τις «μηχανές» ανάλυσης βρόχων, και χρησιμοποιείται από τις «μηχανές» βελτιστοποίησης αυτών, καθώς και από κάθε «μηχανή» που εξαρτάται από τους βρόχους, όπως ο γεννήτορας κώδικα. Περιλαμβάνει όλες τις πληροφορίες που περιλαμβάνονται στο γράφημα ελέγχου ροής (cfg), κάποιες από τις οποίες είναι περιττές. Η ιδέα είναι να διατηρηθεί η σημασιολογία των βρόχων που υπάρχει στο front end αλλά χάνεται κατά τη μεταγλώττιση τους στην ενδιάμεση αναπαράσταση του CoSy (CCMIR), όπως οι εξαρτήσεις μεταξύ διαδοχικών επαναλήψεων των βρόχων, οι επαγωγικές μεταβλητές, και άλλα. Η διατήρηση αυτών των επιπλέον πληροφοριών κάνουν τα loop markers χρήσιμα. Τα loop markers που παράγονται από προγράμματα σε Fortran διατηρούν αρκετές πληροφορίες. Δεν υπάρχουν όμως σχέδια για την αναβάθμιση του front-end της C ώστε να παράγονται loop markers. Η γλώσσα C δεν έχει αυστηρά ορισμένες δομές βρόχων που να επιτρέπουν απευθείας δημιουργία των loop markers, όπως έχει η Fortran. Αντί για αυτό, ο αναλυτής βρόχων πρέπει να τρέχει πίσω από το front-end, χωρίς να είναι διαθέσιμες οι πληροφορίες των loop markers.

Το εργαλείο COSY είναι ένα πολύ χρήσιμο εργαλείο για τη δημιουργία μεταγλωττιστών. Στην περίπτωση όμως της μετατροπής του κώδικα από C σε μTC δεν είναι το κατάλληλο εργαλείο. Το COSY, αν και παρέχει ανάλυση βρόχων, δεν επαρκεί για τη μετατροπή της C σε μTC. Αρχικά, δεν έχει τη δυνατότητα να παράγει loop markers για τη γλώσσα C, η οποία δεν έχει αυστηρά ορισμένες δομές βρόχων. Έτσι χάνεται ένα μέρος των πληροφοριών κατά τη μετατροπή του κώδικα σε C στην ενδιάμεση αναπαράσταση του COSY. Ακόμα όμως και να μπορούσε να τα παράγει, οι πληροφορίες που δίνουν δεν είναι επαρκείς για να μπορέσει να γίνει η μεταγλώττιση.

Μια βασική πληροφορία για τη μετατροπή είναι ο ρόλος της κάθε μεταβλητής στο πρόγραμμα, αν δηλαδή εγγράφεται, διαβάζεται ή και τα δυο μέσα σε έναν βρόχο. Έτσι μπορούμε να καθορίσουμε ποιες μεταβλητές ορίζονται ως διαμοιραζόμενες, δηλαδή οι τιμές που παίρνουν θα προωθούνται από το ένα νήμα στο άλλο μέσο της

μνήμης συγχρονισμού, και ποιες πρέπει να πάρουν τιμές από καθολικές μεταβλητές ή πίνακες που βρίσκονται αποθηκευμένοι στη διαμοιραζόμενη μνήμη. Αυτή η πληροφορία όμως δεν είναι διαθέσιμη από τους loop markers του COSY.

Αν και, με τους loop markers, παρέχει κάποιες λειτουργίες για την εύρεση του δείκτη ενός βρόχου και των ορίων του, αυτές είναι ακόμα υπό δοκιμή και υπόκεινται σε αλλαγές, ακόμα και για τη γλώσσα Fortran που έχει καλά ορισμένους βρόχους, το οποίο κάνει την ανάλυση τους πιο εύκολη.

# ΚΕΦΑΛΑΙΟ 5. ΣΧΕΔΙΑΣΗ

5.1. Σχεδίαση
5.2. Χρήσιμοι ορισμοί
5.3. Καθορισμός του δείκτη ενός βρόχου
5.4. Μετασχηματισμοί απλών βρόχων
5.5. Μετασχηματισμός βρόχων που περιέχουν περισσότερες εντολές
5.6. Αλγόριθμος μετασχηματισμού μονοδιάστατων βρόχων
5.7. Κανονικοποίηση βρόχων
5.8. Μετασχηματισμός εμφωλευμένων βρόχων

## 5.1. Σχεδίαση

Υπάρχουν κάποια προβλήματα τα οποία πρέπει να μελετηθούν κατά το μετασχηματισμό των βρόχων μιας υψηλού επιπέδου γλώσσας σε μTC, κάποια από τα οποία είναι ακόμα ανοιχτά. Αυτά είναι ο καθορισμός του δείκτη ενός βρόχου, καθώς και των ορίων και του βήματος του. Το πρόβλημα του καθορισμού του δείκτη ενός βρόχου σε μερικές γλώσσες προγραμματισμού, συμπεριλαμβανομένης της C, υποπτευόμαστε ότι ανήκει στην κλάση των μη-αποφασίσιμων (undecidable) προβλημάτων. Κάποιοι βρόχοι έχουν περισσότερους από έναν δείκτες και κάποιοι δεν έχουν κανένα δείκτη. Παρακάτω θα εξετάσουμε μονοδιάστατους βρόχους, ως προς τα είδη των εξαρτήσεων που περιέχονται στο σώμα τους.

Ο κώδικας σε μTC που προκύπτει από τη μετατροπή του κώδικα σε C προκύπτει με αυτόματο τρόπο. Αν ο ίδιος κώδικας γραφτεί από έναν προγραμματιστή σίγουρα θα είναι πιο αποδοτικός και θα έχει μεγαλύτερο βαθμό παραλληλίας, καθώς θα μπορούν να αφαιρεθούν κάποιες εντολές που είναι περιττές. Ο στόχος μας όμως είναι να

μετατρέψουμε τον κώδικα αυτόματα. Παρατηρούμε ότι ενώ φαίνεται να υπάρχουν διαφορετικοί τύποι βρόχων, τελικά μπορούμε να τους χειριστούμε όλους με τον ίδιο τρόπο.

Μια οικογένεια νημάτων παρομοιάζεται με ένα «κανάλι» (pipeline) εκτέλεσης. Κάθε νήμα είναι ένα στοιχείο επεξεργασίας και οι διαμοιραζόμενες μεταβλητές είναι τα κανάλια που μεταφέρουν τα δεδομένα μεταξύ των διαδοχικών στοιχείων επεξεργασίας. Οι διαμοιραζόμενες μεταβλητές είναι ο τρόπος να επικοινωνούν διαδοχικά νήματα μεταξύ τους. Κατά τη διάρκεια της εκτέλεσης μιας οικογένειας νημάτων και μέχρι τη στιγμή που αυτή ολοκληρώνεται δεν υπάρχει εγγύηση για το περιεχόμενο της ασύγχρονης διαμοιραζόμενης μνήμης, αλλά αυτή μπορεί να υπάρξει μόνο μετά τον συγχρονισμό της οικογένειας νημάτων.

Οι καθολικές μεταβλητές βρίσκονται αποθηκευμένες στην ασύγχρονη διαμοιραζόμενη μνήμη και από αυτές αρχικοποιούνται οι διαμοιραζόμενες μεταβλητές, οι οποίες βρίσκονται στη μνήμη συγχρονισμού. Κατά το συγχρονισμό της οικογένειας νημάτων, μετά την ολοκλήρωση της εκτέλεσης της, οι τιμές των καθολικών μεταβλητών ανανεώνονται, παίρνοντας τις τιμές των διαμοιραζόμενων μεταβλητών που αρχικοποιήθηκαν αντίστοιχα από αυτές.

### 5.2. Χρήσιμοι ορισμοί [21]

Μια εντολή $S_2$ έχει εξάρτηση ροής (flow-dependency) ή πιο απλά εξαρτάται από μια εντολή $S_1$, όταν ανατίθεται μια τιμή σε μια μεταβλητή στην εντολή $S_1$ και η ίδια μεταβλητή αναφέρεται ξανά στην εντολή $S_2$ και δεν υπάρχει άλλη εντολή ανάμεσα στην $S_1$ και στην $S_2$ που αναθέτει μια άλλη τιμή στη μεταβλητή αυτή.

Μια εντολή $S_2$ έχει αντί-εξάρτηση (anti-dependency) σε μια εντολή $S_1$, όταν μια μεταβλητή αναφέρεται στην εντολή $S_1$ και μια νέα τιμή ανατίθεται στην ίδια μεταβλητή στην εντολή $S_2$ και δεν υπάρχει άλλη εντολή ανάμεσα στην $S_1$ και στην $S_2$ που αναθέτει μια άλλη τιμή στη μεταβλητή αυτή.

Μόνο οι εξαρτήσεις ροής είναι πραγματικές εξαρτήσεις. Οι αντί-εξαρτήσεις συνήθως δημιουργούνται από τους προγραμματιστές, στην προσπάθεια τους να γράψουν πιο αποδοτικό κώδικα και να χρησιμοποιήσουν λιγότερη μνήμη. Σε ένα περιβάλλον παράλληλης επεξεργασίας οι αντί-εξαρτήσεις μπορούν να προκαλέσουν αδιέξοδα και καθυστερήσεις. Ευτυχώς μπορούμε πάντα να περιορίσουμε τις αντί-εξαρτήσεις με κάποιο μετασχηματισμό του κώδικα.

Οι εξαρτήσεις που περιγράψαμε είναι μεταξύ των εντολών στην ίδια επανάληψη ενός βρόχου (loop independent dependencies). Υπάρχει και ένα άλλο είδος εξαρτήσεων, οι εξαρτήσεις μεταξύ των εντολών σε διαφορετικές επαναλήψεις ενός βρόχου (loop carried dependencies). Όταν σε μια επανάληψη ανατίθεται μια τιμή σε μια μεταβλητή και στην επόμενη επανάληψη αναφέρεται αυτή η μεταβλητή, τότε έχουμε μια τέτοια εξάρτηση. Ο τρόπος να χειριστούμε μια τέτοιου είδους εξάρτηση είναι οι διαμοιραζόμενες μεταβλητές. Αφού κάθε νήμα αντιπροσωπεύει μια επανάληψη ενός βρόχου, ο τρόπος να περνάνε οι πληροφορίες από το ένα νήμα στο άλλο είναι οι διαμοιραζόμενες μεταβλητές.

## 5.3. Καθορισμός του δείκτη ενός βρόχου

Στην περίπτωση που ο βρόχος έχει την καθορισμένη μορφή:

```
for(index=a1; index<a2; index+=a3)
{ … }
```

Τα $a_1$, $a_2$ και $a_3$ είναι σταθερές για το βρόχο, δε μεταβάλλονται δηλαδή οι τιμές τους μέσα στο βρόχο, όπως επίσης και το index. Σε αυτή την περίπτωση έχουμε μόνο μια μεταβλητή υποψήφια να είναι ο δείκτης του βρόχου και αυτή είναι το index.

