ΒΙΒΛΙΟΘΗΚΗ
ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ
026000265346

ΥΛΟΠΟΙΗΣΗ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ ΓΙΑ ΤΟΝ ΠΑΡΑΛΛΗΛΟΠΟΙΗΤΙΚΟ ΜΕΤΑΦΡΑΣΤΗ ΟΜΡΙ

116
ΜΠΛΕ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

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

από τον

Σπύρο Μελισσόβα

ως μέρος των Υποχρεώσεων

για τη λήψη

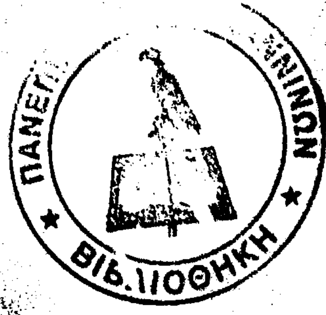
του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

Οκτώβριος 2006

Αρ. εισ.: 1880 200..



ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ πολύ τον επιβλέποντά μου, Επίκουρο Καθηγητή κ. Βασίλειο Β. Δημακόπουλο και τον υποψήφιο διδάκτορα Άλκη Γεωργόπουλο για την πολύτιμη βοήθειά τους στην ολοκλήρωση αυτής της εργασίας. Χωρίς την υπομονή και την καθοδήγησή τους δε θα ήταν δυνατή η ολοκλήρωσή της.

ΠΕΡΙΕΧΟΜΕΝΑ

Σελ

ΕΥΧΑΡΙΣΤΙΕΣ ii
ΠΕΡΙΕΧΟΜΕΝΑ iii
ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ vi
ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ viii
ΕΠΕΞΗΓΗΣΕΙΣ ΣΥΜΒΟΛΙΣΜΩΝ ix
ΠΕΡΙΛΗΨΗ x
EXTENDED ABSTRACT IN ENGLISH xii
ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ 1
1.1. Αναγκαιότητα ύπαρξης του προτύπου 1
1.2. Ένα απλό παράδειγμα μετατροπής κώδικα 2
1.3. Οδηγίες προς τον προεπεξεργαστή 5
1.4. Στόχοι 6
1.5. Δομή της Διατριβής 7
ΚΕΦΑΛΑΙΟ 2. ΤΟ ΠΡΟΤΥΠΟ OPENMP 9
2.1. Εισαγωγή 9
2.2. Προγραμματιστικό μοντέλο του OpenMP 10
2.3. Οδηγίες OpenMP για C/C++ 11
2.3.1. Σύνταξη δομών και οδηγιών OpenMP 12
2.3.2. Παράλληλες Περιοχές 12
2.3.3. Φράσεις οδηγιών του OpenMP 15
2.3.4. Περιοχές Διαμοιρασμού Εργασίας 16
2.3.5. Συνδυασμένες παράλληλες περιοχές διαμοιρασμού εργασίας 20
2.3.6. Οδηγίες συγχρονισμού για C/C++ 23
2.3.7. Εξειδικευμένη Οδηγία: threadprivate 28
2.3.8. Κανόνες εμφωλιασμού 29
2.4. Μεταφραστές OpenMP 29
2.4.1. Εμπορικοί μεταφραστές 30
2.4.2. Ερευνητικοί μεταφραστές 31
ΚΕΦΑΛΑΙΟ 3. ΑΝΑΛΥΣΗ ΚΩΔΙΚΑ ΕΙΣΟΔΟΥ 32
3.1. Εισαγωγή 32
3.2. Ο λεκτικός αναλυτής 33
3.3. Ο συντακτικός αναλυτής 33
3.3.1. Συντακτικός έλεγχος της εισόδου 34
3.3.2. Σημασιολογικός έλεγχος της εισόδου 35
3.3.3. Δημιουργία μη τερματικών κόμβων του δέντρου 35
ΚΕΦΑΛΑΙΟ 4. ΣΥΜΠΑΓΕΣ ΣΥΝΤΑΚΤΙΚΟ ΔΕΝΤΡΟ 40
4.1. Εισαγωγή 40

4.2. Ιεραρχία κλάσεων 41
4.2.1. Τερματικές και μη τερματικές κλάσεις 42
4.2.2. Συλλογές μεταβλητών και τύπων χρήστη 54
4.3. Σύνδεση τερματικών και μη τερματικών αντικειμένων 57
4.4. Εξαγωγή δέντρου σε μορφή XML 57
ΚΕΦΑΛΑΙΟ 5. ΜΕΤΑΤΡΟΠΗ ΣΕ ΠΟΛΥΝΗΜΑΤΙΚΟ ΚΩΔΙΚΑ 60
5.1. Εισαγωγή 60
5.2. Παραγωγή συναρτήσεων νημάτων 61
5.3. Μεταβλητές 64
5.3.1. private (list) 67
5.3.2. firstprivate (list) 67
5.3.3. lastprivate (list) 67
5.3.4. reduction (operator : list) 67
5.3.5. default (shared|none) 68
5.3.6. shared (list) 69
5.3.7. copyin (list) 69
5.3.8. copyprivate (list) 69
5.4. Φωλιασμένες οδηγίες 69
5.5. Κωδικοποίηση ιδιοτήτων μεταβλητών 70
5.6. Ανάλυση οδηγιών 72
5.6.1. Αναζήτηση για κόμβους genomp πρώτου επιπέδου 73
5.6.2. Κλήση αντίστοιχης μεθόδου 73
5.6.3. Εύρεση φωλιασμένων οδηγιών 73
5.6.4. Κλήση αντίστοιχης μεθόδου 73
5.6.5. Επιστροφή ελέγχου στην αρχική μέθοδο μετασχηματισμού 74
5.7. Τροποποίηση ΣΣΔ 74
5.7.1. Αφαίρεση υπόδεντρου από το ΣΣΔ 75
5.7.2. Προσθήκη υπόδεντρου στο ΣΣΔ 76
5.8. Μετασχηματισμοί 80
5.8.1. omp parallel 81
5.8.2. omp for 93
5.8.3. omp sections 106
5.8.4. omp parallel for/sections 109
5.8.5. Παράδειγμα μετασχηματισμού 113
ΚΕΦΑΛΑΙΟ 6. ΑΠΟΤΕΛΕΣΜΑΤΑ 118
6.1. Εισαγωγή 118
6.2. Υπολογισμός π 118
ΚΕΦΑΛΑΙΟ 7. συμπερασματα και μελλοντικη εργασια 121
7.1. Συμπεράσματα 121
7.1.1. Ελλείψεις 121
7.2. Μελλοντική εργασία 122
7.2.1. Επεκτάσεις 122
7.2.2. Βελτιστοποιήσεις 122
7.2.3. Απαλοιφή πλεοναζόντων φραγμάτων 122
ΑΝΑΦΟΡΕΣ 124
ΠΑΡΑΡΤΗΜΑ 126
Παραρτημα Α 127
Παραρτημα Β 203
ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ 204

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

Πίνακας Σελ

Πίνακας 1.1 Σειριακό πρόγραμμα C 2
Πίνακας 1.2 Μετατροπή προγράμματος σε πολυνηματικό 3
Πίνακας 1.3 Πρόγραμμα C με οδηγίες OpenMP 5
Πίνακας 2.1 (α) Σειριακό πρόγραμμα C και (β) Σειριακό πρόγραμμα C με οδηγία OpenMP 11
Πίνακας 2.2 Σύνταξη OpenMP οδηγιών 12
Πίνακας 2.3 Παράδειγμα κώδικα με μια δομή OpenMP 14
Πίνακας 2.4 Η οδηγία for 17
Πίνακας 2.5 Η οδηγιά sections 19
Πίνακας 2.6 Η οδηγία single 20
Πίνακας 2.7 Η οδηγία parallel for 21
Πίνακας 2.8 Η οδηγία parallel sections 22
Πίνακας 2.9 Η οδηγία master 23
Πίνακας 2.10 Η οδηγία critical 24
Πίνακας 2.11 Η οδηγία barrier 25
Πίνακας 2.12 Η οδηγία atomic 25
Πίνακας 2.13 Η οδηγία flush 26
Πίνακας 2.14 Η οδηγία ordered 27
Πίνακας 2.15 Η οδηγία threadprivate 28
Πίνακας 3.1 Παράδειγμα γραμματικού κανόνα 35
Πίνακας 3.2 Παράδειγμα απλού γραμματικού κανόνα 36
Πίνακας 3.3 Παράδειγμα χρήσης απαριθμητή σε γραμματικό κανόνα 37
Πίνακας 3.4 Παράδειγμα αναδρομικού γραμματικού κανόνα 38
Πίνακας 4.1 Απόσπασμα γραμματικού κανόνα 47
Πίνακας 4.2 Παράδειγμα σύνθετου συνόλου εντολών 49
Πίνακας 4.3 Απόσπασμα συντακτικών κανόνων bison 49
Πίνακας 4.4 Απόσπασμα συντακτικών κανόνων bison 50
Πίνακας 4.5 Δήλωση της δομής vardata 52
Πίνακας 4.6 Η δομή ltstr 53
Πίνακας 4.7 Παράδειγμα προγράμματος OpenMP 58
Πίνακας 4.8 Έξοδος XML του μεταφραστή 58
Πίνακας 5.1 Δημιουργία νήματος με χρήση POSIX threads 61
Πίνακας 5.2 Πρόγραμμα με οδηγία OpenMP 62
Πίνακας 5.3 Δημιουργία νημάτων με χρήση της βιβλιοθήκης χρόνου εκτέλεσης του OMPi 63
Πίνακας 5.4 Δήλωση κενής δομής για τις μεταβλητές νήματος 64
Πίνακας 5.5 Παράδειγμα καθολικής και τοπικής μεταβλητής σε τμήμα κώδικα 65
Πίνακας 5.6 Σύνταξη φράσεων οδηγιών 66

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Πίνακας 5.7 Τελεστές και αρχικές τιμές για τη φράση reduction 68
Πίνακας 5.8 Η δήλωση του χάρτη μεταβλητών 70
Πίνακας 5.9 Η δομή που χρησιμοποιείται στον χάρτη μεταβλητών 70
Πίνακας 5.10 Η υπορουτίνα «μικρότερο από» του χάρτη 71
Πίνακας 5.11 Οι τιμές του απαριθμητή var_type (τοπική μεταβλητή στον παραπάνω πίνακα σημαίνει πως η μεταβλητή είναι ορισμένη σε επίπεδο του κώδικα πριν από την οδηγία OpenMP αλλά όχι σε καθολικό επίπεδο) 71
Πίνακας 5.12 Η εικονική συνάρτηση transform() 77
Πίνακας 5.13 Παράδειγμα χρήσης της δομής stringstream 79
Πίνακας 5.14 Εύρεση μιας δομής OpenMP 80
Πίνακας 5.15 Παράδειγμα χρήσης της δομής #pragma omp parallel 114
Πίνακας 6.1 Κώδικας υπολογισμού του π 118

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

Σχήμα Σελ

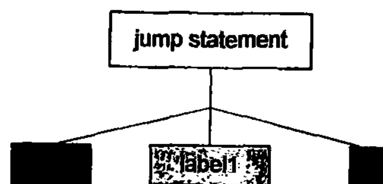
Σχήμα 3.1 Παράδειγμα κατασκευής υποδέντρου 36
Σχήμα 3.2 Απλοποιημένο παράδειγμα χάρτη μεταβλητών 37
Σχήμα 3.3 Υποδέντρο κατασκευασμένο από αναδρομικό κανόνα 39
Σχήμα 3.4 Επίπεδο υπόδεντρο κατασκευασμένο από αναδρομικό κανόνα 39
Σχήμα 4.1 Συμπαγές Συντακτικό Δέντρο για την έκφραση a + b * c 40
Σχήμα 4.2 Αφηρημένο Συντακτικό Δέντρο για την έκφραση a + b * c 40
Σχήμα 4.3 Η ιεραρχία των κλάσεων τερματικών και μη τερματικών αντικειμένων 41
Σχήμα 4.4 Η ιεραρχία των κλάσεων για δηλώσεις μεταβλητών και τύπων 42
Σχήμα 4.5 Η κλάση generic_component 43
Σχήμα 4.6 Η κλάση genterm 44
Σχήμα 4.7 Η κλάση genident 45
Σχήμα 4.8 Η κλάση gennode 46
Σχήμα 4.9 Παράδειγμα κατασκευής υποδέντρου 47
Σχήμα 4.10 Η κλάση genblock 48
Σχήμα 4.11 Απόσπασμα του ΣΣΔ 51
Σχήμα 4.12 Η κλάση gendecl 51
Σχήμα 4.13 Υπόδεντρο δήλωσης μεταβλητής 52
Σχήμα 4.14 Η κλάση genomp 54
Σχήμα 4.15 Η κλάση generic_collection 55
Σχήμα 4.16 Η κλάση declarations_collection 56
Σχήμα 4.17 Η κλάση typenames_collection 57
Σχήμα 5.1 Απόσπασμα του ΣΣΔ για την οδηγία #pragma omp flush(var) 75
Σχήμα 5.2 Υπόδεντρο ανάλυσης της συμβολοσειράς _omp_flush((void *) &var); με χρήση της εικονικής συνάρτησης transform() 78
Σχήμα 5.3 Απόσπασμα του δέντρου στο επίπεδο της δομής #pragma omp parallel 84
Σχήμα 6.1 Γράφημα χρονομέτρησης για τον υπολογισμό του π 120

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΕΠΕΞΗΓΗΣΕΙΣ ΣΥΜΒΟΛΙΣΜΩΝ

Συμπαγές Συντακτικό Δέντρο (Σ.Σ.Δ) – Concrete Syntax Tree (CST)

Στις γραφικές απεικονίσεις των κόμβων ενός υποδέντρου, με γκρι χρώμα εμφανίζονται οι μη τερματικοί κόμβοι, με πράσινο χρώμα οι τερματικοί κόμβοι και με πορτοκαλί χρώμα οι αναγνωριστές (identifiers).



ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΠΕΡΙΛΗΨΗ

Σπύρος Μελισσόβας του Βασιλείου και της Παρασκευής. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Οκτώβριος, 2006. Υλοποίηση συντακτικού αναλυτή για τον παραλληλοποιητικό μεταφραστή OMPi. Επιβλέποντας: Βασίλειος Β. Δημακόπουλος.

Η εργασία αυτή μελετά την υλοποίηση ενός μεταφραστή προγραμμάτων από πηγαίο κώδικα σε πηγαίο κώδικα (source to source) που συμφωνεί με το πρότυπο OpenMP 2.5.

Ο μεταφραστής OMPi έχει υλοποιηθεί παλιότερα σε απλή γλώσσα C με χρήση ενός περάσματος ανάλυσης κώδικα, με τη βοήθεια των εργαλείων GNU flex και bison. Στην παρούσα μελέτη αναλύεται η υλοποίηση εκ νέου του OMPi με τα ίδια εργαλεία σε γλώσσα C++, ώστε να διευκολυνθεί η δημιουργία ενός συντακτικού δέντρου ανάλυσης του κώδικα εισόδου. Με διεπαφή το συντακτικό δέντρο υλοποιείται το δεύτερο πέρασμα του μεταφραστή, το οποίο παράγει, με βάση τις οδηγίες OpenMP του κώδικα, πολυνηματικό κώδικα ισοδύναμο με τον αρχικό σειριακό κώδικα.

Στην εργασία αυτή αναλύονται ο λεκτικός και συντακτικός αναλυτής που υλοποιήθηκε, το ενδιάμεσο Συμπαγές Συντακτικό Δέντρο και οι μετασχηματισμοί του κώδικα εισόδου, ώστε να παραχθεί πολυνηματικός κώδικας ισοδύναμος με τον αρχικό.

Με την υλοποίηση αυτή είναι πλέον δυνατοί προχωρημένοι μετασχηματισμοί καθώς και βελτιστοποιήσεις στον παραγόμενο κώδικα προς αύξηση των επιδόσεων.

EXTENDED ABSTRACT IN ENGLISH

Melissovas, Spyros, V. MSc, Computer Science Department, University of Ioannina, Greece. October, 2006. Syntax analyzer implementation for the OMPi parallelizing compiler. Thesis Supervisor: Vassilios V. Dimakopoulos.

This thesis deals with the implementation of a source to source program compiler which complies with the OpenMP 2.5 standard.

The OMPi compiler has already been implemented using a one-pass plain C language analysis, with the GNU flex and bison tools. In this thesis a new OMPi implementation is analyzed, using the same tools in the C++ language, so a syntax tree creation becomes easier. Using the syntax tree as an interface, a second pass is implemented, which creates multithreaded code equal to the initial serial code, guided by the inserted OpenMP directives.

In this thesis there is an analysis of the lectical and syntax analyzer, the intermediate Concrete Syntax Tree and the transformations of the input code, which produce multithreaded code equal to the original.

Using this implementation it is possible to perform advanced code transformations as well as optimizations to the produced code in order to maximize performance.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

1.1. Αναγκαιότητα ύπαρξης του προτύπου

1.2. Ένα απλό παράδειγμα μετατροπής κώδικα

1.3. Οδηγίες προς τον προεπεξεργαστή

1.4. Στόχοι

1.5. Δομή της Διατριβής

1.1. Αναγκαιότητα ύπαρξης του προτύπου

Τα τελευταία χρόνια παρουσιάζεται μια συνεχόμενα ανοδική τάση μετάβασης από υπολογιστικά συστήματα ενός επεξεργαστή σε υπολογιστικά συστήματα πολλών επεξεργαστών. Η τάση αυτή τροφοδοτείται από τα εξής γεγονότα:

- Η CMOS τεχνολογία φτάνει στα όριά της ως προς την πυκνότητα και την ταχύτητα ρολογιού των τρανζίστορ που μπορούν να παραχθούν σε ένα ολοκληρωμένο κύκλωμα.
- Η αύξηση στη συχνότητα ρολογιού ενός ολοκληρωμένου κυκλώματος συντελεί σε φαινόμενα υπερθέρμανσης, που απαιτούν ειδικές διατάξεις ψύξης, και έχει σχετικά μικρά όρια βελτίωσης με βάση την παρούσα τεχνολογία.
- Τα υπολογιστικά συστήματα πολλών επεξεργαστών έχουν γίνει αρκετά προσιτά ακόμη και για χρήση σε χώρους όπου πριν ήταν απαγορευτικό λόγω όγκου και κόστους.
- Η κατασκευή ολοκληρωμένων κυκλωμάτων δύο ή και παραπάνω πυρήνων (dual core, quad core κ.ο.κ) έχει χαμηλώσει το κόστος κτήσης ακόμη περισσότερο γιατί το υπολογιστικό σύστημα εγκατάστασης δεν απαιτεί ειδική κατασκευή (μόνο μία θέση για επεξεργαστή αντί για δύο και κοινά ολοκληρωμένα υποστήριξης επεξεργαστή).

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Η μετάβαση σε πολυεπεξεργαστικά συστήματα φέρει το προφανές όφελος του πολλαπλασιασμού της επεξεργαστικής ισχύος. Μια απλοϊκή σκέψη είναι πως οποιοδήποτε υπάρχον πρόγραμμα αυτόματα εκμεταλλεύεται αυτή την επιπλέον επεξεργαστική ισχύ και εκτελείται με πολλαπλάσια ταχύτητα, ίση με Ν φορές μεγαλύτερη από ότι σε ένα αντίστοιχο σύστημα ενός επεξεργαστή, αν Ν ο αριθμός των επεξεργαστών.

Κάτι τέτοιο δεν είναι απαραίτητα αληθές και εξαρτάται από τον αριθμό των νημάτων με τον οποίο λειτουργεί το πρόγραμμα, καθώς και την υλοποίηση αυτών. Αν κάνουμε την παραδοχή ότι κάποιο συγκεκριμένο πρόγραμμα θα αποτελεί τη μοναδική διεργασία που εκτελείται στο σύστημα, πρέπει ο αριθμός των νημάτων εκτέλεσης να είναι ίσος ή μεγαλύτερος από τον αριθμό των επεξεργαστών του συστήματος. Σε οποιαδήποτε άλλη περίπτωση, η επιτάχυνση θα είναι μικρότερη από την αναμενόμενη. Ειδικά στην περίπτωση που το πρόγραμμα δεν είναι γραμμένο ώστε να εκτελείται με πολλαπλά νήματα, εκτελείται μόνο με ένα, με αποτέλεσμα η ταχύτητα εκτέλεσης να είναι ίδια είτε σε έναν είτε σε πολλούς επεξεργαστές.

Δημιουργείται λοιπόν το ζήτημα μετατροπής των υπαρχόντων προγραμμάτων σε πολυνηματικά, ώστε να μπορούν να εκμεταλλευτούν την επιπλέον επεξεργαστική ισχύ. Η μετατροπή του υπάρχοντος κώδικα απαιτεί αρκετή μελέτη και δεν είναι πάντα εύκολη.

1.2. Ένα απλό παράδειγμα μετατροπής κώδικα

Ας θεωρήσουμε το απλό σειριακό πρόγραμμα σε γλώσσα C (Πίνακας 1.1), το οποίο αναθέτει σε κάθε θέση ενός πίνακα arr το τετράγωνο της τιμής που ήδη είχε.

Πίνακας 1.1 Σειριακό πρόγραμμα C

```
#include <stdio.h>

int main()
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
{
      int arr[100], i;

      /* Αρχικοποίηση πίνακα */
      ...

      for(I = 0; I < 100; i++)
      {
        /* Ενημέρωση θέσης πίνακα με το τετράγωνο της τιμής της */
      }
}
```

Αυτό το πρόγραμμα εκτελεί εκατό επαναλήψεις για να επιτελέσει την εργασία του. Αν μεταφραστεί με έναν τυπικό μεταφραστή C σε οποιοδήποτε πολυεπεξεργαστικό σύστημα, ο κώδικας που θα παραχθεί θα εκτελεστεί με ένα νήμα, οπότε θα εκμεταλλευτεί μόνο τον ένα επεξεργαστή.

Το ίδιο πρόγραμμα μπορεί απλοποιημένα να μετατραπεί όπως δείχνει ο Πίνακας 1.2, το οποίο θα εκτελεστεί με πολλαπλά νήματα (εκατό στην περίπτωση αυτή), το καθένα από τα οποία θα ενημερώνει μια θέση του πίνακα.

Πίνακας 1.2 Μετατροπή προγράμματος σε πολυνηματικό

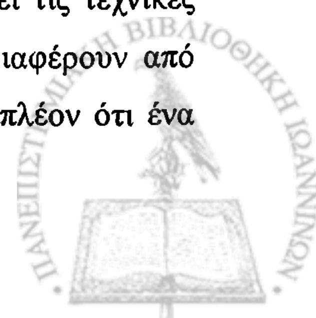
```
#include <stdio.h>
#include <pthread.h>

int arr[100], i;

int myfunc(int num)
{
      /* Ενημέρωση της θέσης πίνακα που αντιστοιχεί στον αριθμό
νήματος */
}

int main()
```

```
{

    /* Αρχικοποίηση πίνακα */
    ...
    for(I = 0; I < 100; i++)
    {
    /* Δημιουργία νήματος που εκτελεί τη συνάρτηση myfunc */
    }

    for(I = 0; I < 100; i++)
    {
    /* Αναμονή και καταστροφή νημάτων */
    }

}
```

Ο Πίνακας 1.2 δείχνει ένα πρόγραμμα που μπορεί να εκτελεστεί σε υπολογιστικά συστήματα είτε ενός είτε πολλών επεξεργαστών. Στη δεύτερη περίπτωση θα εκμεταλλευτεί μέχρι και 100 επεξεργαστές, αφού χρησιμοποιεί 100 νήματα για να κάνει τους υπολογισμούς του. Το νούμερο αυτό είναι μάλλον υπερβολικό. Μια πιο λογική παραλλαγή θα μπορούσε να χρησιμοποιεί 4 νήματα που θα ενημέρωναν από 25 θέσεις του πίνακα το καθένα.

Η μετατροπή που χρειάστηκε το αρχικό σειριακό πρόγραμμα σε γλώσσα προγραμματισμού C ήταν αρκετά μικρή. Αφορούσε τη δήλωση του πίνακα ως καθολικού ώστε να είναι ορατός από όλα τα νήματα και τη δημιουργία μιας συνάρτησης που περικλείει τον κώδικα υπολογισμού που θέλουμε να μοιράσουμε. Επιπλέον ανάλογα με την απαίτηση μπορεί να εισαχθεί και κώδικας που θα διαιρεί την εργασία σε όσο μεγάλα ή μικρά τμήματα θέλουμε.

Η μετατροπή αυτή απαιτεί όμως από τον προγραμματιστή να γνωρίζει τις τεχνικές λεπτομέρειες υλοποίησης των νημάτων, οι οποίες είναι δυνατό να διαφέρουν από λειτουργικό σύστημα σε λειτουργικό σύστημα. Αν ληφθεί υπόψη επιπλέον ότι ένα


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

τυπικό πρόγραμμα αριθμητικών υπολογισμών είναι δυνατό να απαιτεί δεκάδες μετατροπές με άγνωστη επίπτωση στις δηλωμένες μη καθολικές μεταβλητές, όπως και σε άλλες παραμέτρους του προγράμματος, αντιλαμβάνεται κανείς πως απαιτείται η ύπαρξη ενός μακροεργαλείου. Ένα σύνολο εργαλείων που απλοποιούν τη μετάβαση από σειριακό κώδικα ενός νήματος σε πολυνηματικό κώδικα που εκμεταλλεύεται καλύτερα την ισχύ των πολυεπεξεργαστικών συστημάτων είναι οι μεταφραστές που βασίζονται στο πρότυπο OpenMP.

1.3. Οδηγίες προς τον προεπεξεργαστή

Το πρότυπο OpenMP κάνει αρκετά εύκολη τη μετατροπή ήδη υπάρχοντος κώδικα γραμμένου σε γλώσσες προγραμματισμού C, C++ και FORTRAN. Ο προγραμματιστής δεν χρειάζεται να γνωρίζει όλες τις τεχνικές λεπτομέρειες χειρισμού του πολυνηματικού κώδικα, αλλά μια σειρά από οδηγίες που πρέπει να εισαχθούν στον αρχικό σειριακό κώδικα στα κατάλληλα σημεία.

Όταν ο κώδικας με ενσωματωμένες οδηγίες OpenMP διαβαστεί από έναν μεταφραστή συμβατό με το πρότυπο, τα τμήματα κώδικα που περικλείονται από τις οδηγίες αυτές μετασχηματίζονται κατάλληλα, ώστε να εκτελεστούν με πολλαπλά νήματα. Η μέθοδος μετασχηματισμού είναι θέμα υλοποίησης του μεταφραστή.

Για παράδειγμα, ο σειριακός κώδικας (Πίνακας 1.1) θα απαιτούσε μόνο τις αλλαγές που απεικονίζει ο Πίνακας 1.3.

Πίνακας 1.3 Πρόγραμμα C με οδηγίες OpenMP

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int arr[100], i;
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
      /* Αρχικοποίηση πίνακα */
      ...
#pragma omp parallel for
      for(I = 0; I < 100; i++)
      {
        /* Ενημέρωση θέσης πίνακα με το τετράγωνο της τιμής της */
      }
}
```

Οι αλλαγές που έγιναν ήταν οι:

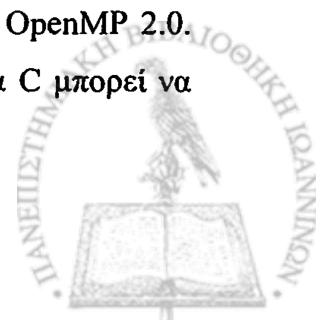
- προσθήκη του αρχείου δηλώσεων omp.h στην κορυφή του κώδικα και
- προσθήκη ενός #pragma omp parallel for ακριβώς πριν από την επανάληψη.

Όταν ο προεπεξεργαστής του μεταφραστή συναντήσει την οδηγία αυτή, αναλαμβάνει αυτόματα να παράγει κώδικα τέτοιο που θα αναλάβει τον καταμερισμό του βρόχου επανάληψης σε τμήματα εργασίας Τα τμήματα αυτά θα ανατεθούν σε μια ομάδα νημάτων για εκτέλεση, μέχρι να εκτελεστούν όλα. Μετά το τέλος του βρόχου επανάληψης που ακολουθεί την οδηγία #pragma, το πρόγραμμα θα εκτελεστεί ως σειριακό και κατά τον τερματισμό του θα καταστραφεί και η ομάδα νημάτων που δημιουργήθηκε.

Ο τελικός κώδικας που μεταφράζεται περιέχει όλες τις απαραίτητες δομές και δηλώσεις και το εκτελέσιμο πρόγραμμα εκτελείται πολυνηματικά.

1.4. Στόχοι

Στόχος αυτής της διατριβής είναι να αναδιοργανώσει και να επεκτείνει τον μεταφραστή OMPi, παρέχοντας μια εντελώς νέα και βελτιωμένη αρχικτεκτονική η οποία είναι εύκολα επεκτάσιμη και παρέχει δυνατότητες για βελτιστοποιήσεις παραγόμενου κώδικα. Ο μεταφραστής OMPi είναι ένα εργαλείο μετάφρασης προγραμμάτων για κώδικα γραμμένο στη γλώσσα προγραμματισμού C με ενσωματωμένες οδηγίες παραλληλοποίησης σύμφωνες με το πρότυπο OpenMP 2.0. Με βάση τις οδηγίες αυτές, ένα απλό σειριακό πρόγραμμα σε γλώσσα C μπορεί να


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

μετασχηματιστεί σε πολυνηματικό κώδικα που εκμεταλλεύεται τις δυνατότητες συστημάτων με πολλούς επεξεργαστές.

Επιμέρους στόχοι:

- Η αναδιοργάνωση του συντακτικού αναλυτή ώστε να υπάρχει πλήρης συμβατότητα με τα πρότυπα ANSI/ISO C99 [1] και OpenMP 2.5 [7].
- Η υλοποίηση από την αρχή σε αντικειμενοστραφή κώδικα C++ ώστε να είναι δυνατή η απομόνωση της ανάλυσης της εισόδου από την επεξεργασία της
- Η παραγωγή ενδιάμεσης αναπαράστασης με χρήση Συμπαγούς Συντακτικού Δέντρου (ΣΣΔ) - Concrete Syntax Tree (CST).
- Η υλοποίηση δεύτερου περάσματος επεξεργασίας του κώδικα εισόδου, που περιλαμβάνει:
    - την υλοποίηση των μετασχηματισμών που απαιτούνται για την παραγωγή πολυνηματικού κώδικα και
    - την παραγωγή ενός μηχανισμού μετασχηματισμών που θα είναι εύκολος στην κατανόηση και κατά συνέπεια τη συντήρηση και την επέκταση από τρίτους προγραμματιστές

1.5. Δομή της Διατριβής

Η διατριβή περιέχει πέντε κεφάλαια:

- Στο κεφάλαιο 2 ανακεφαλαιώνεται το πρότυπο OpenMP με όλες τις οδηγίες που περιλαμβάνει. Επίσης γίνεται ανασκόπηση του OMPi και άλλων μεταφραστών
- Στο κεφάλαιο 3 περιγράφεται ο λεκτικός και συντακτικός αναλυτής μαζί με μια εισαγωγή στο Συντακτικό Δέντρο που παράγεται
- Στο κεφάλαιο 4 περιγράφεται το Συμπαγές Συντακτικό Δέντρο και οι μέθοδοι για την προσπέλασή του
- Στο κεφάλαιο 5 περιγράφονται οι μετασχηματισμοί οι οποίοι ξεκινώντας από την πληροφορία του ΣΣΔ παράγουν ισοδύναμο πολυνηματικό κώδικα.

ΚΕΦΑΛΑΙΟ 2. ΤΟ ΠΡΟΤΥΠΟ OPENMP

2.1. Εισαγωγή
2.2. Προγραμματιστικό μοντέλο του OpenMP
2.3. Οδηγίες OpenMP για C/C++
2.4. Άλλοι μεταφραστές OpenMP

2.1. Εισαγωγή

Το OpenMP είναι μια διεπαφή εφαρμογών προγραμμάτων (Application Program Interface - API) το οποίο χρησιμοποιείται για να παραλληλίσουμε προγράμματα σε αρχιτεκτονικές συστημάτων κοινής μνήμης. Αποτελείται από τρία API συστατικά: οδηγίες για τον μεταφραστή, συναρτήσεις βιβλιοθηκών και μεταβλητές περιβάλλοντος. Το API του OpenMP έχει οριστεί για τις γλώσσες προγραμματισμού C/C++ και Fortran, ενώ έχει υλοποιηθεί για τις πιο σημαντικές πλατφόρμες όπως το Unix και τα Windows NT, κάτι που το κάνει αρκετά φορητό. Έχει οριστεί από κοινού από τις πιο σημαντικές εταιρίες υλικού και λογισμικού και αναμένεται σε λίγα χρόνια να αποτελέσει ANSI πρότυπο. Πρέπει να επισημάνουμε ότι το OpenMP δεν προορίζεται για αρχιτεκτονικές συστημάτων κατανεμημένης μνήμης, ούτε έχει υλοποιηθεί από όλες τις εταιρίες υλικού και λογισμικού. Επίσης, δεν έχει σχεδιαστεί για να αποδίδει τα μέγιστα στη χρήση της κοινής μνήμης.

Οι στόχοι του OpenMP είναι να παράσχει ένα πρότυπο που θα ισχύει για αρχιτεκτονικές και πλατφόρμες κοινής μνήμης. Ένα πρότυπο που θα είναι μικρό και εύκολα κατανοητό, δηλαδή θα παρέχει ένα απλό και περιορισμένο σύνολο από οδηγίες με το οποίο θα μπορεί να γίνεται σημαντική παραλληλοποίηση. Επίσης, θα πρέπει να είναι εύκολο στη χρήση του.

Το OpenMP παρέχει:

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

τη δυνατότητα παραλληλοποίησης ενός προγράμματος σταδιακά (εν αντιθέσει, για παράδειγμα, με το MPI όπου κάθε φορά γίνεται include όλη η βιβλιοθήκη) και τη δυνατότητα τόσο χονδρού όσο και λεπτού κόκκου παραλληλίας. Τέλος, ο κυριότερος στόχος του OpenMP είναι η φορητότητα.

Στη συνέχεια θα περιγράψουμε το μοντέλο του OpenMP και θα δώσουμε την σύνταξη των δομών/οδηγιών του. Θα δούμε τα είδη των οδηγιών και τις κατάλληλες συνθήκες για αυτά. Επίσης, θα συζητήσουμε την έννοια των παράλληλων περιοχών και την έννοια των παράλληλων περιοχών διαμοιρασμού εργασίας. Το κεφάλαιο ολοκληρώνεται με μια ανασκόπηση εμπορικών και ερευνητικών μεταφραστών OpenMP.

2.2. Προγραμματιστικό μοντέλο του OpenMP

Το προγραμματιστικό μοντέλο του OpenMP είναι βασισμένο σε παραλληλισμό με νήματα. Μια διεργασία κοινής μνήμης αποτελείται από πολλά νήματα και το OpenMP βασίζεται στο χαρακτηριστικό αυτό. Στο OpenMP ο προγραμματιστής μπορεί να ελέγχει και να αυξάνει ή να μειώνει τον αριθμό των νημάτων. Η όλη διαδικασία θυμίζει το μοντέλο fork-join που χρησιμοποιεί η C. Ένα νήμα αρχηγός (master) εκτελεί τις εντολές σειριακά μέχρι να φτάσει την πρώτη παράλληλη περιοχή, γεννώντας ομάδες από νήματα όταν χρειάζεται. Η ομάδα των νημάτων που περιλαμβάνεται σε μια παράλληλη περιοχή εκτελείται παράλληλα. Όταν η ομάδα τελειώσει την εργασία της, τότε τα νήματα συγχρονίζονται και τερματίζουν, αφήνοντας μόνο το νήμα αρχηγό και η διαδικασία αυτή επαναλαμβάνεται όσες φορές χρειάζεται.

Τα νήματα επικοινωνούν μεταξύ τους με κοινές μεταβλητές. Ωστόσο, ο διαμοιρασμός των δεδομένων μπορεί να οδηγήσει σε συνθήκες ανταγωνισμού. Για να ελέγξουμε τις συνθήκες ανταγωνισμού, χρησιμοποιούμε συγχρονισμό. Ο συγχρονισμός, όμως, κοστίζει, άρα κάθε φορά που παραλληλοποιούμε πρέπει να οργανώνουμε τα δεδομένα κατά τέτοιο τρόπο ώστε να ελαχιστοποιούμε την ανάγκη για συγχρονισμό.

Για να γίνει η παραλληλοποίηση, ο προγραμματιστής θα πρέπει να δώσει στον μεταφραστή οδηγίες οι οποίες ενσωματώνονται στον κώδικα της C/C++ ή Fortran. Στη συνέχεια, ο μεταφραστής μεταφράζει αυτές τις οδηγίες και παράγει κλήσεις σε συναρτήσεις βιβλιοθήκης έτσι ώστε να παραλληλοποιήσει τμήματα του κώδικα. Ο Πίνακας 2.1 δείχνει μια απλή μετατροπή σειριακού προγράμματος σε πρόγραμμα με χρήση μιας δομής OpenMP.

Πίνακας 2.1 (α) Σειριακό πρόγραμμα C και (β) Σειριακό πρόγραμμα C με οδηγία OpenMP

(α)	(β)

`void main()`
`{`
`  double x[1000];`
`  for(int i = 0; i < 1000; i++)`
`  {`
`    big_calc(x[i]);`
`  }`
`}`
	`void main()`
	`{`
	`  double x[1000];`
	`#pragma omp parallel for`
	`  for(int i = 0; i < 1000; i++)`
	`  {`
	`    big_calc(x[i]);`
	`  }`
	`}`

Ο αριθμός των νημάτων μπορεί να αλλάξει δυναμικά, είτε μέσα από το ίδιο το πρόγραμμα με κλήση συνάρτησης βιβλιοθήκης, είτε μέσω της μεταβλητής περιβάλλοντος OMP_NUM_THREADS. Για τον συγχρονισμό των νημάτων, όμως, καθώς και για τις εξαρτήσεις των δεδομένων υπεύθυνος είναι αποκλειστικά ο προγραμματιστής. Εξάλλου, το API του OpenMP παρέχει την δυνατότητα τοποθέτησης παράλληλων τμημάτων μέσα σε άλλα παράλληλα τμήματα.

2.3. Οδηγίες OpenMP για C/C++

Σε αυτό το κεφάλαιο, θα αναφερθεί ο τρόπος με τον οποίο συντάσσονται οι οδηγίες και θα ασχοληθούμε με κάποιους γενικούς κανόνες πάνω στην χρήση των οδηγιών καθώς και με τις παράλληλες περιοχές. Επίσης, θα περιγραφούν οι συνθήκες (clauses) οδηγιών του OpenMP και θα ασχοληθούμε με τις Περιοχές Διαμοιρασμού Εργασίας

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

(Work-sharing Constructs), καθώς και με τις παράλληλες περιοχές διαμοιρασμού εργασίας. Στο τέλος, θα αναφερθούμε στο συγχρονισμό μεταξύ των δεδομένων.

2.3.1. Σύνταξη δομών και οδηγιών OpenMP

Ο Πίνακας 2.2 δείχνει περιγραφικά τη σύνταξη μιας οδηγίας OpenMP.

Πίνακας 2.2 Σύνταξη OpenMP οδηγιών

#pragma omp	Όνομα οδηγίας	[συνθήκη1, συνθήκη2, ...]
Απαιτείται για όλες τις οδηγίες C/C++	Ένα υπαρκτό όνομα οδηγίας. Τοποθετείται μετα το pragma και πριν τις συνθήκες	Προαιρετικό. Οι συνθήκες μπορεί να εμφανίζονται με κάθε σειρά

Για παράδειγμα, μια οδηγία θα μπορούσε να ήταν η εξής:

```
#pragma omp parallel default(shared) private(beta,pi)
```

Οι οδηγίες ακολουθούν τις συμβάσεις του προτύπου C/C++. Η γραφή κεφαλαίων ή πεζών έχει σημασία, πρέπει να υπάρχει μόνο ένα όνομα ανά οδηγία και κάθε οδηγία εφαρμόζεται μόνο στο αμέσως επόμενο σύνολο εντολών το οποίο πρέπει να είναι ένα δομημένο σύνολο (block) κώδικα. Επίσης, μακρές οδηγίες μπορούν να συνεχιστούν σε επόμενες γραμμές αρκεί στο τέλος κάθε γραμμής να τοποθετείται ο χαρακτήρας της ανάποδης καθέτου ("\").

2.3.2. Παράλληλες Περιοχές

Μια παράλληλη περιοχή είναι ένα τμήμα κώδικα το οποίο θα εκτελεστεί αυτούσιο από πολλαπλά νήματα. Η σύνταξη μιας παράλληλης περιοχής είναι η εξής:

```
#pragma omp parallel                    \
      private (var1, var2, ...)         \
      shared (var1, var2, ...)          \
      firstprivate (var1, var2, ...)    \
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
        copyin (var1, var2, …)          \
        reduction (operator: var1, var2, …)                \
        if (expression)                 \
        default (shared | none)
{
        [δομημένο τμήμα κώδικα]
}
```

Όταν ένα νήμα φτάσει σε μια οδηγία παραλληλοποίησης (#pragma...), δημιουργεί μια ομάδα από νήματα και το ίδιο γίνεται αρχηγός (master) της ομάδας. Ο αργηχός είναι κι ο ίδιος μέλος της ομάδας. Ο κώδικας της παράλληλης περιοχής αντιγράφεται τόσες φορές όσα τα νήματα και δίδεται σε αυτά για να τον εκτελέσουν. Γενικά, δεν υπάρχει συγχρονισμός μεταξύ των νημάτων. Κάθε νήμα μπορεί να φτάσει σε οποιοδήποτε σημείο μέσα στη παράλληλη περιοχή, σε ακαθόριστη χρονική στιγμή. Στο τέλος της παράλληλης περιοχής υπονοείται ένα φράγμα, πέρα από το οποίο μόνο ο αρχηγός θα συνεχίσει την εκτέλεση του προγράμματος. Εξάλλου, το API παρέχει τη δυνατότητα δημιουργίας παράλληλης περιοχής μέσα σε μια άλλη παράλληλη περιοχή. Η φωλιασμένη αυτή παράλληλη περιοχή, καταλήγει στη δημιουργία μιας καινούριας ομάδας από νήματα η οποία αποτελείται ερήμην από ένα νήμα. Ωστόσο, διάφορες υλοποιήσεις μπορούν να επιτρέψουν παραπάνω από ένα νήμα σε μια φωλιασμένη παράλληλη περιοχή.

Ο αριθμός των νημάτων που δημιουργούνται κατά την είσοδο σε μια παράλληλη περιοχή μπορεί να καθοριστεί από τρεις παράγοντες, οι οποίοι κατά σειρά προτεραιότητας είναι:

- η χρήση της συνάρτησης βιβλιοθήκης omp_set_num_threads()
- ο ορισμός της μεταβλητής περιβάλλοντος OMP_NUM_THREADS και
- η χρήση της φράσης default.

Επίσης, δίνεται η δυνατότητα από το API, να προσαρμόζεται δυναμικά ο αριθμός των νημάτων για μία συγκεκριμένη παράλληλη περιοχή. Αυτό μπορεί να επιτευχθεί με τους εξής δύο τρόπους:

- με τη χρήση της συνάρτησης βιβλιοθήκης omp_set_dynamic() και

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- με τον ορισμό της μεταβλητής περιβάλλοντος OMP_DYNAMIC.

Η αρίθμηση των νημάτων ξεκινάει από τον αριθμό 0, οποίος δίνεται στον αρχηγό κάθε ομάδας και φτάνει στον αριθμό N-1 όπου N ο αριθμός των νημάτων. Η ταυτότητα κάθε νήματος μπορεί να βρεθεί χρησιμοποιώντας τη συνάρτηση βιβλιοθήκης omp_get_thread_num(), ενώ ο συνολικός αριθμός των νημάτων μπορεί να βρεθεί με τη χρήση της συνάρτησης βιβλιοθήκης omp_get_num_threads(). Αν και τα νήματα εκτελούν το ίδιο κομμάτι κώδικα, ωστόσο, θα θέλαμε να εκτελέσουν διαφορετικά μονοπάτια αυτού του κώδικα. Αυτό επιτυγχάνεται με το να δίνουμε διαφορετικό κομμάτι κώδικα σε κάθε νήμα (για παράδειγμα, με τη χρήση if-else) ή, όπως θα δούμε παρακάτω, με τη χρήση οδηγιών διαμοιρασμού εργασίας.

Ο Πίνακας 2.3 παραθέτει τον κώδικα του Hello World στον οποίο αναθέτουμε διαφορετικές εργασίες στο νήμα αρχηγό και στα υπόλοιπα νήματα. Στον κώδικα αυτό, όλα τα νήματα τυπώνουν το Hello World και τον αριθμό τους, ενώ ο αρχηγός τυπώνει το συνολικό αριθμό των νημάτων.

Πίνακας 2.3 Παράδειγμα κώδικα με μια δομή OpenMP

```
#include <omp.h>

main ()
{
  int nthreads, tid;

  /* Δημιουργία ομάδας νημάτων με ιδιωτικές μεταβλητές νημάτων */
#pragma omp parallel private(tid)
  {

    /* Εύρεση και εμφάνιση του αναγνωριστικού νήματος */
    tid = omp_get_thread_num();
    printf("Hello World from thread = %d\n", tid);

    /* Μόνο ο αρχηγός εκτελεί αυτό το τμήμα */
    if (tid == 0)
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
    {
     nthreads = omp_get_num_threads();
     printf("Number of threads = %d\n", nthreads);
    }

  } /* Όλα τα νήματα ενώνονται με τον αρχηγό και τερματίζουν */

}
```

2.3.3. Φράσεις οδηγιών του OpenMP

Οι φράσεις που χρησιμοποιούνται στο OpenMP είναι οι εξής:

- shared(var1,var2,...): Καθολικές μεταβλητές (var1,var2,...) οι οποίες θα προσπελαθούν από όλα τα νήματα (τα νήματα προσπελαύνουν ίδιες θέσεις μνήμης).
- private(var1,var2,...): Κάθε νήμα έχει το δικό του ιδιωτικό αντίγραφο από αυτές τις μεταβλητές (var1,var2,...), για τη διάρκεια της εκτέλεσης της παράλληλης περιοχής.
- firstprivate(var1,var2,...): Ιδιωτικές μεταβλητές (var1,var2,...) που αρχικοποιούνται με την τιμή τους πριν από το δομημένο τμήμα κώδικα όταν εισάγεται μια παράλληλη περιοχή.
- lastprivate(var1,var2,...): Ιδιωτικές μεταβλητές (var1,var2,...) οι οποίες επιστρέφουν τις τιμές τους στις εξωτερικές μεταβλητές μετά από τον τερματισμό ενός παράλληλου τμήματος.
- if(expression): Η παραλληλοποίηση γίνεται μόνο όταν η έκφραση αποτιμάται ως αληθής.
- default(shared|private|none): Καθορίζει την εμβέλεια των μεταβλητών στην παράλληλη περιοχή (αν θα είναι κοινές, ιδιωτικές, ή αν δεν θα έχουν εμβέλεια αντιστοίχως). Ισχύει για τις μεταβλητές για τις οποίες δεν ορίζεται ρητά η εμβέλειά τους από άλλη φράση.
- schedule(type [,chunk]): Ελέγχει τον τρόπο με τον οποίο οι επαναλήψεις ενός βρόγχου θα διαμοιραστούν στα νήματα και χρησιμοποιείται μόνο στην οδηγία for. Ο τύπος του schedule μπορεί να είναι:

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- o STATIC: Οι επαναλήψεις ενός βρόγχου διαμοιράζονται σε κομμάτια μεγέθους chunk και ανατίθενται στατικά στα νήματα. Αν το chunk δεν έχει καθοριστεί, τότε οι επαναλήψεις διαμοιράζονται ισομερώς (αν αυτό είναι δυνατό) και συνεχόμενα μεταξύ των νημάτων.
- o DYNAMIC: Οι επαναλήψεις ενός βρόγχου διαμοιράζονται σε κομμάτια μεγέθους chunk και ανατίθενται δυναμικά στα νήματα. Όταν ένα νήμα τελειώσει με ένα κομμάτι αναλαμβάνει δυναμικά ένα άλλο. Το ερήμην μέγεθος ενός κομματιού είναι 1.
- o GUIDED: Το μέγεθος του κομματιού μειώνεται δυναμικά με κάθε κομμάτι του χώρου επαναλήψεων που αποστέλλεται στα νήματα. Το μέγεθος chunk καθορίζει τον ελάχιστο αριθμό επαναλήψεών που θα πρέπει να αποσταλούν κάθε φορά. Το ερήμην μέγεθος ενός κομματιού είναι 1.
- o RUNTIME: Η απόφαση διαμοιρασμού, αναβάλλεται μέχρι την εκτέλεση του προγράμματος, με τη βοήθεια της μεταβλητής περιβάλλοντος OMP_SCHEDULE. Απαγορεύεται να καθοριστεί μέγεθος chunk για αυτή τη συνθήκη.

- reduction(operator|intrinsic:var1,var2,...): Εξασφαλίζει ότι μια reduction λειτουργία (του τελεστή) μεταξύ των μεταβλητών var1, var2, ... θα εκτελεστεί με ασφάλεια (για παράδειγμα ένα καθολικό άθροισμα).
- copyin(var1, var2,...): Αντιγράφει τις μεταβλητές var1, var2, ...στο τρέχον νήμα. Η μεταβλητή του νήματος αρχηγού χρησιμοποιείται σαν η πηγή αντιγραφής. Η ομάδα των νημάτων αρχικοποιείται με αυτή τη τιμή με την είσοδο στη παράλληλη περιοχή.

2.3.4. Περιοχές Διαμοιρασμού Εργασίας

Μια περιοχή διαμοιρασμού εργασίας χωρίζει τον κώδικα που περικλείεται στην παράλληλη περιοχή μεταξύ των νημάτων της ομάδας της περιοχής. Οι περιοχές διαμοιρασμού εργασίας δεν δημιουργούν καινούρια νήματα και δεν υπάρχει κάποιος συγχρονισμός κατά την είσοδο σε μια τέτοια περιοχή, υπάρχει όμως συγχρονισμός κατά την έξοδο. Υπάρχουν τρεις τύποι περιοχών διαμοιρασμού εργασίας:

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- Οδηγία for: Διαμοιράζει τις επαναλήψεις ενός βρόγχου στην ομάδα της τρέχουσας παράλληλης περιοχής.
- Οδηγία sections: Διαμοιράζει την εργασία σε ξεχωριστά, διακριτά τμήματα. Κάθε τμήμα εκτελείται από διαφορετικό νήμα.
- Οδηγία single: Το τμήμα εκτελείται μόνο από ένα νήμα.

Όπως και στις παράλληλες περιοχές, έτσι και στις περιοχές διαμοιρασμού εργασίας, υπάρχουν κάποιοι περιορισμοί που πρέπει να ικανοποιηθούν:

- για να εκτελεσθεί παράλληλα μια οδηγία, θα πρέπει η περιοχή διαμοιρασμού εργασίας να εσωκλείεται δυναμικά μέσα σε μια παράλληλη περιοχή.
- οι περιοχές διαμοιρασμού εργασίας θα πρέπει να διαμοιράζούν τα δεδομένα (ή την εργασία) σε όλα τα νήματα ή σε κανένα.
- διαδοχικές περιοχές διαμοιρασμού εργασίας θα πρέπει να προσπελαύνονται από τα μέλη μιας ομάδας με την ίδια σειρά.

2.3.4.1. Οδηγία for

Η οδηγία for καθορίζει ότι οι επαναλήψεις του βρόγχου που ακολουθεί πρέπει να εκτελεσθούν παράλληλα από την ομάδα. Αυτό προϋποθέτει ότι η παράλληλη περιοχή έχει ήδη αρχικοποιηθεί, αλλιώς εκτελεί σειριακά τον κώδικα. Η σύνταξη της οδηγίας for είναι όπως δείχνει ο Πίνακας 2.4.

Πίνακας 2.4 Η οδηγία for

```
#pragma omp for [clause [clause] … ]
  for loop
```

όπου κάθε clause είναι μία από τις παρακάτω συνθήκες:

- private(list)
- firstprivate(list)
- lastprivate(list)

- reduction(operator: list)
- ordered
- schedule(type [,chunk])
- nowait

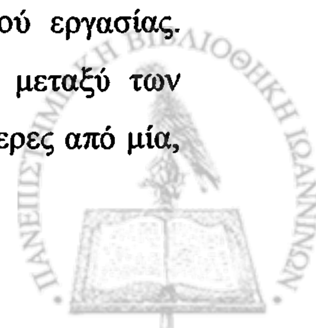
Οι συνθήκες private(list), firstprivate(list), lastprivate(list), reduction(operator: list) και schedule(type [,chunk]) περιγράφηκαν στην Ενότητα 2.3.3.
Η συνθήκη ordered πρέπει να υπάρχει όταν οδηγίες ordered συμπεριλαμβάνονται μέσα σε μια for οδηγία και καθορίζουν κάποια τμήματα που πρέπει να εκτελεστούν ακολουθιακά.
Η συνθήκη nowait χρησιμοποιείται για να επιτρέψουμε στα νήματα να συνεχίσουν την εκτέλεσή τους χωρίς να περιμένουν τον τερματισμό και των υπόλοιπων νημάτων στο τέλος του παράλληλου βρόγχου.

Οι περιορισμοί που ισχύουν είναι οι εξής:

- η ορθότητα ενός προγράμματος δεν θα πρέπει να εξαρτάται από το ποιο νήμα θα εκτελέσει μια συγκεκριμένη επανάληψη.
- απαγορεύεται μια διακλάδωση να καταλήγει έξω από ένα βρόγχο ο οποίος σχετίζεται με μια οδηγία for.
- το μέγεθος ενός κομματιού πρέπει να καθορίζεται σαν σταθερή έκφραση ακεραίου ενός βρόγχου, αφού δεν υπάρχει συγχρονισμός για την αποτίμηση του μεγέθους από τα νήματα.
- ο βρόγχος for θα πρέπει να είναι συγκεκριμένης μορφής, όπως περιγράφεται στο πρότυπο.
- οι φράσεις ORDERED και SCHEDULE πρέπει να εμφανίζονται το πολύ μια φορά η καθεμία.

2.3.4.2. Οδηγία sections

Η οδηγία sections είναι μια μη επαναληπτική περιοχή διαμοιρασμού εργασίας. Καθορίζει ότι τα εσωκλειώμενα τμήματα κώδικα θα διαμοιραστούν μεταξύ των νημάτων της ομάδας. Μια οδηγία sections μπορεί να περιέχει περισσότερες από μία,


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ανεξάρτητες, οδηγίες section. Κάθε τμήμα εκτελείται μια φορά από ένα νήμα της ομάδας. Τη σύνταξη μιας οδηγίας sections δείχνει ο Πίνακας 2.5.

Πίνακας 2.5 Η οδηγιά sections

```
#pragma omp sections [clause [clause ...]]
{
#pragma omp section
   δομημένο τμήμα κώδικα
[#pragma omp section
   δομημένο τμήμα κώδικα
...]
}
```

Οι φράσεις μπορούν να είναι οι εξής:

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator: list)
- nowait

όπως αυτές περιγράφηκαν στις παραγράφους 2.3.3 και 2.3.4.1. Στο τέλος κάθε οδηγίας section υπονοείται κάποιο φράγμα που θα βοηθήσει στο συγχρονισμό των νημάτων, εκτός κι αν χρησιμοποιηθεί η συνθήκη nowait οπότε τα νήματα δεν περιμένουν για συγχρονισμό μετα το πέρας της εργασίας τους.

Οι περιορισμοί που ισχύουν στην οδηγία sections είναι οι εξής:

- όπως και στην οδηγία for, απαγορεύεται μια διακλάδωση να εισέρχεται σε ένα δομημένο τμήμα κώδικα ή να καταλήγει έξω από αυτό.
- οι οδηγίες τμήματος πρέπει να λαμβάνουν χώρα μέσα σε μια οδηγία τμημάτων.
- μόνο μία συνθήκη nowait μπορεί να περιέχεται σε μια οδηγία τμημάτων.

2.3.4.3. Οδηγία single

Η οδηγία single καθορίζει ότι ο κώδικας που εσωκλείεται από αυτή, θα εκτελεστεί μόνο από ένα νήμα. Το νήμα που θα φτάσει πρώτο στην single οδηγία, αυτό θα εκτελέσει τον κώδικα. Η οδηγία αυτή, είναι χρήσιμη όταν έχουμε τμήματα από κώδικα που δεν εξαρτώνται αποκλειστικά από τα νήματα όπως για παράδειγμα λειτουργίες εισόδου-εξόδου. Η σύνταξη μιας οδηγίας single δείχνει ο Πίνακας 2.6.

Πίνακας 2.6 Η οδηγία single

```
#pragma omp single [clause [clause…]]
  δομημένο τμήμα κώδικα
```

Οι φράσεις μπορούν να είναι οι παρακάτω:

- private(list)
- firstprivate(list)
- nowait

όπως αυτές περιγράφηκαν στις παραγράφους 2.3.3 και 2.3.4.1. Τα νήματα που δεν εκτελούν την οδηγία, περιμένουν στο τέλος του εσωκλειώμενου κώδικα, εκτός κι αν χρησιμοποιηθεί η συνθήκη nowait οπότε τα νήματα δεν θα περιμένουν για να συγχρονιστούν.

Οι περιορισμοί που ισχύουν είναι οι εξής:

- απαγορεύεται μια διακλάδωση να εισέρχεται σε ένα δομημένο τμήμα κώδικα οδηγίας single ή να καταλήγει έξω από αυτό και
- μόνο μία συνθήκη nowait μπορεί να χρησιμοποιηθεί σε κάθε οδηγία single.

2.3.5. Συνδυασμένες παράλληλες περιοχές διαμοιρασμού εργασίας

Οι Συνδυασμένες παράλληλες περιοχές διαμοιρασμού εργασίας είναι συντομεύσεις για να καθορίσουμε μια παράλληλη περιοχή η οποία περιέχει μόνο μία περιοχή διαμοιρασμού εργασίας (ένα παράλληλο for ή παράλληλα sections). Σημασιολογικά,

οι συνδυασμένες παράλληλες περιοχές διαμοιρασμού εργασίας, είναι ισοδύναμες, με τη δήλωση μιας παράλληλης περιοχής, αμέσως ακολουθούμενης από μία περιοχή διαμοιρασμού εργασίας.

Επιτρέπονται όλες οι υπαρκτές συνθήκες για μια παράλληλη περιοχή και για την σχετική περιοχή διαμοιρασμού εργασίας, εκτός από τη nowait αφού, έτσι κι αλλιώς, στο τέλος κάθε παράλληλου τμήματος υπονοείται ένα φράγμα για το συγχρονισμό των νημάτων.

Υπάρχουν δύο τύποι τέτοιων οδηγιών:

- οδηγία parallel for και
- οδηγία parallel sections.

2.3.5.1. Οδηγία parallel for

Η οδηγία parallel for καθορίζει μια παράλληλη περιοχή που περιέχει μια απλή for οδηγία. Η οδηγία for θα πρέπει να είναι η αμέσως επόμενη εντολή (μετα τη δήλωση της παράλληλης περιοχής) του κώδικα. Η σύνταξη μιας parallel for οδηγίας είναι όπως δείχνει ο Πίνακας 2.7.

Πίνακας 2.7 Η οδηγία parallel for

```
#pragma omp parallel for [clause [clause…]]
  βρόχος επανάληψης
```

Οι φράσεις που μπορούν να χρησιμοποιηθούν είναι οι εξής:

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator: list)
- schedule(type [,chunk])
- if(expression)
- default(shared|private|none)

- copyin(list)

όπως αυτές περιγράφηκαν στη παράγραφο 2.3.4.1.

Οι περιορισμοί που ισχύουν στην απλή οδηγία for ισχύουν κι εδώ (βλέπε παράγραφο 2.3.4.1).

2.3.5.2. Οδηγία parallel sections

Η οδηγία parallel sections καθορίζει μια παράλληλη περιοχή η οποία περιέχει μια απλή οδηγία sections. Αυτή η οδηγία sections, θα πρέπει να είναι η αμέσως επόμενη εντολή (μετα τη δήλωση της παράλληλης περιοχής) του κώδικα. Η σύνταξη μιας οδηγίας parallel sections είναι όπως δείχνει ο Πίνακας 2.8:

Πίνακας 2.8 Η οδηγία parallel sections

```
#pragma omp parallel sections [clause [clause…]]
{
[#pragma omp section]
  δομημένο τμήμα κώδικα
[#pragma omp section
  δομημένο τμήμα κώδικα
  …]
}
```

Οι φράσεις που μπορούν να χρησιμοποιηθούν είναι οι:

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator: list)
- default(shared|private|none)
- copyin(list)
- ordered

όπως αυτές περιγράφηκαν στη παράγραφο 2.3.4.2.

Οι περιορισμοί που ισχύουν στην απλή οδηγία for ισχύουν κι εδώ (βλέπε παράγραφο 2.3.4.2).

2.3.6. Οδηγίες συγχρονισμού για C/C++

Οι Οδηγίες συγχρονισμού χρησιμοποιούνται για να ελέγξουμε τη σειρά εκτέλεσης των νημάτων μιας ομάδας, έτσι ώστε να εξασφαλίσουμε την ακεραιότητα των δεδομένων. Οι οδηγίες συγχρονισμού που μας δίνει το API είναι οι εξής:

- master
- critical
- barrier
- atomic
- flush
- ordered

οι οποίες θα περιγραφούν στις επόμενες παραγράφους.

2.3.6.1. Οδηγία master

Η οδηγία master καθορίζει μια περιοχή η οποία θα εκτελεστεί μόνο από το master νήμα της ομάδας. Όλα τα άλλα νήματα της ομάδας δεν εκτελούν αυτό το τμήμα κώδικα. Για την οδηγία αυτή, δεν υπάρχει κάποιο φράγμα που να υπονοείται. Η σύνταξη της master οδηγίας είναι όπως δείχνει ο Πίνακας 2.9:

Πίνακας 2.9 Η οδηγία master

```
#pragma omp master
  δομημένο τμήμα κώδικα
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.3.6.2. Οδηγία critical

Η οδηγία critical καθορίζει μια περιοχή η οποία πρέπει να εκτελεστεί μόνο από ένα νήμα κάθε φορά, ορίζοντας έτσι μια κρίσιμη περιοχή. Σε μια κρίσιμη περιοχή, μόνο μια διεργασία μπορεί να εγγράψει ή να διαβάσει μια κοινή μεταβλητή διασφαλίζοντας έτσι την ακεραιότητα αυτής της μεταβλητής. Η σύνταξη της οδηγίας critical είναι όπως δείχνει ο Πίνακας 2.10:

Πίνακας 2.10 Η οδηγία critical

```
#pragma omp critical [<όνομα>]
  δομημένο τμήμα κώδικα
```

όπου όνομα είναι ένα αναγνωριστικό όνομα της κρίσιμης περιοχής που καθορίζει την οδηγία critical (κάθε κρίσιμη περιοχή πρέπει να έχει μοναδικό αναγνωριστικό όνομα). Οι κρίσιμες περιοχές που δεν έχουν όνομα θεωρείται ότι έχουν την ίδια ταυτότητα, ενώ διαφορετικές critical οδηγίες που έχουν το ίδιο όνομα, θεωρείται ότι αναφέρονται στην ίδια κρίσιμη περιοχή. Αν ένα νήμα εκτελεί μια κρίσιμη περιοχή και ένα άλλο νήμα φτάσει σε αυτή την κρίσιμη περιοχή και προσπαθήσει να την εκτελέσει, τότε το δεύτερο νήμα θα αναγκαστεί να περιμένει μέχρι το πρώτο νήμα να φύγει από την κρίσιμη περιοχή.

2.3.6.3. Οδηγία barrier

Η οδηγία barrier συγχρονίζει όλα τα νήματα της ομάδας απαιτώντας από κάθε νήμα να σταματήσει, προσωρινά την εκτέλεσή του, στο σημείο όπου υπάρχει η barrier οδηγία, μέχρις ότου όλα τα νήματα φτάσουν σε αυτό το σημείο. Στη συνέχεια, όλα τα νήματα ξεκινούν παράλληλα από εκείνο το σημείο την εκτέλεσή του κώδικα που ακολουθεί. Η σύνταξη της οδηγίας barrier είναι όπως δείχνει ο Πίνακας 2.11.

Πίνακας 2.11 Η οδηγία barrier

```
#pragma omp barrier
```

Προκειμένου να εφαρμοστεί σωστά η οδηγία barrier πρέπει να ισχύουν οι ακόλουθοι περιορισμοί:

- κάθε οδηγία barrier σε μια παράλληλη περιοχή, πρέπει να συναντηθεί από όλα τα νήματα ή από κανένα και
- η σειρά με την οποία τα νήματα προσπελαύνουν τις περιοχές διαμοιρασμού εργασίας και τις οδηγίες barrier σε μια παράλληλη περιοχή, πρέπει να είναι ίδια για κάθε νήμα της ομάδας.

2.3.6.4. <u>Οδηγία atomic</u>

Η οδηγία atomic καθορίζει ότι η συγκεκριμένη τοποθεσία στη μνήμη πρέπει να ανανεώνεται ατομικά (από κάθε νήμα), μη επιτρέποντας πολλά νήματα να κάνουν εγγραφή ταυτόχρονα στη συγκεκριμένη τοποθεσία. Έτσι απαγορεύει σε οποιοδήποτε νήμα να διακόψει κάποιο άλλο νήμα που βρίσκεται στη διαδικασία προσπέλασης ή αλλαγής της τιμής μιας μεταβλητής κοινής μνήμης. Η σύνταξη της atomic οδηγίας είναι όπως δείχνει ο Πίνακας 2.12.

Πίνακας 2.12 Η οδηγία atomic

```
#pragma omp atomic
  έκφραση
```

όπου έκφραση μπορεί να είναι μια από τις παρακάτω εκφράσεις:

- x binop = expr
- x++
- ++x

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- x--
- --x

Στις προηγούμενες εκφράσεις το x είναι μια μεταβλητή, το expr είναι μια έκφραση η οποία δεν περιέχει το x και binop είναι κάποιος από τους εξής (μη υπερφορτωμένους) τελεστές: +, *, -, /, &, ^, |, <<, >>.

2.3.6.5. <u>Οδηγία flush</u>

Η οδηγία flush καθορίζει ένα σημείο συγχρονισμού στο οποίο η υλοποίηση του κώδικα πρέπει να παρέχει ένα συνεπές στιγμιότυπο της μνήμης. Σε αυτό το σημείο η τρέχουσα τιμή μιας κοινής μεταβλητής εγγράφεται αμέσως στη μνήμη. Η σύνταξη της οδηγίας flush είναι όπως δείχνει ο Πίνακας 2.13.

Πίνακας 2.13 Η οδηγία flush

```
#pragma omp flush (var1 [, var2]…)
```

όπου var1, var2, ...είναι μια λίστα από κοινές μεταβλητές οι οποίες θα εγγραφούν στη μνήμη προκειμένου να αποφύγουμε να εγγράψουμε όλες τις κοινές μεταβλητές. Αν κάποιο από τα var1, var2, ...είναι δείκτης, τότε εγγράφεται ο δείκτης και όχι το αντικείμενο στο οποίο δείχνει.

Οι περιορισμοί που ισχύουν για την οδηγία flush είναι:

- οι υλοποιήσεις των προγραμμάτων πρέπει να εγγυώνται ότι οποιεσδήποτε αλλαγές σε μεταβλητές ορατές από τα νήματα θα είναι ορατές στα νήματα μετά από αυτό το σημείο
- η οδηγία flush υπονοείται για τις ακόλουθες οδηγίες (εκτός αν υπάρχει στον κώδικα η συνθήκη nowait):
    - barrier (στην είσοδο και στην έξοδο),
    - critical (στην είσοδο και στην έξοδο),

- parallel sections (μόνο στην έξοδο),
- parallel for (μόνο στην έξοδο),
- for (μόνο στην έξοδο),
- sections(μόνο στην έξοδο),
- single (μόνο στην έξοδο).

2.3.6.6. Οδηγία ordered

Η οδηγία ordered καθορίζει ότι οι επαναλήψεις του περιβάλλοντος βρόγχου θα εκτελεστούν με τη σειρά όπως θα εκτελούνταν σε ένα σειριακό υπολογιστή. Η σύνταξη της ordered οδηγίας είναι όπως δείχνει ο Πίνακας 2.14.

Πίνακας 2.14 Η οδηγία ordered

```
#pragma omp ordered
  δομημένο τμήμα κώδικα
```

Όταν ένα νήμα, το οποίο πρόκειται να εκτελέσει την πρώτη επανάληψη του βρόγχου, συναντήσει μια οδηγία ordered, τότε εκτελεί την ordered περιοχή χωρίς να περιμένει. Αντίθετα, όταν νήματα που εκτελούν επόμενες επαναλήψεις, συναντήσουν την ίδια οδηγία ordered, τότε περιμένουν στην αρχή της περιοχής ordered μέχρι να τελειώσουν από αυτή την περιοχή όλα τα νήματα που εκτελούν προηγούμενες επαναλήψεις (έτσι, μόνο ένα νήμα επιτρέπεται να είναι σε μια ordered περιοχή κάθε φορά).

Οι περιορισμοί που ισχύουν σε μια οδηγία ordered είναι:

- μια ordered οδηγία μπορεί να εμφανίζεται μόνο σε μια for ή parallel for οδηγία.
- ένας βρόγχος ο οποίος περιέχει μια ordered οδηγία, θα πρέπει να είναι βρόγχος με μια ordered φράση.