Αντίστοιχα στην περίπτωση που έχουμε ένα βρόχο της μορφής:

```
while(index<a1)
{   …
    index++;
}
```

Για να θεωρηθεί μια μεταβλητή δείκτης ενός βρόχου θα πρέπει να ισχύουν οι παρακάτω συνθήκες:

1. Ο δείκτης είναι μια από τις μεταβλητές που εγγράφονται μέσα στο σώμα του βρόχου, είτε στην εντολή που καθορίζει το βήμα του βρόχου
2. Τουλάχιστον μια από τις μεταβλητές της συνθήκης εξόδου εξαρτάται από τον δείκτη, άμεσα ή έμμεσα
3. Υπάρχει μόνο μια μεταβλητή που ικανοποιεί τις παραπάνω συνθήκες.

Αν υπάρχουν περισσότερες από μια μεταβλητές που ικανοποιούν αυτές τις συνθήκες, ή δεν υπάρχει καμιά μεταβλητή που να ικανοποιεί κάποια από αυτές, τότε δεν μπορεί να καθοριστεί δείκτης για αυτόν το βρόχο, ακόμα και αν αυτός υπάρχει.

Η C επιτρέπει στους βρόχους να ορίζονται χωρίς να είναι αυστηρά ορισμένος ο δείκτης τους. Το παρακάτω παράδειγμα είναι κώδικας που επιτρέπεται από τη C:

```
for(i<10;k<10;x<10)
{     printf("%d %d %d\n",i,k,x);
      k++;
}
```

Ο δείκτης αυτού του βρόχου είναι η μεταβλητή k. Είναι η μοναδική μεταβλητή που εγγράφεται μέσα στο σώμα του βρόχου και η συνθήκη εξόδου από το βρόχο εξαρτάται από αυτή.

Αντίθετα, στον κώδικα που θα δούμε παρακάτω περισσότερες από μια μεταβλητές ικανοποιούν τις δυο πρώτες συνθήκες έτσι δε μπορούμε να καθορίσουμε τον δείκτη του βρόχου:

```
for(i<10;j+k<10;x<10)
{     printf("%d %d %d %d\n",i,j,k,x);
      k++;
      j++;
}
```

Ακόμα μπορεί ένας βρόχος να μην έχει καθόλου δείκτη, όπως ο παρακάτω:

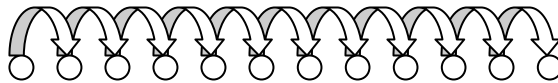
```
for(i<10;k<10;x<10)
```

```
{       printf("%d %d %d\n",i,k,x);
}
```

### 5.4. Μετασχηματισμοί απλών βρόχων με καθορισμένο δείκτη ([21], [22])

Αρχικά θα μελετήσουμε την περίπτωση όπου το σώμα του βρόχου περιέχει μόνο μια απλή ανάθεση και ο δείκτης μπορεί να καθοριστεί σύμφωνα με τους κανόνες που περιγράψαμε παραπάνω. Αντίστοιχη είναι η μετατροπή και στην περίπτωση του `while`. Στην πιο απλή περίπτωση, που παρουσιάζεται στο Σχήμα 5.1, υπάρχει μόνο μια μοναδιαία εξάρτηση μεταξύ των στοιχείων ενός πίνακα. Αυτός ο τύπος εξάρτησης υποστηρίζεται απευθείας από το μοντέλο μας. Ο μετασχηματισμός γίνεται απευθείας όπως φαίνεται παρακάτω:

```
for(i=1;i<N;i++)  →          int g_1=w[0];
     w[i]=w[i-1];            family fid0;
                             create(fid0;1;N-1;1)
                             {   index i;
                                 shared int s_1=g_1;
                                 int t_1;
                                 int t0;

                                 t_1=s_1;

                                 t0=t_1;
                                 w[i]=t0;

                                 s_1=t0;
                             }
                             sync(fid0);
```



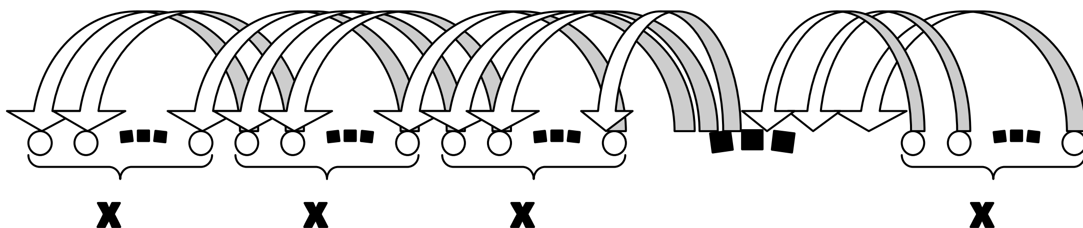
Σχήμα 5.1 Εξάρτηση με Απόσταση 1

Το στοιχείο i του πίνακα w εξαρτάται από το στοιχείο i-1 του ίδιου πίνακα. Το στοιχείο i-1 πρέπει να υπολογιστεί πριν από τον υπολογισμό του στοιχείου i. Η τιμή του στοιχείου i-1 θα περαστεί από το νήμα i-1 στο νήμα i μέσω της διαμοιραζόμενης μεταβλητής s_1. Η τιμή της s_1 αρχικοποιείται στην τιμή w[0], μέσω της καθολικής μεταβλητής g_1, στο νήμα με δείκτη 1. Σε κάθε άλλο νήμα η τιμή περνάει από το προηγούμενο νήμα μέσω της διαμοιραζόμενης μεταβλητής s_1. Κάθε νήμα έχει τις τοπικές μεταβλητές t_1 και t0, όπου αρχικά αποθηκεύονται οι τιμές των διαμοιραζόμενων μεταβλητών, στη συνέχεια γίνονται οι πράξεις και τέλος οι τιμές τους αποθηκεύονται πίσω στις διαμοιραζόμενες μεταβλητές για να περαστούν στο επόμενο νήμα. Η τιμή της t0 εγγράφεται στην θέση w[i].

Ας υποθέσουμε τώρα ότι έχουμε μια αντί-εξάρτηση, σαν αυτή του Σχήματος 5.2, όπου το στοιχείο i εξαρτάται από το στοιχείο i+x, όπου x>0 αφού έχουμε αντί-εξάρτηση. Το νήμα με δείκτη i+x μπορεί να εκτελεστεί μόνο αφού ολοκληρωθεί η εκτέλεση του νήματος i. Αυτός είναι ένας σημαντικός περιορισμός για την εκτέλεση του βρόχου. Αφού οι αντί-εξαρτήσεις δεν είναι πραγματικές εξαρτήσεις, μπορούμε να αλλάξουμε τον κώδικα, ώστε να μη περιέχει την αντί-εξάρτηση. Η λύση είναι να αντιγράψουμε τον πίνακα w σε έναν νέο προσωρινό πίνακα t και μετά να επιτρέψουμε σε όλα τα νήματα να τρέξουν ανεξάρτητα, χρησιμοποιώντας τον πίνακα t για ανάγνωση και τον πίνακα w για εγγραφή.

```
for(i=1;i<N-x;i++)    →        family fid0;
     w[i]=w[i+x];              family fid1;

                               create(fid0;1;N-x-1;1)
                               {    index i;
                                    t[i]=w[i+x];
```

```
}
sync(fid0);

create(fid1;1;N-x-1;1)
{     index i;
      int t0;
      t0=t[i];
      w[i]=t0;
}
sync(fid1);
```

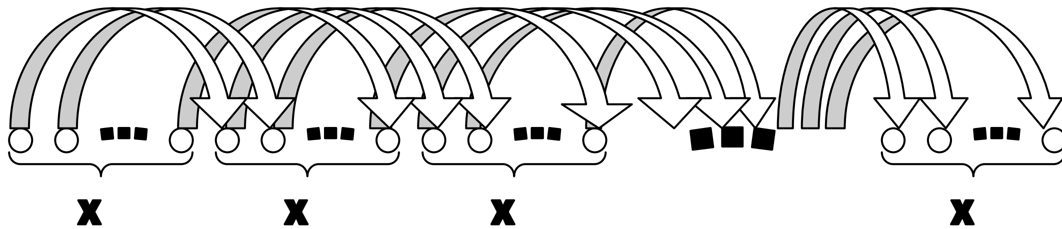



Σχήμα 5.2 Αντί-Εξάρτηση με Απόσταση x

Αν έχουμε μια εξάρτηση ροής, αλλά όχι μοναδιαία, τότε ο βρόχος μετασχηματίζεται με τον παρακάτω τρόπο, όπως στο Σχήμα 5.3. Υπάρχουν x διαμοιραζόμενες μεταβλητές, οι οποίες αρχικοποιούνται από το πρώτο νήμα, με τις πρώτες x τιμές του πίνακα w. Το πρώτο νήμα υπολογίζει την τιμή της θέσης w[x]. Μετά οι διαμοιραζόμενες μεταβλητές μετατοπίζονται κατά μια θέση, μέσω των τοπικών μεταβλητών, και η τιμή της διαμοιραζόμενης μεταβλητής s_1 γίνεται ίση με την τιμή t0, δηλαδή ίση με την τιμή w[i] που μόλις υπολογίστηκε. Κάθε νήμα i, όπου $i \neq x$, διαβάζει τις διαμοιραζόμενες μεταβλητές από το προηγούμενο νήμα, υπολογίζει το στοιχείο w[i] χρησιμοποιώντας τις διαμοιραζόμενες μεταβλητές, μέσω των τοπικών, και προετοιμάζει τις διαμοιραζόμενες μεταβλητές για το επόμενο νήμα, με τη μετατόπιση τους, και πάλι μέσω των τοπικών μεταβλητών.

```
for(i=x;i<N;i++)   →        int g_1=w[x-1];
     w[i]=w[i-x];            int g_2=w[x-2];
                             …
                             int g_x=w[0];

                             family fid0;
                             create(fid0;x;N-1;1)
                             {    index i;
                                  shared int s_1=g_1;
                                  shared int s_2=g_2;
                                  …
                                  shared int s_x=g_x;

                                  int t0;
                                  int t_1;
                                  int t_2;
                                  …
                                  int t_x;

                                  t_1=s_1;
                                  t_2=s_2;
                                  …
                                  t_x=s_x;

                                  t0=t_x;
                                  w[i]=t0;

                                  s_1=t0;
                                  s_2=t_1;
                                  …
                                  s_x=t_(x-1);
                             }
```

```
sync(fid0);
```