- μια επανάληψη ενός βρόγχου δεν πρέπει να εκτελεί την ίδια οδηγία ordered παραπάνω από μια φορά και δεν πρέπει να εκτελεί πάνω από μία οδηγία ordered.

2.3.7. Εξειδικευμένη Οδηγία: threadprivate

Η οδηγία threadprivate καθορίζει ότι καθολικά αντικείμενα (ή μεταβλητές) μπορούν να γίνουν ιδιωτικά για κάποιο νήμα. Με αυτό τον τρόπο, μπορούμε να ορίσουμε καθολικά αντικείμενα, αλλά να μετατρέψουμε την εμβέλειά τους και να τα κάνουμε τοπικά για κάποιο νήμα. Οι μεταβλητές για τις οποίες ισχύει η οδηγία threadprivate συνεχίζουν να είναι ιδιωτικές, για κάθε νήμα, ακόμα και σε διαφορετικές παράλληλες περιοχές. Η σύνταξη της threadprivate οδηγίας είναι όπως δείχνει ο Πίνακας 2.15:

Πίνακας 2.15 Η οδηγία threadprivate

```
#pragma omp threadprivate (var1 [, var2, …])
```

Η οδηγία πρέπει να εμφανίζεται αμέσως μετά τις δηλώσεις των καθολικών μεταβλητών (άρα πριν τη main()). Μετά την οδηγία, η συγκεκριμένη μεταβλητή που έχει οριστεί ως threadprivate, αντιγράφεται σε κάθε νήμα και κάθε νήμα κρατάει το δικό του αντίγραφο, έτσι ώστε δεδομένα τα οποία εγγράφονται από ένα νήμα να μην είναι ορατά στα υπόλοιπα. Για να συνεχίσουν οι μεταβλητές να είναι ιδιωτικές για όλες τις παράλληλες περιοχές, θα πρέπει να ισχύουν οι παρακάτω συνθήκες:

- να μην υπάρχει παράλληλη περιοχή μέσα σε μια άλλη παράλληλη περιοχή.
- ο αριθμός των νημάτων στις παράλληλες περιοχές να είναι ο ίδιος.
- ο εσωτερικός, δυναμικός μηχανισμός των νημάτων, πρέπει να είναι απενεργοποιημένος κατά την πρώτη κρίσιμη περιοχή και να παραμένει ο ίδιος στις υπόλοιπες παράλληλες περιοχές.

Κατά την πρώτη είσοδο σε μια παράλληλη περιοχή, οι τιμές των μεταβλητών threadprivate θεωρούνται ακαθόριστες, εκτός κι αν έχει χρησιμοποιηθεί στην parallel (section ή for) οδηγία, η φράση copyin (βλέπε παράγραφο 2.3.3).

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Οι περιορισμοί που ισχύουν για την οδηγία threadprivate είναι:

- μια οδηγία threadprivate πρέπει να εμφανίζεται μετά τις καθολικές δηλώσεις και πριν την πρώτη χρήση των μεταβλητών.
- μια μεταβλητή threadprivate μπορεί να εμφανίζεται μόνο στις συνθήκες copyin, schedule και if.

2.3.8. Κανόνες εμφωλιασμού

Κατά τη συγγραφή προγραμμάτων OpenMP, ισχύουν οι εξής κανόνες για ό,τι αφορά τον εμφωλιασμό δομών και οδηγιών:

- Μια παράλληλη περιοχή μέσα σε μια άλλη, λογικά ορίζει μια νέα ομάδα νημάτων η οποία αποτελείται από το τρέχον νήμα.
- Οι οδηγίες for, sections και single που βρίσκονται μέσα στην ίδια παράλληλη περιοχή δεν επιτρέπεται να φωλιάσουν μεταξύ τους.
- Οι οδηγίες for, sections και single δεν επιτρέπονται μέσα στις δομές critical, ordered, master.
- Οδηγίες critical μα το ίδιο όνομα δεν επιτρέπεται να φωλιάσουν η μία μέσα στην άλλη.
- Οδηγίες barrier δεν επιτρέπονται μέσα στις δομές critical, ordered, master, for, sections, single.
- Οδηγίες master δεν επιτρέπονται μέσα στις δομές for, sections, single.
- Οδηγίες ordered δεν επιτρέπονται μέσα σε δομές critical.
- Κάθε οδηγία η οποία επιτρέπεται όταν εκτελείται δυναμικά μέσα σε μια παράλληλη περιοχή, επιτρέπεται κι όταν εκτελείται έξω από μια παράλληλη περιοχή. Όταν εκτελείται έξω από μια παράλληλη περιοχή, τότε η οδηγία εκτελείται σε μια ομάδα νημάτων αποτελούμενη μόνο από το νήμα αρχηγό.

2.4. Μεταφραστές OpenMP

Πέρα από τον OMPi υπάρχουν και κάποιες άλλες προσπάθειες για τη δημιουργία μεταφραστών συμβατών με το πρότυπο OpenMP. Θα αναφερθούν οι κυριότερες.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.4.1. Εμπορικοί μεταφραστές

2.4.1.1. Fujitsu/Lahey

Ένας μεταφραστής που υποστηρίζει γλώσσες προγραμματισμού FORTRAN, C και C++ για λειτουργικά συστήματα Linux και Solaris.

2.4.1.2. Hewlett-Packard

Οι μεταφραστές FORTRAN, C και C++ για το λειτουργικό σύστημα TRU64 υποστηρίζουν το πρότυπο OpenMP.

2.4.1.3. Intel

Μεταφραστές C++ και FORTRAN για συστήματα IA32 και Itanium στα λειτουργικά συστήματα Linux και Windows

2.4.1.4. Portland Group, Inc

Οι μεταφραστές PGF77 και PGF90 (FORTRAN) για λειτουργικά συστήματα Linux, Solaris και Windows.

2.4.1.5. SGI

Ο μεταφραστής SGI MIPSpro για συστήματα IRIX και γλώσσες C, C++ και FORTRAN.

2.4.1.6. IBM

Οι μεταφραστές IBM XL C/C++ και Fortran, οι οποίοι υποστηρίζουν και φωλιασμένο παραλληλισμό.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.4.1.7. Sun

Οι μεταφραστές του Sun Studio υποστηρίζουν γλώσσες Fortran 95, C και C++.

2.4.2. Ερευνητικοί μεταφραστές

2.4.2.1. GNU C Compiler

Η έκδοση 4.1 του μεταφραστή C της GNU περιλαμβάνει διακόπτες που ενεργοποιούν τη μεταγλώττιση κώδικα με οδηγίες OpenMP. Υποστηρίζει εμφωλευμένο παραλληλισμό (δομή parallel μέσα σε δομή parallel).

2.4.2.2. OdinMP

Ο OdinMP είναι επίσης γραμμένος σε C++, όπως και η νέα υλοποίηση του OMPi. Χρησιμοποιεί ενδιάμεση αναπαράσταση κώδικα με Αφηρημένο Συντακτικό Δέντρο (Abstract Syntax Tree). Για την παραγωγή ενός «κανονικού» συντακτικού δέντρου γίνονται προσθήκες στον κώδικα εισόδου, όπως για παράδειγμα η προσθήκη σύνθετου συνόλου εντολών (compound statement) σε μια εντολή if που δεν έχει, επειδή περιέχει μόνο μια εντολή.

2.4.2.3. Omni

Ο Omni περιλαμβάνει εκτός από τον μεταφραστή για γλώσσα προγραμματισμού C και τον μεταφραστή για γλώσσα FORTRAN77. Είναι γραμμένος σε C και Java και σκοπός της ομάδας υλοποίησης είναι η φορητότητα σε πολλές πλατφόρμες.

2.4.2.4. OMPi

Ο OMPi είναι γραμμένος σε γλώσσα προγραμματισμού C και υλοποιημένος γύρω από τα εργαλεία GNU flex και bison. Είναι ο πρώτος μεταφραστής που υποστήριξε την έκδοση 2.0 του προτύπου OpenMP.

ΚΕΦΑΛΑΙΟ 3. ΑΝΑΛΥΣΗ ΚΩΔΙΚΑ ΕΙΣΟΔΟΥ

3.1. Εισαγωγή

3.2. Ο λεκτικός αναλυτής

3.3. Ο συντακτικός αναλυτής

3.1. Εισαγωγή

Για να μπορέσει να μετασχηματιστεί ο κώδικας εισόδου σύμφωνα με τις οδηγίες OpenMP, πρέπει πρώτα να αναλυθεί στα επιμέρους στοιχεία του, πρώτα λεκτικά και έπειτα συντακτικά και σημασιολογικά.

Η υπάρχουσα έκδοση του OMPi είναι κωδικοποιημένη μονολιθικά με χρήση του εργαλείου bison. Ο κώδικας ανάλυσης της εισόδου και μετασχηματισμών είναι διάσπαρτος στους συντακτικούς κανόνες. Το αποτέλεσμα είναι ότι η αλλαγή, για παράδειγμα, στον κώδικα κάποιου μετασχηματισμού απαιτεί καθολική γνώση του κώδικα του μεταφραστή, ώστε να γίνουν αλλαγές σε όλους τους εμπλεκόμενους συντακτικούς κανόνες.

Κατέστη λοιπόν απαραίτητη η τμηματοποίηση του κώδικα του μεταφραστή. Μια μέθοδος τμηματοποίησης είναι ο διαχωρισμός στα επιμέρους τμήματα:

- Λεκτική, συντακτική και σημασιολογική ανάλυση
- Μετασχηματισμοί
- Η κοινή διεπαφή μεταξύ των δύο τμημάτων είναι μια ενδιάμεση αναπαράσταση του κώδικα εισόδου, με τη χρήση Συντακτικού Δέντρου.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Η απαίτηση για τη δημιουργία ενδιάμεσης αναπαράστασης του κώδικα στον μεταφραστή οδήγησε στη χρήση της γλώσσας προγραμματισμού C++ για τους εξής λόγους:

- Ευκολία στη δημιουργία του Συμπαγούς Συντακτικού Δέντρου (ΣΣΔ – Concrete Syntax Tree).
- Βολικότερος χειρισμός του παραγόμενου δέντρου σε σχέση με την επιλογή μιας μη αντικειμενοστραφούς γλώσσας προγραμματισμού.
- Πιο συμπαγής και ευκολότερος στην κατανόηση κώδικας.
- Ευκολότερος στην επέμβαση και επέκταση κώδικας.

Η γλώσσα C++ χρησιμοποιήθηκε σε συνδυασμό με τα GNU εργαλεία flex και bison. για την ανάλυση του κώδικα εισόδου και την παραγωγή του ΣΣΔ.

Το εργαλείο **flex** παρέχει λεκτική ανάλυση της εισόδου. Χρησιμοποιώντας μια σειρά από κανόνες που έχουν σύνταξη κανονικών εκφράσεων, παράγει τερματικά σύμβολα με βάση την είσοδο.
Το εργαλείο **bison** παρέχει συντακτική και σημασιολογική ανάλυση της εισόδου χρησιμοποιώντας τα τερματικά σύμβολα που παρέχει ο flex σε αυτό. Η γλώσσα του bison αποτελείται από μια σειρά από γραμματικούς κανόνες που υπακούουν στη γραμματική της γλώσσας εισόδου.

3.2. Ο λεκτικός αναλυτής

Ο λεκτικός αναλυτής παράχθηκε με χρήση του εργαλείου GNU flex. Η εργασία που εκτελεί είναι η αναγνώριση λεκτικών φράσεων (tokens) και η μετατροπή τους σε κόμβους-φύλλα του συντακτικού δέντρου.

3.3. Ο συντακτικός αναλυτής

Ο συντακτικός αναλυτής παράχθηκε με χρήση του εργαλείου GNU bison. Η εργασία που επιτελεί αποτελείται από τις εξής υποεργασίες:

- Συντακτικός έλεγχος της εισόδου
- Σημασιολογικός έλεγχος της εισόδου

- Δημιουργία κόμβων διακλαδώσεως του δέντρου

Αναλυτικότερα:

3.3.1. Συντακτικός έλεγχος της εισόδου

Οι κανόνες γραμματικής που χρησιμοποιούνται στην παραγωγή του συντακτικού αναλυτή αποτελούνται από:

- Κανόνες γραμματικής από ANSI/IEC ISO 9899-1999 για τη γλώσσα προγραμματισμού C [1]
- Προσθήκες γραμματικής OpenMP με βάση το πρότυπο έκδοσης 2.5 [2]
- Επιπλέον κανόνες POMP που αφορούν τη χρονομέτρηση κατά την εκτέλεση του προγράμματος [4]

Με βάση τους γραμματικούς κανόνες είναι δυνατή η επαλήθευση της συντακτικής ορθότητας της εισόδου.

Ο Πίνακας 3.1 δείχνει ένα παράδειγμα γραμματικού κανόνα. Ο κανόνας ονομάζεται labeled_statement και μπορεί να ταιριάξει μια από τις περιπτώσεις που ακολουθούν, χωρίζόμενες με τον χαρακτήρα pipe «|». Κατά σύμβαση, με κεφαλαία γράμματα εμφανίζονται τα τερματικά στοιχεία ενώ με πεζά τα μη τερματικά. Ο κανόνας τερματίζει με τον χαρακτήρα “;” μόνο του σε μια γραμμή.

Έτσι για παράδειγμα ο κανόνας «IDENTIFIER ‘:’ statement» ταιριάζει το τερματικό στοιχείο IDENTIFIER, δηλαδή κάποιο όνομα ορισμένο από το χρήστη, ακολουθούμενο από το χαρακτήρα ‘:’, ακολουθούμενο από ένα μη τερματικό σύμβολο statement. Πρόκειται για την ετικέτα μιας εντολής goto, η οποία ακολουθείται από τμήμα κώδικα, που περιλαμβάνεται στο statement.

Αντίστοιχα η γραμμή «DEFAULT ‘:’ statement» ταιριάζει το τερματικό στοιχείο DEFAULT ακολουθούμενο από το τερματικό στοιχείο ‘:’, ακολουθούμενο από το μη τερματικό στοιχείο statement. Εδώ πρόκειται για την έκφραση “default: κώδικας” που συναντούμε στην εντολή switch.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Πίνακας 3.1 Παράδειγμα γραμματικού κανόνα

```
labeled_statement:
    IDENTIFIER ':' statement
  | DEFAULT ':' statement
;
```

3.3.2. Σημασιολογικός έλεγχος της εισόδου

Το εργαλείο bison χρησιμοποιεί κώδικα χρήστη που παρεμβάλλεται στους γραμματικούς κανόνες ώστε να γίνει ο σημασιολογικός έλεγχος της εισόδου.
Έτσι είναι δυνατό ο αναλυτής να ελέγξει για την ύπαρξη μεταβλητών που χρησιμοποιούνται χωρίς να έχουν δηλωθεί προηγουμένως, όπως και για τη διπλή δήλωση μεταβλητών στο ίδιο επίπεδο ορατότητας (visibility scope).
Επίσης γίνεται έλεγχος για τύπους ορισμένους από το χρήστη (typedef), ώστε να μπορούν να χρησιμοποιηθούν ως τύποι μεταβλητών μέσα στο πρόγραμμα.

3.3.3. Δημιουργία μη τερματικών κόμβων του δέντρου

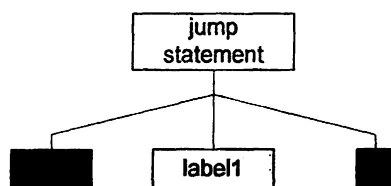
Λαμβάνοντας τα τερματικά αντικείμενα από τον λεκτικό αναλυτή, ο συντακτικός αναλυτής προσπαθεί να ταιριάξει την είσοδο με τους γραμματικούς κανόνες που γνωρίζει.
Σε κάθε επίπεδο ταιριάσματος δημιουργείται ένας μη τερματικός κόμβος που συνδέει τα τερματικά και μη τερματικά αντικείμενα που έχουν χρησιμοποιηθεί. Ο κόμβος αυτός θα χρησιμοποιηθεί ώστε να δημιουργηθεί ένα υποδέντρο. Το υποδέντρο σταδιακά θα ενσωματωθεί στο ολικό, όταν το επίπεδο ταιριάσματος φτάσει στο κατάλληλο σημείο.

Ο Πίνακας 3.2 δείχνει μια απλή περίπτωση κανόνα. Αν θεωρήσουμε ότι έχουμε συναντήσει την είσοδο “goto label1;”, τότε είναι προφανές ότι αν υπάρχει καταχωρημένο το label1 ως ετικέτα, θα ταιριάξει ο πρώτος κανόνας.

Πίνακας 3.2 Παράδειγμα απλού γραμματικού κανόνα

```
jump_statement:
    GOTO IDENTIFIER ';'
  | CONTINUE ';'
  | BREAK ';'
  | RETURN ';'
  | RETURN expression ';'
;
```

Τα τρία τερματικά αντικείμενα που έχουμε δημιουργήσει ("goto", "label1" και ";") θα συνενωθούν με τη χρήση ενός μη τερματικού κόμβου, του οποίου θα γίνουν παιδιά. Όπως φαίνεται στο Σχήμα 3.1, έχει δημιουργηθεί ένας μη τερματικός κόμβος "jump statement" ο οποίος περιέχει ως κόμβους παιδιά τους τερματικούς κόμβους που περιγράφηκαν.



Σχήμα 3.1 Παράδειγμα κατασκευής υποδέντρου

3.3.3.1. <u>Απόδοση γνωρισμάτων στους μη τερματικούς κόμβους</u>

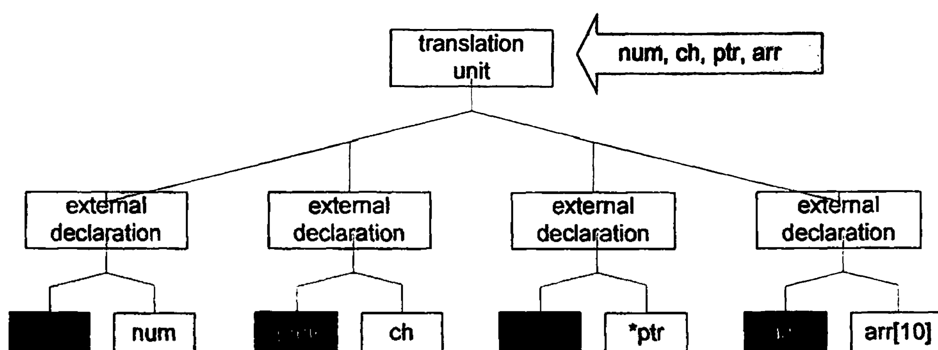
Κατά τη δημιουργία των μη τερματικών κόμβων, ανατίθενται κάποια γνωρίσματα σε αυτούς, για τον ευκολότερο χειρισμό του ΣΣΔ στο δεύτερο πέρασμα του μεταφραστή. Αυτά είναι:

Ένας απαριθμητής που απεικονίζει τον κανόνα που δημιούργησε το συγκεκριμένο κόμβο (PR_TRANSLATION_UNIT στο παράδειγμα, PR = Parser Rule).

Πίνακας 3.3 Παράδειγμα χρήσης απαριθμητή σε γραμματικό κανόνα

```
translation_unit:
  external_declaration
  {
    $$-= new genblock(PR_TRANSLATION_UNIT, …);
  }
…
```

Αναφορές σε χάρτες δηλώσεων μεταβλητών και τύπων χρήστη για κόμβους που περιέχουν σύνθετες εκφράσεις (compound statements) ώστε να είναι δυνατός ο έλεγχος ορατότητας μεταβλητών.



Σχήμα 3.2 Απλοποιημένο παράδειγμα χάρτη μεταβλητών

Ενδιαφέρον παρουσιάζει η περίπτωση του χάρτη δηλώσεων μεταβλητών μιας συνάρτησης, όπου ο αναγνωριστής της συνάρτησης είναι δηλωμένος στον καθολικό χώρο διευθύνσεων, ενώ οι παράμετροι της συνάρτησης όπως και οι τοπικές μεταβλητές της συνάρτησης είναι δηλωμένες στην τοπική εμβέλεια της συνάρτησης. Για την τήρηση δηλώσεων μεταβλητών μιας συνάρτησης χρησιμοποιείται η ίδια δομή με αυτή των καθολικών μεταβλητών.

Ένα γνώρισμα επιπέδου εμφωλιασμού για τους κόμβους των οδηγιών προς τον προεπεξεργαστή (parallel_level), το οποίο αυξάνεται κατά 1 από το 0 για κάθε επίπεδο εμφωλιασμού που συναντάται.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

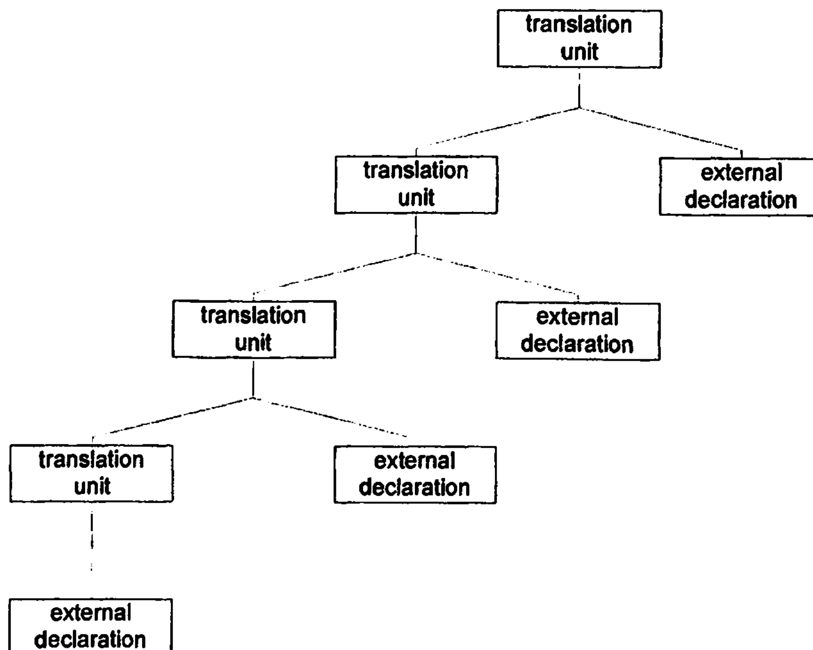
3.3.3.2. Απλοποίηση δενδρικής δομής

Οι γραμματικοί κανόνες της C99 και του OpenMP περιέχουν κάποια σημεία όπου αναδρομικά επαναλαμβάνονται μη τερματικά στοιχεία, όπως δείχνει ο Πίνακας 3.4. Αυτό συμβαίνει ώστε να καλυφθεί συντακτικά οποιοσδήποτε αριθμός εκφράσεων μέσα στον κώδικα εισόδου.

Πίνακας 3.4 Παράδειγμα αναδρομικού γραμματικού κανόνα

```
translation_unit:
  external_declaration
| translation_unit external_declaration
;
```

Στο συγκεκριμένο παράδειγμα επαναλαμβάνεται αναδρομικά το μη τερματικό στοιχείο translation_unit. Το παραγόμενο υποδέντρο έχει δημιουργηθεί σωστά, γιατί γίνεται σωστός έλεγχος της σύνταξης. Η δεντρική δομή όμως που παράγεται είναι μη συμμετρική, όπως φαίνεται στο παράδειγμα στο Σχήμα 3.3.

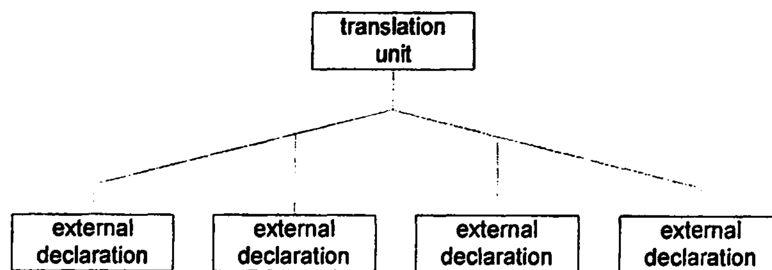


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Σχήμα 3.3 Υποδέντρο κατασκευασμένο από αναδρομικό κανόνα

Επίσης η δομή αυτή περιέχει περιττούς μη τερματικούς κόμβους που δε διευκολύνουν την αναζήτηση μέσα στο δέντρο.

Στις περιπτώσεις που εμφανίζεται αυτό το φαινόμενο, έχει γίνει επέμβαση στον κώδικα που ακολουθεί τους γραμματικούς κανόνες ώστε η δομή που θα παραχθεί να είναι επίπεδη, για πιο εύκολο χειρισμό του δέντρου. Όλοι οι μη τερματικοί κόμβοι γίνονται άμεσα παιδιά του αρχικού. Το αποτέλεσμα μετά την επέμβαση είναι όπως στο ακόλουθο σχήμα. Η σύνδεση των μη τερματικών κόμβων έχει πλησιάσει πιο πολύ τη μορφή ενός προγράμματος εισόδου και δεν υπάρχει εννοιολογική αλλοίωση του δέντρου. Για πλήρη κατάλογο συντακτικών κανόνων στους οποίους εφαρμόζεται η τεχνική αυτή βλ. Παράρτημα Β.



Σχήμα 3.4 Επίπεδο υπόδεντρο κατασκευασμένο από αναδρομικό κανόνα

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΚΕΦΑΛΑΙΟ 4. ΣΥΜΠΑΓΕΣ ΣΥΝΤΑΚΤΙΚΟ ΔΕΝΤΡΟ

4.1. Εισαγωγή

4.2. Ιεραρχία κλάσεων

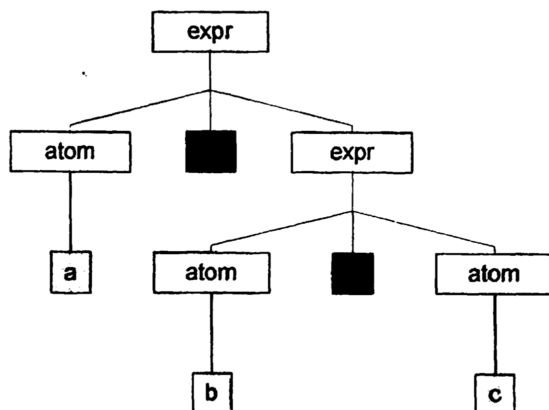
4.3. Σύνδεση τερματικών και μη τερματικών αντικειμένων

4.4. Εξαγωγή δέντρου σε μορφή XML

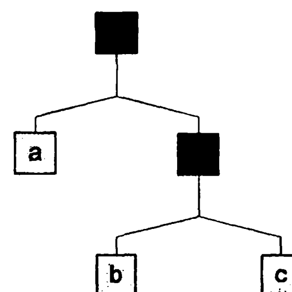
4.1. Εισαγωγή

Ένα Συμπαγές Συντακτικό Δέντρο (ΣΣΔ) είναι μια αναπαράσταση του κώδικα εισόδου του μεταφραστή. Συνδέει τα τερματικά σύμβολα της γραμματικής του με κόμβους μη τερματικών συμβόλων. Αντίθετα, ένα Αφηρημένο Συντακτικό Δέντρο (ΑΣΔ) συνδέει τους τερματικούς του κόμβους με κόμβους τερματικών συμβόλων.

Σχήμα 4.1 Συμπαγές Συντακτικό Δέντρο για την έκφραση a + b * c



Σχήμα 4.2 Αφηρημένο Συντακτικό Δέντρο για την έκφραση a + b * c



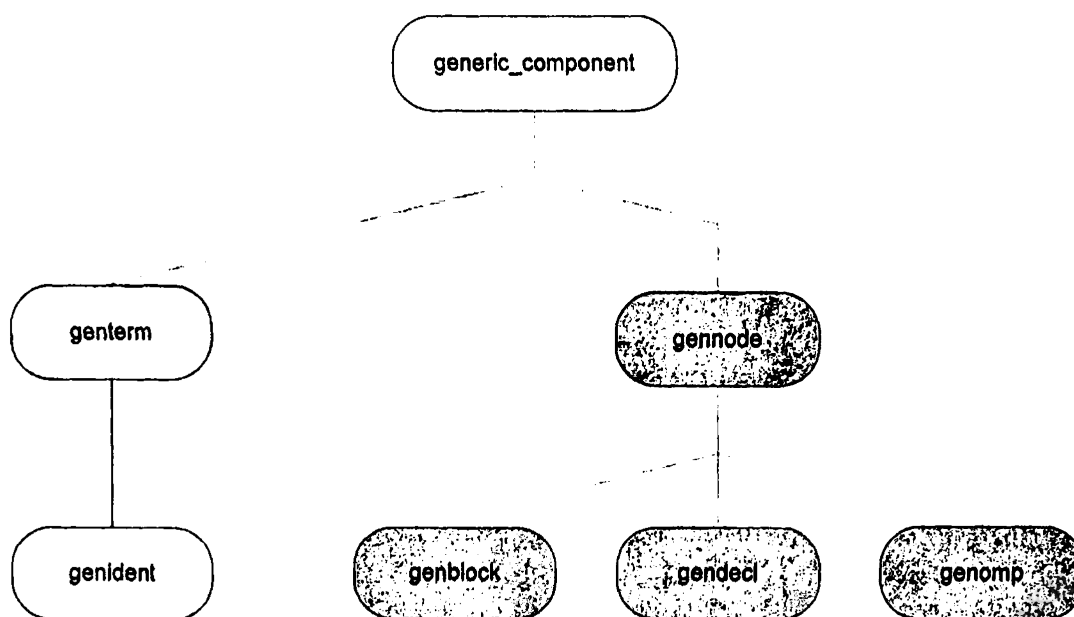
Για να δημιουργήσουμε το Συμπαγές Συντακτικό Δέντρο (ΣΣΔ) χρησιμοποιούμε δύο είδη αντικειμένων, τα τερματικά και τα μη τερματικά. Τα τερματικά είναι οι κόμβοι φύλλα αυτού του δέντρου. Τα μη τερματικά παίζουν συνδετικό ρόλο μεταξύ των τερματικών.

Για την ενδιάμεση αναπαράσταση επιλέχθηκε η μορφή του Συμπαγούς δέντρου, γιατί είναι απαραίτητη η ύπαρξη μη τερματικών στοιχείων που αποθηκεύουν επιπλέον πληροφορία για τον κώδικα εισόδου.

4.2. Ιεραρχία κλάσεων

Για τη δημιουργία του συντακτικού δέντρου υλοποιήθηκε μια σειρά κλάσεων. Κάποιες από αυτές χρησιμοποιούνται για την αποθήκευση τερματικών φράσεων, άλλες για τη σύνδεση των κλάσεων τερματικών στοιχείων και άλλες για την αποθήκευση επιπλέον πληροφορίας, όπως οι δηλωμένες μεταβλητές.
Οι κλάσεις αυτές είναι δεμένες σε μια ιεραρχία, στην οποία κληρονομούνται και κληροδοτούνται χαρακτηριστικά.

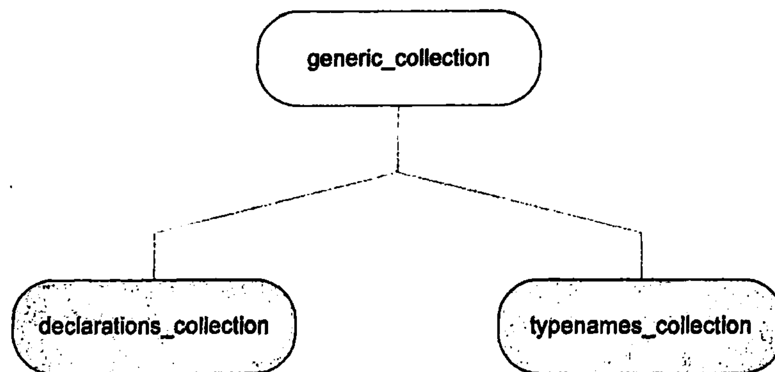
Κοινός πρόγονος όλων των κλάσεων που αναπαριστούν τερματικά και μη τερματικά αντικείμενα στο δέντρο είναι η κλάση generic_component.



Σχήμα 4.3 Η ιεραρχία των κλάσεων τερματικών και μη τερματικών αντικειμένων

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Αντίστοιχα, κοινός πρόγονος των κλάσεων που αναπαριστούν τις δηλώσεις μεταβλητών και τύπων είναι η κλάση generic_collection.



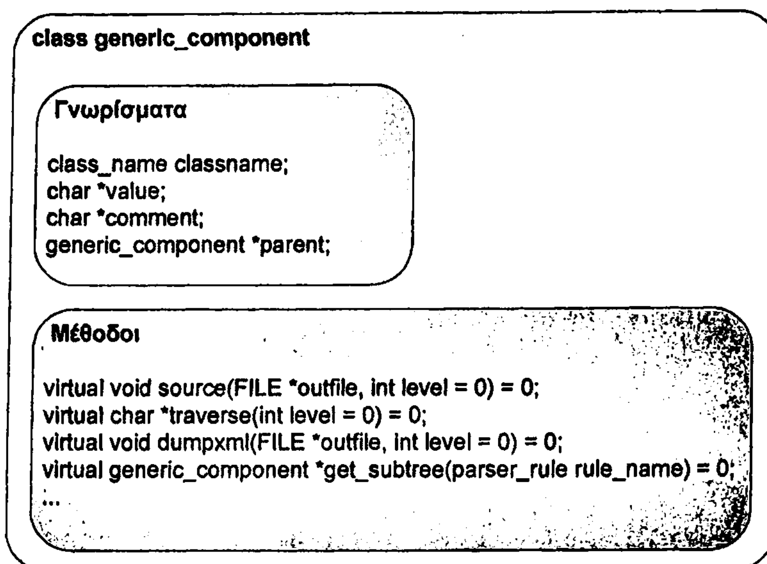
Σχήμα 4.4 Η ιεραρχία των κλάσεων για δηλώσεις μεταβλητών και τύπων

Παρακάτω αναλύονται και οι δύο ιεραρχίες

4.2.1. Τερματικές και μη τερματικές κλάσεις

Η κορυφή του δέντρου κληρονομικότητας για τη δημιουργία τερματικών και μη τερματικών αντικειμένων είναι μια αφηρημένη κλάση γενικής χρήσης με όνομα generic_component. Περιλαμβάνει κάποια γνωρίσματα που χρησιμοποιούνται από όλες τις κληροδοτούμενες κλάσεις, όπως επίσης και τους ορισμούς των εικονικών μεθόδων που πρέπει αυτές να υλοποιήσουν.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 4.5 Η κλάση generic_component

4.2.1.1. Η κλάση genterm

Η κλάση genterm προκύπτει ως άμεσος κληρονόμος της generic_component. Στιγμιότυπα της κλάσης αυτής προκύπτουν από τον λεκτικό αναλυτή όταν συναντήσει:

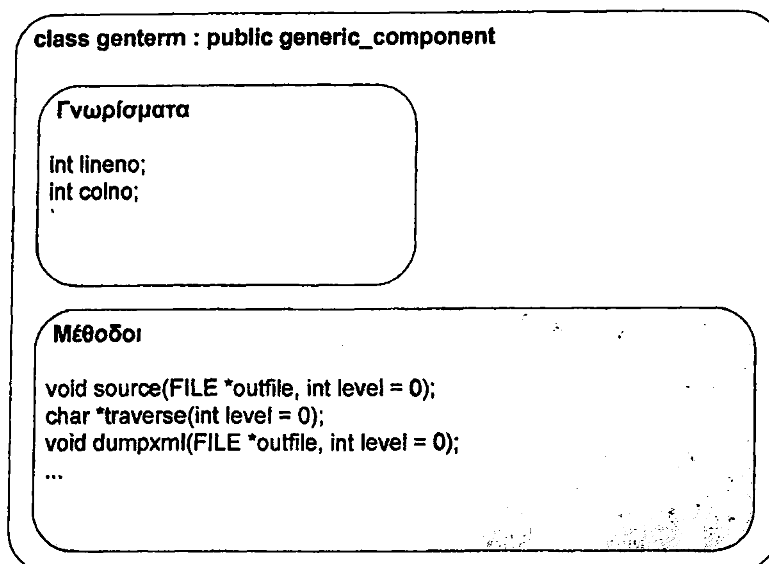
λέξεις-κλειδιά της γλώσσας C,

αριθμητικές σταθερές

συμβολοσειρές μέσα σε διπλά εισαγωγικά (“)

τερματικά σύμβολα της γλώσσας, όπως ‘;’, ‘,’, ‘(‘, ‘)’, ‘[‘, ‘]’, ‘{‘, ‘}’ και άλλα

οδηγίες προς τον προεπεξεργαστή σύμφωνες με το πρότυπο OpenMP



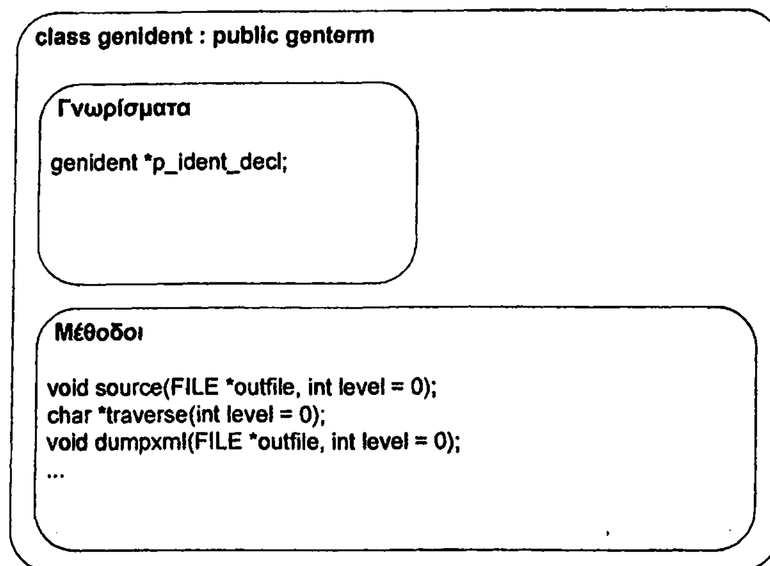
Σχήμα 4.6 Η κλάση genterm

Πέρα από τα γνωρίσματα που κληρονομεί από την generic_component, η κλάση genterm επιπλέον διαθέτει γνωρίσματα για τον αριθμό γραμμής και στήλης στην οποία συναντήθηκε η φράση (token).

4.2.1.1.1. Η κλάση genident

Η κλάση genident προκύπτει από την κλάση genterm. Δημιουργείται από τον λεκτικό αναλυτή όταν συναντήσει συμβολοσειρά που δεν είναι λέξη-κλειδί. Αντικείμενα τύπου genident δημιουργούνται λοιπόν για αναγνωριστές, που μπορεί να είναι π.χ. μεταβλητές, ονόματα συναρτήσεων ή τύποι δηλωμένοι από το χρήστη (typedef).

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 4.7 Η κλάση genident

Επιπλέον της κλάσης genterm, η κλάση genident περιλαμβάνει έναν δείκτη σε αντικείμενο τύπου genident. Ο δείκτης αυτός δεν έχει καθορισμένη αρχική τιμή, οπότε ο λεκτικός αναλυτής τον αρχικοποιεί σε τιμή NULL.

Κατά τη συντακτική και σημασιολογική ανάλυση, όταν συναντηθεί η δήλωση για παράδειγμα μιας μεταβλητής, προστίθεται στον πίνακα των δηλώσεων ένας δείκτης στη δήλωση της μεταβλητής. Όταν λοιπόν συναντηθεί μια αναφορά (χρήση) αυτής της μεταβλητής στον κώδικα, ενημερώνεται ο δείκτης p_ident_decl ώστε να δείχνει στην αντίστοιχη θέση του πίνακα δηλώσεων. Έτσι είναι πολύ εύκολο στην έπειτα επεξεργασία του δέντρου να βρεθεί η δήλωση μιας μεταβλητής από οποιοδήποτε σημείο αυτή συναντηθεί.

4.2.1.2. Η κλάση gennode

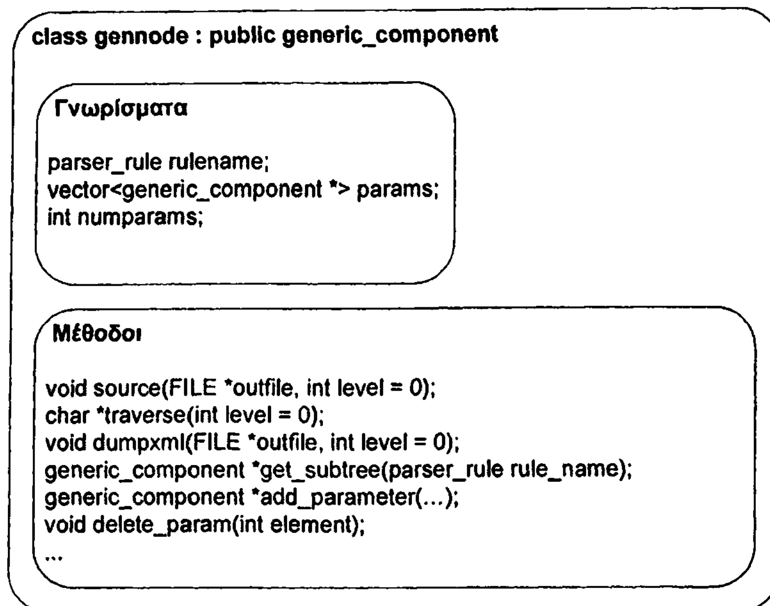
Η κλάση gennode χρησιμοποιείται ως γενικής φύσης μη τερματικός κόμβος του δέντρου.

Επιπλέον της κλάσης generic_component από την οποία κληρονομεί, περιλαμβάνει τα εξής γνωρίσματα:

- rulename, ένας απαριθμητής

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- params, ένας δυναμικός πίνακας (vector)
- numparams, ένας ακέραιος



Σχήμα 4.8 Η κλάση gennode

Το γνώρισμα rulename λαμβάνει τιμή από τον συντακτικό αναλυτή, μέσα από έναν απαριθμητή του οποίου κάθε τιμή αντιστοιχεί ένα προς ένα και μοναδικά στους συντακτικούς κανόνες της γλώσσας. Οι τιμές του απαριθμητή είναι της μορφής PR_<ΣΥΝΤΑΚΤΙΚΟΣ ΚΑΝΟΝΑΣ>, έτσι ο συντακτικός κανόνας με όνομα translation_unit που θα δημιουργήσει ένα μη τερματικό κόμβο, θα του αναθέσεί την τιμή PR_TRANSLATION_UNIT. Με βάση το γνώρισμα αυτό μπορεί να γίνει αναζήτηση μέσα στο δέντρο για υπόδεντρα, και να αναγνωριστούν οι παράμετροι που έχουν παραχθεί.

Ο δυναμικός πίνακας params περιέχει αναφορές σε αντικείμενα τύπου generic_component, τα οποία είναι τα παιδιά του συγκεκριμένου κόμβου. Επειδή όλα τα αντικείμενα που ανήκουν στο ΣΣΔ έχουν πρόγονο την κλάση generic_component, μπορούν να ανατεθούν αναφορές σε αυτό τον πίνακα για στιγμιότυπο οποιασδήποτε κλάσης.

Ο ακέραιος numparams χρησιμοποιείται βοηθητικά και αποθηκεύει τον αριθμό των παιδιών που υπάρχουν στον πίνακα params.

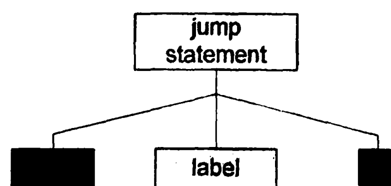
Το απόσπασμα συντακτικού κανόνα bison που δείχνει ο Πίνακας 4.1 ταιριάζει την περίπτωση που εμφανίζεται μια εντολή άλματος σε άλλο σημείο του προγράμματος. Ακολουθείται από την ενέργεια του κανόνα, η οποία καθοδηγεί τον μεταφραστή να δημιουργήσει ένα μη τερματικό κόμβο gennode με όνομα κανόνα PR_JUMP_STATEMENT και παιδιά τους τερματικούς κόμβους που αντιστοιχούν στα:

- GOTO, λεξη-κλειδί της γλώσσας
- IDENTIFIER, αναγνωριστή του προγράμματος και
- ';', τερματικό σύμβολο της γλώσσας.

Πίνακας 4.1 Απόσπασμα γραμματικού κανόνα

```
jump_statement:
    GOTO IDENTIFIER ';'
    {
      $$ = new gennode(PR_JUMP_STATEMENT, $1, $2, $3);
    }
;
```

Αν υποθέσουμε πως η γραμμή κώδικα που συναντήθηκε είναι η:
jump label;
Τότε ο κατασκευασμένος μη τερματικός κόμβος με τα παιδιά του θα είναι όπως στο Σχήμα 4.9.



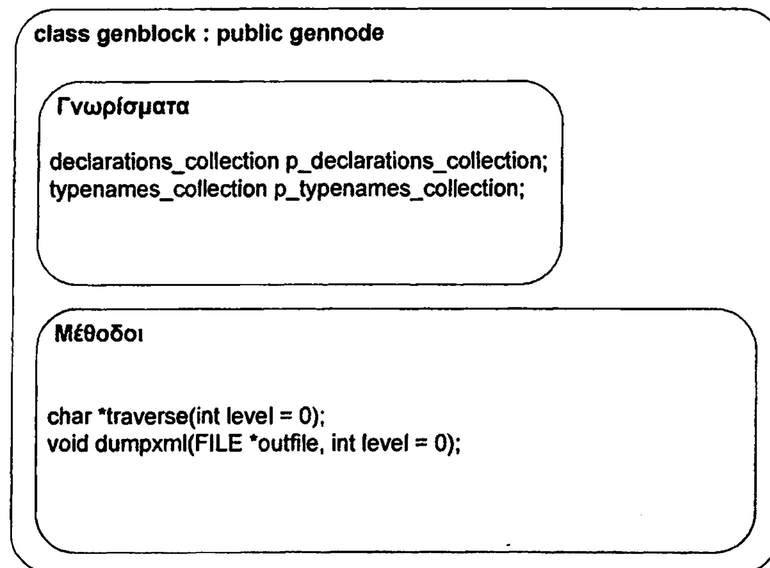
Σχήμα 4.9 Παράδειγμα κατασκευής υποδέντρου

4.2.1.2.1. Η κλάση genblock

Η κλάση genblock κληρονομεί από την gennode. Εκτός των γνωρισμάτων της gennode, περιλαμβάνει δύο επιπλέον δείκτες:

δείκτη σε αντικείμενο τύπου declarations_collection

δείκτη σε αντικείμενο τύπου typenames_collection



Σχήμα 4.10 Η κλάση genblock

Στιγμιότυπα της κλάσης αυτής χρησιμοποιούνται σε συντακτικούς κανόνες, όπου είναι δυνατό να αλλάξει η ορατότητα μεταβλητών ή να γίνουν νέες δηλώσεις. Αυτοί οι κανόνες είναι οι translation_unit και block_item_list.

Ο κανόνας translation_unit παράγει το ριζικό κόμβο του δέντρου, και περιέχει τις καθολικες δηλώσεις, δηλώσεις συναρτήσεων και οτιδήποτε μπορεί να δηλωθεί σε καθολική ορατότητα.

Ο κανόνας block_item_list χρησιμοποιείται όπου υπάρχει σύνθετο σύνολο εντολών (compound statement). Η αρχή και το τέλος ενός τέτοιου συνόλου σηματοδοτούνται από το άνοιγμα και το κλείσιμο του άγκιστρου αντίστοιχα ({. }). Μπορεί να συναντηθεί σε περιπτώσεις κώδικα όπως:

- σώμα συνάρτησης
- σώμα εντολής if
- σώμα εντολής for

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Ας θεωρήσουμε το απλό παράδειγμα σύνθετου συνόλου εντολών όπως δείχνει ο Πίνακας 4.2.

Πίνακας 4.2 Παράδειγμα σύνθετου συνόλου εντολών

```
{
  a++;
}
```

Το απόσπασμα της δενδρικής δομής που παράγεται κατά την μετάφραση του κώδικα ταιριάζει τους εξής συντακτικούς κανόνες (δεν παρουσιάζονται όλες οι εκδοχές των κανόνων για απλότητα):

Πίνακας 4.3 Απόσπασμα συντακτικών κανόνων bison

```
compound_statement:
    '{' block_item_list '}'
    {
      $$ = new gennode(PR_COMPOUND_STATEMENT_FULL, $1, $2, $3);
    }
;

block_item_list:
    block_item
    {
      $$ = new genblock(PR_BLOCK_ITEM_LIST,
        active_typenames_collection,
        active_declarations_collection, $1);
    }
;
```

Ο κανόνας compound_statement παράγει τον block_item_list. Αυτός με τη σειρά του παράγει ένα block_item, που γίνεται statement, expression_statement, expression

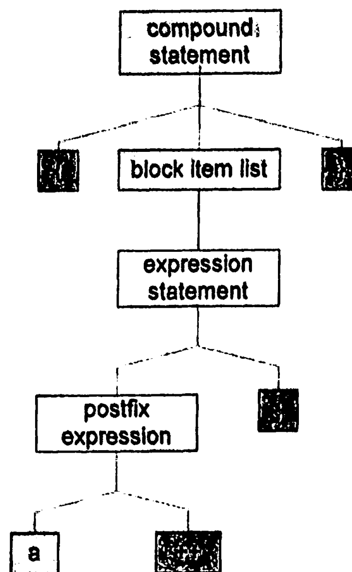
κ.ο.κ μέχρι να γίνει postfix_expression και να καταλήξει στους κανόνες που απεικονίζει ο Πίνακας 4.4.

Πίνακας 4.4 Απόσπασμα συντακτικών κανόνων bison

```
postfix_expression:
    primary_expression
    {
        $$ = $1;
    }
    | postfix_expression INC_OP
    {
        $$ = new gennode(PR_POSTFIX_EXPRESSION, $1, $2);
    }
;

primary_expression:
    IDENTIFIER
    {
        $$ = $1;
    }
;
```

Τελικά στο ΣΣΔ μένουν μόνο οι κανόνες που είναι απαραίτητοι. Η γραφική αναπαράσταση του αποσπάσματος ΣΣΔ που συνδέεται με τον κώδικα εισόδου είναι όπως στο Σχήμα 4.11.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

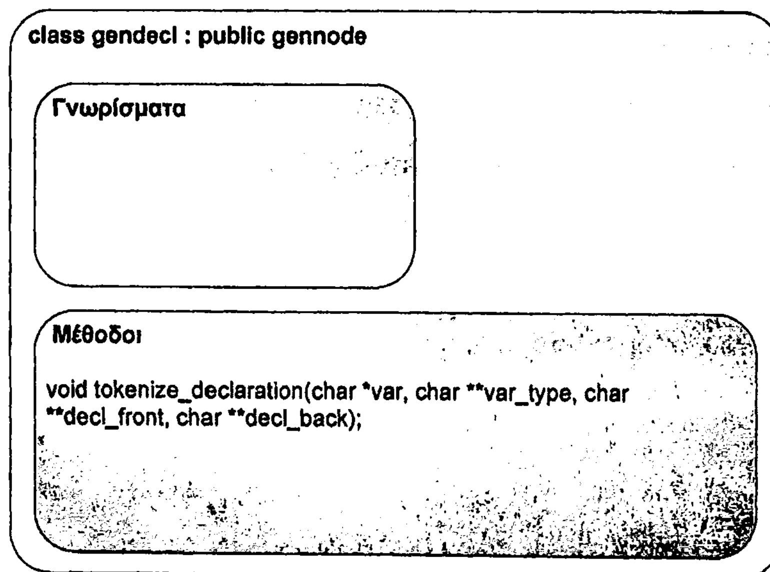


Σχήμα 4.11 Απόσπασμα του ΣΣΔ

4.2.1.2.2. Η κλάση gendecl

Η κλάση gendecl δεν έχει κάποια επιπλέον γνωρίσματα σε σχέση με την προγονική gennode.

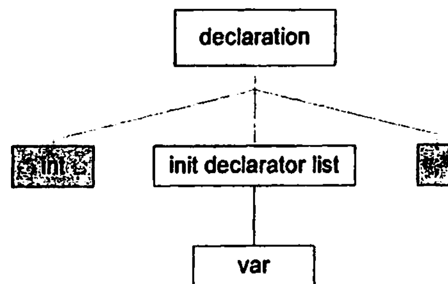
Δημιουργείται στη θέση της gennode, όταν οι κόμβοι παιδιά είναι μια δήλωση μεταβλητής για ευκολία στον έπειτα χειρισμό.



Σχήμα 4.12 Η κλάση gendecl

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Για παράδειγμα, η δήλωση "int a;" μετατρέπεται μέσω του συντακτικού αναλυτή στο υπόδεντρο δήλωσης του Σχήμα 4.13



Σχήμα 4.13 Υπόδεντρο δήλωσης μεταβλητής

4.2.1.2.3. Η κλάση genomp

Η κλάση genomp κληρονομεί από την κλάση gennode. Περιέχονται επιπλέον κάποια γνωρίσματα:

- parallel_level, ακέραιος
- var_list, χάρτης
- options_mask, ακέραιος

Το γνώρισμα parallel_level περιέχει μετά τη δημιουργία του ΣΣΔ έναν μη αρνητικό αριθμό είναι η τιμή του επιπέδου εμφωλιασμού της εντολής OpenMP. Έτσι, όταν έχουμε για παράδειγμα τρεις δομές/οδηγίες εμφωλευμένες τη μία μέσα στην άλλη σε τρία επίπεδα, αυτές θα λάβουν από το εξωτερικό προς το εσωτερικό επίπεδο τις τιμές 0, 1 και 2 αντίστοιχα.

Ο χάρτης var_list έχει ως κλειδί μια συμβολοσειρά χαρακτήρων και περιέχει τη δομή vardata σε κάθε κόμβο, η οποία δηλώνεται ως εξής:

Πίνακας 4.5 Δήλωση της δομής vardata

```
struct vardata
{
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
  int freq;
  gendecl *decl;
};
```

Το πεδίο freq περιέχει τη συχνότητα εμφανίσεων της μεταβλητής στο τμήμα κώδικα της εντολής OpenMP, αν είναι μη μηδενικό. Αν είναι μικρότερο του μηδενός, παίρνει μια από τις τιμές ενός απαριθμητή που σημαίνει τον τύπο της μεταβλητής μέσα στον κώδικα.

Το πεδίο decl είναι δείκτης στον μη τερματικό κόμβο δήλωσης (βλέπε παραπάνω) της μεταβλητής. Έχει νόημα μόνο όταν η τιμή του freq είναι μικρότερη από μηδέν.

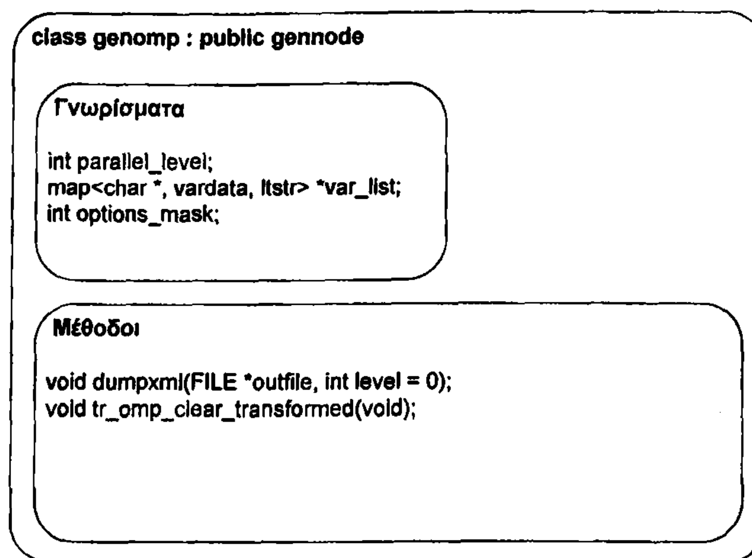
Για την αναζήτηση ανάμεσα στα στοιχεία του χάρτη παρέχεται η δομή ltstr (less-than string) που είναι ορισμένη όπως δείχνει ο Πίνακας 4.6.

Πίνακας 4.6 Η δομή ltstr

```
struct ltstr
{
  bool operator()(const char* s1, const char* s2) const
  {
    return strcmp(s1, s2) < 0;
  }
};
```

Το γνώρισμα options_mask είναι μια μάσκα ψηφίων (bit mask) όπου αποθηκεύονται οι επιλογές που σχετίζονται με την εντολή OpenMP.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 4.14 Η κλάση genomp

Η κλάση genomp χρησιμοποιείται από τον συντακτικό αναλυτή κατά τη δημιουργία ενός μη τερματικού κόμβου εντολής ή οδηγίας OpenMP.

Με βάση τους κόμβους genomp γίνεται στο δεύτερο πέρασμα η επεξεργασία του ΣΣΔ για το μετασχηματισμό κώδικα.

4.2.2. Συλλογές μεταβλητών και τύπων χρήστη

Κατά τη συντακτική ανάλυση του κώδικα εισόδου συναντώνται δηλώσεις μεταβλητών και τύπων χρήστη. Επειδή είναι απαραίτητη η αναφορά στις δηλώσεις των μεταβλητών στο δεύτερο πέρασμα, η κλάση genblock περιλαμβάνει ως γνωρίσματα δείκτες σε δύο τέτοιες συλλογές.

Κοινός πρόγονος των συλλογών είναι η κλάση generic_collection, που περιέχει όλα τα γνωρίσματα και υλοποιεί σχεδόν όλες τις μεθόδους τους.

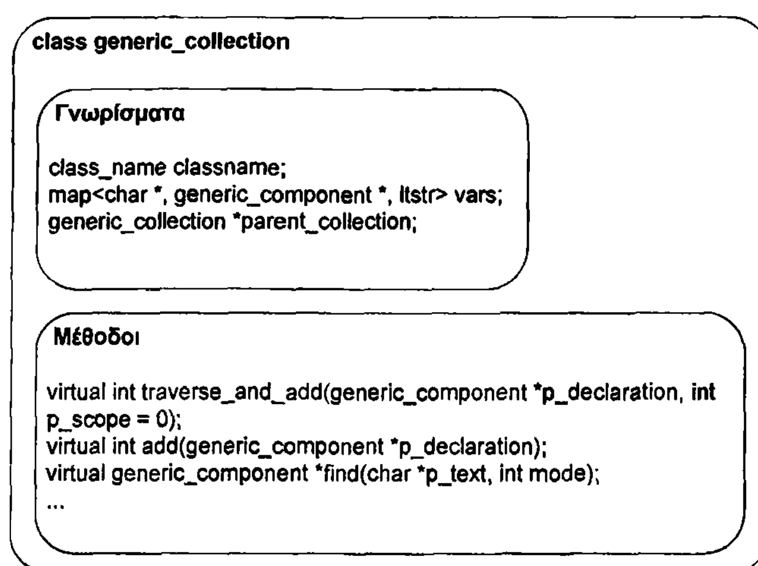
Τα γνωρίσματα της κλάσης generic_collection είναι:

- classname, απαριθμητής
- vars, χάρτης
- parent_collection, δείκτης

Ο απαριθμητής χρησιμοποιείται για να ταυτοποιεί το είδος της συλλογής.

Ο χάρτης χρησιμοποιεί ως κλειδί μια συμβολοσειρά χαρακτήρων για πρόσβαση σε μια αναφορά σε αντικείμενο generic_component. Χρησιμοποιεί την δομή που περιγράφηκε στην κλάση genomp για αναζήτηση.
Η αναφορά σε αντικείμενο generic_component χρησιμοποιείται για να αποθηκεύεται η δήλωση της μεταβλητής, για γρήγορη πρόσβαση.

Το γνώρισμα parent_collection είναι δείκτης στη συλλογή του προηγούμενου επιπέδου.

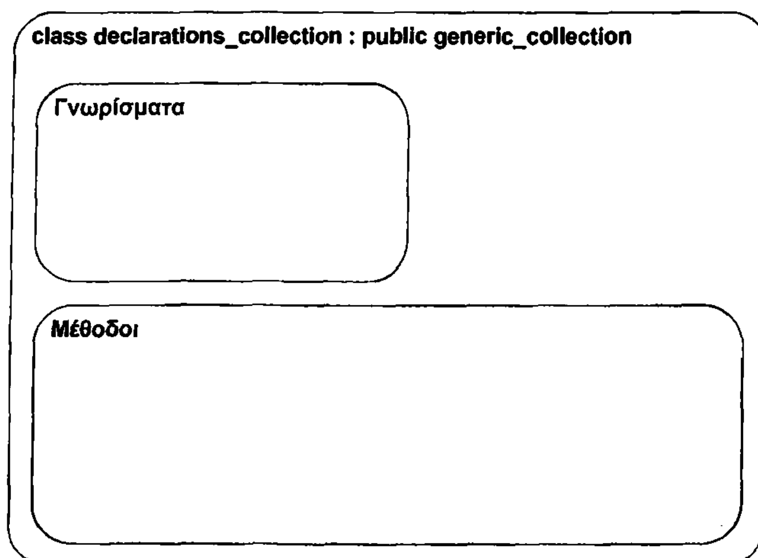


Σχήμα 4.15 Η κλάση generic_collection

Η μέθοδος traverse_and_add() διατρέχει το υπόδεντρο μιας δήλωσης μεταβλητής (απλής – int a; ή πολλαπλής – int a, b;) ώστε να προσθέσει στις συλλογές δηλώσεων καθεμία από τις δηλώσεις που συναντά. Επιστρέφει τον αριθμό μεταβλητών που βρέθηκαν για προσθήκη.
Η μέθοδος add() είναι η χαμηλού επιπέδου λειτουργία που προσθέτει νέες μεταβλητές σε συλλογή δηλώσεων, και επιστρέφει μηδέν για επιτυχία, ένα για αποτυχία.
Η μέθοδος find() εκτελεί αναζήτηση σε συλλογές δηλώσεων για μια συγκεκριμένη μεταβλητή. Η παράμετρος mode καθορίζει αν η αναζήτηση θα γίνει αναδρομικά προς προηγούμενα επίπεδα ορατότητας ή όχι. Επιστρέφει τον κόμβο της δήλωσης μέσα στη συλλογή όπου βρέθηκε.

4.2.2.1. Οι κλάσεις declarations_collection και typenames_collection

Οι κλάσεις declarations_collection και typenames_collection κληρονομούν από την κλάση generic_collection. Δεν έχουν κάποια επιπλέον γνωρίσματα ή μεθόδους και υπάρχουν για μελλοντική χρήση.

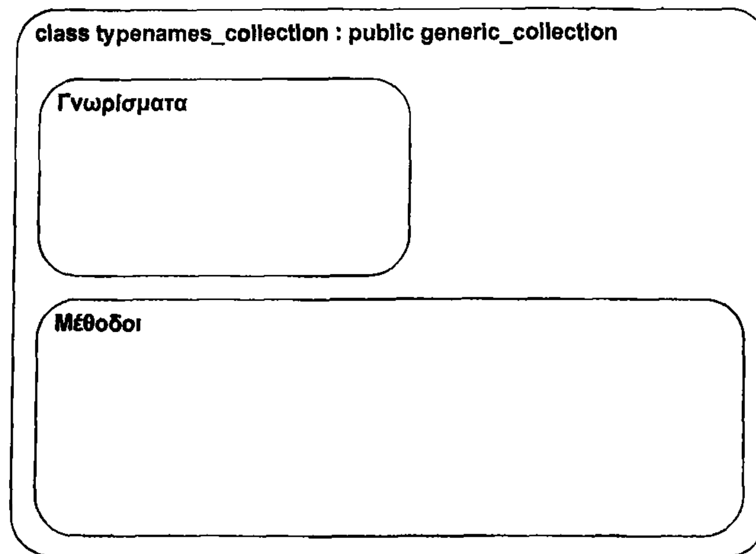


Σχήμα 4.16 Η κλάση declarations_collection

Όπως και με την κορυφαία κλάση της κληρονομικότητας των αντικειμένων που χρησιμοποιούνται για την δημιουργία τερματικών και μη τερματικών κόμβων του ΣΣΔ, η κλάση generic_collection δεν χρησιμοποιείται ποτέ άμεσα. Αντί γι' αυτή, χρησιμοποιούνται ως εξής:

- declarations_collection: Περιέχει δηλώσεις μεταβλητών, πινάκων, δεικτών κλπ
- typenames_collection: Περιέχει δηλώσεις τύπων που ορίζονται από το χρήστη με τη δεσμευμένη λέξη typedef.

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ



Σχήμα 4.17 Η κλάση typenames_collection

4.3. Σύνδεση τερματικών και μη τερματικών αντικειμένων

Με βάση τους συντακτικούς κανόνες που ταιριάζουν σε κάθε λεκτική είσοδο, είναι δυνατό να συνδέσουμε τα τερματικά και μη τερματικά αντικείμενα που δημιουργήθηκαν σε μια δομή δέντρου.

Σκοπός της δημιουργίας αυτής της δομής είναι η διατήρηση όλης της πληροφορίας που ανακτήθηκε από την είσοδο του μεταφραστή. Αυτό καθιστά δυνατή την επεξεργασία της εισόδου σε ένα ανεξάρτητο πέρασμα από αυτό της λεκτικής, συντακτικής και σημασιολογικής ανάλυσης.

4.4. Εξαγωγή δέντρου σε μορφή XML

Είναι δυνατό το ΣΣΔ να εξαχθεί από τον μεταφραστή σε μορφή XML και να αποθηκευτεί σε αρχείο, για μετέπειτα εξέταση της δομής του.

Ας θεωρήσουμε το πρόγραμμα που απεικονίζει ο Πίνακας 4.7.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Πίνακας 4.7 Παράδειγμα προγράμματος OpenMP

```
int global_var;

int main(int argc, char *argv[])
{
  int local_var;

  #pragma omp parallel
  {
    local_var++;
  }
}
```

Η XML έξοδος του μεταφραστή θα είναι όπως δείχνει ο Πίνακας 4.8.