Σχήμα 5.3 Εξάρτηση με Απόσταση x

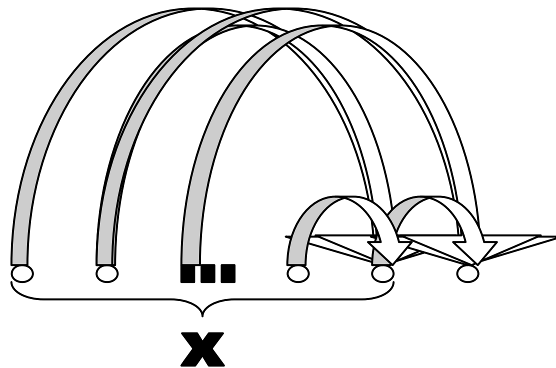
Ας υποθέσουμε τώρα ότι έχουμε περισσότερες από μια εξαρτήσεις σε μια εντολή, όπως φαίνεται στο Σχήμα 5.4. Υποθέτουμε ότι ισχύει x>0. Παρατηρούμε ότι η μόνη διαφορά από το προηγούμενο παράδειγμα είναι το σημείο όπου γίνεται η πρόσθεση των τοπικών μεταβλητών, όπου δεν έχουμε πια μόνο την ανάθεση του t_x στο t0, αλλά την ανάθεση του αθροίσματος των τοπικών μεταβλητών στο t0. Κατά τα άλλα πρέπει και πάλι να γίνει η αρχικοποίηση των διαμοιραζόμενων μεταβλητών, η αρχικοποίηση των τοπικών μεταβλητών, η εγγραφή του t0 στη θέση w[i] και οι εγγραφές των υπόλοιπων τοπικών μεταβλητών στις διαμοιραζόμενες, ώστε να γίνει η μετατόπιση τους και το πέρασμα τους στο επόμενο νήμα.

```
for(i=x;i<N;i++)
      w[i]=w[i-1]+w[i-2]+…+w[i-x];
```

→

```
int g_1=w[x-1];
int g_2=w[x-2];
…
int g_x=w[0];
family fid0;
create(fid0;x;N-1;1)
{     index i;
      shared int s_1=g_1;
```

```
        shared int s_2=g_2;
        …
        shared int s_x=g_x;

        int t0;
        int t_1;
        int t_2;
        …
        int t_x;

        t_1=s_1;
        t_2=s_2;
        …
        t_x=s_x;

        t0=t_1+t_2+…+t_x;
        w[i]=t0;

        s_1=t0;
        s_2=t_1;
        …
        s_x=t_(x-1);
    }
    sync(fid0);
```



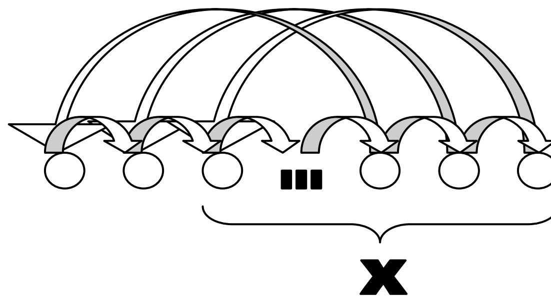

Σχήμα 5.4 Εξάρτηση από τα Προηγούμενα x-1 Στοιχεία του Πίνακα

Τέλος υπάρχει μια ακόμα περίπτωση, στην οποία υπάρχουν εξαρτήσεις ροής και αντί-εξαρτήσεις στην ίδια εντολή, όπως στο Σχήμα 5.5. Σε αυτή την περίπτωση αρχικά χειριζόμαστε την αντί-εξάρτηση και στη συνέχεια την εξάρτηση. Υποθέτουμε ότι ισχύει x>0.

```
for(i=1;i<N-x;i++)   →
w[i]=w[i-1]+w[i+x];
```

```
int g_1=w[0];
family fid0;
family fid1;

create(fid0;1;N-x-1;1)
{     index i;
      t[i]=w[i+x];
}
sync(fid0);

create(fid1;1;N-x-1;1)
{     index i;
       shared int s_1=g_1;

      int t_1;
      int t0;
```

```
            t_1=s_1;
            t0=t_1+t[i];
            w[i]=t0;

            s_1=t0;
        }
        sync(fid1);
```




Σχήμα 5.5 Εξάρτηση με Απόσταση 1 και Αντί-Εξάρτηση με απόσταση x

**5.5. Μετασχηματισμός βρόχων που περιέχουν περισσότερες εντολές και καθορισμένο βρόχο**

Ας μελετήσουμε τώρα την περίπτωση ένας βρόχος να περιλαμβάνει περισσότερες από μια εντολές, οι οποίες μπορούν να περιέχουν εξαρτήσεις και αντί-εξαρτήσεις. Και πάλι θα χειριστούμε πρώτα τις αντί-εξαρτήσεις, αντιγράφοντας τον πίνακα σε έναν προσωρινό, και μετά τις εξαρτήσεις. Θα ακολουθήσουμε τα ίδια βήματα με την περίπτωση που περιλαμβάνεται μόνο μια εντολή στο βρόχο.

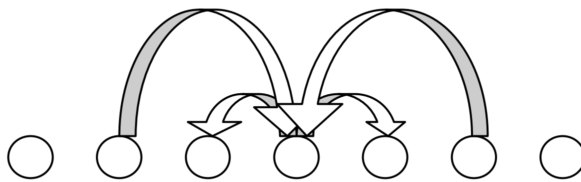
Στο πρώτο νήμα της οικογένειας οι διαμοιραζόμενες μεταβλητές αρχικοποιούνται μέσω κάποιων καθολικών μεταβλητών. Σε κάθε νήμα έχουμε κάποιες τοπικές μεταβλητές οι οποίες παίρνουν τις τιμές των διαμοιραζόμενων μεταβλητών που τους

αντιστοιχούν. Στη συνέχεια έχουμε το πρώτο σημείο όπου διαφέρει ο μετασχηματισμός ενός βρόχου με περισσότερες εντολές, από τον μετασχηματισμό ενός βρόχου με μια εντολή. Και πάλι γίνονται οι πράξεις μεταξύ των τοπικών μεταβλητών, όμως πλέον έχουμε περισσότερες από μια αναθέσεις. Το δεύτερο σημείο όπου διαφέρουν οι δυο περιπτώσεις είναι η εγγραφή στη θέση i του πίνακα. Στην περίπτωση που έχουμε περισσότερες από μια αναθέσεις μέσα στο βρόχο θα έχουμε και εγγραφή σε περισσότερες θέσεις του πίνακα. Επειδή μπορεί ένα νήμα να γράφει σε μια θέση μνήμης που διαβάζεται και ξαναγράφεται από άλλα νήματα, θα επιτρέψουμε σε κάθε νήμα να κάνει μόνο μια εγγραφή στη μνήμη, στη θέση i του πίνακα. Οι νέες τιμές που δεν εγγράφτηκαν στην μνήμη θα περάσουν στο επόμενο νήμα μέσω των διαμοιραζόμενων μεταβλητών, στις οποίες αποθηκεύονται οι τοπικές μεταβλητές μετατοπισμένες κατά ένα. Μετά το τέλος της εκτέλεσης του τελευταίου νήματος γίνεται η εγγραφή και σε άλλες θέσεις του πίνακα αν αυτό απαιτείται. Ένα παράδειγμα ενός βρόχου με περισσότερες εντολές παρουσιάζεται παρακάτω:

```
for(i=1;i<=9;i++)    →
{     w[i]=w[i+1];
      w[i+2]=w[i+1];
      w[i+1]=w[i-1]+w[i+3];
}
```

```
int g_1=w[0];
int g0=w[1];
int g1=w[2];

family fid0;
family fid1;

create(fid0;1;9;1)
{     index i;
      t[i]=w[i+3];
}
sync(fid0);

create(fid1;1;9;1)
{     index i;

      shared int s_1=g_1;
      shared int s0=g0;
```

```
        shared int s1=g1;

        int t_1;
        int t0;
        int t1;
        int t2;

        t_1=s_1;
        t0=s0;
        t1=s1;

        t0=t1;
        t2=t1;
        t1=t_1+t[i];

        w[i]=t0;

        s_1=t0;
        s0=t1;
        s1=t2;
    }
    sync(fid1);
```



Σχήμα 5.6 Εξαρτήσεις και αντί-Εξαρτήσεις Βρόχου με Περισσότερες Εντολές

**5.6. Μετασχηματισμοί απλών βρόχων χωρίς καθορισμένο δείκτη**

Όταν δεν υπάρχει καμιά μεταβλητή που να ικανοποιεί τις συνθήκες που αναφέραμε πριν για τον καθορισμό του δείκτη ή υπάρχουν περισσότερες από μια μεταβλητές που τις ικανοποιούν, τότε ο δείκτης του βρόχου δε μπορεί να καθοριστεί, όμως το μοντέλο SVP παρέχει τη δυνατότητα μετατροπής του βρόχου σε νήματα, ακόμα και όταν δε γνωρίζουμε τον αριθμό των νημάτων που πρέπει να δημιουργηθούν. Η μTC παρέχει τα μέσα για τη δημιουργία μιας οικογένειας νημάτων με «άπειρο» αριθμό νημάτων, η οποία τερματίζεται από μια δυναμική συνθήκη. Η λύση είναι η δημιουργία ενός αυθαίρετα μεγάλου αριθμού νημάτων που ανήκουν στην ίδια οικογένεια και η διακοπή της δημιουργίας περισσότερων νημάτων, μόλις τεθεί σε ισχύ μια συνθήκη, με την εντολή break.

Ας δούμε μερικά παραδείγματα μετατροπής βρόχων που δεν μπορεί να καθοριστεί ο δείκτης. Στο πρώτο παράδειγμα υπάρχουν δυο μεταβλητές που ικανοποιούν τις συνθήκες και θα μπορούσαν να είναι δείκτες, και το i και το j. Ο βρόχος μπορεί να υλοποιηθεί είτε με την εντολή for είτε με την εντολή while, και ο κώδικας σε μTC που θα δημιουργηθεί θα είναι ακριβώς ο ίδιος.

```
j=0;
for (i=1;i+j<10;i++)
{     a[i] = a[i-1];
      j++;
}
```

ή

```
j=0;
i=1;
while(i+j<10)
{       a[i] = a[i-1];
        j++;
        i++;
}
```

```
→     int g_1=A[0];
      j=0;
      i=1;
      create(fid;1;MAX_THREADS;1)
      {   index ind;
          shared int i=1;
          shared int j=0;
          shared int s_1=g_1;
          int t0;
```

```
        int t_1;

        if(!(i+j<10))
            break;

        t_1=s_1;

        t0=t_1;
        j++;

        a[i]=t0;

        s_1=t0;
        i++;

    }
    sync(fid);
```

Ο κώδικας που παράγεται είναι ίδιος με την περίπτωση που μπορούμε να καθορίσουμε τον δείκτη του βρόχου με τη διαφορά ότι αρχικά παράγουμε ένα μεγάλο αριθμό νημάτων και σε κάθε νήμα ελέγχουμε τη συνθήκη με την οποία θα τερματιζόταν ο βρόχος, και όταν ισχύει σταματάμε τη δημιουργία περισσότερων νημάτων και ολοκληρώνουμε την εκτέλεση και τον συγχρονισμό των νημάτων που έχουν ήδη δημιουργηθεί. Στην περίπτωση που γίνεται μετασχηματισμός ενός βρόχου for, προσθέτουμε στον κώδικα των νημάτων και την αύξηση του βήματος.