Πίνακας 4.8 Έξοδος XML του μεταφραστή

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<CN_BLOCK rule="PR_TRANSLATION_UNIT" numparams="2">
  <CN_TCOLL var0="__builtin_va_list"> </CN_TCOLL>
  <CN_DCOLL var0="global_var" var1="main"> </CN_DCOLL>
  <CN_DECL rule="PR_DECLARATION_2" numparams="3">
    <CN_TERM value="int" lineno="0"> </CN_TERM>
    <CN_NODE rule="PR_INIT_DECLARATOR_LIST" numparams="1">
      <CN_IDENT value="global_var" lineno="0"> </CN_IDENT>
    </CN_NODE>
    <CN_TERM value=";" lineno="0"> </CN_TERM>
  </CN_DECL>
  <CN_NODE rule="PR_FUNCTION_DEFINITION_1" numparams="3">
    <CN_TERM value="int" lineno="2"> </CN_TERM>
    <CN_NODE rule="PR_DIRECT_DECLARATOR_12" numparams="4">
      <CN_IDENT value="main" lineno="2"> </CN_IDENT>
      <CN_TERM value="(" lineno="2"> </CN_TERM>
      <CN_NODE rule="PR_PARAMETER_LIST" numparams="3">
        <CN_DECL rule="PR_PARAMETER_DECLARATION_1" numparams="2">
          <CN_TERM value="int" lineno="2"> </CN_TERM>
          <CN_IDENT value="argc" lineno="2"> </CN_IDENT>
        </CN_DECL>
        <CN_TERM value="," lineno="2"> </CN_TERM>
        <CN_DECL rule="PR_PARAMETER_DECLARATION_1" numparams="2">
          <CN_TERM value="char" lineno="2"> </CN_TERM>
          <CN_NODE rule="PR_DECLARATOR_POINTER" numparams="2">
            <CN_TERM value="*" lineno="2"> </CN_TERM>
            <CN_NODE rule="PR_DIRECT_DECLARATOR_3" numparams="3">
              <CN_IDENT value="argv" lineno="2"> </CN_IDENT>
```

```
                <CN_TERM value="[" lineno="2"> </CN_TERM>
                <CN_TERM value="]" lineno="2"> </CN_TERM>
              </CN_NODE>
            </CN_NODE>
          </CN_DECL>
        </CN_NODE>
        <CN_TERM value=")" lineno="2"> </CN_TERM>
      </CN_NODE>
      <CN_NODE rule="PR_COMPOUND_STATEMENT_FULL" numparams="3">
        <CN_TERM value="{" lineno="3"> </CN_TERM>
        <CN_BLOCK rule="PR_BLOCK_ITEM_LIST" numparams="2">
          <CN_DCOLL var0="argc" var1="argv" var2="local_var">
</CN_DCOLL>
          <CN_DECL rule="PR_DECLARATION_2" numparams="3">
            <CN_TERM value="int" lineno="4"> </CN_TERM>
            <CN_NODE rule="PR_INIT_DECLARATOR_LIST" numparams="1">
              <CN_IDENT value="local_var" lineno="4"> </CN_IDENT>
            </CN_NODE>
            <CN_TERM value=";" lineno="4"> </CN_TERM>
          </CN_DECL>
          <CN_OMP rule="PR_PARALLEL_CONSTRUCT" parallel_level="0"
numparams="2">
            <CN_NODE rule="PR_PARALLEL_DIRECTIVE" numparams="3">
              <CN_TERM value=";" lineno="4"> </CN_TERM>
              <CN_TERM value="parallel" lineno="6"> </CN_TERM>
              <CN_TERM value="\n" lineno="7"> </CN_TERM>
            </CN_NODE>
            <CN_NODE rule="PR_COMPOUND_STATEMENT_FULL" numparams="3">
              <CN_TERM value="{" lineno="7"> </CN_TERM>
              <CN_BLOCK rule="PR_BLOCK_ITEM_LIST" numparams="1">
                <CN_NODE rule="PR_EXPRESSION_STATEMENT" numparams="2">
                  <CN_NODE rule="PR_POSTFIX_EXPRESSION_INC_OP"
numparams="2">
                    <CN_IDENT value="local_var" lineno="8"> </CN_IDENT>
                    <CN_TERM value="++" lineno="8"> </CN_TERM>
                  </CN_NODE>
                  <CN_TERM value=";" lineno="8"> </CN_TERM>
                </CN_NODE>
              </CN_BLOCK>
              <CN_TERM value="}" lineno="9"> </CN_TERM>
            </CN_NODE>
          </CN_OMP>
        </CN_BLOCK>
        <CN_TERM value="}" lineno="10"> </CN_TERM>
      </CN_NODE>
    </CN_NODE>
</CN_BLOCK>
```

Ο Πίνακας 4.8 δείχνει τις μονάδες XML που προκύπτουν από την ανάλυση του κώδικα εισόδου μαζί με τα γνωρίσματά τους.

ΚΕΦΑΛΑΙΟ 5. ΜΕΤΑΤΡΟΠΗ ΣΕ ΠΟΛΥΝΗΜΑΤΙΚΟ ΚΩΔΙΚΑ

5.1. Εισαγωγή
5.2. Παραγωγή συναρτήσεων νημάτων
5.3. Μεταβλητές
5.4. Φωλιασμένες οδηγίες
5.5. Κωδικοποίηση ιδιοτήτων μεταβλητών
5.6. Ανάλυση οδηγιών
5.7. Τροποποίηση ΣΣΔ
5.8. Μετασχηματισμοί

5.1. Εισαγωγή

Η συγγραφή κώδικα που είναι σύμφωνος με το πρότυπο OpenMP περιλαμβάνει οδηγίες προς τον προεπεξεργαστή που καθοδηγούν τον μεταφραστή ώστε να παράγει κώδικα που εκτελείται με χρήση πολλαπλών νημάτων.

Για να μπορέσουμε να μετατρέψουμε δεδομένο κώδικα σε πολυνηματικό είναι αναγκαίο, όπου απαιτείται δημιουργία νημάτων, να παραχθεί μια συνάρτηση την οποία αυτά θα εκτελούν. Για την παραγωγή της συνάρτησης απαιτείται γνώση των δηλώσεων μεταβλητών και τύπων, μαζί με την πληροφορία της ορατότητάς τους.

Κατά τη συντακτική και σημασιολογική ανάλυση έγινε η ανάλυση του κώδικα εισόδου και η εισαγωγή των δηλώσεων μεταβλητών και τύπων στις αντίστοιχες συλλογές δηλώσεων. Σε αυτή τη δεύτερη φάση εκτέλεσης του μεταφραστή,

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

αξιοποιούμε τα δεδομένα που συλλέξαμε κατά τη δημιουργία του ΣΣΔ για να επιτύχουμε τους ανάλογους μετασχηματισμούς.

5.2. Παραγωγή συναρτήσεων νημάτων

Το πρότυπο OpenMP περιλαμβάνει τρεις οδηγίες που προκαλούν τη δημιουργία πολλαπλών νημάτων. Αυτές είναι οι:

- #pragma omp parallel
- #pragma omp parallel for
- #pragma omp parallel sections

Η πρώτη είναι η βασική οδηγία δημιουργίας μιας ομάδας νημάτων, για εκτέλεση ενός τμήματος κώδικα. Οι άλλες δύο είναι συμπτύξεις των δομών:

- #pragma omp parallel, αμέσως ακολουθούμενη από μια #pragma omp for
- #pragma omp parallel, αμέσως ακολουθούμενη από μια #pragma omp sections
- αντίστοιχα.

Για να εκτελεστεί ένα τμήμα κώδικα από νήματα, πρέπει αυτό να απομονωθεί και να μετακινηθεί σε μια ξεχωριστή συνάρτηση. Αυτή καλείται κατά την εκτέλεση του μετασχηματισμένου προγράμματος από τη συνάρτηση βιβλιοθήκης για τη δημιουργία νημάτων. Ένα παράδειγμα που αφορά τη χρήση της βιβλιοθήκης νημάτων POSIX threads δείχνει ο Πίνακας 5.1.

Πίνακας 5.1 Δημιουργία νήματος με χρήση POSIX threads

```
#include <stdio.h>
#include <pthread.h>

int counter = 0;

/* Συνάρτηση νήματος */
void *threadfunc(void *param)
{
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
  counter++;
  return 0;
}

int main(int argc, char *argv[])
{
  pthread_t tid; /* αναγνωριστής νήματος */
  pthread_attr_t attr; /* χαρακτηριστικά νήματος */

  /* αρχικοποίηση χαρακτηριστικών */
  pthread_attr_init(&attr);
  /* εκτέλεση νήματος με τη συνάρτηση threadfunc */
  pthread_create(&tid, &attr, threadfunc, (void *)NULL);
  /* αναμονή για τερματισμό του νήματος */
  pthread_join(tid, NULL);

  return 0;
}
```

Η λογική που ακολουθεί ο μεταφραστής OMPi είναι η ίδια, με τη διαφορά ότι την παραγωγή νημάτων την αναλαμβάνει μια κλήση προς τη βιβλιοθήκη χρόνου εκτέλεσης (runtime library), η _omp_create_team. Αντίστοιχα, την καταστροφή των νημάτων της ομάδας την αναλαμβάνει η κλήση _omp_destroy_team. Ο Πίνακας 5.3 περιέχει τον μετασχηματισμένο κώδικα του προγράμματος που δείχνει ο Πίνακας 5.2.

Πίνακας 5.2 Πρόγραμμα με οδηγία OpenMP

```
#include <stdio.h>
#include <pthread.h>
#include <omp.h>

int counter = 0;

int main(int argc, char *argv[])
{
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
#pragma omp parallel
{
  counter++;
  return 0;
}

  return 0;
}
```

Πίνακας 5.3 Δημιουργία νημάτων με χρήση της βιβλιοθήκης χρόνου εκτέλεσης του OMPi

```
#include <stdio.h>
#include <omp.h>
#include <ompi.h>

int counter = 0;

/* Δήλωση μορφής της συνάρτησης νήματος */
void *main_parallel_0(void *);

int main(int argc, char *argv[])
{
  _omp_initialize();
  {
    /* Αρχικοποίηση μεταβλητών της συνάρτησης νήματος */
    _OMP_PARALLEL_DECL_VARSTRUCT(main_parallel_0);
    /* Δημιουργία ομάδας νημάτων */
    _omp_create_team((-1),  _OMP_THREAD,  main_parallel_0,  (void  *)
&main_parallel_0_var);
    _omp_destroy_team(_OMP_THREAD->parent);
  }
  return 0;
}

/* Συνάρτηση νημάτων */
void *main_parallel_0(void *_omp_thread_data)
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
{
  int _omp_dummy = _omp_assign_key(_omp_thread_data);
  {
    counter++;
  }
  return 0;
}
```

Όπως δείχνει ο Πίνακας 5.3, για τη χρήση των κλήσεων βιβλιοθήκης απαιτείται, εκτός από τις ίδιες τις κλήσεις, και η αρχικοποίηση της βιβλιοθήκης. Επίσης απαιτείται η δήλωση μεταβλητών που τυχόν χρησιμοποιούνται στο τμήμα κώδικα που μετακινήθηκε. Οι μεταβλητές δηλώνονται σε καθολικής ορατότητας δομή με όνομα main_parallel_0_vars, η οποία στο συγκεκριμένο παράδειγμα είναι κενή:

Πίνακας 5.4 Δήλωση κενής δομής για τις μεταβλητές νήματος

```
typedef struct {
} main_parallel_0_vars;
```

Για συντομία και ευκολία κατανόησης του κώδικα, γίνεται εκτενής χρήση macros, όπως τα _OMP_PARALLEL_DECL_VARSTRUCT και _OMP_THREAD, τα οποία είναι δηλωμένα στο αρχείο ompi.h.

5.3. Μεταβλητές

Κατά την αποκοπή και μεταφορά του κώδικα νήματος σε δική του συνάρτηση, είναι αναγκαίο να διατηρηθεί και η πρόσβαση στις μεταβλητές τις οποίες αυτό περιέχει.
Η απαίτηση αυτή προκύπτει από το γεγονός ότι αν πάρουμε ένα τμήμα κώδικα και το μεταφέρουμε σε μια συνάρτηση νήματος, οι μεταβλητές και οι τύποι του που είναι δηλωμένοι καθολικά θα είναι προσβάσιμα από αυτό. Κάποιες μεταβλητές όμως είναι πιθανά δηλωμένες σε σύνθετα εμφωλευμένα τμήματα κώδικα μέσα σε συναρτήσεις.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Μέσα σε αυτές συμπεριλαμβάνονται και οι μεταβλητές που σκιάζουν την ορατότητά τυχόν καθολικών με το ίδιο όνομα.

Ας θεωρήσουμε το παράδειγμα κώδικα OpenMP που δείχνει ο Πίνακας 5.5.

Πίνακας 5.5 Παράδειγμα καθολικής και τοπικής μεταβλητής σε τμήμα κώδικα

```
#include <stdio.h>
#include <omp.h>

int global_variable = 0;

int main(int argc, char *argv[])
{
  int local_variable;

  #pragma omp parallel
  {
    global_variable++;
    local_variable++;
  }

  return 0;
}
```

Στο δομημένο τμήμα κώδικα υπάρχουν αναφορές σε δύο μεταβλητές, μια καθολική του προγράμματος και μια τοπική της συνάρτησης main. Δεδομένου ότι η νέα συνάρτηση θα τοποθετηθεί στο τέλος του προγράμματος, κατά την αποκοπή και μεταφορά του κώδικα που περικλείεται στην οδηγία #pragma omp parallel, η καθολική μεταβλητή εξακολουθεί να είναι ορατή. Η τοπική μεταβλητή local_variable όμως δεν είναι δηλωμένη σε εκείνο το επίπεδο κώδικα.

Προφανώς πρέπει η μεταβλητή που χάνει τη δήλωσή της να δηλωθεί ξανά μέσα στη νέα συνάρτηση. Για να παρακαμφθεί αυτό το πρόβλημα, χρησιμοποιούμε την εξής

τέχνική: Δηλώνεται στην αρχή του προγράμματος μια καθολικής ορατότητας δομή η οποία περιέχει δείκτες που παραπέμπουν στις τοπικές μεταβλητές.

Η δομή αρχικοποιείται στο τμήμα κώδικα που αντικαθιστά την οδηγία OpenMP. Μετά από την αρχικοποίηση δημιουργείται η ομάδα νημάτων που εκτελεί τη νέα συνάρτηση. Στην αρχή της συνάρτησης γίνεται και η δήλωση των δεικτών που αντικαθιστούν τις τοπικές μεταβλητές.

Αυτό καθιστά αναγκαία και τη μετατροπή του τμήματος κώδικα που μετατέθηκε, ώστε πλέον οι μη καθολικές μεταβλητές που χρησιμοποιούνται εκεί να χρησιμοποιούν την τιμή του δείκτη που ορίστηκε. Έτσι η έκφραση με χρήση της μεταβλητής local_variable:

```
local_variable++;
```

θα μετατραπεί στην εξής, δεδομένης της δήλωσης δείκτη με όνομα local_variable και ίδιο τύπο μέσα στη νέα συνάρτηση:

```
(*local_variable)++;
```

Η ορατότητα των μεταβλητών όπως αυτές χρησιμοποιούνται είναι δυνατό να μεταβληθεί με τη χρήση φράσεων που ακολουθούν τις οδηγίες. Για παράδειγμα, η φράση private (*list*) δηλώνει πως στην οδηγία αυτή η μεταβλητή που δηλώθηκε πρέπει να θεωρηθεί ιδιωτική.

Η γενική σύνταξη είναι όπως δείχνει ο Πίνακας 5.6.

Πίνακας 5.6 Σύνταξη φράσεων οδηγιών

```
#pragma omp <directive or construct> private (list)
```

Ας δούμε αναλυτικά τις φράσεις που υποστηρίζονται στις οδηγίες OpenMP και τις αλλαγές που επιφέρουν, σε σχέση με τα παραπάνω, στον παραγόμενο κώδικα.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

5.3.1. private (list)

Η φράση private με λίστα μεταβλητών καθοδηγεί τον προεπεξεργαστή στη δημιουργία μιας μεταβλητής τοπικής ως προς τον κώδικα της οδηγίας, η οποία σκιάζει οποιαδήποτε μεταβλητή με το ίδιο όνομα θα ήταν ορατή σε αυτό το επίπεδο κώδικα επειδή είναι δηλωμένη σε προηγούμενο σύνθετο σύνολο κώδικα.

Κατά την παραγωγή του κώδικα γίνονται τα εξής:

- με βάση τις συλλογές μεταβλητών που έχουν αποθηκευτεί στο ΣΣΔ, βρίσκεται η δήλωση της μεταβλητής
- γίνεται δήλωση της μεταβλητής από την αρχή με τον ίδιο τύπο και όνομα, στην αρχή του κώδικα που μετατοπίστηκε
- η μεταβλητή χρησιμοποιείται όπως ήταν μέσα στον κώδικα

5.3.2. firstprivate (list)

Η φράση firstprivate χρησιμοποιείται με τον ίδιο τρόπο όπως και η private. Η διαφορά τους είναι ότι η μεταβλητή που έχει δηλωθεί ως firstprivate, αρχικοποιείται στην τιμή που είχε πριν από το τμήμα κώδικα, αμέσως μετά τη νέα δήλωσή της.

Έτσι η μεταβλητή φέρει την ίδια τιμή, αλλά με το τέλος του μπλοκ κώδικα οποιαδήποτε αλλαγή σε αυτή την τιμή χάνεται, γιατί ο κώδικας φεύγει από την ορατότητά της.

5.3.3. lastprivate (list)

Η φράση lastprivate συναντάται μόνο σε οδηγίες for και sections και είναι αντίστοιχη με τη firstprivate. Η διαφορά είναι ότι η μεταβλητή δηλώνεται ξανά μέσα στο μπλοκ κώδικα, και όταν αυτό τελειώσει, ενημερώνει την τιμή της μεχρι τότε σκιασμένης μεταβλητής.

5.3.4. reduction (operator : list)

Η φράση reduction ακολουθείται από έναν τελεστή και μια λίστα από μία ή περισσότερες μεταβλητές. Για κάθε στοιχείο της λίστας, δημιουργείται μια ιδιωτική

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

μεταβλητή και αρχικοποιείται ανάλογα με τον τελεστή. Μετά το τέλος του μπλοκ κώδικα, η σκιασμένη μεταβλητή ενημερώνεται με την τιμή της ιδιωτικής μεταβλητής, χρησιμοποιώντας τον καθορισμένο τελεστή.

Οι τιμές αρχικοποίησης των ιδιωτικών μεταβλητών είναι όπως δείχνει ο Πίνακας 5.7, ανάλογα με τον τελεστή που χρησιμοποιείται.

Πίνακας 5.7 Τελεστές και αρχικές τιμές για τη φράση reduction

Τελεστής	Αρχική τιμή
+	0
*	1
-	0
&	~0
|	0
^	0
&&	1
||	0

- Μετά το τέλος του μπλόκ, συνδυάζονται οι τιμές όλων των ιδιωτικών μεταβλητών των νημάτων με βάση την τιμή της αρχικής μεταβλητής, και με το αποτέλεσμα ενημερώνεται η σκιασμένη μεταβλητή.

5.3.5. default (shared|none)

Η φράση default συναντάται μόνο σε οδηγίες parallel, parallel for και parallel sections. Επιτρέπει στον προγραμματιστή να ελέγξει τις ιδιότητες διαμοιρασμού των μεταβλητών που εμφανίζονται στο μπλοκ κώδικα, και των οποίων οι ιδιότητες έχουν καθοριστεί έμμεσα, όπως περιγράφηκε στην αρχή της ενότητας.

Η φράση ακολουθείται από μια παράμετρο με δύο δυνατές τιμές:

- *shared*: όλες οι μεταβλητές που εμφανίζονται και που δεν έχουν προκαθορισμένες ιδιότητες θεωρούνται διαμοιραζόμενες.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- *none*: για όλες οι μεταβλητές που εμφανίζονται και δεν έχουν προκαθορισμένες ιδιότητες πρέπει να καθοριστούν ιδιότητες διαμοιρασμού, μέσω άλλων φράσεων ή οδηγιών.

5.3.6. shared (list)

Η φράση shared ακολουθούμενη από μια λίστα μεταβλητών καθορίζει ότι οι μεταβλητές που την ακολουθούν είναι διαμοιραζόμενες ανάμεσα στα νήματα εκτέλεσης.

5.3.7. copyin (list)

Η φράση copyin ακολουθούμενη από μια λίστα μεταβλητών παρέχει ένα μηχανισμό για την αντιγραφή τιμών από μια μεταβλητή threadprivate του νήματος-αρχηγού σε καθεμία threadprivate μεταβλητή των άλλων μελών της ομάδας νημάτων που εκτελούν μια παράλληλη περιοχή.

5.3.8. copyprivate (list)

Η φράση copyprivate ακολουθούμενη από μια λίστα μεταβλητών παρέχει ένα μηχανισμό για τη χρήση μιας ιδιωτικής μεταβλητής για μεταφορά τιμής από ένα νήμα στα άλλα νήματα της ίδιας ομάδας.

5.4. Φωλιασμένες οδηγίες

Το πρότυπο OpenMP επιτρέπει το φώλιασμα οδηγιών κατά τη συγγραφή κώδικα και καθορίζει τις ιδιότητες διαμοιρασμού για μεταβλητές μέσα σε δομές parallel ως εξής:

- Οι μεταβλητές που είναι ορισμένες σε οδηγίες threadprivate, έχουν την ιδιότητα διαμοιρασμού threadprivate
- Οι μεταβλητές που είναι ορισμένες μέσα στη δομή είναι ιδιωτικές
- Οι μεταβλητές που χρησιμοποιούν δυναμική μνήμη είναι διαμοιραζόμενες
- Τα μέλη δομών static είναι διαμοιραζόμενα

- Η μεταβλητή του βρόχου for σε μια δομή for ή parallel for είναι ιδιωτική στη δομή αυτή

5.5. Κωδικοποίηση ιδιοτήτων μεταβλητών

Για να καταστεί δυνατή η μετονομασία των μεταβλητών που εμφανίζονται σε ένα τμήμα κώδικα, πρέπει να αποθηκευτεί στο ΣΣΔ η πληροφορία που συναντάται διασχίζοντας το δέντρο.

Για τό λόγο αυτό χρησιμοποιούνται οι χάρτες μεταβλητών που είδαμε στην ανάλυση της κλάσης genomp, σε συνδυασμό με έναν απαριθμητή.

Ένας χάρτης (map) της βασικής βιβλιοθήκης της C++ είναι μια ταξινομημένη συνδετική δομή (sorted associative container) η οποία συνδέει αντικείμενα του τύπου *Κλειδί* με αντικείμενα του τύπου *Δεδομένο* [8].

Πίνακας 5.8 Η δήλωση του χάρτη μεταβλητών

```
map<char *, vardata, ltstr> *var_list;
```

Στην υλοποίησή μας, ο χάρτης που χρησιμοποιείται έχει ως κλειδί ένα πεδίο τύπου char *, όπου αποθηκεύεται το όνομα της μεταβλητής και ως δεδομένο τη δομή vardata που ορίζεται όπως δείχνει ο Πίνακας 5.9.

Πίνακας 5.9 Η δομή που χρησιμοποιείται στον χάρτη μεταβλητών

```
struct vardata
{
  int freq;
  gendecl *decl;
};
```

Η δήλωση του χάρτη επιτρέπει και τη δήλωση της υπορουτίνας ltstr, η οποία καθορίζει τη σχέση «μικρότερο από» για το κλειδί, όπως δείχνει ο Πίνακας 5.10.

Πίνακας 5.10 Η υπορουτίνα «μικρότερο από» του χάρτη

```
struct ltstr
{
  bool operator()(const char* s1, const char* s2) const
  {
    return strcmp(s1, s2) < 0;
  }
};
```

Η δομή vardata έχει δύο μέλη, έναν ακέραιο freq και ένα δείκτη decl σε αντικείμενο τύπου gendecl. Ο ακέραιος παίρνει τιμές από τον απαριθμητή var_type ο οποίος ορίζεται όπως στον Πίνακας 5.11, ή κάποια θετική τιμή για τη συχνότητα εμφάνισης μιας μεταβλητής.

Πίνακας 5.11 Οι τιμές του απαριθμητή var_type (τοπική μεταβλητή στον παραπάνω πίνακα σημαίνει πως η μεταβλητή είναι ορισμένη σε επίπεδο του κώδικα πριν από την οδηγία OpenMP αλλά όχι σε καθολικό επίπεδο)

Τιμή	Χρήση
VAR_PRIVATE	Μεταβλητή private
VAR_FPRIV_LOCAL	Τοπική μεταβλητή firstprivate
VAR_FPRIV_GLOBAL	Καθολική μεταβλητή firstprivate
VAR_LPRIV_LOCAL	Τοπική μεταβλητή lastprivate
VAR_LPRIV_GLOBAL	Καθολική μεταβλητή lastprivate
VAR_RED_LPLUS	Τοπική μεταβλητή reduction με τελεστή '+'
VAR_RED_LMINUS	Τοπική μεταβλητή reduction με τελεστή '-'
VAR_RED_LMULT	Τοπική μεταβλητή reduction με τελεστή '*'
VAR_RED_LXOR	Τοπική μεταβλητή reduction με τελεστή '^'
VAR_RED_LBITOR	Τοπική μεταβλητή reduction με τελεστή '|'

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

VAR_RED_LBITAND	Τοπική μεταβλητή reduction με τελεστή '&'
VAR_RED_LOR	Τοπική μεταβλητή reduction με τελεστή '||'
VAR_RED_LAND	Τοπική μεταβλητή reduction με τελεστή '&&'
VAR_RED_GPLUS	Καθολική μεταβλητή reduction με τελεστή '+'
VAR_RED_GMINUS	Καθολική μεταβλητή reduction με τελεστή '-'
VAR_RED_GMULT	Καθολική μεταβλητή reduction με τελεστή '*'
VAR_RED_GXOR	Καθολική μεταβλητή reduction με τελεστή '^'
VAR_RED_GBITOR	Καθολική μεταβλητή reduction με τελεστή '|'
VAR_RED_GBITAND	Καθολική μεταβλητή reduction με τελεστή '&'
VAR_RED_GOR	Καθολική μεταβλητή reduction με τελεστή '||'
VAR_RED_GAND	Καθολική μεταβλητή reduction με τελεστή '&&'
VAR_TPRIV_LOCAL	Τοπική μεταβλητή threadprivate
VAR_TPRIV_GLOBAL	Καθολική μεταβλητή threadprivate
VAR_NOCHANGE	Χρήση για φωλιασμένες οδηγίες

Οι τιμές του απαριθμητή είναι αρνητικές ώστε να διαχωρίζουν τις μεταβλητές που έχουν προκαθορισμένες ιδιότητες διαμοιρασμού από αυτές για τις οποίες οι ιδιότητές τους ορίζονται έμμεσα.

5.6. Ανάλυση οδηγιών

Μετά το τέλος της συντακτικής και σημασιολογικής ανάλυσης του κώδικα, το ΣΣΔ περιέχει οδηγίες OpenMP με τη μορφή μη τερματικών κόμβων gencomp. Για να μπορέσουν να αναλυθούν και να μετασχηματιστούν οι δομές αυτές, ακολουθείται ο εξής αλγόριθμος:

- Εκτελείται αναζήτηση μέσα στο ΣΣΔ για κόμβους genomp πρώτου επιπέδου
- Για κάθε έναν από τους κόμβους αυτούς καλείται η μέθοδος που αντιστοιχεί στην ανάλυσή του
- Κατά τη διάρκεια της ανάλυσης και πριν την έναρξη των μετασχηματισμών, γίνεται εύρεση για φωλιασμένες οδηγίες φωλιασμένες ένα επίπεδο μέσα σε αυτή που επεξεργαζόμαστε
- Με αντίστοιχο τρόπο κάθε οδηγία εκτελεί την ανάλυση και τους μετασχηματισμούς που απαιτούνται

- Ο έλεγχος επιστρέφει στην οδηγία προηγούμενου επιπέδου για να γίνουν οι μετασχηματισμοί

5.6.1. Αναζήτηση για κόμβους genomp πρώτου επιπέδου

Καλείται η μέθοδος search_and_act τόσες φορές, όσες και οι οδηγίες που είναι δυνατό να συναντήσουμε στο πρώτο επίπεδο. Η ταυτοποίηση του επιπέδου του κόμβου γίνεται εύκολα με χρήση του γνωρίσματος parallel_level των κόμβων αυτών.

5.6.2. Κλήση αντίστοιχης μεθόδου

Όταν βρεθεί κόμβος που ταιριάζει στα δεδομένα της αναζήτησης, καλείται η μέθοδος που αντιστοιχεί στην οδηγία που συναντήθηκε. Γίνεται η ανάλυση της οδηγίας χρησιμοποιώντας τις φράσεις που την ακολουθούν. Επίσης ανατίθενται στο χάρτη του κόμβου οι ιδιότητες των μεταβλητών που είναι ρητά καθορισμένες. Γίνεται ανάλυση του μπλοκ κώδικα και ορίζονται οι ιδιότητες διαμοιρασμού για τις υπόλοιπες μεταβλητές

5.6.3. Εύρεση φωλιασμένων οδηγιών

Ξεκινώντας από τον κόμβο που βρέθηκε στην τελευταία αναζήτηση, ψάχνουμε για φωλιασμένες οδηγίες ένα επίπεδο βαθύτερα, πάλι με χρήση του γνωρίσματος parallel_level.

5.6.4. Κλήση αντίστοιχης μεθόδου

Με αντίστοιχο τρόπο όπως και στην αρχική αναζήτηση, καλείται η αντίστοιχη μέθοδος. Αυτή με τη σειρά της κάνει την ανάλυση της οδηγίας που συναντήθηκε, ψάχνει για φωλιασμένες οδηγίες ένα επίπεδο βαθύτερα κ.ο.κ.

5.6.5. Επιστροφή ελέγχου στην αρχική μέθοδο μετασχηματισμού

Μετά το τέλος της αναζήτησης για φωλιασμένες οδηγίες, ο έλεγχος επιστρέφει στη μέθοδο πρώτου επιπέδου που κλήθηκε. Αυτή εκτελεί πλέον τους απαραίτητους μετασχηματισμούς, παράγει τον κώδικα που αντιστοιχεί στην οδηγία και επιστρέφει τον έλεγχο στην αρχική αναζήτηση.

5.7. Τροποποίηση ΣΣΔ

Κατά τη διαδικασία του μετασχηματισμού τμήματος κώδικα, είναι απαραίτητη η εισαγωγή ή αφαίρεση τμημάτων από αυτόν. Ουσιαστικά, πρέπει να αποκοπεί ολόκληρο υπόδεντρο, να δημιουργηθεί νέο δέντρο βάσει του μετασχηματισμένου κώδικα και αυτό να προσαρτηθεί ως νέο υπόδεντρο του ΣΣΔ στη θέση του παλιού. Είναι αναγκαίο σε όλες τις προσθήκες και αφαιρέσεις να διατηρηθεί η ακεραιότητα και η δομή του ΣΣΔ. Για να επιτευχθεί αυτό, πρέπει οποιαδήποτε προσθήκη σε αυτό ή αφαίρεση από αυτό να ακολουθεί αυστηρά τους κανόνες με τους οποίους αυτό αρχικά δημιουργήθηκε. Στην περίπτωσή μας, οι κανόνες αυτοί είναι οι συντακτικοί κανόνες του OMPi.

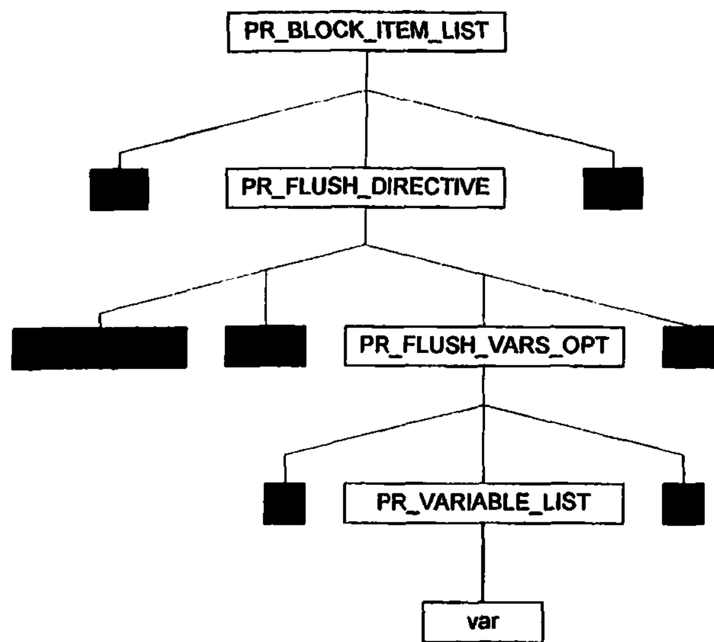
Ας δούμε τη διαδικασία χρησιμοποιώντας ένα παράδειγμα αποκοπής της οδηγίας

```
#pragma omp flush(var)
```

όπου var μια μεταβλητή ορισμένη στο επίπεδο του κώδικα που επεξεργαζόμαστε, και αντικατάστασής της από τον παραγόμενο κώδικα

```
_omp_flush((void *)&var);
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 5.1 Απόσπασμα του ΣΣΔ για την οδηγία #pragma omp flush(var)

5.7.1. Αφαίρεση υπόδεντρου από το ΣΣΔ

Η αφαίρεση υπόδεντρου από το ΣΣΔ αφορά σχεδόν αποκλειστικά τη διαγραφή από αυτό ολόκληρων δέντρων οδηγιών, ώστε να μην αλλοιωθεί ο κώδικας εισόδου (δεν αφαιρούνται απλά κάποια τερματικά ή μη τερματικά σύμβολα αλλά ολόκληρες εκφράσεις).

Γίνεται σε δύο βήματα:

- αποδέσμευση της μνήμης που χρησιμοποιούν τα αντικείμενα του υπόδεντρου, αφού έχουν αποθηκευτεί όποια στοιχεία από αυτό είναι χρήσιμα
- ενημέρωση του πατέρα μη τερματικού κόμβου με αφαίρεση της θέσης του δυναμικού πίνακα

5.7.1.1. Αποδέσμευση της μνήμης υποδέντρου

Κάθε κλάση της οποίας τα στιγμιότυπα απαρτίζουν το ΣΣΔ περιλαμβάνει έναν καταστροφέα (deconstructor). Στην περίπτωση τερματικών κόμβων, γίνεται αποδέσμευση των δυναμικά δεσμευμένων γνωρισμάτων, όπως π.χ. το γνώρισμα value.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Στην περίπτωση μη τερματικών κόμβων, διατρέχονται αναδρομικά όλοι οι κόμβοι παιδιά που περιέχονται στο δυναμικό πίνακα params. Μετά από αυτό, αποδεσμεύονται ο ίδιος ο δυναμικός πίνακας και όποια γνωρίσματα χρησιμοποιούν δυναμική δέσμευση μνήμης.

5.7.1.2. <u>Ενημέρωση του πίνακα παιδιών του πατέρα μη τερματικού κόμβου</u>

Μετά από την αποδέσμευση της μνήμης που καταλάμβανε το υποδέντρο, καλείται η μέθοδδς erase του δυναμικού πίνακα με παράμετρο τη θέση του που θα διαγραφεί. Ενημερώνεται και η βοηθητική μεταβλητή numparams με το νέο αριθμό των θέσεων στον πίνακα.

5.7.2. Προσθήκη υπόδεντρου στο ΣΣΔ

Για να προστεθεί ένα τμήμα κώδικα στο ΣΣΔ πρέπει αρχικά να αποθηκευτεί ο κώδικας σε μια συμβολοσειρά και έπειτα να τροφοδοτηθεί πίσω στον συντακτικό αναλυτή. Έτσι προκύπτει ένα υπόδεντρο σύμφωνο με τους κανόνες δημιουργίας του αρχικού δέντρου.

Λόγω δομής της εισόδου που πρέπει να μετατρέψουμε σε υπόδεντρο, προκύπτουν οι εξής δύο περιπτώσεις:

- ανάλυση μιας ολόκληρης νέας συνάρτησης ή καθολικής δήλωσης
- ανάλυση μιας ή περισσότερων εκφράσεων, χωρίς περιβάλλοντα κώδικα

Στην πρώτη περίπτωση, μπορούμε απλά να καλέσουμε τον συντακτικό αναλυτή, ο οποίος θα μας επιστρέψει το επιθυμητό υπόδεντρο. Παίρνοντας όμως ένα νέο ξεχωριστό υπόδεντρο, οι καθολικές δηλώσεις του βρίσκονται σε ξεχωριστές συλλογές δηλώσεων και τύπων, άρα χρειάζεται να γίνει συνένωση των νέων καθολικών δηλώσεων με τις αρχικές, του ΣΣΔ.

Για να αποφευχθεί αυτό το πρόβλημα, ο αναλυτής καλείται με μια παράμετρο που τον οδηγεί στο να χρησιμοποιήσει τις συλλογές δηλώσεων που ήδη υπάρχουν και απλά να τις ενημερώσει με το νέο κώδικα.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Στη δεύτερη περίπτωση, η αντιμετώπιση είναι λίγο πιο πολύπλοκη. Επειδή δεν είναι συντακτικά σωστή η ύπαρξη εκφράσεων (π.χ. a = b + c;) στο καθολικό επίπεδο του κώδικα, δημιουργείται μια εικονική συνάρτηση void transform() η οποία περικλείει τον κώδικα που θέλουμε να αναλύσουμε. Με αυτή την προϋπόθεση, η ανάλυση γίνεται κανονικά, αλλά επιστρέφει στο υπόδεντρο και τη δήλωση της εικονικής συνάρτησης. Είναι πολύ εύκολο να απομονώσουμε το υπόδεντρο που μας ενδιαφέρει, απλά ψάχνοντας για τον πρώτο μη τερματικό κόμβο με κανόνα δημιουργίας PR_BLOCK_ITEM_LIST. Η ανάλυση με αυτό τον τρόπο απαιτεί άλλη τιμή στην παράμετρο κλήσης του αναλυτή. Με αυτό τον τρόπο ορίζονται επίσης αυτόματα οι συλλογές δηλώσεων και τύπων του σημείου ενημέρωσης του ΣΣΔ, και δεν επηρεάζονται από την ψεύτικη συνάρτηση οι συλλογές του.
Μπορούμε έπειτα να χρησιμοποιήσουμε το υπόδεντρο αυτό όπως χρειάζεται, για να το προσθέσουμε στο ΣΣΔ.

Στο παράδειγμα της προηγούμενης ενότητας, αφού έχουμε αφαιρέσει την οδηγία #pragma omp flush (var) από το ΣΣΔ και έχουμε κρατήσει τη μεταβλητή ενημέρωσης var, παράγουμε τη συμβολοσειρά αντικατάστασης, η οποία είναι όπως δείχνει ο Πίνακας 5.12.