Ας δούμε άλλο ένα παράδειγμα:

```
while(a<b)        →            create(fid;1;MAX_THREADS;1)
{    a++;                       {    index ind;
     b--;                            shared int a;
}                                    shared int b;

                                     if(!(a<b))
```

```
        break;
    a++;
    b--;
}
sync(fid);
```

Τέλος:

```
j=0;
for(i=1;i+j<=9;i++)       →
{     w[i]=w[i+1];
      w[i+2]=w[i+1];
      w[i+1]=w[i-1]+w[i+3];
      j++;
}
```

```
i=1;
j=0;
int g_1=w[0];
int g0=w[1];
int g1=w[2];
family fid0;
family fid1;

create(fid0;1;MAX_THREAD;1)
{     index ind;
      shared int i;
      shared int j;

      if(!(i+j<=9))
         break;

      j++;
      i++;

      t[i]=w[i+3];


}
sync(fid0);

create(fid1;1;MAX_THREAD;1)
```

```
{     index ind;
      shared int i=1;
      shared int j=0;
      shared int s_1=g_1;
      shared int s0=g0;
      shared int s1=g1;

      int t_1;
      int t0;
      int t1;
      int t2;

      if(!(i+j<=9))
         break;

      t_1=s_1;
      t0=s0;
      t1=s1;

      t0=t1;
      t2=t1;
      t1=t_1+t[i];
      j++;

      w[i]=t0;

      s_1=t0;
      s0=t1;
      s1=t2;
      i++;
}
sync(fid1);
```

**5.7. Αλγόριθμος μετασχηματισμού μονοδιάστατων βρόχων**

Είδαμε ως τώρα μερικά παραδείγματα μετασχηματισμών, για διάφορους τύπους μονοδιάστατων βρόχων. Ουσιαστικά όλοι οι τύποι βρόχων μετασχηματίζονται σύμφωνα με τον ίδιο αλγόριθμο και για όλους τους τύπους ακολουθούνται οι ίδιοι κανόνες. Παρακάτω θα περιγράψουμε τα βήματα αυτού του αλγορίθμου.

Στο πρώτο βήμα ορίζουμε M καθολικές μεταβλητές, όπου M το πλήθος των θέσεων πίνακα που περιλαμβάνονται μεταξύ της ελάχιστης θέσης πίνακα που διαβάζεται ως τη μέγιστη θέση μνήμης που εγγράφεται μέσα στο σώμα του βρόχου, και τις αρχικοποιούμε στις θέσεις του πίνακα που τους αντιστοιχούν. Ορίζουμε τόσες μεταβλητές τύπου family όσες οι οικογένειες νημάτων που θα δημιουργήσουμε.

Στο δεύτερο βήμα δημιουργείται μια οικογένεια νημάτων για τον χειρισμό των αντί-εξαρτήσεων, εφόσον υπάρχουν. Στην οικογένεια αυτή αντιγράφεται ο πίνακας που έχει την αντί-εξάρτηση σε έναν προσωρινό πίνακα t. Ακολουθεί ο συγχρονισμός της οικογένειας αυτής.

Στο επόμενο βήμα δημιουργείται η οικογένεια νημάτων μέσα στην οποία εκτελούνται οι εντολές που περιλαμβάνονται στο σώμα του βρόχου. Αρχικά ορίζονται M διαμοιραζόμενες μεταβλητές, οι οποίες αρχικοποιούνται από τις αντίστοιχες καθολικές μεταβλητές που ορίστηκαν προηγουμένως. Ακολουθεί η δήλωση N τοπικών μεταβλητών, όπου N το πλήθος των θέσεων πίνακα που περιλαμβάνονται μεταξύ της ελάχιστης θέσης πίνακα που χρησιμοποιείται (διαβάζεται ή εγγράφεται) και της μέγιστης θέσης που εγγράφεται μέσα στο σώμα του βρόχου. Οι τοπικές μεταβλητές που αντιστοιχούν στις διαμοιραζόμενες παίρνουν τις τιμές των αντίστοιχων διαμοιραζόμενων. Γίνεται η αντιστοίχιση των τοπικών μεταβλητών και του πίνακα t στις εντολές του σώματος του βρόχου. Ακολουθεί η εγγραφή K τοπικών μεταβλητών στις θέσεις i+K του πίνακα, όπου K οι τιμές από την ελάχιστη τιμή που εγγράφεται ως την μικρότερη τιμή μεταξύ του βήματος και της μέγιστης τιμή που εγγράφεται. Τέλος, ακολουθεί η αποθήκευση των τιμών των τοπικών μεταβλητών στις διαμοιραζόμενες μεταβλητές, με μετατόπιση κατά τόσες θέσεις όσο το βήμα του βρόχου, σε σχέση με την αντιστοιχία τους και η αποθήκευση στις διαμοιραζόμενες μεταβλητές των τιμών του πίνακα που χρειάζονται στο επόμενο νήμα. Με αυτό τον

τρόπο το επόμενο νήμα θα πάρει τις σωστές τιμές από τις διαμοιραζόμενες μεταβλητές.

Γίνεται ο συγχρονισμός της οικογένειας, και μετά γίνονται οι εγγραφές στις υπόλοιπες θέσεις του πίνακα, εκτός από τις i+K, ώστε να γίνουν και οι εγγραφές που πρέπει στις θέσεις μετά από αυτή που εγγράφεται από το τελευταίο νήμα. Οι τιμές που εγγράφονται στις θέσεις αυτές είναι οι τιμές των καθολικών μεταβλητών που τους αντιστοιχούν, οι οποίες κατά τον συγχρονισμό της οικογένειας νημάτων παίρνουν τις τιμές που εγγράφει το τελευταίο νήμα στις αντίστοιχες διαμοιραζόμενες μεταβλητές. Η τιμή που έχει μια διαμοιραζόμενη μεταβλητή μετά τον συγχρονισμό της οικογένειας εγγράφεται στην καθολική μεταβλητή από την οποία αρχικοποιήθηκε.

Στην περίπτωση που δε μπορέσουμε να καθορίσουμε τον δείκτη του βρόχου, υπάρχουν μικρές διαφορές στον κώδικα που παράγεται. Αντί να γίνει η δημιουργία μιας οικογένειας νημάτων σύμφωνα με τα όρια του βρόχου, δημιουργείται μια οικογένεια με MAX_THREADS νήματα και βήμα 1. Ως δείκτης της οικογένειας δηλώνεται μια προσωρινή μεταβλητή `index ind`. Όλες οι μεταβλητές που υπάρχει πιθανότητα να είναι δείκτες του βρόχου δηλώνονται ως διαμοιραζόμενες και αρχικοποιούνται κατάλληλα. Προσθέτουμε έναν έλεγχο για τη διακοπή της περεταίρω δημιουργίας νημάτων με την εντολή break, σύμφωνα με τον έλεγχο τερματισμού του βρόχου. Αν η οικογένεια νημάτων προέρχεται από ένα βρόχο for προσθέτουμε στο τέλος του κώδικα των νημάτων την έκφραση του βήματος του βρόχου, ώστε να γίνει η ανανέωση της τιμής της μεταβλητής που περιέχεται σε αυτό. Αντίστοιχες αλλαγές γίνονται και στον κώδικα για τον χειρισμό των αντί-εξαρτήσεων. Κατά τα άλλα ο κώδικας που παράγεται είναι ίδιος με την περίπτωση που έχει καθοριστεί ο δείκτης του βρόχου. Οι προσθήκες στην περίπτωση που δε μπορεί να καθοριστεί ο δείκτης βρίσκονται μέσα σε [ ] στον παρακάτω αλγόριθμο.

Αλγόριθμος μετασχηματισμού μονοδιάστατων βρόχων

---

1. Δημιουργία της λίστας read με όλα τα στοιχεία πίνακα που διαβάζονται μέσα στο βρόχο

2. Δημιουργία της λίστας write με όλα τα στοιχεία πίνακα που εγγράφονται μέσα στο βρόχο
3. min_read := το ελάχιστο στοιχείο της λίστας read
4. max_read := το μέγιστο στοιχείο της λίστας read
5. min_write := το ελάχιστο στοιχείο της λίστας write
6. max_write := το μέγιστο στοιχείο της λίστας write
7. `int` $g_M$`=A[M+start];` ∀M: min_read ≤ M ≤ max_write-1
8. `family fid*;` ανάλογα με τον αριθμό των οικογενειών που δημιουργούνται
9. Αν max_read≥max_write (αν διαβάζεται κάποιο στοιχείο μετά από το τελευταίο στοιχείο που γράφεται)

```
create(fid*;start;limit;step)
[ή create(fid*;0;N-max_read;1) ο πίνακας A έχει N θέσεις]
{     index i;
      t[i] = A[i+max_read];
}
sync(fid*);
10.
create(fid*;start;limit;step)
[ή create(fid*;1;MAX_THREADS;1)]
{     index i; [ή index ind;]
      [shared int var=αρχικοποίηση; ∀πιθανό index]
      shared int sM=gM;  ∀M: min_read ≤ M ≤ max_write-1
      int tN;  ∀N: min(min_read, min_write) ≤ N ≤ max_write

      [if(!condition)
          break;]

      tM=sM;  ∀M: min_read ≤ M ≤ max_write-1
```

/*αναπαράγουμε το σώμα του βρόχου χρησιμοποιώντας τις τοπικές μεταβλητές $t_M$ και αν υπάρχει τον πίνακα t*/

`A[i+K]=t`$_K$`;` $\forall K$: min_write $\leq K <$ min(step, max_write)

`s`$_N$`=t`$_{N+step}$`;` $\forall N$: min_read $\leq N \leq$ max_write-step

`s`$_N$`=A[i+step+N];` $\forall N$: max_write-step +1 $\leq N \leq$ max_write-1

[`step_expression` αν η οικογένεια προέρχεται από for]

```
}
sync(fid*);
```

11. `A[limit+K]=g`$_{K-step}$`;` $\forall K$: min_write+step $\leq K \leq$ max_write

---

## 5.8. Κανονικοποίηση βρόχων

Για να μπορέσουμε να χειριστούμε όλους τους βρόχους με τον ίδιο τρόπο μπορούμε να κάνουμε μια κανονικοποίηση ώστε σε όλους τους βρόχους η ελάχιστη θέση πίνακα όπου γίνεται εγγραφή να είναι η θέση 0. Για να γίνει η κανονικοποίηση αφαιρούμε από όλους τους δείκτες των πινάκων τον δείκτη της ελάχιστης θέσης όπου γίνεται εγγραφή (το min_write), καθώς και προσθέτουμε το min_write στα όρια του βρόχου (start και limit). Ας δούμε ένα παράδειγμα, όπου το min_write είναι -1:

```
for(i=2;i<N;i++)           →        for(i=1;i<N-1;i++)
{   a[i-1]=a[i];                    {   a[i]=a[i+1];
    a[i+1]=a[i];                        a[i+2]=a[i+1];
    a[i]=a[i-2]+a[i+2];                 a[i+1]=a[i-1]+a[i+3];
}                                   }
```

# ΚΕΦΑΛΑΙΟ 6. ΥΛΟΠΟΙΗΣΗ

6.1. Υλοποίηση
6.2. GCC
6.3. Σχεδιασμός εξαρχής
6.4. Λεκτική και συντακτική ανάλυση
6.5. Ενδιάμεση αναπαράσταση και επιπρόσθετες πληροφορίες
6.6. Επεξεργασία ενδιάμεσης αναπαράστασης και πληροφοριών
6.7. Παραγωγή του κώδικα σε μTC

## 6.1. Υλοποίηση

Ξεκινώντας να σχεδιάσουμε ένα μεταγλωττιστή για τη μετατροπή του κώδικα από τη γλώσσα C στη γλώσσα μTC, υπάρχουν δυο προσεγγίσεις που μπορούμε να ακολουθήσουμε. Ο ένας τρόπος είναι να αλλάξουμε έναν ήδη υπάρχον μεταγλωττιστή για την C και να κάνουμε όλες τις απαραίτητες μετατροπές ώστε να παράγει κώδικα σε μTC, αντί να παράγει τελικό κώδικα σε γλώσσα μηχανής. Ο πιο πιθανός υποψήφιος μεταγλωττιστής για να γίνει αυτό θα ήταν ο GCC. Η άλλη προσέγγιση είναι να ξεκινήσουμε την σχεδίαση ενός μεταγλωττιστή από την αρχή. Και οι δυο προσεγγίσεις έχουν πλεονεκτήματα και μειονεκτήματα. Εμείς επιλέξαμε την εξ' αρχής σχεδίαση του μεταγλωττιστή.

## 6.2. GCC

Ο GCC είναι ένας από τους πιο ισχυρούς μεταγλωττιστές που είναι διαθέσιμοι στους ερευνητές. Ο κώδικας που παράγει είναι βελτιστοποιημένος για αρκετές αρχιτεκτονικές, και σε μερικές περιπτώσεις έχει την ίδια ποιότητα με τον

μεταγλωττιστή που παράγει ο κατασκευαστής του μηχανήματος. Είναι ελεύθερο λογισμικό και έχει συνεχείς ανανεώσεις, όμως δεν είναι σχεδιασμένος για μετατροπές από κώδικα σε κώδικα. Οι περισσότερες λειτουργίες του εκτελούνται σε μια χαμηλού επιπέδου αναπαράσταση. Μόνο οι πιο πρόσφατες εκδόσεις του περιλαμβάνουν ένα συντακτικό δέντρο.

Ο GCC παρέχει ένα όχι φιλικό API, το οποίο αποτελείται κυρίως από macros και δεν υπάρχει λογική ομαδοποίηση στον κώδικα του μεταγλωττιστή. Η ενδιάμεση αναπαράσταση που χρησιμοποιεί είναι δυσνόητη αφού γενικά όλα τα πεδία αναπαριστούνται με τον ίδιο τύπο. Υπάρχει άφθονη τεκμηρίωση για τον GCC, όμως για να μπορέσει να υλοποιηθεί ακόμα και ένας απλός μετασχηματισμός χρειάζεται να αφιερωθεί πολύς χρόνος για την κατανόηση του μεταγλωττιστή αυτού.

Ουσιαστικά το κομμάτι του GCC το οποίο μας είναι χρήσιμο είναι οι βελτιστοποιήσεις που κάνει στον κώδικα. Ο δικός μας στόχος είναι να κάνουμε το μετασχηματισμό του κώδικα από C σε μTC και όχι να βελτιστοποιήσουμε τον κώδικα, αφού αυτό θα γίνει σε επόμενο βήμα. Αφού δηλαδή παραχθεί ο κώδικας μας σε μTC, θα περάσει από ένα εργαλείο βασισμένο στον GCC το οποίο θα κάνει όλες τις απαραίτητες βελτιστοποιήσεις.

### 6.3. Σχεδιασμός εξαρχής

Όταν σχεδιάζουμε ένα σύστημα εξαρχής έχουμε την ελευθερία να το προσαρμόσουμε, ώστε να είναι όσο αποδοτικότερο γίνεται σύμφωνα με τις απαιτήσεις μας. Έχουμε μεγαλύτερη ευελιξία στο σχεδιασμό και στην υλοποίηση αφού το σύστημα μας δεν εξαρτάται από κάποιο άλλο, είναι αυτόνομο. Το μειονέκτημα του σχεδιασμού εξαρχής είναι πρέπει να υλοποιήσουμε τον λεκτικό και τον συντακτικό αναλυτή και κάποιες βελτιστοποιήσεις, που στους ήδη υπάρχοντες μεταγλωττιστές υπάρχουν ήδη υλοποιημένοι.

Τα βήματα που ακολουθήσαμε ώστε να γίνει η μετατροπή από τη γλώσσα C στη γλώσσα μTC είναι τα εξής:

-Λεκτική ανάλυση

-Συντακτική ανάλυση

-Μετατροπή σε μια ενδιάμεση αναπαράσταση

-Προσθήκη πληροφοριών που προκύπτουν κατά τη μεταγλώττιση

-Επεξεργασία της ενδιάμεσης αναπαράστασης σε συνδυασμό με τις πληροφορίες

-Παραγωγή του κώδικα σε μTC

## 6.4. Λεκτική και συντακτική ανάλυση

Για να δημιουργήσουμε έναν λεκτικό αναλυτή για τη γλώσσα C χρησιμοποιήθηκε το εργαλείο Lex. Το εργαλείο αυτό παίρνει ως είσοδο ένα σύνολο από περιγραφές συμβόλων και δεσμευμένων λέξεων που ορίζουν την γλώσσα και παράγει μια συνάρτηση γραμμένη στη γλώσσα C, η οποία ονομάζεται λεκτικός αναλυτής. Το σύνολο των περιγραφών ονομάζεται περιγραφή Lex και είναι ένα σύνολο κανονικών εκφράσεων. Μια κανονική έκφραση είναι η περιγραφή ενός γλωσσικού προτύπου με την χρήση μιας μετά-γλώσσας. Το Lex μετατρέπει αυτές τις κανονικές εκφράσεις σε μια μορφή που μπορεί να χρησιμοποιηθεί από το λεκτικό αναλυτή ώστε να σαρώνει το αρχείο εισόδου του μεταγλωττιστή μας πολύ γρήγορα, ανεξάρτητα από τον αριθμό των εκφράσεων που προσπαθεί να ταυτοποιήσει. Ένας λεκτικός αναλυτής φτιαγμένος από το εργαλείο Lex είναι σχεδόν πάντα γρηγορότερος από έναν άλλο, γραμμένο από έναν προγραμματιστή.

Καθώς η είσοδος χωρίζεται σε σύμβολα και λέξεις, πρέπει να οριστεί μια σχέση μεταξύ τους, πρέπει δηλαδή να γίνει η συντακτική ανάλυση του προγράμματος. Ένας μεταγλωττιστής για τη γλώσσα C πρέπει να εντοπίσει τις εκφράσεις, τις εντολές, τους ορισμούς, τα blocks και τις συναρτήσεις των προγραμμάτων. Αυτή η διαδικασία ονομάζεται συντακτική ανάλυση και η λίστα κανόνων που ορίζει την διαδικασία αυτή ονομάζεται γραμματική της γλώσσας. Για τη δημιουργία ενός συντακτικού αναλυτή χρησιμοποιήσαμε το εργαλείο Yacc. Το Yacc παίρνει ως είσοδο μια συνοπτική περιγραφή μιας γραμματικής και παράγει μια συνάρτηση σε C, η οποία κάνει συντακτική ανάλυση ενός προγράμματος σύμφωνα με αυτή τη γραμματική. Ο συντακτικός αναλυτής αυτός αυτόματα εντοπίζει πότε μια ακολουθία συμβόλων και λέξεων ταιριάζει με έναν από τους κανόνες της γραμματικής. Επίσης εντοπίζει τα

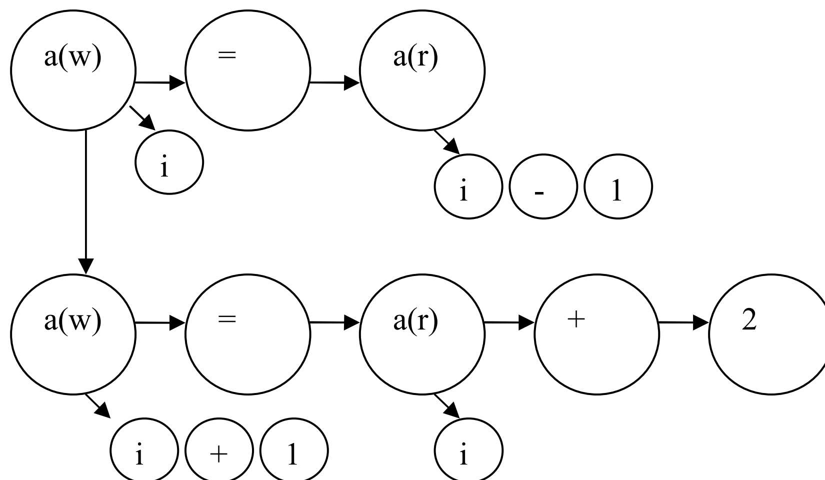
συντακτικά λάθη στην είσοδο του μεταγλωττιστή στην περίπτωση που δεν ταιριάζει με κανέναν από τους κανόνες. Ένας συντακτικός αναλυτής φτιαγμένος από το εργαλείο Yacc γενικά δεν είναι όσο γρήγορος όσο ένας άλλος γραμμένος από έναν προγραμματιστή, αλλά η ευκολία στην συγγραφή και τις μετατροπές του αντισταθμίζει κάθε απώλεια ταχύτητας. Σε κάθε περίπτωση ο χρόνος που καταλαμβάνει η συντακτική ανάλυση δεν είναι σημαντικός ώστε να δημιουργεί οποιοδήποτε πρόβλημα.

### 6.5. Ενδιάμεση αναπαράσταση και επιπρόσθετες πληροφορίες

Καθώς γίνεται η συντακτική ανάλυση, έχοντας προσθέσει τον κατάλληλο κώδικα στον συντακτικό αναλυτή που παράγεται από το εργαλείο Yacc, αποθηκεύουμε τον κώδικα σε μια λίστα από λίστες, όπως φαίνεται στο Σχήμα 6.1, όπου κάθε κόμβος είναι μια λέξη ή ένα σύμβολο. Οι κόμβοι συνδέονται σε λίστες ανά εντολή και οι λίστες των εντολών σε μια λίστα με όλο το πρόγραμμα. Με αυτό τον τρόπο είναι πιο εύκολο να επεξεργαστούμε τον κώδικα στο δεύτερο πέρασμα που πρέπει να γίνει για την μετατροπή του σε μTC.