Πίνακας 5.12 Η εικονική συνάρτηση transform()

```
void transform()
{
  _omp_flush((void*)&var);
}
```

Το συντακτικό δέντρο που προκύπτει για την transform() μετά την ανάλυση της συμβολοσειράς έχει όπως στο Σχήμα 5.2.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

PR_TRANSLATION_UNIT
PR_FUNCTION_DEFINITION_1
PR_DIRECT_DECLARATOR_13
PR_COMPOUND_STATEMENT_FULL
transform
PR_BLOCK_ITEM_LIST
PR_EXPRESSION_STATEMENT
PR_POSTFIX_EXPRESSION_4
_omp_flush
PR_ARGUMENT_EXPRESSION_LIST
PR_CAST_EXPRESSION
PR_TYPE_NAME
PR_UNARY_EXPRESSION
var

Σχήμα 5.2 Υπόδεντρο ανάλυσης της συμβολοσειράς _omp_flush((void *) &var); με χρήση της εικονικής συνάρτησης transform()

Από το υπόδεντρο που προκύπτει, βρίσκουμε πολύ εύκολα τον μη τερματικό κόμβο PR_BLOCK_ITEM_LIST και από αυτόν παίρνουμε την έκφραση που μας ενδιαφέρει.

Για να προστεθεί το υπόδεντρο στο ΣΣΔ μας, ακολουθούμε την εξής διαδικασία:

- Βρίσκουμε τη θέση στην οποία θέλουμε να το εισάγουμε, αν δεν γνωρίζουμε ήδη

- για κάθε παιδί του PR_BLOCK_ITEM_LIST (στη γενική περίπτωση) το εισάγουμε στη θέση που θέλουμε
- ενημερώνουμε τον δείκτη parent του παιδιού ώστε να δείχνει το νέο του πατέρα
- εγημερώνουμε τη βοηθητική μεταβλητή numparams
- αποδεσμεύουμε το υπόλοιπο του υπόδεντρου που μόλις εισάγαμε

5.7.2.1. Ανάκτηση κώδικα με stringstreams

Στις περισσότερες περιπτώσεις δεν αρκεί απλά η αποθήκευση μιας μεταβλητής από την οδηγία που επεξεργαζόμαστε, αλλά χρειάζεται να μετατραπεί όλο το τμήμα κώδικα σε μια συμβολοσειρά, ώστε να αναλυθεί εξαρχής, αφού γίνουν οι απαραίτητες μετατροπές.

Για να μετατραπεί ένα υπόδεντρο σε συμβολοσειρά, χρησιμοποιούμε μια δομή της βασικής βιβλιοθήκης της C++, το stringstream. Με τη δομή αυτή μπορούμε να επαναδομήσουμε τον κώδικα που έχει μετατραπεί σε δεντρική δομή πίσω στην αρχική του μορφή.

Για τη λειτουργία αυτή καλούμε τη μέθοδο source του ριζικού κόμβου του υπόδεντρου με παράμετρο ένα stringstream που έχουμε ορίσει. Ο ριζικός κόμβος καλεί αναδρομικά τη source στους κόμβους-παιδιά του και τελικά η stringstream μεταβλητή μας περιέχει όλο τον κώδικα του υπόδεντρου.

Έστω os η μεταβλητή stringstream που ορίσαμε. Ο κώδικας του παραδείγματος που απεικονίζει ο Πίνακας 5.13 δείχνει πώς χρησιμοποιούμε την os για να ανακτήσουμε κώδικα από υπόδεντρο και έπειτα να τον τυπώσουμε στην οθόνη.

Πίνακας 5.13 Παράδειγμα χρήσης της δομής stringstream

```
// δήλωση της os
ostringstream os;

// subtree_root είναι δείκτης στο ριζικό αντικείμενο του υπόδεντρου
// που μας ενδιαφέρει
```

```
subtree_root->source(&os);

// τώρα η δομή os περιέχει τον ανακτημένο κώδικα.
// για να μετατραπεί σε C++ string καλούμε την μέθοδο str()
// και για μετατροπή του string σε char * την c_str()
printf("original code:\n%s\n", os.str().c_str());
```

Με αντίστοιχο τρόπο μπορούμε να μετατρέψουμε ένα υπόδεντρο σε τμήμα κώδικα το οποίο μπορεί αφού περάσει από μετασχηματισμούς και μεταφερθεί ή ενσωματωθεί σε άλλο σημείο του κώδικα. Αφού συμβεί αυτό είναι εύκολο να ξαναμετατραπεί σε τμήμα δέντρου, έτσι ώστε να διατηρηθεί η ακεραιότητα της δομής του ΣΣΔ.

5.8. Μετασχηματισμοί

Η κυριότερη εργασία που επιτελεί ο μεταφραστής είναι αυτή των μετασχηματισμών κώδικα. Αφού έχει γίνει η ανάλυση σε λεκτικό, συντακτικό και σημασιολογικό επίπεδο, είναι δυνατό πλέον να μετατραπεί ο κώδικας εισόδου με τέτοιο τρόπο ώστε να προκύψει ένα πολυνηματικό πρόγραμμα. Η μετατροπή γίνεται με βάση τις οδηγίες OpenMP τις οποίες έχει εισάγει ο προγραμματιστής στο σειριακό πρόγραμμά του.

Για να μετασχηματιστεί κάποιο τμήμα κώδικα, είναι απαραίτητο πρώτα να εντοπιστεί. Η διαδικασία εντοπισμού γίνεται με εύρεση κατά πλάτος σε κάθε επίπεδο του δέντρου, με στόχο το αμέσως βαθύτερο επίπεδο. Στον κώδικα του σχήματος που ακολουθεί, συναντούμε την οδηγία #pragma omp parallel όταν βρεθούμε στο επίπεδο της σύνθετης έκφρασης της συνάρτησης myfunc, σε αντιδιαστολή με την εύρεσή της όταν θα διατρέχαμε τον ίδιο τον κόμβο της οδηγίας.

Πίνακας 5.14 Εύρεση μιας δομής OpenMP

```
..
void myfunc(void)
{
#pragma omp parallel
  {
```

```
    ...
  }
}
```

Η ιδιαίτερότητα αυτή προκύπτει από τον τρόπο λειτουργίας των συναρτήσεων μετασχηματισμών. Για εκτελεστεί πλήρως ο μετασχηματισμός, είναι απαραίτητο η ίδια η οδηγία να αφαιρεθεί από το συντακτικό δέντρο. Διατρέχοντας τους κόμβους-παιδιά από το προηγούμενο επίπεδο, προσδιορίζουμε τη θέση του υπόδεντρου με βάση τον πατρικό του κόμβο. Έτσι είναι δυνατό να κληθεί ο μετασχηματισμός από τον πατρικό κόμβο με παράμετρο τον αριθμό παιδιού και να ολοκληρωθεί η μετατόπιση και διαγραφή του κώδικα όπως είναι απαραίτητο.

Αντίθετα, κάτι τέτοιο δε θα ήταν δυνατό αν καλούνταν η συνάρτηση μετασχηματισμού κατά τη συνάντηση του κόμβου στο μονοπάτι. Αυτό, γιατί θα έπρεπε η ίδια η συνάρτηση να διαγράψει τον κόμβο από τον οποίο κλήθηκε, με απρόβλεπτα αποτελέσματα για τη συνέχεια της αναζήτησης για επόμενη οδηγία.

Ας εξετάσουμε τον κώδικα και τον τρόπο λειτουργίας μερικών μετασχηματισμών:

5.8.1. omp parallel

```
// transform a "#pragma omp parallel" construct.
// element is the child number in the params vector held by the
parent
// non-terminal node.
void gennode::tr_omp_parallel(int element)
{
  // a gennode
  gennode *gn,
          *optseq;
  // handy generic component
  generic_component *gencomp,
                    *newfunc;
  char *parallel_construct_name,
```

```
        *parallel_construct_prefix = NULL,
        *parallel_construct_infix = "_parallel_",
        *num_threads = NULL,  // number of threads
        *if_cond = NULL,
        *tmpstr = NULL,
        *outstr = NULL;
int ps;


// quick way to address the omp node
genomp *go = (genomp *)params[element];
int nested = go->parallel_level;
int *parallel_mode_mask;
parallel_mode_mask = &(go->options_mask);
ostringstream oldcode;

// ---------------------------------------------------------------
// a map of variables and vardata elements
// if freq = VAR_PRIVATE it is a private var and should not
// be renamed
// for detailed frequency types refer to the var_type enum
// in objects.hpp
// ---------------------------------------------------------------

go->var_list = new map<char *, vardata, ltstr>;
```

Στις δηλώσεις της συνάρτησης χρησιμοποιούνται αρκετοί δείκτες και μεταβλητές που θα βοηθήσουν στο χειρισμό του δέντρου αργότερα. Ιδιαίτερης σημασίας είναι οι:

- go: δείκτης σε αντικείμενο τύπου genomp ο οποίος δείχνει στην κορυφή του υπόδεντρου που επεξεργαζόμαστε,
- nested: ακέραιος που αποθηκεύει το βάθος δομών OpenMP στο οποίο βρισκόμαστε. Το γνώρισμα parallel_level του κόμβου genomp περιέχει ήδη τη σωστή τιμή μετά το τέλος της συντακτικής ανάλυσης
- parallel_mode_mask: δείχνει στο γνώρισμα options_mask του κόμβου genomp, το οποίο είναι μια μάσκα bit που αφορά τις φράσεις της δομής #pragma omp parallel, π.χ. τη φράση private(*list*), αν αυτή υπάρχει.

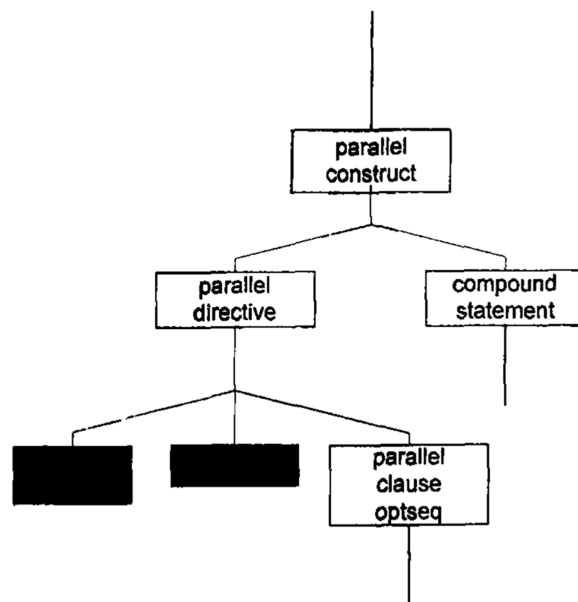
- var_list: γνώρισμα του κόμβου genomp το οποίο περιέχει το χάρτη μεταβλητών για τη δομή OpenMP που επεξεργαζόμαστε. Εδώ δημιουργείται ένας νέος χάρτης με χαρακτηριστικά όπως περιγράφονται σε προηγούμενη ενότητα

```
*parallel_mode_mask = tr_get_construct_options(element);
```

Η κλήση της tr_get_construct_options() διατρέχει τις φράσεις της δομής, αν υπάρχουν, και ορίζει τη μάσκα bit ανάλογα. Με βάση την τελική τιμή του parallel_mode_mask καλούνται αργότερα συναρτήσεις για να θέσουν τιμές από τις οποίες εξαρτάται η ροή του μετασχηματισμού.

```
// node of the parallel directive
gn = (gennode *)((gennode *)params[element])->
  get_subtree(PR_PARALLEL_DIRECTIVE);
```

Για παραπέρα χρήση, κρατούμε μια αναφορά στην οδηγία omp parallel. Χρησιμοποιούμε τη γνωστή ρίζα του υπόδεντρου και από εκεί καλούμε τη μέθοδο get_subtree με παράμετρο έναν απαριθμητή που αντιστοιχεί στην τιμή του κανόνα της οδηγίας που θέλουμε.



Σχήμα 5.3 Απόσπασμα του δέντρου στο επίπεδο της δομής #pragma omp parallel

Στη συγκεκριμένη περίπτωση θέλουμε από τον δείκτη στο ριζικό αντικείμενο του υπόδεντρου να προχωρήσουμε προς το υπόδεντρο φράσεων (parallel clause optseq). Το κάνουμε αυτό σε βήματα, φτάνοντας πρώτα στον κόμβο parallel directive και κατόπιν στον επιθυμητό. Με αυτό τον τρόπο εξασφαλίζουμε πως δε θα επιλεγεί λάθος κόμβος, γιατί στο δεύτερο βήμα παρακάτω,

```
// options subtree (might be null)
optseq = (gennode *)gn->get_subtree(PR_PARALLEL_CLAUSE_OPTSEQ);
```

υπάρχει η περίπτωση να μη βρεθεί ο ζητούμενος κόμβος, αν η δομή parallel δεν έχει φράσεις. Αν γινόταν η αναζήτηση για κόμβο με κανόνα PR_PARALLEL_CLAUSE_OPTSEQ από τον ριζικό του υπόδεντρου, στην περίπτωση που δεν υπήρχαν φράσεις, η αναζήτηση θα συνεχιζόταν στο δεύτερο υπόδεντρο που αρχίζει κάτω από το compound statement. Αυτό θα είχε αποτέλεσμα πιθανή εύρεση φράσης από κάποια άλλη φωλιασμένη δομή ή οδηγία OpenMP.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
if(nested)
{
  // get the direct parent of this construct
  genomp *pgo = tr_get_parent_omp();
  // copy the parent omp node var_list
  // copy assignment
  *go->var_list = *pgo->var_list;
  go->tr_omp_clear_transformed();
}
else
  in_parallel_construct = 1;
```

- Αν η δομή που εξετάζουμε είναι φωλιασμένη μέσα σε άλλη (έχει επίπεδο φωλιάσματος μεγαλύτερο από μηδέν), τότε αρχικοποιούμε το χαρτη μεταβλητών με βάση αυτόν του προηγούμενου επιπέδου. Έτσι μπορούμε εύκολα να γνωρίζουμε αν υπάρχει μη αποδεκτή αλλαγή στην ορατότητα κάποιας μεταβλητής που είχε ρητά ορισμένη την ορατότητά της σε προηγούμενο επίπεδο. Οι μεταβλητές που είχαν έμμεσα ορισμένη την ορατότητά τους πριν, σβήνονται από το χάρτη γιατί μπορεί να οριστεί ρητά τώρα, με τη χρήση της μεθόδου tr_omp_clear_transformed().
- Η αρχικοποίηση του χάρτη μεταβλητών γίνεται με χρήση ανάθεσης με αντιγραφή (copy assignment), ώστε κάθε επίπεδο να έχει το δικό του αποκλειστικό χάρτη.

Αν το επίπεδο της δομής μας είναι μηδέν, τότε δεν υπάρχει δομή ή οδηγία πριν από την παρούσα και ορίζουμε την καθολική μεταβλητή in_parallel_construct για χρήση σε τυχόν φωλιασμένες δομές/οδηγίες.

```
// ---------------------------------------------------------------
// get parallel options
// ---------------------------------------------------------------

if(*parallel_mode_mask & TR_PAR_IF)
{
  optseq->get_if_cond(&if_cond);
  if(nested) // transform if variables "local" and nested
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
    ((gennode   *)if_cond)->tr_omp_transform_variables(NULL,    go-
>var_list, nested);
  }
```

Με τη σειρά συγκρίνουμε τη μάσκα bit που αρχικοποιήσαμε με την εκκίνηση της συνάρτησης με τις πιθανές τιμές της στη δομή αυτή, χρησιμοποιώντας αντίστοιχους απαριθμητές. Στην περίπτωση της φράσης if, κρατάμε το υπόδεντρο της συνθήκης στο δείκτη if_cond για να εισαχθεί στον παραγόμενο κώδικα αργότερα. Αν το υπόδεντρο που εξετάζουμε είναι φωλιασμένο, ίσως είναι αναγκαία και η μετονομασία μεταβλητών στη συνθήκη.

```
  if(*parallel_mode_mask & TR_PAR_PRIVATE)
  {
    optseq->search_and_act(PR_DATA_CLAUSE_PRIVATE,
AN_OMP_PARALLEL_PRIVATE, 0, go->var_list, SEARCH_CURRENT_SCOPE);
  }
```

Αν υπάρχει η φράση private, τότε ξεκινώντας από τον κόμβο parallel clause optseq κάνουμε μια αναζήτηση για όσες φορές αυτή απαντηθεί. Σε κάθε εύρεση θα κληθεί η μέθοδος χειρισμού get_private_opts() (όπως ορίζεται από τον απαριθμητή ενέργειας AN_OMP_PARALLEL_PRIVATE) ώστε να ελεγχθεί αν υπάρχει επαναδήλωση ορατότητας και αν όχι να προσθέσει τη μεταβλητή στο χάρτη μεταβλητών.
Με αντίστοιχο τρόπο γίνεται έλεγχος για όλες τις πιθανές φράσεις μιας δομής omp parallel.

```
if(!nested)
{
  //-------------------------------------------------------
  // make some preparations
  //-------------------------------------------------------
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
// get the containing function name
get_containing_function_name(&parallel_construct_prefix);
if(parallel_construct_prefix == NULL)
{
  exit_errorline(this, "error: could not get the containing function name");
}

// generate construct name
initstr(&parallel_construct_name);
generate_name(&tmpstr, parallel_construct_infix, parallel_construct_num);
genstr(&parallel_construct_name, parallel_construct_prefix, tmpstr);
termstr(&tmpstr);

// set the active parallel construct name, if on omp level 0
if(!nested)
  active_parallel_construct_name = parallel_construct_name;



// -----------------------------------------------------------
// add the generated forward function declaration to the
// beginning of the program
// -----------------------------------------------------------


// parser runs, t_start points to the top level object of the new subtree
parse_buffer(p_start, "void *", parallel_construct_name, "(void *);\n");
// keep number of children in t_start tree to update program_put_element
ps = ((gennode *)t_start)->numparams;
// insert the subtree in place program_put_element in p_start
((gennode *)p_start)->insert_tree(t_start, program_put_element);
```

```
// increase program_put_element by the number of items inserted
// so the next addition will take place just below this
program_put_element += ps;
// deallocate t_start
delete t_start;
}
```

Αν το υπόδεντρο που εξετάζουμε δεν είναι φωλιασμένο, τότε γίνονται κάποιες ενέργειες:

- ανακαλύπτεται το όνομα της συνάρτησης που εσωκλείει τη δομή
- παράγεται μια συμβολοσειρά χαρακτήρων της μορφής <όνομα συνάρτησης>_parallel_<αριθμός> που θα αποτελέσει το όνομα της συνάρτησης νημάτων που θα παραχθεί
- προστίθεται η δήλωση της συνάρτησης στην αρχή του προγράμματος. p_start είναι ο δείκτης στη ρίζα του αρχικού συντακτικού δέντρου και t_start είναι η ρίζα του συντακτικού δέντρου μετά από κάθε επανεκτέλεση του αναλυτή.

```
// global_declarations: global program declarations
declarations_collection *global_declarations = ((genblock *)p_start)->p_declarations_collection;

// same goes for typenames
typenames_collection *global_typenames = ((genblock *)p_start)->p_typenames_collection;


// keep the structured block in newfunc pointer for later use
newfunc = params[element]->get_subtree(PR_COMPOUND_STATEMENT_FULL);
```

Ορίζουμε δύο δείκτες επιπλέον που δείχνουν στις συλλογές μεταβλητών και τύπων που κρατάει η ρίζα του δέντρου p_start. Άρα είναι οι καθολικές συλλογές μεταβλητών και τύπων.

Επίσης κρατάμε το δομημένο τμήμα που εσωκλείεται στο τμήμα που εξετάζουμε σε άλλο δείκτη για μετέπειτα χρήση.

```
// -----------------------------------------------------------------
// transform variables where needed
// -----------------------------------------------------------------

// update pointer to newfunc's block_item_list
gencomp = newfunc->get_subtree(PR_BLOCK_ITEM_LIST);

// set active collections
set_collections(gencomp);

// categorize variables
if((*parallel_mode_mask & TR_PAR_FIRSTPRIVATE) ||
   (*parallel_mode_mask & TR_PAR_REDUCTION))
  tr_omp_check_variables(gencomp, go->var_list);

// do the transformation
((genblock *)gencomp)->tr_omp_transform_variables(gencomp,
  go->var_list, nested);

// map is ready, let's have a look for nested pragmas a level
// deeper
((gennode *)newfunc)->get_nested_pragmas(nested + 1);
```

Αφού βρούμε τον πατρικό κόμβο των δηλώσεων και εντολών του τμήματος του προγράμματος που ελέγχουμε (PR_BLOCK_ITEM_LIST), ορίζουμε σαν ενεργές συλλογές μεταβλητών και τύπων αυτές που είναι αποθηκευμένες σε αυτό.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Αν υπάρχουν μεταβλητές firstprivate ή reduction, με τη μέθοδο tr_omp_check_variables() τις διαχωρίζουμε σε καθολικές και μη, αφού αυτό θα παίξει ρόλο αργότερα στις δηλώσεις τους στη νέα συνάρτηση.

Η μέθοδος tr_omp_transform_variables() εκτελεί τη μετονομασία μεταβλητών όπου χρειάζεται, όπως περιγράφηκε σε προηγούμενη ενότητα.

Αφού γίνουν όλα τα παραπάνω, εκτελείται η μέθοδος get_nested_pragmas(), η οποία ψάχνει για δομές και οδηγίες OpenMP που είναι φωλιασμένες ακριβώς ένα επίπεδο παρακάτω από αυτό που ήδη βρισκόμαστε.

```
// ---------------------------------------------------------------
// substitute the #pragma omp parallel (parallel_directive) and the
// accompanying structured_block
// with a compound statement which creates and destroys the team of
// threads
// ---------------------------------------------------------------

// delete #pragma omp ...
// and deallocate the associated tree
delete gn->params[0];
// delete the corresponding params element
gn->delete_param(0);

// remove the remaining tree element from params from the current
// node
delete_param(element);
```

Αφαιρούμε από το υπόδεντρο που επεξεργαζόμαστε το υπόδεντρο που περιέχει την οδηγία #pragma omp parallel, αφού πρώτα αποδεσμεύσουμε τις δομές του. Έχουμε ήδη πάρει όλες τις φράσεις του και πλέον μας είναι άχρηστο. Επίσης αφαιρούμε το δείκτη από τον vector παιδιών του πατέρα κόμβου.

Σβήνουμε από τον vector παιδιών του κόμβου από όπου κλήθηκε η μέθοδος την καταχώρηση για το παιδί που επεξεργαζόμαστε. Τώρα ο κώδικας που βρισκόταν εκεί υπάρχει μόνο στον δείκτη newfunc.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
// generate function stub to replace the #pragma omp
char *prestr = NULL,
     *poststr = NULL;

initstr(&prestr);
initstr(&poststr);
initstr(&outstr);

generate_parallel_function_stub(&prestr, &poststr,
  parallel_construct_name, *parallel_mode_mask, if_cond,
  num_threads, element, go->var_list, nested);
```

Μετά από λίγη προεργασία, καλείται η μέθοδος generate_parallel_function_stub, η οποία δημιουργεί το τμήμα κώδικα που θα αντικαταστήσει το #pragma omp parallel στο σημείο που βρισκόταν. Μεταξύ άλλων εκεί περιέχονται και εντολές για τη δημιουργία και καταστροφή νημάτων, την αρχικοποίηση μεταβλητών, η συνθήκη του if, αν υπάρχει, και άλλα.

```
if(nested)
{
  char *tmpstr = NULL;
  ostringstream os;

  initstr(&tmpstr);

  // old code source
  newfunc->source(&os);

  // merge strings
  genstr(&outstr, prestr, os.str().c_str(), poststr);
}
else
{
  genstr(&outstr, prestr, poststr);
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
   tr_omp_generate_function(newfunc, parallel_construct_name,
     *parallel_mode_mask, go->var_list, nested);
   tr_omp_generate_struct_declaration(parallel_construct_name,
     go->var_list);
 }
```

Αν η δομή που εξετάζουμε είναι φωλιασμένη, τότε απλά παίρνουμε τον παλιό κώδικα και τον εισάγουμε στην παλιά θέση μαζί με νέες δηλώσεις και εντολές.
Αν όχι, παράγουμε τη νέα συνάρτηση, καθώς και τη δήλωση της καθολικής δομής που θα περιέχει τους (τυχόν) δείκτες για τις μη τοπικές και μη καθολικές μεταβλητές του κώδικα.

```
// parse above generated strings

// setup the collections for the new parser
set_collections(this);

// parser runs, t_start points to the top level object of the new
// subtree
parse_buffer(this, OMPI_START_BUFFER, outstr, OMPI_END_BUFFER);
termstr(&outstr);

// path to the t_start block_item_list
gencomp = ((gennode *)t_start)->get_subtree(PR_BLOCK_ITEM_LIST);

// insert the new subtree in params[element]
insert_tree(gencomp, element);
// deallocate t_start
delete t_start;

termstr(&prestr);
termstr(&poststr);
termstr(&outstr);

// deallocate old function tree
delete newfunc;
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Τελικά, περνάμε από τον συντακτικό αναλυτή τη συμβολοσειρά που δημιουργήσαμε στα προηγούμενα βήματα, απομονώνουμε τον κώδικα που μας ενδιαφέρει και τον εισάγουμε στη θέση της παλιάς δομής.

```
termstr(&parallel_construct_name);
free(num_threads);

if(!nested)
{
  // increase construct number
  parallel_construct_num++;

  in_parallel_construct = 0;
}
}
```

Αφού αποδεσμεύσουμε μνήμη από δομές που τους εκχωρήθηκε, αυξάνουμε τον αριθμό των παράλληλων δομών κατά ένα και μηδενίζουμε την καθολική μεταβλητή in_parallel_construct.

5.8.2. omp for

Η λογική του μετασχηματισμού αυτού είναι παρόμοια με αυτή του omp parallel. Θα περιγραφούν τα τμήματα όπου παρουσιάζεται αρκετή διαφορά σε σχέση με παραπάνω.

```
if(*for_mode_mask & TR_FOR_SCHEDULE)
{
  gennode *gn = (gennode *)optseq->
    get_subtree(PR_UNIQUE_FOR_CLAUSE_SCHEDULE);
  if(gn == NULL) // PR_UNIQUE_FOR_CLAUSE_SCHEDULE_CHUNKSIZE
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
    {
      gn = (gennode *)optseq->
        get_subtree(PR_UNIQUE_FOR_CLAUSE_SCHEDULE_CHUNKSIZE);
      if(gn == NULL)
      {
        exit_errorline(this, "schedule: error getting schedule
parameters");
      }
    }

    // gn should be OK now
    //
    // structure: OMP_SCHEDULE ( OMP_<SCHEDULE_TYPE> [, chunk_size])
    //
    genterm *schedule_kind = (genterm *)gn->params[2];
    // number of parameters if no chunk size is specified
    int no_chunk_params = 4;

    if(!strcasecmp(schedule_kind->value, "static"))
    {
      sched = strdup("static");
    }
    else
      if(!strcasecmp(schedule_kind->value, "dynamic"))
      {
        sched = strdup("dynamic");
      }
      else
        if(!strcasecmp(schedule_kind->value, "guided"))
        {
          sched = strdup("guided");
        }
        else
          if(!strcasecmp(schedule_kind->value, "runtime"))
          {
            if(gn->numparams > no_chunk_params) // a chunk size is
specified
            {
              exit_errorline(this, "error: no chunk size can be
```

```
            specified for runtime scheduling");
          }

          sched = strdup("runtime");
        }

    // get the chunk size, if it exists
    if(gn->numparams > no_chunk_params) // a chunk size is specified
    {
      ostringstream os;
      gn->params[4]->source(&os);
      chunksize = strdup(os.str().c_str());
    }
    else
      chunksize = strdup("#");

  }
  else // no schedule clause
  {
    // default scheduling
    sched = strdup("static");
    chunksize = strdup("#");
  }
```

Αν στις παραμέτρους του omp for περιέχεται μια φράση schedule, ο παραπάνω κώδικας αναλαμβάνει να εξάγει τις τιμές που παρέχονται, αλλιώς ορίζει κάποιες ερήμην τιμές, όπως έχουν αντιστοιχιστεί στη βιβλιοθήκη εκτέλεσης (runtime library) του OMPi.

Πιο συγκεκριμένα, αν δεν εμφανίστεί πρόβλημα στην ανάκτηση της παραμέτρου schedule, ορίζει στο δρομολογητή μια από τις τιμές static, dynamic, guided ή runtime. Κατόπιν, αν υπάρχει και μέγεθος τμήματος επεξεργασίας (chunk size), αποθηκεύει στη συμβολοσειρά chunksize την τιμή.

```
  // get the iterator and iteration values
  iter_stmt = (gennode *)params[element]->
```

```
    get_subtree(PR_ITERATION_STATEMENT_OMP_FOR);
  if(iter_stmt == NULL)
  {
    exit_errorline(this, "error: the iteration statement used is not
      suitable for openmp");
  }

  if(iter_stmt->params[2]->check_rule(PR_ASSIGNMENT_EXPRESSION))
  {

    iterator = (genident *)iter_stmt->params[2]->find_first_ident();

    // iterator is PRIVATE or LASTPRIVATE
    if(go->var_list->find(iterator->value) == go->var_list->end())
    // not assigned by any clause
    {
      (*go->var_list)[strdup(iterator->value)].freq = VAR_PRIVATE;
      varnames_add_gendecl(iterator, go->var_list);
    }
    else  // already assigned by this or containing construct,
          // possible values <= VAR_PRIVATE
    {
      if((*go->var_list)[iterator->value].freq == VAR_PRIVATE)
      { // private, that's what the iterator should be
        // ensure the right declaration node is set
        varnames_add_gendecl(iterator, go->var_list);
      }
      else
        if((*go->var_list)[iterator->value].freq > VAR_NOCHANGE)
        {
      // set in this construct, ensure it is a lastprivate variable.
      // declaration node should be ok
          if(((*go->var_list)[iterator->value].freq              !=
VAR_LPRIV_GLOBAL)    &&    ((*go->var_list)[iterator->value].freq    !=
VAR_LPRIV_LOCAL))
          { // an error
            exit_errorline(go, "error: omp for clause for iterator
              can be private or lastprivate only");
          }
```

```
        }
        else
        { // set in an enclosing construct, replace map values
          (*go->var_list)[iterator->value].freq = VAR_PRIVATE;
          varnames_add_gendecl(iterator, go->var_list);
        }
    }
}
```

Γίνεται έλεγχος για τη μορφή του for, ώστε να διαπιστωθεί αν είναι σύμφωνη με το πρότυπο OpenMP. Εξάγεται ο μετρητής επανάληψης από τη συνθήκη με τη μέθοδο find_first_ident() η οποία βρίσκει τον πρώτο αναγνωριστή μέσα σε ένα υπόδεντρο.

Πρέπει επίσης να αποκλειστεί η περίπτωση ο μετρητής επανάληψης να έχει ορισμένη ρητά την ορατότητά του σε κάτι άλλο εκτός από private ή lastprivate. Έτσι γίνεται αναζήτηση στο χάρτη μεταβλητών της δομής omp for. Αν δεν υπάρχει η μεταβλητή εκεί, προστίθεται με ρητή ορατότητα private. Αν υπάρχει, ελέγχουμε αν η ορατότητα ορίστηκε σε φράση της παρούσας δομής for. Αν ορίστηκε σε αυτή τη δομή που εξετάζουμε, πρέπει να διαπιστώσουμε αν έχει οριστεί σε κάτι άλλο εκτός από private ή lastprivate, αλλιώς αντικαθιστούμε την ορατότητα με private.

```
gennode *gb = (gennode *)((gennode *)gencomp)->
  get_subtree(PR_BLOCK_ITEM_LIST);

// do the transformation
if(gb != NULL)
{
  set_collections(gb);
  gb->tr_omp_transform_variables(gb, go->var_list, nested);
}
else
{
  gb = (gennode *)gencomp;
  gb->tr_omp_transform_variables(NULL, go->var_list, nested);
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
  // map is ready, let's have a look for nested pragmas a level
deeper
  ((gennode *)params[element])->get_nested_pragmas(nested + 1);
```

Κάνουμε αναζήτηση μέσα στο υπόδεντρο του omp for για να βρούμε έναν μη τερματικό κόμβο block item list. Αν υπάρχει, σημαίνει πως η for περικλείει ένα σύνολο κώδικα. Αν όχι, περιέχει μόνο μια εντολή (για παράδειγμα μια ανάθεση), και ανάλογα καλούμε τη μέθοδο tr_omp_transform_variables() για να μετονομαστούν όποιες μεταβλητές χρειάζονται.

Τελικά, κάνουμε αναζήτηση για φωλιασμένες δομές και οδηγίες OpenMP σε ένα επίπεδο βαθύτερα από αυτό που είμαστε.

```
{ // iterator stepping
  char *incr = NULL;
  ostringstream os;
  gennode *is = (gennode *)iter_stmt->params[6];

  switch(is->rulename)
  {
    case PR_UNARY_EXPRESSION_INC_OP: // e.g. ++i
      incr = strdup("1");
      break;

    case PR_UNARY_EXPRESSION_DEC_OP: // e.g. --i
      incr = strdup("(-1)");
      break;

    case PR_POSTFIX_EXPRESSION_INC_OP: // e.g. i++
      incr = strdup("1");
      break;

    case PR_POSTFIX_EXPRESSION_DEC_OP: // e.g. i--
      incr = strdup("(-1)");
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
        break;

      case PR_ASSIGNMENT_EXPRESSION:
        if(!strcmp(is->params[1]->value, "+=")) // i += 1
        {
          is->params[2]->source(&os);
          incr = strdup(os.str().c_str());
        }
        else
          if(!strcmp(is->params[1]->value, "-=")) // i -= 1
          {
            is->params[2]->source(&os);

            initstr(&incr);
            genstr(&incr, "-(", os.str().c_str(), ")");
          }
        break;

      default:
        exit_errorline(this, "error, not implemented yet");
        break;

    }



    genstr(&tmpstr, "  _omp_incr = (", incr, ");\n");
    free(incr);

    // keep stepping expression for later use
    is->source(&os);
    step_expr = strdup(os.str().c_str());
}
```

Ελέγχουμε την έκφραση με την οποία μεταβάλλεται ο μετρητής επανάληψης, γιατί πρέπει να παράγουμε κώδικα που θα δουλεύει ανάλογα.

```
// get initial lower bound
{
  ostringstream os;
  gennode *lb_expr = (gennode *)iter_stmt->params[2];

  // for(i = blah; ...)
  lb_expr->params[2]->source(&os);

  initial_lower_bound = strdup(os.str().c_str());

}
```

- Κρατάμε την αρχική τιμή του μετρητή επανάληψης.