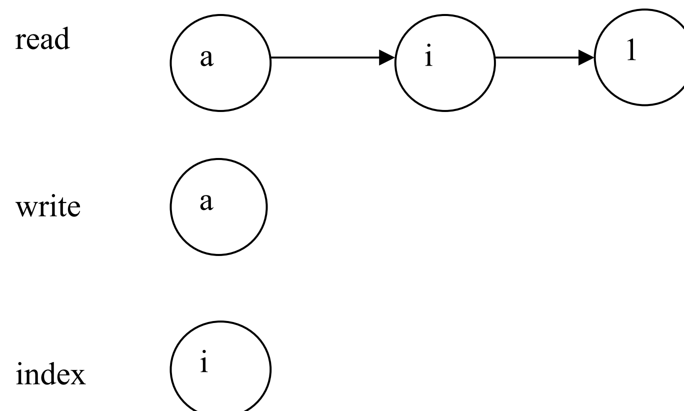
Κατά τη διάρκεια της συντακτικής ανάλυσης και ενώ κατασκευάζουμε τη λίστα με τις λέξεις και τα σύμβολα, εξάγουμε από τον κώδικα πληροφορίες που μας βοηθάνε στην μετατροπή του σε μTC. Κάθε κόμβο της λίστας τον χαρακτηρίζουμε ανάλογα με το είδος του, αν δηλαδή είναι μια δεσμευμένη λέξη της C, αν είναι κάποιο σύμβολο ή κάποια μεταβλητή. Επίσης για κάθε μεταβλητή που συναντούμε την χαρακτηρίζουμε ως προς την λειτουργία που εφαρμόζεται σε αυτή, αν δηλαδή η μεταβλητή διαβάζεται, εγγράφεται ή εφαρμόζονται και οι δυο λειτουργίες.

Συνολικά, για τη λίστα από λίστες έχουν υλοποιηθεί η δημιουργία ενός κόμβου για κάθε λέξη ή σύμβολο, η εισαγωγή των κόμβων σε μια λίστα, η σύνδεση των υπολιστών που αντιστοιχούν σε μια εντολή η κάθε μια, ο χαρακτηρισμός των κόμβων ανάλογα με το είδος και τη λειτουργία τους και η εύρεση των ορίων και του βήματος ενός βρόχου εφόσον ακολουθείται συγκεκριμένη σύνταξη.




Σχήμα 6.1 Λίστα από λίστες

Επίσης για κάθε βρόχο που συναντούμε φτιάχνουμε μια λίστα με τους υποψήφιους δείκτες του βρόχου. Αργότερα θα ελέγξουμε αν κάποια από αυτές ικανοποιεί τις συνθήκες ώστε να καθοριστεί πια είναι ο δείκτης του βρόχου. Ακόμα δημιουργούμε δυο λίστες στις οποίες αποθηκεύουμε τις μεταβλητές στις οποίες μέσα στο βρόχο γίνεται εγγραφή και εκείνες στις οποίες γίνεται ανάγνωση, αντίστοιχα. Αυτές οι λίστες φαίνονται στο Σχήμα 6.2.




Σχήμα 6.2 Λίστες μεταβλητών

Για να καθορίσουμε τη λειτουργία κάθε μεταβλητής, αν δηλαδή γίνεται ανάγνωση ή εγγραφή σε αυτή, μπορούμε να χρησιμοποιήσουμε τις δυο λίστες με τις μεταβλητές ή την λίστα από λίστες όπου υπάρχει ο κατάλληλος χαρακτηρισμός, ανάλογα τι βολεύει καλύτερα σε κάθε περίπτωση. Για κάθε βρόχο, εφόσον ακολουθεί την καθορισμένη σύνταξη, καθορίζουμε την αρχική τιμή του δείκτη του, την τελική του τιμή και το βήμα με το οποίο αυτός αλλάζει. Αυτές οι τιμές πρέπει να είναι σταθερές μέσα στο βρόχο, δεν πρέπει δηλαδή να αλλάζουν στο σώμα του βρόχου.

Οι βρόχοι που μπορούμε να μετατρέψουμε σε μTC πρέπει να ακολουθούν την παρακάτω γραμματική:

```
<loop> ::= <for> | <while>
<for> ::= for (<assign>;<cond>;<step>) <statements>
<assign>::= index = a1
<cond> ::= index <relop> a2
<relop> ::= < | > | == | <= | >=
<step> ::= index ++ | index = index + a3 | index += a3
```

**6.6. Επεξεργασία ενδιάμεσης αναπαράστασης και πληροφοριών**

Έχοντας συλλέξει όλες τις πληροφορίες που χρειαζόμαστε και έχοντας το βρόχο στην ενδιάμεση αναπαράσταση ξεκινάμε την επεξεργασία του. Αρχικά βρίσκουμε τον δείκτη του βρόχου. Χρησιμοποιούμε τη λίστα με τους πιθανούς δείκτες και τους κανόνες:

1. Ο δείκτης είναι μια από τις μεταβλητές που εγγράφονται μέσα στο σώμα του βρόχου, είτε στην εντολή που καθορίζει το βήμα του βρόχου
2. Τουλάχιστον μια από τις μεταβλητές της συνθήκης εξόδου εξαρτάται από τον δείκτη, άμεσα ή έμμεσα

Αν μια μόνο από τις μεταβλητές τη λίστας ικανοποιεί τους κανόνες, τότε αυτή είναι ο δείκτης. Αν υπάρχουν περισσότερες μεταβλητές που να τους ικανοποιούν τότε θα πρέπει να παράγουμε τον κατάλληλο κώδικα για να χειριστούμε αυτή την περίπτωση.

Το επόμενο βήμα είναι η δημιουργία δυο λιστών, μια για την ανάγνωση και μια για την εγγραφή σε πίνακα. Για κάθε θέση πίνακα που συναντούμε στο βρόχο, αποθηκεύουμε τον δείκτη θέσης στην κατάλληλη λίστα, ανάλογα με τον χαρακτηρισμό που δώσαμε κατά τη συντακτική ανάλυση. Για κάθε μια από τις δυο λίστες βρίσκουμε το μέγιστο και το ελάχιστο στοιχείο της. Αυτές είναι οι μέγιστες και οι ελάχιστες θέσεις του πίνακα όπου γίνονται εγγραφή και ανάγνωση. Από αυτές τις τιμές καθορίζουμε το είδος των εξαρτήσεων που έχουμε στο βρόχο. Αν η μέγιστη τιμή όπου γίνεται ανάγνωση είναι μεγαλύτερη ή ίση με τη μέγιστη τιμή όπου γίνεται εγγραφή τότε υπάρχει αντί-εξάρτηση και θα πρέπει να δημιουργηθεί ο κατάλληλος κώδικας για αυτό.

Δημιουργούμε μια λίστα με τις μονοδιάστατες διαμοιραζόμενες μεταβλητές. Οι μεταβλητές αυτές θα πρέπει να δηλωθούν ως διαμοιραζόμενες κατά τη δημιουργία της οικογένειας νημάτων, ώστε να μπορεί ένα νήμα να βλέπει την τιμή που έδωσε το προηγούμενο νήμα σε αυτή και να είναι ενημερωμένο για τη σωστή τιμή της. Αυτό που θέλουμε είναι κάθε φορά που γίνεται εγγραφή σε μια τέτοια μεταβλητή, το επόμενο νήμα να ενημερώνεται για τη σωστή τιμή. Έτσι οι μονοδιάστατες μεταβλητές που θα δηλωθούν ως διαμοιραζόμενες θα είναι όλες οι μεταβλητές στις οποίες γίνεται εγγραφή μέσα στο βρόχο.

Για να γίνεται σωστά η προώθηση των τιμών ενός πίνακα μεταξύ των νημάτων χρειαζόμαστε να ορίσουμε έναν αριθμό διαμοιραζόμενων μεταβλητών. Ο αριθμός αυτός καθορίζεται από τον αριθμό των θέσεων πίνακα μεταξύ της ελάχιστης θέσης που διαβάζεται μέσα στο βρόχο και της μέγιστης θέσης που εγγράφεται.

Ακολουθεί η κανονικοποίηση του βρόχου, ώστε η τελευταία θέση πίνακα όπου γίνεται εγγραφή να είναι η θέση που καθορίζει η τιμή του δείκτη, και όχι κάποια επόμενη ή προηγούμενη, ώστε να μπορούμε να χειριστούμε όλους τους βρόχους με τον ίδιο ακριβώς τρόπο.

**6.7. Παραγωγή του κώδικα σε μTC**

Αρχικά δηλώνονται οι καθολικές μεταβλητές που θα χρησιμοποιηθούν για την αρχικοποίηση των διαμοιραζόμενων. Κάθε μεταβλητή παίρνει το όνομα $g_M$, όπου M ένας αριθμός από την ελάχιστη θέση όπου γίνεται ανάγνωση μέχρι την προηγούμενη από την μεγαλύτερη θέση πίνακα όπου γίνεται εγγραφή. Κάθε μια από αυτές τις μεταβλητές αρχικοποιείται από τη θέση πίνακα που της αντιστοιχεί. Χρησιμοποιούμε τη σύμβαση όταν το M είναι μεγαλύτερο του μηδενός η μεταβλητή ονομάζεται gM, ενώ αν το M είναι μικρότερο του μηδενός τότε η μεταβλητή ονομάζεται g_M. Το ίδιο ισχύει και για την ονοματολογία των διαμοιραζόμενων μεταβλητών, καθώς και των τοπικών. Ο κώδικας που παράγεται από αυτό το βήμα θα είναι:

`int g_M=w[M+start];` $\forall M$: min_read $\leq$ M $\leq$ max_write - 1

Στη συνέχεια δηλώνουμε όσες μεταβλητές τύπου family χρειάζονται για τη δημιουργία των οικογενειών νημάτων. Ο κώδικας που παράγεται είναι:

```
family fid*;
```

Στην περίπτωση που υπάρχει αντί-εξάρτηση δημιουργούμε τον κώδικα για να τη χειριστούμε. Αν λοιπόν υπάρχει ανάγνωση μετά από την τελευταία εγγραφή τότε δημιουργούμε τον κώδικα για την οικογένεια νημάτων που αντιγράφει τον πίνακα με την αντί-εξάρτηση σε έναν άλλο πίνακα `t` ο οποίος είναι προσωρινός. Αρχικά καλείται η εντολή `create` με ορίσματα τη μεταβλητή `fid*`, που είναι η μεταβλητή όπου θα αποθηκευτεί το αναγνωριστικό αυτής της οικογένειας νημάτων, τα όρια του βρόχου μετά την κανονικοποίηση, καθώς και το βήμα του βρόχου. Αν δεν έχει καθοριστεί ο δείκτης του βρόχου, τα όρια είναι από 0 ως N-max_read, όπου N το μέγεθος του πίνακα που θα αντιγραφεί, και το βήμα είναι 1. Μέσα στο σώμα της οικογένειας νημάτων περιλαμβάνεται η δήλωση του δείκτη του βρόχου, αν καθορίστηκε προηγουμένως, και η εντολή για την αντιγραφή του πίνακα στον προσωρινό. Τέλος παράγεται η εντολή για τον συγχρονισμό της οικογένειας νημάτων. Ο κώδικας που παράγεται είναι:

```
create(fid*;start;limit;step)
{     index i;
```

```
    t[i] = A[i+max_read];
}
```

Σειρά έχει η δημιουργία του κώδικα για την οικογένεια νημάτων που αντικαθιστά το βρόχο. Αρχικά καλείται η εντολή `create` με ορίσματα το αναγνωριστικό `fid*` αυτής της οικογένειας νημάτων, τα όρια του βρόχου μετά την κανονικοποίηση και το βήμα του βρόχου. Μέσα στο σώμα του βρόχου περιλαμβάνεται η δήλωση του δείκτη του βρόχου, αν καθορίστηκε προηγουμένως. Ακολουθεί η δήλωση των διαμοιραζόμενων μεταβλητών και η αρχικοποίηση τους από τις αντίστοιχες καθολικές, καθώς και η δήλωση ως διαμοιραζόμενες μεταβλητές όλων των πιθανών δεικτών, αν δεν έχει καθοριστεί δείκτης για το βρόχο. Γίνεται η δήλωση των τοπικών μεταβλητών και η αρχικοποίηση τους από τις αντίστοιχες διαμοιραζόμενες και αμέσως μετά αναπαράγουμε το σώμα του βρόχου από τη λίστα από λίστες, χρησιμοποιώντας τις τοπικές μεταβλητές που αντιστοιχούν στις θέσεις του πίνακα και τον προσωρινό πίνακα, αν αυτός χρησιμοποιήθηκε, για το χειρισμό της αντί-εξάρτησης. Γίνεται η εγγραφή στις θέσεις μνήμης που αντιστοιχούν στο νήμα. Παράγεται ο κώδικας για την αποθήκευση των τιμών των τοπικών μεταβλητών στις διαμοιραζόμενες μετά από μια μετατόπιση κατά τόσες θέσεις όσο είναι το βήμα καθώς και η αποθήκευση των τιμών του πίνακα που απαιτούνται από το επόμενο νήμα. Τέλος παράγεται η εντολή για το συγχρονισμό της οικογένειας νημάτων.

```
create(fid*;start;limit;step)
{       index i;
        shared int sM=gM;  ∀M:  min_read ≤ M ≤ max_write-1
        int tN;  ∀N:  min(min_read, min_write) ≤ N ≤ max_write
        tM=sM;  ∀M:  min_read ≤ M ≤ max_write-1

   /*αναπαράγουμε το σώμα του βρόχου χρησιμοποιώντας τις τοπικές μεταβλητές tM
   και αν υπάρχει τον πίνακα t*/

        A[i+K]=tK;  ∀K: min_write ≤ K < min(step, max_write)
```

$s_N = t_{N+step}$; $\forall N$: min_read $\leq N \leq$ max_write-step

$s_N = A[i+step+N]$; $\forall N$: max_write-step +1 $\leq N \leq$ max_write-1

```
}
sync(fid*);
```

Το τελευταίο βήμα, αν αυτό απαιτείται, είναι η παραγωγή του κώδικα για την αποθήκευση των τιμών στον πίνακα, στις θέσεις μετά από τη θέση όπου γράφει το τελευταίο νήμα. Μετά το συγχρονισμό της οικογένειας, οι τιμές των διαμοιραζόμενων μεταβλητών αποθηκεύονται στις καθολικές. Έτσι με αυτό τον κώδικα αποθηκεύονται οι κατάλληλες τιμές στον πίνακα.

$A[limit+K] = g_{K-step}$; $\forall K$: min_write+step $\leq K \leq$ max_write

# ΚΕΦΑΛΑΙΟ 7. ΜΕΛΛΟΝΤΙΚΑ ΒΗΜΑΤΑ

7.1. Μετασχηματισμός εμφωλευμένων βρόχων

7.2. Εξαρτήσεις μεταξύ διαφορετικών πινάκων στο σώμα ενός βρόχου

7.3. Ονομασμένα νήματα (named threads)

7.4. Εξαγωγή παραλληλισμού από συναρτήσεις

## 7.1. Μετασχηματισμός εμφωλευμένων βρόχων

Οι μετασχηματισμοί των εμφωλευμένων βρόχων δεν είναι μια απλή γενίκευση των αντίστοιχων μονοδιάστατων βρόχων. Ο βασικός λόγος είναι ότι το μοντέλο μας επιτρέπει την επικοινωνία μεταξύ διαδοχικών νημάτων μόνο, μέσω των διαμοιραζόμενων μεταβλητών. Στους μονοδιάστατους βρόχους η λύση σε αυτό το πρόβλημα είναι η μετατόπιση των δεδομένων που παράγονται από τις προηγούμενες επαναλήψεις, μέσω των διαμοιραζόμενων μεταβλητών, ώστε να φτάσουν στο νήμα που τις χρειάζεται για να κάνει τους υπολογισμούς του. Παρόλα αυτά, αυτό δεν είναι πάντα εφικτό, ή δεν είναι αρκετό για τους εμφωλευμένους βρόχους και αυτό κάνει το πρόβλημα πιο περίπλοκο.

Θα ξεκινήσουμε από τις πιο απλές περιπτώσεις όπου οι γενικεύσεις από τους απλούς μονοδιάστατους βρόχους είναι δυνατές. Χειριζόμαστε τις αντί-εξαρτήσεις αντιγράφοντας και πάλι τον πίνακα σε έναν προσωρινό. Θεωρούμε ότι a>0 και b>0. Δημιουργούμε ένα αντίγραφο t του πίνακα και όλα τα νήματα διαβάζουν από τον προσωρινό πίνακα t.

```
for(i=0;j<N-a;i++)      →      create(fid0;0;N-a-1;1)
  for(j=0;j<N-b;j++)           {  index i;
```

```
w[i][j]=w[i+a][j+b];          create(fid1;0;N-b-1;1)
                              {  index j;
                                 t[i][j]=w[i+a][j+b];
                              }
                              sync(fid1);
                           }
                           sync(fid0);

                           create(fid2;1;N-a-1;1)
                           {   index i;
                               create(fid3;0;N-b-1;1)
                               {   index j;

                                   int t00;
                                   t00=t[i][j];
                                   w[i][j]=t00;
                               }
                               sync(fid3);
                           }
                           sync(fid2);
```

Στην περίπτωση που υπάρχει εξάρτηση όμως, ο μετασχηματισμός δεν είναι μια γενίκευση της αντίστοιχης περίπτωσης των μη εμφωλευμένων βρόχων. Αν υπάρχουν εξαρτήσεις και αντί-εξαρτήσεις τότε χειριζόμαστε τις αντί-εξαρτήσεις πρώτα, όπως και στους μονοδιάστατους βρόχους. Ο λόγος που δεν είναι απλή η γενίκευση είναι ότι το υλικό υποστηρίζει μόνο μονοδιάστατες εξαρτήσεις. Έτσι στους βρόχους μεγαλύτερης διάστασης το πρόβλημα γίνεται ακόμα πιο περίπλοκο και πρέπει να εξετάσουμε περισσότερες περιπτώσεις.

Ο επεξεργαστής μας υποστηρίζει μόνο απλές μοναδιαίες εξαρτήσεις και δεν υποστηρίζει επικοινωνία μεταξύ των επαναλήψεων σε περίπλοκους εμφωλευμένους βρόχους. Θα μπορούσαμε να επιλέξουμε τη λεξικογραφικά μεγαλύτερη εξάρτηση και να δημιουργήσουμε μια οικογένεια νημάτων για αυτή. Μια οικογένεια νημάτων

μοιάζει με ένα pipeline εκτέλεσης, όπου οι επαναλήψεις είναι τα στοιχεία υπολογισμού και οι διαμοιραζόμενες μεταβλητές είναι τα κανάλια που μεταφέρουν τα δεδομένα μεταξύ διαδοχικών στοιχείων υπολογισμού. Οι υπόλοιπες εξαρτήσεις ανήκουν σε μια από τις παρακάτω κατηγορίες:

- Αν η επανάληψη i εξαρτάται από την επανάληψη j, η οποία ανήκει στην ίδια οικογένεια νημάτων, τα δεδομένα που παράγονται από την επανάληψη j θα μετακινηθούν από επανάληψη σε επανάληψη μέχρι να φτάσουν στην i.
- Αν η επανάληψη i εξαρτάται από την επανάληψη j, η οποία ανήκει σε διαφορετική οικογένεια νημάτων, η εκτέλεση της οποίας έχει ολοκληρωθεί νωρίτερα, θα διαβάσει τα δεδομένα που παράχθηκαν από την επανάληψη j από την κύρια μνήμη. Αφού η έχει ολοκληρωθεί η εκτέλεση της οικογένειας νημάτων η μνήμη θεωρείται συνεπής και όλες οι επόμενες εκτελέσεις μπορούν να διαβάσουν τα δεδομένα από εκεί.

Οι αντί-εξαρτήσεις αντιμετωπίζονται με τον ίδιο τρόπο:

- Αν μια επανάληψη i σχετίζεται με μια επανάληψη j, η οποία ανήκει στην ίδια οικογένεια νημάτων, ένα αντίγραφο του πίνακα δημιουργείται και όλα τα νήματα διαβάζουν τις τιμές του πίνακα από το αντίγραφο
- Αν η επανάληψη i ανήκει στην οικογένεια I και εξαρτάται με μια σχέση αντί-εξάρτησης από την επανάληψη j που ανήκει στην οικογένεια J, τότε μπορούμε να υποθέσουμε ότι η εκτέλεση της οικογένειας I θα ολοκληρωθεί πριν ξεκινήσει η εκτέλεση της οικογένειας J. Έτσι κάθε επανάληψη i μπορεί να διαβάσει τις τιμές του j, αφού αυτές θα μεταβληθούν μετά την ολοκλήρωση της εκτέλεσης της οικογένειας I.

Μελλοντικά, μπορούν να μελετηθούν οι εμφωλευμένοι βρόχοι, ώστε να καθοριστεί ένας τρόπος αντιμετώπισης τους όπως και για τους μη-εμφωλευμένους, ώστε να μπορεί να γίνει η μετατροπή τους από σειριακούς σε παράλληλους, σε μTC, σύμφωνα με το μοντέλο SVP.

### 7.2. Εξαρτήσεις μεταξύ διαφορετικών πινάκων στο σώμα ενός βρόχου

Στην περίπτωση που στο σώμα ενός βρόχου περιλαμβάνονται εξαρτήσεις μεταξύ διαφορετικών πινάκων, πρέπει να χειριστούμε τον βρόχο διαφορετικά, καθώς δεν είναι απλή γενίκευση. Ας δούμε ένα παράδειγμα:

```
for(i=1;i<10;i++)
{     b[i]=b[i-1]+a[i];
      a[i+1]=b[i-2];
}
```

Ο τρόπος που θα χειριστούμε τον παραπάνω κώδικα είναι ο ίδιος με τον τρόπο που θα χειριστούμε τον παρακάτω:

```
for(i=1;i<10;i++)
{     w[i][1]=w[i-1][1]+w[i][2];
      w[i+1][2]=w[i-2][1];
}
```

Θα μπορούσαμε δηλαδή μελλοντικά να χειριστούμε τον βρόχο αυτό σαν να περιέχει έναν πίνακα δυο διαστάσεων, όπου κάθε πίνακας του βρόχου θα είναι μια στήλη του βοηθητικού πίνακα.