```
// get bound and logical operator
{
  ostringstream os;
  gennode *log_expr = (gennode *)iter_stmt->params[4];

  // bound
  log_expr->params[2]->source(&os);
  bound = strdup(os.str().c_str());

  // logical_expression
  switch(log_expr->rulename)
  {
    case PR_RELATIONAL_EXPRESSION_LT:
    case PR_RELATIONAL_EXPRESSION_LE:
      logical_op = strdup("<");
      break;

    case PR_RELATIONAL_EXPRESSION_GT:
    case PR_RELATIONAL_EXPRESSION_GE:
      logical_op = strdup(">");
      break;
```

```
    default:
      exit_errorline(log_expr, "error: no suitable relational
        operator found");
      break;
  }

}
```

Βρίσκουμε το όριο των επαναλήψεων και τον λογικό τελεστή σύγκρισης.

```
{
  char ts[200];

  sprintf(ts, "  _omp_init_directive(_OMP_FOR, _omp_for_id, %s,
    _omp_incr, %d, %d);\n", initial_lower_bound,
    (*for_mode_mask & TR_FOR_ORDERED)?1:0,  sched[0]);
  genstr(&tmpstr, ts);
}

if(sched[0] != 'r')
{
  genstr(&tmpstr, "_omp_sched_bounds_func = _omp_", sched,
    "_bounds;\n");

  if((sched[0] == 's') && chunksize[0] != '#') // static chunk
  {
    genstr(&tmpstr, "  _omp_chunksize = (", chunksize, ");\n");
    genstr(&tmpstr, "  _omp_static_bounds_chunk(", bound, ", ",
      initial_lower_bound, ", _omp_incr, _omp_chunksize,"
                  " &_omp_nchunks, &_omp_init_start);\n");
    genstr(&tmpstr, "  while((*_omp_sched_bounds_func)(", bound,
      ", ", initial_lower_bound, ", _omp_for_id, _omp_incr,"
                  " _omp_chunksize, &_omp_start, &_omp_end,
      _omp_nchunks, _omp_init_start, &_omp_c))\n"
                  "  {\n");
  }
  else
```

```
  {
    if(sched[0] == 's')
    {
      genstr(&tmpstr, "  _omp_static_bounds_default(", bound, ", ",
        initial_lower_bound, ", _omp_incr, &_omp_start,
        &_omp_end);\n");
      genstr(&tmpstr, "  while ((*_omp_sched_bounds_func)(", bound,
        ", ", initial_lower_bound, ", _omp_for_id, _omp_incr, -1,"
                      " &_omp_start, &_omp_end, 1, 0, &_omp_c))\n"
                      "  {\n");
    }
    else
    {
      genstr(&tmpstr, "  while ((*_omp_sched_bounds_func)(", bound,
        ", ", initial_lower_bound, ", _omp_for_id, _omp_incr, (",
                      (chunksize[0] == '#')?"1":chunksize, "),
        &_omp_start, &_omp_end, 1, 0, &_omp_c))\n"
                      "  {\n");
    }
  }
}
else
{
  genstr(&tmpstr, "  {\n"
    "    _omp_sched_info_t _omp_sched_info = _OMP_THREAD->parent
    ->con_run[_OMP_FOR][_omp_for_id]->sched_info;\n"
    "    switch (_omp_sched_info.sched)\n"
    "    {\n"
    "      case _OMP_STATIC:\n"
    "        _omp_sched_bounds_func = _omp_static_bounds;\n"
    "        if (_omp_sched_info.chunksize == -1)\n"
    "        {\n"
    "          _omp_static_bounds_default(", bound, ", ",
    initial_lower_bound, ", _omp_incr, &_omp_start,
    &_omp_end);\n"
    "          _omp_chunksize = -1;\n"
    "        }\n");

  genstr(&tmpstr, "        else\n"
```

```
    "        {\n"
    "            _omp_chunksize = _omp_sched_info.chunksize;\n"
    "             _omp_static_bounds_chunk(", bound, ", ",
      initial_lower_bound, ", _omp_incr, _omp_chunksize,
      &_omp_nchunks, &_omp_init_start);\n"
    "        }\n"
    "        break;\n");

  genstr(&tmpstr, "        case _OMP_DYNAMIC:\n"
    "        _omp_sched_bounds_func = _omp_dynamic_bounds;\n"
    "        _omp_chunksize = _omp_sched_info.chunksize;\n"
    "        if (_omp_chunksize == -1) _omp_chunksize = 1;\n"
    "        break;\n"
    "      case _OMP_GUIDED:\n"
    "        _omp_sched_bounds_func = _omp_guided_bounds;\n"
    "        _omp_chunksize = _omp_sched_info.chunksize;\n"
    "        if (_omp_chunksize == -1) _omp_chunksize = 1;\n"
    "        break;\n"
    "    }\n"
    "  }\n");

  genstr(&tmpstr, "  while ((*_omp_sched_bounds_func)(", bound,
    ", ", initial_lower_bound, ", _omp_for_id, _omp_incr,"
    " _omp_chunksize, &_omp_start, &_omp_end, _omp_nchunks,
     _omp_init_start, &_omp_c))\n"
    "  {\n");
}


if(*for_mode_mask & TR_FOR_ORDERED)
{
  genstr(&tmpstr, "_omp_push_for_data(_omp_for_id,
    _omp_start);\n");
}

genstr(&tmpstr, "    if(_omp_start ", logical_op, " (", bound, ")
  && _omp_end == (", bound, "))\n""      _omp_last_iter = 1;\n");
genstr(&tmpstr, "    for (", iterator->value, " =  _omp_start; ",
  iterator->value, " ", logical_op, " _omp_end; ", step_expr, ")\n"
```

```
    "    {\n");

 // append transformed code
 {
   ostringstream os;

   // get block source
   gb->source(&os);

   genstr(&tmpstr, os.str().c_str());
 }

 genstr(&tmpstr, "    }\n");

 if(*for_mode_mask & TR_FOR_ORDERED)
 {
   genstr(&tmpstr, "    _omp_pop_for_data();\n");
 }

 genstr(&tmpstr, "    }\n");


 // append lastprivate variable updates
 if(*for_mode_mask & TR_PAR_LASTPRIVATE)
   tr_omp_lastprivate_end(&tmpstr, active_parallel_construct_name,
   go->var_list);

 // append reduction variable updates
 if((*for_mode_mask & TR_PAR_REDUCTION) && in_parallel_construct)
   tr_omp_reduction_end(&tmpstr, active_parallel_construct_name,
   go->var_list);

 // insert barrier if applicable
 if(!(*for_mode_mask & TR_OMP_NOWAIT))
   genstr(&tmpstr, "#pragma omp barrier\n");

 genstr(&tmpstr, "}\n");

 // generate variables initializations and put above code inside the
```

```
compound statement
  if(go->var_list->size())
  {
    char *varstr = NULL,
         *memcpystr = NULL;

    initstr(&varstr);
    initstr(&memcpystr);

    tr_omp_generate_variables_initialization(go->var_list, &varstr,
      &memcpystr, nested);

    genstr(&outstr, "{\n", varstr, memcpystr, tmpstr, "}\n");

    termstr(&varstr);
    termstr(&memcpystr);
  }
  else
  {
    genstr(&outstr, tmpstr);
  }

  // parse the buffer using the local collections
  parse_buffer(this, OMPI_START_BUFFER, outstr, OMPI_END_BUFFER);
  // get the block_item_list in t_start
  gencomp = ((gennode *)t_start)->get_subtree(PR_BLOCK_ITEM_LIST);


  // remove old code
  delete params[element];
  delete_param(element);
  // insert new code
  insert_tree(gencomp, element);

  ((genblock *)params[element])->transformed = 1;

  delete t_start;
```

```
  termstr(&tmpstr);
  termstr(&outstr);

  free(sched);
  free(chunksize);
  free(step_expr);
  free(logical_op);
  free(bound);
  free(initial_lower_bound);

  for_construct_num++;
}
```

- Το τελευταίο τμήμα κώδικα προετοιμάζει το μετασχηματισμένο πρόγραμμα, καλεί τον συντακτικό αναλυτή και εισάγει το παραγόμενο υπόδεντρο στο σημείο προσθήκης.

5.8.3. omp sections

Ο μετασχηματισμός για τη δομή sections έχει τα εξής σημαντικά σημεία:

- Δεδομένου ότι μια δομή sections περιέχει φωλιασμένες δομές section ή ένα δομημένο τμήμα κώδικα (structured block) και μετά δομές section, διατρέχουμε τον πίνακα παιδιών του κόμβου-πατέρα.

```
// sec_seq should contain a sequence of:
// structured_block [section_directive structured_block]*
// or
// [section_directive structured_block]+
int i = 0;
int count = 0;

// if the first parameter is a directive, start from params[1]
if(sec_seq->params[0]->check_rule(PR_SECTION_DIRECTIVE))
  i++;
```

```
for(;i < sec_seq->numparams; i += 2, count++)
// skip over directive declarations
{
  ostringstream section_code;

  // update pointer to param's block_item_list
  gencomp = sec_seq->params[i]->get_subtree(PR_BLOCK_ITEM_LIST);

  // set active collections
  set_collections(gencomp);

  // categorize variables
  if((*sections_mode_mask       &       TR_PAR_FIRSTPRIVATE)       ||
(*sections_mode_mask & TR_PAR_REDUCTION))
    tr_omp_check_variables(gencomp, go->var_list);

  // do the transformation
  ((genblock *)gencomp)->tr_omp_transform_variables(gencomp,
    go->var_list, nested);

  // map is ready, let's have a look for nested pragmas a level
  // deeper on this code segment
  // NB two levels deeper is required for sections construct,
  // because the section_scope directives count as a level
  ((gennode *)gencomp)->get_nested_pragmas(nested + 2);

  generate_name(&namestr, "case ", count, ":\n");
  sec_seq->params[i]->source(&section_code);
  genstr(&tmpstr, namestr,
                  section_code.str().c_str(),
                  "break;\n\n");
  free(namestr);
}

// add number of cases to sections vector for static int
sections_construct_vec.push_back(count);

// end statements
genstr(&tmpstr,     "}\n");
```

```
if(!(*sections_mode_mask & TR_OMP_NOWAIT))
   genstr(&tmpstr, "#pragma omp barrier\n");

genstr(&tmpstr,    "}\n"
                   "}\n");

// generate variables initializations and put above code inside the
// compound statement
if(go->var_list->size())
{
  char *varstr = NULL,
       *memcpystr = NULL;

  initstr(&varstr);
  initstr(&memcpystr);

  tr_omp_generate_variables_initialization(go->var_list,    &varstr,
&memcpystr, nested);

  genstr(&outstr, "{\n", varstr, memcpystr, tmpstr, "}\n");

  termstr(&varstr);
  termstr(&memcpystr);
}
else
{
  genstr(&outstr, tmpstr);
}
termstr(&tmpstr);

// parse the buffer using the local collections
parse_buffer(this, OMPI_START_BUFFER, outstr, OMPI_END_BUFFER);
// get the block_item_list in t_start
gencomp = ((gennode *)t_start)->get_subtree(PR_BLOCK_ITEM_LIST);
// remove old code
delete params[element];
delete_param(element);
// insert new code
```

```
  insert_tree(gencomp, element);

  termstr(&outstr);

  sections_construct_num++;
}
```

Για καθένα από τα παιδιά εκτός των ίδιων των οδηγιών OpenMP μετονομάζουμε τις μεταβλητές που χρειάζονται, αναζητούμε φωλιασμένες δομές OpenMP στο επόμενο επίπεδο και έπειτα εξάγουμε τον κώδικα με τη χρήση stringstreams. Στο τέλος του μετασχηματισμού όλος ο κώδικας αναλύεται από το συντακτικό αναλυτή και το υπόδεντρο που παράγεται εισάγεται στη θέση της αρχικής δομής.

5.8.4. omp parallel for/sections

Η λογική που ακολουθείται όταν συναντηθεί μια δομή του τύπου:

- #pragma omp parallel for [clauses ...]
- #pragma omp parallel sections [clauses ...]

είναι να διαιρεθεί ο κώδικας σε μια δομή *#pragma omp parallel* και μια άμεσα φωλιασμένη *#pragma omp for* ή *sections*, ανάλογα με την περίπτωση. Αμέσως μετά, καλείται ο μετασχηματισμός του #pragma omp parallel, ο οποίος θα βρει και τη φωλιασμένη δομή.

Προγραμματιστικά αυτό διευκολύνει πολύ την επέμβαση σε κώδικα μετασχηματισμών, αφού δεν υπάρχει ξεχωριστός κώδικας για τις συνδυασμένες δομές parallel for και parallel sections.

```
// ---------------------------------------------------------------
// take a combined parallel construct and break it down to a parallel
// and a for/sections construct
// ---------------------------------------------------------------
void gennode::tr_omp_break_combined_parallel(int element)
{
  gennode *gn,
```

```
        *gnp;
  generic_component *gencomp;

  char *outstr,
       *par_opts = NULL,
       *for_opts = NULL;

  // quick way to address the omp node
  genomp *go = (genomp *)params[element];
  int nested = go->parallel_level;

  // flag for for/sections construct
  int parallel_for = params[element]->
    check_rule(PR_PARALLEL_FOR_CONSTRUCT);

  int parallel_mode_mask = 0;

  ostringstream oldcode;

  initstr(&par_opts);
  initstr(&for_opts);

  parallel_mode_mask = tr_get_construct_options(element);

  if(parallel_mode_mask != TR_PAR_DEFAULTSHARED) // no params
  {
    gnp = (gennode *)((gennode *)params[element])->
      get_subtree(PR_PARALLEL_SECTIONS_CLAUSE_OPTSEQ);

    for(int i = 0; i < gnp->numparams; i++)
    {
      ostringstream os;

      // ordered

      if(gnp->params[i]->classname == CN_TERM)
      {
        // this can only be an "ordered" clause
        if(!strcasecmp(gnp->value, "ordered"))
```

```
      genstr(&for_opts, "ordered ");
    else
    {
      fprintf(stderr, "error: %s is not a valid omp parallel for
        clause\n", gnp->params[i]->value);
      exit(1);
    }
  }
  else
  {
    // a gennode
    gn = (gennode *)gnp->params[i];

    switch(gn->rulename)
    {
      // omp parallel options
      case PR_UNIQUE_PARALLEL_CLAUSE_IF:
      case PR_DATA_CLAUSE_DEFAULTSHARED:
      case PR_DATA_CLAUSE_DEFAULTNONE:
      case PR_DATA_CLAUSE_SHARED:
      case PR_UNIQUE_PARALLEL_CLAUSE_NUMTHREADS:
        gn->source(&os);
        genstr(&par_opts, os.str().c_str(), " ");
        break;


      // omp for options
      case PR_DATA_CLAUSE_PRIVATE:
      case PR_DATA_CLAUSE_FIRSTPRIVATE:
      case PR_DATA_CLAUSE_LASTPRIVATE:
      case PR_DATA_CLAUSE_REDUCTION:
      case PR_UNIQUE_FOR_CLAUSE_SCHEDULE:
      case PR_UNIQUE_FOR_CLAUSE_SCHEDULE_CHUNKSIZE:
        gn->source(&os);
        genstr(&for_opts, os.str().c_str(), " ");
        break;

      default:
        fprintf(stderr, "error: %s: unexpected omp parallel for
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
            clause\n", gn->value);
          exit(1);
          break;
      }
    }
  }
}

if(parallel_for)
  params[element]->get_subtree(PR_ITERATION_STATEMENT_OMP_FOR)->
    source(&oldcode);
else
  params[element]->get_subtree(PR_SECTION_SCOPE)->source(&oldcode);

reparsing_parallel_level = nested;

char *allstr = NULL;
initstr(&allstr);

genstr(&allstr, "#pragma omp parallel ", par_opts, "\n"
                "{\n{\n");

if(parallel_for)
  genstr(&allstr, "#pragma omp for ", for_opts, "\n");
else
  genstr(&allstr, "#pragma omp sections ", for_opts, "\n");

genstr(&allstr, oldcode.str().c_str(),
                "}\n}\n");

termstr(&for_opts);
termstr(&par_opts);

parse_buffer(this, OMPI_START_BUFFER, allstr, OMPI_END_BUFFER);

termstr(&allstr);

delete params[element];
delete_param(element);
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
  gencomp = t_start->get_subtree(PR_BLOCK_ITEM_LIST);
  insert_tree(gencomp, element);


  delete t_start;
}
```

Όλες οι παράμετροι της συνδυασμένης δομής #pragma omp parallel for/sections εξετάζονται ώστε να μοιραστούν στις δύο νέες δομές. Στην φωλιασμένη δομή for ή sections ανατίθενται οι εξής παράμετροι, αν συναντηθούν:

- private (list)
- firstprivate (list)
- lastprivate (list)
- reduction (operator: list)
- ordered
- schedule (kind [, chunk_size])

Από τις παραμέτρους, οι ordered και schedule συναντώνται μόνο στη δομή #pragma omp parallel for.

Η παράμετρος nowait δεν είναι αποδεκτή στις δομές αυτές.

5.8.5. Παράδειγμα μετασχηματισμού

Ας δούμε πώς γίνεται ο μετασχηματισμός σε μια δομή #pragma omp parallel με εμφωλιασμένη μια οδηγία #pragma omp critical:

```
#include <stdio.h>
#include <omp.h>

int global_a;

void myfunc(void)
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
{
  int local_a;

#pragma omp parallel
  {
#pragma omp critical
    {
      global_a++;
      local_a++;
    }
  }
}

int main(int argc, char *argv[])
{
  myfunc();
}
```

Πίνακας 5.15 Παράδειγμα χρήσης της δομής #pragma omp parallel

Λόγω της ύπαρξης μιας δομής parallel στο πρόγραμμα, είναι αναγκαίο να δημιουργηθεί μια επιπλέον συνάρτηση, η οποία θα περιέχει τον κώδικα που θα εκτελέσουν τα νήματα. Το σημείο όπου βρίσκεται τώρα η δομή θα αντικατασταθεί με μια σειρά εντολών που:

- αρχικοποιούν τις μεταβλητές
- δημιουργούν την ομάδα νημάτων με τη νέα συνάρτηση
- καταστρέφουν την ομάδα νημάτων όταν τελειώσει η εργασία

Ακολουθούμε τα εξής βήματα για την ανάλυση και το μετασχηματισμό του προγράμματος:

- Στη συνάρτηση myfunc, μέσα στο πρώτο επίπεδο του τμήματος parallel δε χρησιμοποιούνται μεταβλητές.
- Ψάχνοντας στο ΣΣΔ για εμφωλευμένες δομές/οδηγίες βρίσκουμε μέσα στο πρώτο επίπεδο τη δομή #pragma omp critical.

- Ελέγχουμε το νέο πρώτο επίπεδο της δομής critical για μεταβλητές και βρίσκουμε πως χρησιμοποιούνται δύο μεταβλητές. Η μία (global_a) είναι ορισμένη σε καθολική ορατότητα, ενώ η άλλη (local_a) στην τοπική ορατότητα της συνάρτησης myfunc. Επειδή η μεταφορά του κώδικα από μέσα στη δομή parallel προς τη νέα συνάρτηση θα μεταφέρει και τη μεταβλητή local_a που δεν είναι ορισμένη σε καθολική ορατότητα, είναι απαραίτητο να δηλωθεί αλλιώς. Αν υποθέσουμε πως το όνομα της συνάρτησης νημάτων είναι myfunc_parallel_0, θα δημιουργήσουμε μια δομή καθολικής ορατότητας με όνομα myfunc_parallel_0_vars η οποία θα περιέχει ένα μέλος: δείκτη σε μεταβλητή τύπου ίδιου με της local_a, με όνομα local_a.

```
typedef struct
{
  int (*local_a);
}
myfunc_parallel_0_vars;
```

Ψάχνοντας μέσα στο επίπεδο της δομής critical δεν βρίσκουμε κάποια άλλη εμφωλευμένη δομή/οδηγία. Έτσι πριν επιστρέψουμε τον έλεγχο στο μετασχηματισμό του parallel, κάνουμε τις απαραίτητες αλλαγές στο τμήμα critical

```
    othread_set_lock(&_omp_critical_lock);
    {
      global_a++;
      (*(local_a))++;
    }
    othread_unset_lock(&_omp_critical_lock);
```

Το τμήμα κώδικα αντικαθιστά όλη τη δομή critical στο αρχικό πρόγραμμα.

Επιστρέφουμε στο μετασχηματισμό του parallel. Αφού δεν υπάρχει άλλη οδηγία μέσα στην parallel, αποκόπτουμε το τμήμα κώδικα δημιουργώντας τη συνάρτηση

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

myfunc_parallel_0 και προσθέτουμε στη θέση του τις δηλώσεις μεταβλητών από τη νέα καθολική δομή και την έναρξη και τερματισμό των νημάτων.

Στη νέα συνάρτηση γίνεται η αρχικοποίηση όσων μεταβλητών χρειάζονται και ο κώδικας γίνεται όπως παρακάτω:

```
#include <stdio.h>
#include <omp.h>

void *myfunc_parallel_0(void *_omp_thread_data);

typedef struct
{
  int (*local_a);
}
myfunc_parallel_0_vars;

int global_a;

void myfunc(void)
{
  int local_a;

  {
    _OMP_PARALLEL_DECL_VARSTRUCT(myfunc_parallel_0);
    _OMP_PARALLEL_INIT_VAR(myfunc_parallel_0, local_a);
    _omp_create_team((-1), _OMP_THREAD, myfunc_parallel_0,
      (void *)&myfunc_parallel_0_var);
    _omp_destroy_team(_OMP_THREAD->parent);
  }
}

int main(int argc, char *argv[])
{
  myfunc();
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
void *myfunc_parallel_0(void *_omp_thread_data)
{
  int *_omp_dummy = _omp_assign_key(_omp_thread_data);
  int (*local_a) = &_OMP_VARREF(myfunc_parallel_0, local_a);
  {
    othread_set_lock(&_omp_critical_lock);
    {
      global_a++;
      (*(local_a))++;
    }
    othread_unset_lock(&_omp_critical_lock);
  }
}
```

ΚΕΦΑΛΑΙΟ 6. ΑΠΟΤΕΛΕΣΜΑΤΑ

6.1. Εισαγωγή

6.2. Υπολογισμός π

6.1. Εισαγωγή

Ο νέος μεταφραστής υποβλήθηκε σε δοκιμή για να επιβεβαιωθεί η ορθή λειτουργία του αλλά και η ποιότητα και οι επιδόσεις του παραγόμενου κώδικα. Στο κεφάλαιο αυτό δίνουμε ένα δείγμα εκτέλεσης μετασχηματισμένου προγράμματος.

6.2. Υπολογισμός π

Για τη δοκιμή του μεταφραστή και τη χρονομέτρηση εκτέλεσης χρησιμοποιήθηκε ο κώδικας που δείχνει ο Πίνακας 6.1. Η χρονομέτρηση έγινε για 100000000 επαναλήψεις του βρόχου, και για αριθμούς νημάτων 1, 2 και 4.

Η μετάφραση του κώδικα και η εκτέλεσή του έγινε στον υπολογιστή atlantis του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων. Πρόκειται για έναν υπολογιστή με τέσσερις επεξεργαστές Intel Xeon ταχύτητας 700MHz με 2MB level 2 cache και 1.5GB συνολική μνήμη συστήματος. Η αρχικτεκτονική του συστήματος είναι κοινής μνήμης.

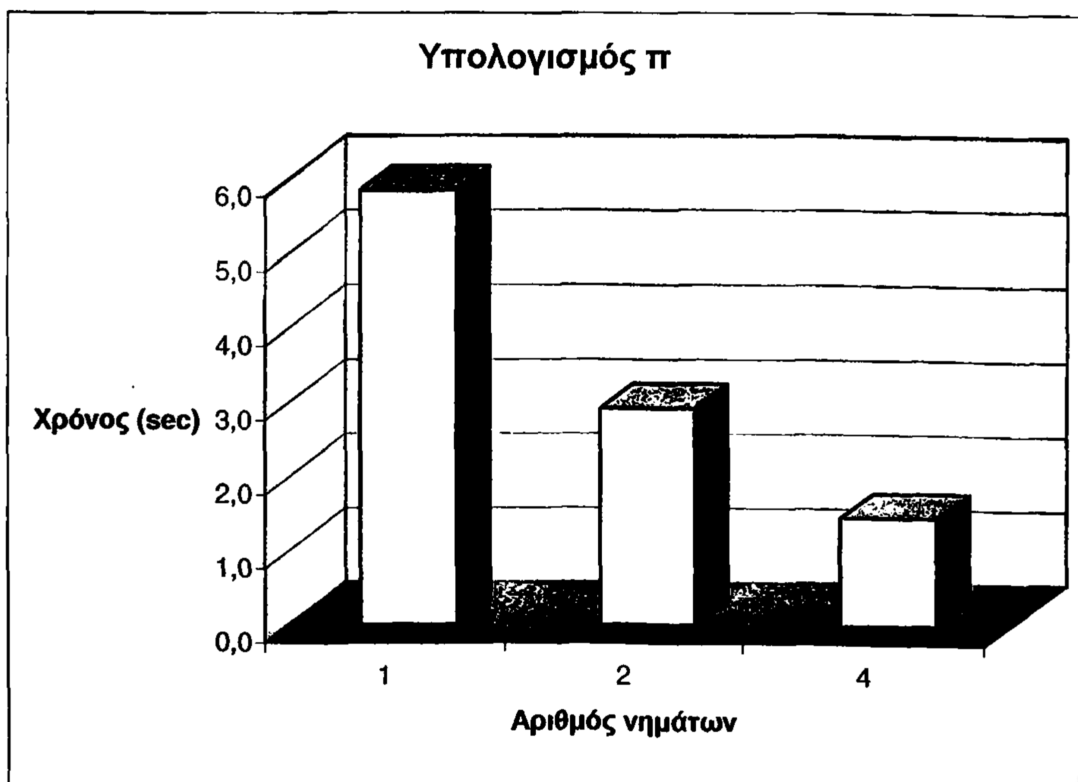
Πίνακας 6.1 Κώδικας υπολογισμού του π

```
int main()
{
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
  int i;
  double x,
         pi,
         sum = 0.0;

  step = 1.0/(double) num_steps;

#pragma omp parallel for reduction(+:sum) private(x)
  for(I = 1; I <= num_steps; i++)
  {
    x = (i-0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
  }

  pi = step * sum;
}
```

Τα αποτελέσματα φαίνονται στο Σχήμα 6.1. Όπως είναι φανερό, το πρόγραμμα είναι πλήρως παραλληλοποιήσιμο και η επιτάχυνση είναι γραμμική και ίση με τον αριθμό των νημάτων.



Σχήμα 6.1 Γράφημα χρονομέτρησης για τον υπολογισμό του π

ΚΕΦΑΛΑΙΟ 7. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

7.1 Συμπεράσματα
7.2 Μελλοντική εργασία

7.1. Συμπεράσματα

Στη εργασία [illegible] ασχοληθήκαμε με την υλοποίηση ενός συντακτικού αναλυτή για τον παραλληλο[illegible] μεταφραστή OMPi, ο οποίος υποστηρίζει τις προδιαγραφές OpenMP για [illegible] προγραμματισμό κοινής μνήμης.

Σε σημαντικό [illegible] προηγούμενη υλοποίηση, [illegible] λεκτικού ή [illegible] συντακτικό [illegible]. Η νέα υλοποίηση του μεταφραστή OMPi αποτελείται από σύνολο περίπου 8.000 γραμμές [illegible] και 1500 γραμμές [illegible] όλα τα τμήματα του [illegible].

Η νέα υλοποίηση [illegible] σε C++. Η επιλογή αυτή μαζί με [illegible] μας [illegible], κάνουν [illegible] την [illegible] κώδικα πιο [illegible] και επομένως πιο συντηρήσιμο.

7.1.1. [illegible]

Η υλοποίηση [illegible] ελλείψεις όσο αφορά τις οδηγίες και φράσεις OpenMP που [illegible]. Πιο συγκεκριμένα:

* Η φράση [illegible] αναγνωρίζεται από τον μεταφραστή αλλά [illegible] [illegible].

ΚΕΦΑΛΑΙΟ 7. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

7.1. Συμπεράσματα
7.2. Μελλοντική εργασία

7.1. Συμπεράσματα

Στη εργασία αυτή ασχοληθήκαμε με την υλοποίηση ενός συντακτικού αναλυτή για τον παραλληλοποιητικό μεταφραστή OMPi, ο οποίος υποστηρίζει τις προδιαγραφές OpenMP για παράλληλο προγραμματισμό κοινής μνήμης.

Σε σημαντικό ποσοστό η προηγούμενη υλοποίηση δν διέθετε λεπτομερή ή επαρκή σχολιασμό του κώδικα. Η νέα υλοποίηση του μεταφραστή OMPi αποτελείται από σύνολο περίπου 8000 γραμμές κώδικα και 3500 γραμμές σχόλια που συνοδεύουν όλα τα σημεία του κώδικα.

Η νέα υλοποίηση χρησιμοποιεί ως βασική γλώσσα υλοποίησης την C++. Η επιλογή αυτή μαζί με τη δημιουργία μιας ιεραρχίας κλάσεων βοήθησε στην παραγωγή κώδικα πιο τμηματοποιημένου και επομένως πιο συντηρήσιμου.

7.1.1. Ελλείψεις

Η υλοποίηση έχει κάποιες ελλείψεις όσο αφορά τις οδηγίες και φράσεις OpenMP που περιγράφονται στο πρότυπο. Πιο συγκεκριμένα:

- Η οδηγία threadprivate αναγνωρίζεται από τον μεταφραστή αλλά αυτή τη στιγμή αγνοείται.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- Η φράση copyin εξαρτάται άμεσα από την υλοποίηση της οδηγίας threadprivate. Για αυτό το λόγο η υλοποίησή της είναι ατελής.

7.2. Μελλοντική εργασία

Η υλοποίηση του μεταφραστή αφήνει αρκετό χώρο για σημαντικές επεκτάσεις και βελτιώσεις.

7.2.1. Επεκτάσεις

7.2.1.1. Φωλιασμένος παραλληλισμός

Είναι δυνατό κατά τη συγγραφή κώδικα με επεκτάσεις OpenMP να εισαχθούν περιοχές παραλληλισμού μέσα σε άλλες περιοχές παραλληλισμού. Η προηγούμενη υλοποίηση του OMPi, λόγω δομής τόσο του συντακτικού-σημασιολογικού αναλυτή όσο και της βιβλιοθήκης εκτέλεσης (runtime library), δημιουργεί για κάθε εμφωλευμένη παράλληλη περιοχή μια ομάδα νημάτων αποτελούμενη από ένα μόνο νήμα.

Με την παρούσα υλοποίηση και την ανάλογη βελτίωση της βιβλιοθήκης εκτέλεσης είναι δυνατό να υλοποιηθεί πλήρως με ομάδα πολλών νημάτων η εκτέλεση φωλιασμένων παράλληλων περιοχών.

7.2.2. Βελτιστοποιήσεις

7.2.2.1. Απαλοιφή πλεοναζόντων φραγμάτων

Η ύπαρξη όλης της πληροφορίας για τον κώδικα συγκεντρωμένης στο συντακτικό δέντρο μας επιτρέπει να το διατρέχουμε και να την ανακτήσουμε. Ένας τρόπος να εκμεταλλευτούμε αυτό το γεγονός είναι η εύρεση και απαλοιφή πλεοναζόντων φραγμάτων.

Κατά το μετασχηματισμό των δομών και οδηγιών OpenMP αφήνεται χωρίς να αλλοιωθεί οποιαδήποτε οδηγία φράγματος υπάρχει (#pragma omp barrier). Με αυτό

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

τον τρόπο, όταν τελειώσουν όλοι οι μετασχηματισμοί το συντακτικό δέντρο περιέχει μόνο καθαρό κώδικα C και τυχόν οδηγίες φράγματος.

Βρίσκοντας μέσα στο δέντρο τις οδηγίες φράγματος είναι εύκολο να καθοριστεί αν κάποιες από αυτές είναι πλεονάζουσες, ώστε να απαλοιφούν [].

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΑΝΑΦΟΡΕΣ

[1] American National Standards Institute, INCITS/ISO/IEC 9899-1999 (E) - Programming Languages - C (formerly ANSI/ISO/IEC 9899-1999), 1999

[2] The GNU bison online manual,
http://www.gnu.org/software/bison/bison.html

[3] V.V. Dimakopoulos, E. Leontiadis and G. Tzoumas, "A Portable C Compiler for OpenMP V.2.0", in Proc. EWOMP 2003, 5th European Workshop on OpenMP, Aachen, Germany, Sept. 2003

[4] A. Georgopoulos, Θέματα υλοποίησης μεταφραστή για το πρότυπο παράλληλου προγραμματισμού OpenMP, Dept. of Computer Science, Univ. of Ioannina, Sept. 2004

[5] The GOMP project,
http://gcc.gnu.org/projects/gomp/

[6] E. Leontiadis, G. Tzoumas, Μεταφραστής γλώσσας προγραμματισμού C με επεκτάσεις OpenMP για παραλληλοποίηση, Dept. of Computer Science, Univ. of Ioannina, June 2002

[7] OpenMP Architecture Review Board, "The OpenMP 2.5 specification", http://www.openmp.org/drupal/mp-documents/spec25.pdf, May 2005

[8] Stroustrup Bjarne, The C++ programming language Third edition, Addison-Wesley Publishing Company, 1997

[9] Van der Linden Peter, Expert C Programming, Prentice Hall, 1994

ΠΑΡΑΡΤΗΜΑ

ΠΑΡΑΡΤΗΜΑ Α

Κώδικας του εργαλείου παραγωγής συντακτικού αναλυτή bison.

```
%{
/*
  OMPi OpenMP Compiler
  Copyright 2001-2006 Vassilios V. Dimakopoulos, Elias Leontiadis,
George Tzoumas,
  Alkis Georgopoulos, Spyros Melissovas

  This file is part of OMPi.