### 7.3. Ονομασμένα νήματα (named threads)

Τα ονομασμένα νήματα ορίζονται όπως οι συναρτήσεις της γλώσσας C. Χρησιμοποιεί μια λίστα ορισμάτων, με τον συνηθισμένο τρόπο, και μπορεί να μεταγλωττιστεί χωριστά και να περιγραφεί με μια κεφαλίδα. Οι διαφορές τους από τις συναρτήσεις είναι ότι δεν ορίζεται για τα νήματα το `return` των συναρτήσεων. Αντίθετα, τα νήματα παρέχουν την εντολή `break`, που μπορεί να χρησιμοποιηθεί με τον ίδιο τρόπο. Όταν χρησιμοποιείται ένα ονομασμένο νήμα, οι `shared` μεταβλητές πρέπει να αρχικοποιηθούν μέσω των ορισμάτων του. Ένα μελλοντικό βήμα είναι η υποστήριξη των ονομασμένων νημάτων και η δημιουργία οικογενειών νημάτων με τη χρήση τους, στη θέση της σύνθεσης εντολών που χρησιμοποιείται τώρα.

### 7.4. Εξαγωγή παραλληλισμού από συναρτήσεις

Κάθε συνάρτηση μπορεί να μετατραπεί σε έναν ονομασμένο νήμα στην μTC και η κλήση της θα αντικατασταθεί από μια κλήση στην εντολή create για το συγκεκριμένο ονομασμένο νήμα. Μελλοντικά θα μπορούσε να αναπτυχθεί ο έλεγχος εξαρτήσεων μεταξύ συναρτήσεων, με σκοπό την εξαγωγή παραλληλισμού μεταξύ τους. Αν υπάρχει εξάρτηση μεταξύ δυο συναρτήσεων θα μπορούσαν να ξεκινήσουν να εκτελούνται παράλληλα μέχρι το σημείο της εξάρτησης, όπου απαιτείται ο συγχρονισμός τους μέσω διαμοιραζόμενων μεταβλητών, όπως γίνεται για δυο γειτονικά νήματα. Με αυτό τον τρόπο μπορούμε να εκμεταλλευτούμε την πιθανή παραλληλία μεταξύ τους.

# ΑΝΑΦΟΡΕΣ

[1] C R Jesshope. SVP and μTC - A dynamic model of concurrency and its implementation as a compiler target, draft report (unpublished), 2007.

[2] C R Jesshope. A model for the design and programming of multicores, Draft paper submitted to: Advances in Parallel Computing, IOS Press, Amsterdam. 2007.

[3] C R Jesshope. μTC – an intermediate language for programming chip multiprocessors, Proc. Pacific Computer Systems Architecture Conference 2006 - ACSAC06, ISBN 3-540-4005, LNCS 4186, pp147-160. 2006.

[4] T.A.M. Bernard, C.R. Jesshope, and P.M.W. Knijnenburg, Strategies for Compiling μTC to Novel Chip Multiprocessors, International Symposium on Systems, Architectures, MOdeling and Simulation, S. Vassiliadis et al. (Eds.): SAMOS 2007, LNCS 4599, pp. 127-138, 2007.

[5] Keith A. Faigin, Stephen A. Weatherford, Jay P. Hoeflinger, David A. Padua and Paul M. Petersen. The polaris internal representation. International Journal of Parallel Programming, 22(5). pp 553-586, Oct. 1994.

[6] W. Blume, R. Eigenmann, J. Hoeflinger, and D. Padua. Automatic Detection of Parallelism: A grand Challenge for High-Performance Computing. IEEE Parallel and Distributed Technology, Vol. 2, No. 3. pp 37-47. 1994.

[7] S. P. Amarasinghe, J. M. Anderson, M. S. Lam, and A. W. Lim. An overview of the SUIF compiler for scalable parallel machines. In Proceedings of the 6th Workshop on Languages and Compilers for Parallel Computing, Portland, OR. August 1993.

[8] S. Benkner. VFC: The Vienna Fortran Compiler. Scientific Programming, 7(1). pp 67-81, 1999.

[9] P. Banerjee, J. A. Chandy, M. Gupta, J. G. Holm, A. Lain, D. J. Palermo, S. Ramaswamy, and E. Su. The PARADIGM Compiler for Distributed-Memory Message Passing Multicomputers. In Proceedings of the First International Workshop on Parallel Processing, pages 322-330, Bangalore, India, December 1994.

[10] S.-I. Lee, T. A. Johnson, and R. Eigenmann. Cetus - an extensible compiler infrastructure for source-to-source transformation. In Proceedings of the 16th Annual Workshop on Languages and Compilers for Parallel Computing (LCPC '2003).

[11] C.L. McAvaney and A.M. Goscinski. "Loop parallelisation using a simple parallelising compiler on the RHODOS distributed system", Technical Report, School of Computing and Mathematics, Deakin University, 1998.

[12] C. McAvaney and A. Goscinski, "A Simple Parallelising Compiler", Technical Report TR C97/06, Deakin University, May 1997.

[13] H. Kasahara, M. Obata, K. Ishizaka, K. Kimura, H. Kaminaga, H. Nakano, K. Nagasawa, A. Murai, H. Itagaki and J. Shirako. Multigrain Automatic Parallelization in Japanese Millennium Project IT21 Advanced Parallelizing Compiler. PARELEC '02: Proceedings of the International Conference on Parallel Computing in Electrical Engineering, IEEE Computer Society, page 105, Washington, DC, USA, 2002.

[14] Hideki Saito, Nicholas Stavrakos, Steven Carroll, Constantine Polychronopoulos, and Alex Nicolau. The design of the PROMIS compiler. In Proceedings of the International Conference on Compiler Construction., 1999.

[15] Mohammad R. Haghighat and Constantine D. Polychronopoulos. Symbolic Program Analysis and Optimization for Parallelizing Compilers. In Conference Record of the 5th Workshop on Languages and Compilers for Parallel Computing, Yale University, Department of Computer Science, 1992.

[16] E. Ayguadé and C. R. Calidonna and J. Corbalan and M. Giordano and M. Gonzalez and H. C. Hoppe and J. Labarta and M. Mango Furnari and X. Martorell and N. Navarro and D. S. Nikolopoulos and J. Oliver and T. S. Papatheodorou. NANOS: Effective Integration of Fine-grain Parallelism Exploitation and Multiprogramming, 1999.

[17] Jesus Labarta. Manual Parallelization. NANOS: Effective Integration of Fine-grain Parallelism Exploitation and Multiprogramming (Project No.21907), Deliverable M2.D5, 1998.

[18] Universitat Politecnica de Catalunya (CEPBA), PALLAS GmbH, Consiglio Nazionale delle Ricerche (Istituto di Cibernetica), University of Patras (LHPCA). The NANOS Environment User Guide. 1999.

[19] Universitat Polit`ecnica de Catalunya (CEPBA), PALLAS GmbH, Consiglio Nazionale delle Ricerche (Istituto di Cibernetica), University of Patras (LHPCA). RTL Calls Generator. NANOS: Effective Integration of Fine-grain Parallelism Exploitation and Multiprogramming (Project No.21907), Deliverable M2.D4, 1998.

[20] M. B. Girkar, M. R. Haghighat, C. L. Lee, B. P. Leung, and D. A. Schouten. Parafrase-2 user's manual. Technical report, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champagn, July 1991.

[21] D. Kulkarni, M. Stumm, "*Loop and Data Transformations: A tutorial*", Technical Report CSRI-337, Computer Systems Research Institute, University of Toronto, June 1993.

[22] D.F. Bacon, S.L. Graham and O.J. Sharp, "Compiler Transformations for High Performance Computing," Computer Science Division Technical Report UCB--CSD--93--781, U.C. Berkeley, Oct. 1993.

[23] http://www.ace.nl/compiler/cosy.html

# ΠΑΡΑΡΤΗΜΑ

Στο παράρτημα θα συμπεριλάβουμε μερικά ολοκληρωμένα παραδείγματα μετατροπής γνωστών αλγορίθμων από C σε μTC:

<u>Παράδειγμα 1</u>: Υπολογισμός αριθμών Fibonacci

```
int N;
int a[N];
void main(void)
{  if(N<=2) return;
   a[1]=1;
   a[2]=1;
   for(i=3;i<=N;i++)
     a[i]=a[i-1]+a[i-2];
}
```

→

```
int N;
int a[N];
create(fid1;1;1;0)
{
    if(N<=2) break;
    a[1]=1;
    a[2]=2;
    int g_1=A[2];
    int g_2=A[1];

    create(fid2;3;N;1)
   {    index i;

        shared int s_1=g_1;
        shared int s_2=g_2;

        int t0;
        int t_1;
        int t_2;

        t_1=s_1;
        t_2=s_2;
```

```
            t0=t_1+t_2;
            a[i]=t0;

            s_1=t0;
            s_2=t_1;
        } sync(fid2);
    } sync(fid1);
```

<u>Παράδειγμα 2</u>: Υπολογισμός Παραγοντικού

```
int N;                    →
int a[N];
void main(void)
{  if(N<=1) return;
   a[1]=1;
   for(i=2;i<=N;i++)
     a[i]=a[i-1]*i;
}
```

```
int N;
int a[N];
create(fid1;1;1;0)
{
    if(N<=1) break;
    a[1]=1;
    int g_1=A[1];

    create(fid2;2;N;1)
    {   index i;
        shared int s_1=g_1;

        int t0;
        int t_1;

        t_1=s_1;

        t0=t_1*i;
        a[i]=t0;

        s_1=t0;
   }sync(fid2);
}sync(fid1);
```

# ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ονοματεπώνυμο Ευγενίδου Δέσποινα

Ημερομηνία γέννησης 02-12-1983, Θεσσαλονίκη

Σπουδές

2005- Τμήμα Μεταπτυχιακών Σπουδών του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων

2005 Πτυχίο Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων, Βαθμός 6,79

2001 Απολυτήριο Λυκείου Ελληνο-γαλλική σχολή Καλαμαρί, Θεσσαλονίκη, Βαθμός 18,5

Ξένες γλώσσες

Αγγλικά

- Certificate of Proficiency in English Michigan, March 2000
- Certificate of Proficiency in English Cambridge, Grade C, Dec 1999

Γαλλικά

- Delf 2(A5 & A6), Institut français de Thessalonique, 2002

Ιταλικά

- Βεβαίωση παρακολούθησης «Ιταλικά 1», ΚΕΕ Ιωαννίνων, 2008