  OMPi is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published
by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  OMPi is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with OMPi; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-
1307  USA
*/

// C++ headers
#include <iostream>
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
// C headers
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <stdarg.h>
#include <ctype.h>
#include <assert.h>

// local headers
#include "objects.hpp"


// generic object of the parser
#define YYSTYPE p_generic_component

// debug flag
// need to uncomment yydebug in main also
#define YYDEBUG 1

// xml parse tree output file
#define XMLOUTFILE                    "source.xml"
// source output file
#define SRCOUTFILE                    stdout


int yylex (void);
void yyerror (int create_tree_only, char *s);

// redirect yyparse input to a file instead of stdin
extern FILE *yyin;

// obtain line numbers from preprocessor defines
extern int orig_line;
extern int line_no;
extern int new_line;

// column number
extern int column;
```

```
// original filename
extern char orig_fname[1024];

// get the actual line number based on preprocessor defines
extern int get_actual_lineno(void);

extern void set_input_buffer(const char *str);
extern void delete_input_buffer(void);

extern int process_tree(generic_component *start_ptr);

// class names array
extern char *class_names[];

// rule names array
extern char *rule_names[];

extern int reparsing_parallel_level;

// flag for the parser. see comments below
extern int program_start;
// contains the element number of the translation unit params vector
// where the real program starts, in order to be able to add
// declarations later
int program_start_element = 0;
// where to put new code elements
int program_put_element = 0;

// points to the root object (see translation_unit rule)
generic_component *p_start = NULL;

// transformation tree pointer
// When the parser is called again in order to generate
// transformation trees,
// the new tree top node is returned in t_start
generic_component *t_start = NULL;

// points to the active declarations_collection object (standard
// declarations)
```

```
declarations_collection *active_declarations_collection = NULL;

// points to the active typenames_collection object (type
// declarations)
typenames_collection *active_typenames_collection = NULL;

// temporarily holds the function collections in order for it to
// be placed in the block_item_list afterwards
declarations_collection *function_declarations_collection = NULL;

// points to the current declaration NODE
gendecl *active_declaration = NULL;

// This is used when scanning buffers, in the second pass.
// It is used as a convenience considering the declarations and
// typenames collections.
// When this turns to "1" in the translation unit rule, the
// collections are not
// created from scratch, instead they are pointed to the active ones,
// so no merging will
// be necessary in the subtree insertion.
int buffer_scanning_enabled = 0;

// flag to signal if this pragma define is nested inside another
int parallel_level = 0;

// when 1, several printfs are enabled for debugging
int debugflag = 0;


%}

%defines
%output="parser.cpp"

// expect 2 shift/reduce
%expect 2

// parameter which directs yyparse to create a modified version of
// the tree
```

```
// for use in transformations, when a tree structure is needed
%parse-param { int create_tree_only }

// token declarations
%token   IDENTIFIER TYPE_NAME CONSTANT STRING_LITERAL
%token   PTR_OP INC_OP DEC_OP LEFT_OP RIGHT_OP LE_OP GE_OP EQ_OP
NE_OP
%token   AND_OP OR_OP MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
%token   SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN
%token   XOR_ASSIGN OR_ASSIGN SIZEOF

%token   TYPEDEF EXTERN STATIC AUTO REGISTER RESTRICT
%token   CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE
%token   CONST RESTRICT VOLATILE VOID INLINE
%token   UBOOL UCOMPLEX UIMAGINARY
%token   STRUCT UNION ENUM ELLIPSIS

%token   CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK
RETURN

%token   PRAGMA_OMP PRAGMA_OMP_THREADPRIVATE OMP_PARALLEL
OMP_SECTIONS OMP_NOWAIT OMP_ORDERED
%token   OMP_SCHEDULE OMP_STATIC OMP_DYNAMIC OMP_GUIDED OMP_RUNTIME
OMP_SECTION
%token   OMP_SINGLE OMP_MASTER OMP_CRITICAL OMP_BARRIER OMP_ATOMIC
OMP_FLUSH
%token   OMP_PRIVATE OMP_FIRSTPRIVATE
%token   OMP_LASTPRIVATE OMP_SHARED OMP_DEFAULT OMP_NONE
OMP_REDUCTION
%token   OMP_COPYIN OMP_NUMTHREADS OMP_COPYPRIVATE OMP_FOR OMP_IF

// additional POMP tokens
%token   POMP_INST POMP_INIT POMP_FINALIZE POMP_ON POMP_OFF
%token   POMP_BEGIN POMP_END POMP_NOINSTRUMENT POMP_INSTRUMENT

// parsing top rule is the translation unit rule
%start translation_unit

%%
```

```
// Note regarding rule actions
//
// Each rule normally either creates a new object of type node or
// derived
// or simply passes the lex generated genterm or genident object to
// the parent
// rule.
// There are some exceptions as noted within the applying rules.
//
// Tree flattening -----------------------------------------------
//
// Rules:
//
//  1. argument_expression_list
//  2. init_declarator_list
//  3. struct_declaration_list
//  4. specifier_qualifier_list
//  5. struct_declarator_list
//  6. enumerator_list
//  7. type_qualifier_list
//  8. parameter_list
//  9. identifier_list
// 10. designator_list
// 11. block_item_list
// 12. declaration_list
// 13. variable_list
//
// flatten the parse tree by inserting new parameters to the
// previously matched list
// so instead of creating more gennode objects and generating a tree
// like this:

//                                  argument_expression_list
//                                    /                  \
//            argument_expression_list                argument_expression
//                       /                \
//          argument_expression_list   argument_expression
//                     /            \
//    argument_expression_list    argument_expression
//               /
```

```
// argument_expression
//
// the flattened tree contains all argument_expression as children of
// the
// argument_expression_list:
//
//                                      argument_expression_list
//                                      /      |         |      \
//                                     /       |         |       \
//                                    /        |         |        \
//          argument_expression                |         |argument_expression
//                                             |         |
//                          argument_expression   argument_expression
//
// There is a drawback in this method, that the top level node (in
// this example
// argument_expression_list) is always created, even if there is only
// one child.
// This is currently useful in identifying such subtrees, e.g.
// function parameter
// lists inside a function definition rule.


// -------------------------------------------------------------------
// --------- Bison rules begin ----------------------------------------
// -------------------------------------------------------------------



// -------------------------------------------------------------------
// --------- ISO/IEC 9899:1999 A.1 Lexical grammar -----------------
// -------------------------------------------------------------------


// -------------------------------------------------------------------
// --------- ISO/IEC 9899:1999 A.1.5 Constants --------------------
// -------------------------------------------------------------------
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
// ISO/IEC 9899:1999 6.4.4.3
enumeration_constant:
    IDENTIFIER
    {
      $$ = $1;
    }
;


// ----------------------------------------------------------------------
// ---------- ISO/IEC 9899:1999 A.1.6 String literals --------------
// ----------------------------------------------------------------------


// ISO/IEC 9899:1999 6.4.5
string_literal:                         // "abcd" "abcd"
    STRING_LITERAL
    {
      $$ = $1;
    }
  | STRING_LITERAL string_literal
    {
      $$ = new gennode(PR_STRING_LITERAL, $1, $2);
    }
;


// ----------------------------------------------------------------------
// ------ ISO/IEC 9899:1999 A.2 Phrase structure grammar ------------
// ----------------------------------------------------------------------


// ----------------------------------------------------------------------
// ------- ISO/IEC 9899:1999 A.2.1 Expressions ----------------------
// ----------------------------------------------------------------------


// ISO/IEC 9899:1999 6.5.1
primary_expression:
//
(2+3)
```

```
    IDENTIFIER
    {
      int declaration_errors = 0;

      $$ = $1;

/*
      // update the genidents in the subtree starting from $$
      // to use the current declarations_collection node
      if((declaration_errors = $$->update_genident_pointers()) > 0)
      {
        fprintf(stderr, "primary_expression_1: %d errors
encountered\n", declaration_errors);
        exit(1);
      }
*/
    }
  | CONSTANT
    {
      $$ = $1;
    }
  | string_literal
    {
      $$ = $1;
    }
  | '(' expression ')'
    {
      $$ = new gennode(PR_PRIMARY_EXPRESSION, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.5.2
postfix_expression:                // printf(), a->b, a.b, a++, a--
    primary_expression
    {
      $$ = $1;
    }
  | postfix_expression '[' expression ']'
    {
      $$ = new gennode(PR_POSTFIX_EXPRESSION_2, $1, $2, $3, $4);
```

```
        }
      | postfix_expression '(' ')'
        {
          $$ = new gennode(PR_POSTFIX_EXPRESSION_3, $1, $2, $3);
        }
      | postfix_expression '(' argument_expression_list ')'
        {
          $$ = new gennode(PR_POSTFIX_EXPRESSION_4, $1, $2, $3, $4);
        }
      | postfix_expression '.' IDENTIFIER
        {
          $1->update_genident_pointers();
          $$ = new gennode(PR_POSTFIX_EXPRESSION_MEMBER, $1, $2, $3);
        }
      | postfix_expression PTR_OP IDENTIFIER
        {
          $1->update_genident_pointers();
          $$ = new gennode(PR_POSTFIX_EXPRESSION_PTR_MEMBER, $1, $2, $3);
        }
      | postfix_expression INC_OP
        {
          $$ = new gennode(PR_POSTFIX_EXPRESSION_INC_OP, $1, $2);
        }
      | postfix_expression DEC_OP
        {
          $$ = new gennode(PR_POSTFIX_EXPRESSION_DEC_OP, $1, $2);
        }
      | '(' type_name ')' '{' initializer_list '}'
        {
          $$ = new gennode(PR_POSTFIX_EXPRESSION_9, $1, $2, $3, $4, $5,
$6);
        }
      | '(' type_name ')' '{' initializer_list ',' '}'
        {
          $$ = new gennode(PR_POSTFIX_EXPRESSION_10, $1, $2, $3, $4, $5,
$6, $7);
        }
;

// ISO/IEC 9899:1999 6.5.2
```

```
argument_expression_list:
    assignment_expression
    {
      $$ = new gennode(PR_ARGUMENT_EXPRESSION_LIST, $1);
    }
  | argument_expression_list ',' assignment_expression
    {
      // flatten the argument_expression_list tree (see translation
      // unit rule)
      $$ = ((gennode *)$1)->add_parameter($2, $3);
    }
;

// ISO/IEC 9899:1999 6.5.3
unary_expression:                  // ++a, sizeof(char), *(char *)a
    postfix_expression
    {
      $$ = $1;
    }
  | INC_OP unary_expression
    {
      $$ = new gennode(PR_UNARY_EXPRESSION_INC_OP, $1, $2);
    }
  | DEC_OP unary_expression
    {
      $$ = new gennode(PR_UNARY_EXPRESSION_DEC_OP, $1, $2);
    }
  | unary_operator cast_expression
    {
      $$ = new gennode(PR_UNARY_EXPRESSION_4, $1, $2);
    }
  | SIZEOF unary_expression
    {
      $$ = new gennode(PR_UNARY_EXPRESSION_5, $1, $2);
    }
  | SIZEOF '(' type_name ')'
    {
      $$ = new gennode(PR_UNARY_EXPRESSION_6, $1, $2, $3, $4);
    }
;
```

```
// ISO/IEC 9899:1999 6.5.3
unary_operator:                          // &, *, ~, !
    '&'
    {
      $$ = $1;
    }
  | '*'
    {
      $$ = $1;
    }
  | '+'
    {
      $$ = $1;
    }
  | '-'
    {
      $$ = $1;
    }
  | '~'
    {
      $$ = $1;
    }
  | '!'
    {
      $$ = $1;
    }
;

// ISO/IEC 9899:1999 6.5.4
cast_expression:                         // (int)++a
    unary_expression
    {
      $$ = $1;
    }
  | '(' type_name ')' cast_expression
    {
      $$ = new gennode(PR_CAST_EXPRESSION, $1, $2, $3, $4);
    }
;
```

```
// ISO/IEC 9899:1999 6.5.5
multiplicative_expression:          // (int)++a / (char)--b
    cast_expression
    {
      $$ = $1;
    }
  | multiplicative_expression '*' cast_expression
    {
      $$ = new gennode(PR_MULTIPLICATIVE_EXPRESSION_2, $1, $2, $3);
    }
  | multiplicative_expression '/' cast_expression
    {
      $$ = new gennode(PR_MULTIPLICATIVE_EXPRESSION_3, $1, $2, $3);
    }
  | multiplicative_expression '%' cast_expression
    {
      $$ = new gennode(PR_MULTIPLICATIVE_EXPRESSION_4, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.5.6
additive_expression:                // 1 / 2 + 3 * 5
    multiplicative_expression
    {
      $$ = $1;
    }
  | additive_expression '+' multiplicative_expression
    {
      $$ = new gennode(PR_ADDITIVE_EXPRESSION_2, $1, $2, $3);
    }
  | additive_expression '-' multiplicative_expression
    {
      $$ = new gennode(PR_ADDITIVE_EXPRESSION_3, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.5.7
shift_expression:                   // 1/2+3*5 << 5*4-3/3
    additive_expression
```

```
    {
      $$ = $1;
    }
  | shift_expression LEFT_OP additive_expression
    {
      $$ = new gennode(PR_SHIFT_EXPRESSION_2, $1, $2, $3);
    }
  | shift_expression RIGHT_OP additive_expression
    {
      $$ = new gennode(PR_SHIFT_EXPRESSION_3, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.5.8
relational_expression:              // ++a < --b
    shift_expression
    {
      $$ = $1;
    }
  | relational_expression '<' shift_expression
    {
      $$ = new gennode(PR_RELATIONAL_EXPRESSION_LT, $1, $2, $3);
    }
  | relational_expression '>' shift_expression
    {
      $$ = new gennode(PR_RELATIONAL_EXPRESSION_GT, $1, $2, $3);
    }
  | relational_expression LE_OP shift_expression
    {
      $$ = new gennode(PR_RELATIONAL_EXPRESSION_LE, $1, $2, $3);
    }
  | relational_expression GE_OP shift_expression
    {
      $$ = new gennode(PR_RELATIONAL_EXPRESSION_GE, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.5.9
equality_expression:                // ++a == --b
    relational_expression
```

```
        {
          $$ = $1;
        }
      | equality_expression EQ_OP relational_expression
        {
          $$ = new gennode(PR_EQUALITY_EXPRESSION_2, $1, $2, $3);
        }
      | equality_expression NE_OP relational_expression
        {
          $$ = new gennode(PR_EQUALITY_EXPRESSION_3, $1, $2, $3);
        }
;

// ISO/IEC 9899:1999 6.5.10
AND_expression:                         // ++a == --b  &  1 != 2
        equality_expression
        {
          $$ = $1;
        }
      | AND_expression '&' equality_expression
        {
          $$ = new gennode(PR_AND_EXPRESSION, $1, $2, $3);
        }
;

// ISO/IEC 9899:1999 6.5.11
exclusive_OR_expression:                // ++a == --b  ^  1 != 2
        AND_expression
        {
          $$ = $1;
        }
      | exclusive_OR_expression '^' AND_expression
        {
          $$ = new gennode(PR_EXCLUSIVE_OR_EXPRESSION, $1, $2, $3);
        }
;

// ISO/IEC 9899:1999 6.5.12
inclusive_OR_expression:                // 1 | 2
        exclusive_OR_expression
```

```
      {
        $$ = $1;
      }
    | inclusive_OR_expression '|' exclusive_OR_expression
      {
        $$ = new gennode(PR_INCLUSIVE_OR_EXPRESSION, $1, $2, $3);
      }
;

// ISO/IEC 9899:1999 6.5.13
logical_AND_expression:                // a && b
      inclusive_OR_expression
      {
        $$ = $1;
      }
    | logical_AND_expression AND_OP inclusive_OR_expression
      {
        $$ = new gennode(PR_LOGICAL_AND_EXPRESSION, $1, $2, $3);
      }
;

// ISO/IEC 9899:1999 6.5.14
logical_OR_expression:                 // a || b
      logical_AND_expression
      {
        $$ = $1;
      }
    | logical_OR_expression OR_OP logical_AND_expression
      {
        $$ = new gennode(PR_LOGICAL_OR_EXPRESSION, $1, $2, $3);
      }
;

// ISO/IEC 9899:1999 6.5.15
conditional_expression:                // a?b:1==0
      logical_OR_expression
      {
        $$ = $1;
      }
    | logical_OR_expression '?' expression ':' conditional_expression
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
    {
      $$ = new gennode(PR_CONDITIONAL_EXPRESSION, $1, $2, $3, $4,
$5);
    }
;

// ISO/IEC 9899:1999 6.5.16
assignment_expression:              // a += f
    conditional_expression
    {
      $$ = $1;
    }
  | unary_expression assignment_operator assignment_expression
    {
      $$ = new gennode(PR_ASSIGNMENT_EXPRESSION, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.5.16
assignment_operator:                // =, +=, -=, <<=, >>=
    '='
    {
      $$ = $1;
    }
  | MUL_ASSIGN
    {
      $$ = $1;
    }
  | DIV_ASSIGN
    {
      $$ = $1;
    }
  | MOD_ASSIGN
    {
      $$ = $1;
    }
  | ADD_ASSIGN
    {
      $$ = $1;
    }
```

```
    | SUB_ASSIGN
      {
        $$ = $1;
      }
    | LEFT_ASSIGN
      {
        $$ = $1;
      }
    | RIGHT_ASSIGN
      {
        $$ = $1;
      }
    | AND_ASSIGN
      {
        $$ = $1;
      }
    | XOR_ASSIGN
      {
        $$ = $1;
      }
    | OR_ASSIGN
      {
        $$ = $1;
      }
;

// ISO/IEC 9899:1999 6.5.17
expression:
        // 1 = 2, a = 3
      assignment_expression
      {
        $$ = $1;
        $$->update_genident_pointers();
      }
    | expression ',' assignment_expression
      {
        $$ = new gennode(PR_EXPRESSION, $1, $2, $3);
      }
;
```

```
// ISO/IEC 9899:1999 6.6
constant_expression:
    conditional_expression
    {
      $$ = $1;
    }
;


// ----------------------------------------------------------------
// ------------ ISO/IEC 9899:1999 A.2.2 Declarations ----------------
// ----------------------------------------------------------------

// ISO/IEC 9899:1999 6.7
declaration:       // typedef int a; int a; char f='k';
    declaration_specifiers ';'
    {
      // enums, structs and unions will be matched by this rule
      $$ = new gendecl(PR_DECLARATION_SEU, $1, $2);
    }
  | declaration_specifiers init_declarator_list ';'
    {
      genident *p_ident;
      int numvars = 0;

      $$ = new gendecl(PR_DECLARATION_NORMAL, $1, $2, $3);

      // if p_node is of class gennode then check if its first
      // parameter is "typedef" or "extern"
      if($1->classname == CN_NODE)
      {
        if(!strcmp(((gennode *)$1)->params[0]->value, "typedef"))
        {
          // typedef found, $2's first CN_IDENT should be a type name
          if(debugflag)
          {

          fprintf(stderr, "typedef, init_declarator_list:\n");
            $2->traverse();
            fflush(stdout);
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
            }

            // add the new typename(s) to the active typenames
            // collection

          if(numvars = active_typenames_collection->
            traverse_and_add($2, SEARCH_CURRENT_SCOPE))
            {

              // actually, no redefined type name will come this way...
              // after the initial typedef check, the token will return
as
              // TYPE_NAME next time, so it will match declaration_1
              // rule instead
              // and happily redefine itself. Any redefinition will
              // become obvious
              // when the final gcc runs, as the preprocessor isn't
              // aware of it

              fprintf(stderr, "%s: error: %d variable%s redefined\n",
orig_fname, numvars, (numvars - 1)?"s":"");
              exit(1);
            }
          } // end typedef
          else
              if(numvars = active_declarations_collection-
>traverse_and_add($2, SEARCH_CURRENT_SCOPE))
              {
                fprintf(stderr, "%s: error: %d variable%s redefined\n",
orig_fname, numvars, (numvars - 1)?"s":"");
                exit(1);
              }
        }
        else
        { // CN_TERM or type name
          //
          // a normal declaration
          //
          // add the new declaration(s) to the active collection
```

```
        if(debugflag)
        {
                                                     fprintf(stderr,
"init_declarator_list:\n");
          $2->traverse();
          fflush(stdout);
        }


        if(numvars = active_declarations_collection-
>traverse_and_add($2, SEARCH_CURRENT_SCOPE))
        {
          fprintf(stderr, "%s: error: %d variable%s redefined\n",
orig_fname, numvars, (numvars - 1)?"s":"");
          exit(1);
        }
      }

                                              }
  | threadprivate_directive // OpenMP Version 2.5 ISO/IEC 9899:1999
                            // addition
    {
      $$ = $1;
    }
;

// ISO/IEC 9899:1999 6.7
declaration_specifiers:                 // int a    extern   auto
    storage_class_specifier
    {
      $$ = $1;
    }
  | storage_class_specifier declaration_specifiers
    {
      $$ = new gennode(PR_DECLARATION_SPECIFIERS_2, $1, $2);
    }
  | type_specifier
    {
      $$ = $1;
    }
```

```
    | type_specifier declaration_specifiers
      {
        $$ = new gennode(PR_DECLARATION_SPECIFIERS_4, $1, $2);
      }
    | type_qualifier
      {
        $$ = $1;
      }
    | type_qualifier declaration_specifiers
      {
        $$ = new gennode(PR_DECLARATION_SPECIFIERS_6, $1, $2);
      }
    | function_specifier
      {
        $$ = $1;
      }
    | function_specifier declaration_specifiers
      {
        $$ = new gennode(PR_DECLARATION_SPECIFIERS_8, $1, $2);
      }
;

// ISO/IEC 9899:1999 6.7
init_declarator_list:                    // a = 5, b = 8, c
      init_declarator
      {
        $$ = new gennode(PR_INIT_DECLARATOR_LIST, $1);
      }
    | init_declarator_list ',' init_declarator
      {
        // flatten the init_declarator_list tree (see translation unit
        // rule)
        $$ = ((gennode *)$1)->add_parameter($2, $3);
      }
;

// ISO/IEC 9899:1999 6.7
init_declarator:                         // a = 5
      declarator
      {
```

```
        $$ = $1;
      }
    | declarator '=' initializer
      {
        int declaration_errors = 0;
        int numvars = 0;

        $$ = new gennode(PR_INIT_DECLARATOR_INITIALIZER, $1, $2, $3);

        if(numvars = active_declarations_collection-
>traverse_and_add($1, SEARCH_CURRENT_SCOPE))
        {
          fprintf(stderr, "%s: error: %d variable%s redefined\n",
orig_fname, numvars, (numvars - 1)?"s":"");
          exit(1);
        }

        // update the genidents in the subtree starting from $3
        // to use the current declarations_collection node
        if((declaration_errors = $3->update_genident_pointers()) > 0)
        {
          fprintf(stderr, "init_declarator_2: %d errors encountered\n",
declaration_errors);
          exit(1);
        }
      }
;

// ISO/IEC 9899:1999 6.7.1
storage_class_specifier:              //
      TYPEDEF
      {
        $$ = $1;
      }
    | EXTERN
      {
        $$ = $1;
      }
    | STATIC
```

```
    {
      $$ = $1;
    }
  | AUTO
    {
      $$ = $1;
    }
  | REGISTER
    {
      $$ = $1;
    }
;

// ISO/IEC 9899:1999 6.7.2
type_specifier:
      // enum { ... }
    VOID
    {
      $$ = $1;
    }
  | CHAR
    {
      $$ = $1;
    }
  | SHORT
    {
      $$ = $1;
    }
  | INT
    {
      $$ = $1;
    }
  | LONG
    {
      $$ = $1;
    }
  | FLOAT
    {
      $$ = $1;
    }
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
  | DOUBLE
    {
      $$ = $1;
    }
  | SIGNED
    {
      $$ = $1;
    }
  | UNSIGNED
    {
      $$ = $1;
    }
  | UBOOL
    {
      $$ = $1;
    }
  | UCOMPLEX
    {
      $$ = $1;
    }
  | UIMAGINARY
    {
      $$ = $1;
    }
  | struct_or_union_specifier
    {
      $$ = $1;
    }
  | enum_specifier
    {
      $$ = $1;
    }
  | typedef_name
    {
      $$ = $1;
    }
;

// ISO/IEC 9899:1999 6.7.2.1
struct_or_union_specifier:           // struct blah { int counter }
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
    struct_or_union '{' struct_declaration_list '}'
    {
      $$ = new gennode(PR_STRUCT_OR_UNION_SPECIFIER_1, $1, $2, $3,
$4);
    }
  | struct_or_union '{' '}' // NON-ISO empty declaration
                            // (added by Spyros Melissovas)
    {
      $$ = new gennode(PR_STRUCT_OR_UNION_SPECIFIER_1, $1, $2, $3);
    }
  | struct_or_union IDENTIFIER '{' struct_declaration_list '}'
    {
      $$ = new gennode(PR_STRUCT_OR_UNION_SPECIFIER_2, $1, $2, $3,
$4, $5);
    }
     // ADDITION: non-ISO (Spyros Melissovas)
     // When an identifier is inserted in the typenames declaration,
     // next time
     // it is encountered it will be a type_name and won't be matched
     // by the above rule
  | struct_or_union TYPE_NAME '{' struct_declaration_list '}'
    {
      ((genident *)$2)->set_typename();
      $$ = new gennode(PR_STRUCT_OR_UNION_SPECIFIER_2, $1, $2, $3,
$4, $5);
    }
  | struct_or_union IDENTIFIER
    {
      $$ = new gennode(PR_STRUCT_OR_UNION_SPECIFIER_3, $1, $2);
    }
  | struct_or_union TYPE_NAME // ADDITION: non-ISO (see above)
    {
      ((genident *)$2)->set_typename();
      $$ = new gennode(PR_STRUCT_OR_UNION_SPECIFIER_4, $1, $2);
    }
;

// ISO/IEC 9899:1999 6.7.2.1
struct_or_union:                                          //
    STRUCT
```

```
    {
      $$ = $1;
    }
  | UNION
    {
      $$ = $1;
    }
;

// ISO/IEC 9899:1999 6.7.2.1
struct_declaration_list:              // char a; int b;
    struct_declaration
    {
      $$ = new gennode(PR_STRUCT_DECLARATION_LIST, $1);
    }
  | struct_declaration_list struct_declaration
    {
      // flatten the struct_declaration_list tree (see translation
      // unit rule)
      $$ = ((gennode *)$1)->add_parameter($2);
    }
;

// ISO/IEC 9899:1999 6.7.2.1
struct_declaration:                   // char b;
    specifier_qualifier_list struct_declarator_list ';'
    {
      $$ = new gennode(PR_STRUCT_DECLARATION, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.7.2.1
specifier_qualifier_list:
    type_specifier
    {
      $$ = $1;
    }
  | type_specifier specifier_qualifier_list
    {
      $$ = new gennode(PR_SPECIFIER_QUALIFIER_LIST_2, $1, $2);
```

```
    }
  | type_qualifier
    {
      $$ = $1;
    }
  | type_qualifier specifier_qualifier_list
    {
      $$ = new gennode(PR_SPECIFIER_QUALIFIER_LIST_4, $1, $2);
    }
;

// ISO/IEC 9899:1999 6.7.2.1
struct_declarator_list:
    struct_declarator
    {
      $$ = new gennode(PR_STRUCT_DECLARATOR_LIST, $1);
    }
  | struct_declarator_list ',' struct_declarator
    {
      // flatten the struct_declarator_list tree (see translation
      // unit rule)
      $$ = ((gennode *)$1)->add_parameter($2, $3);
    }
;

// ISO/IEC 9899:1999 6.7.2.1
struct_declarator:                    // : a?b:1==0
    declarator
    {
      $$ = $1;
    }
  | declarator ':' constant_expression
    {
      $$ = new gennode(PR_STRUCT_DECLARATOR_2, $1, $2, $3);
    }
  | ':' constant_expression
    {
      $$ = new gennode(PR_STRUCT_DECLARATOR_3, $1, $2);
    }
;
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
// ISO/IEC 9899:1999 6.7.2.2
enum_specifier:
      // enum blah { ... }
    ENUM '{' enumerator_list '}'
    {
      $$ = new gennode(PR_ENUM_SPECIFIER_1, $1, $2, $3, $4);
    }
  | ENUM IDENTIFIER '{' enumerator_list '}'
    {
      int numvars = 0;

      $$ = new gendecl(PR_ENUM_SPECIFIER_2, $1, $2, $3, $4, $5);

      // add the identifier to the active typenames collection
      if(numvars = active_typenames_collection->traverse_and_add($2,
SEARCH_CURRENT_SCOPE))
      {
        fprintf(stderr, "%s: error: redefinition of %s in line
%d,%d\n", orig_fname, $2->value, ((genident *)$2)->lineno, ((genident
*)$2)->colno);
        exit(1);
      }
    }
  | ENUM '{' enumerator_list ',' '}'
    {
      $$ = new gendecl(PR_ENUM_SPECIFIER_3, $1, $2, $3, $4, $5);
    }
  | ENUM IDENTIFIER '{' enumerator_list ',' '}'
    {
      int numvars = 0;

      $$ = new gendecl(PR_ENUM_SPECIFIER_4, $1, $2, $3, $4, $5, $6);

      // add the identifier to the active typenames collection
      if(numvars = active_typenames_collection->traverse_and_add($2,
SEARCH_CURRENT_SCOPE))
      {
```

```
        fprintf(stderr, "%s: error: redefinition of %s in line
%d,%d\n", orig_fname, $2->value, ((genident *)$2)->lineno, ((genident
*)$2)->colno);
        exit(1);
      }
    }
  | ENUM IDENTIFIER
    {
      int numvars = 0;

      $$ = new gendecl(PR_ENUM_SPECIFIER_5, $1, $2);

      // add the identifier to the active typenames collection
      if(numvars = active_typenames_collection->traverse_and_add($2,
SEARCH_CURRENT_SCOPE))
      {
        fprintf(stderr, "%s: error: redefinition of %s in line
%d,%d\n", orig_fname, $2->value, ((genident *)$2)->lineno, ((genident
*)$2)->colno);
        exit(1);
      }

    }
;

// ISO/IEC 9899:1999 6.7.2.2
enumerator_list:                        // enum_a, enum_b
    enumerator
    {
      $$ = new gennode(PR_ENUMERATOR_LIST, $1);
    }
  | enumerator_list ',' enumerator
    {
      // flatten the enumerator_list tree (see translation unit rule)
      $$ = ((gennode *)$1)->add_parameter($2, $3);
    }
;

// ISO/IEC 9899:1999 6.7.2.2
```

```
enumerator:
        // enum_a = {2}
      enumeration_constant
      {
        $$ = $1;
      }
    | enumeration_constant '=' constant_expression
      {
        $$ = new gennode(PR_ENUMERATOR, $1, $2, $3);
      }
;

// ISO/IEC 9899:1999 6.7.3
type_qualifier:
      CONST
      {
        $$ = $1;
      }
    | RESTRICT
      {
        $$ = $1;
      }
    | VOLATILE
      {
        $$ = $1;
      }
;

// ISO/IEC 9899:1999 6.7.4
function_specifier:
      INLINE
      {
        $$ = $1;
      }
;

// ISO/IEC 9899:1999 6.7.5
declarator:
        // counter, *counter
      direct_declarator
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
        {
          $$ = $1;
        }
      | pointer direct_declarator
        {
          $$ = new gennode(PR_DECLARATOR_POINTER, $1, $2);
        }
;

// ISO/IEC 9899:1999 6.7.5
direct_declarator:
        // a    (a)    a[]    a[4], ...
      IDENTIFIER
      {
        int numvars = 0;

        $$ = $1;
/*
        if(numvars = active_declarations_collection-
>traverse_and_add($$, SEARCH_CURRENT_SCOPE))
        {
          fprintf(stderr, "%s: error: %d variable%s redefined\n",
orig_fname, numvars, (numvars - 1)?"s":"");
          exit(1);
        }
*/
      }
    | '(' declarator ')'
      {
        $$ = new gennode(PR_DIRECT_DECLARATOR_2, $1, $2, $3);
      }
    | direct_declarator '[' ']'
      {
        $$ = new gennode(PR_DIRECT_DECLARATOR_3, $1, $2, $3);
      }
    | direct_declarator '[' type_qualifier_list ']'
      {
        $$ = new gennode(PR_DIRECT_DECLARATOR_4, $1, $2, $3, $4);
      }
    | direct_declarator '[' assignment_expression ']'
```

```
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_5, $1, $2, $3, $4);
    }
  | direct_declarator '[' type_qualifier_list assignment_expression
']'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_6, $1, $2, $3, $4, $5);
    }
  | direct_declarator '[' STATIC assignment_expression ']'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_7, $1, $2, $3, $4, $5);
    }
  | direct_declarator '[' STATIC type_qualifier_list
assignment_expression ']'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_8, $1, $2, $3, $4, $5,
$6);
    }
  | direct_declarator '[' type_qualifier_list STATIC
assignment_expression ']'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_9, $1, $2, $3, $4, $5,
$6);
    }
  | direct_declarator '[' '*' ']'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_10, $1, $2, $3, $4);
    }
  | direct_declarator '[' type_qualifier_list '*' ']'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_11, $1, $2, $3, $4, $5);
    }
  | direct_declarator '(' parameter_type_list ')'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_12, $1, $2, $3, $4);
    }
  | direct_declarator '(' ')'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_13, $1, $2, $3);
    }
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
  | direct_declarator '(' identifier_list ')'
    {
      $$ = new gennode(PR_DIRECT_DECLARATOR_14, $1, $2, $3, $4);
    }
;

// ISO/IEC 9899:1999 6.7.5
pointer:                                 // *, **, *char*,
    '*'
    {
      $$ = $1;
    }
  | '*' type_qualifier_list
    {
      $$ = new gennode(PR_POINTER_2, $1, $2);
    }
  | '*' pointer
    {
      $$ = new gennode(PR_POINTER_3, $1, $2);
    }
  | '*' type_qualifier_list pointer
    {
      $$ = new gennode(PR_POINTER_4, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.7.5
type_qualifier_list:                     // char, enum, ...
    type_qualifier
    {
      $$ = new gennode(PR_TYPE_QUALIFIER_LIST, $1);
    }
  | type_qualifier_list type_qualifier
    {
      // flatten the type_qualifier_list tree (see translation unit
      // rule)
      $$ = ((gennode *)$1)->add_parameter($2);
    }
;
```

```
// ISO/IEC 9899:1999 6.7.5
parameter_type_list:                    // a, b, ...
    parameter_list
    {
      $$ = $1;
    }
  | parameter_list ',' ELLIPSIS
    {
      $$ = new gennode(PR_PARAMETER_TYPE_LIST, $1, $2, $3);
    }
;

// ISO/IEC 9899:1999 6.7.5
parameter_list:
      // int argc, char *argv[]
    parameter_declaration
    {
      $$ = new gennode(PR_PARAMETER_LIST, $1);
    }
  | parameter_list ',' parameter_declaration
    {
      // flatten the parameter_list tree (see translation unit rule)
      $$ = ((gennode *)$1)->add_parameter($2, $3);
    }
;

// ISO/IEC 9899:1999 6.7.5
parameter_declaration:                  // int argc
    declaration_specifiers declarator
    {
      $$ = new gendecl(PR_PARAMETER_DECLARATION_1, $1, $2);
    }
  | declaration_specifiers
    {
      $$ = new gendecl(PR_PARAMETER_DECLARATION_2, $1);
    }
  | declaration_specifiers abstract_declarator
    {
      $$ = new gendecl(PR_PARAMETER_DECLARATION_3, $1, $2);
    }
```

```
;

// ISO/IEC 9899:1999 6.7.5
identifier_list:                       // a, b, c
    IDENTIFIER
    {
      $$ = new gennode(PR_IDENTIFIER_LIST, $1);
    }
  | identifier_list ',' IDENTIFIER
    {
      // flatten the identifier_list tree (see translation unit rule)
      $$ = ((gennode *)$1)->add_parameter($2, $3);
    }
;

// ISO/IEC 9899:1999 6.7.6
type_name:
      // char*
    specifier_qualifier_list
    {
      $$ = $1;
    }
  | specifier_qualifier_list abstract_declarator
    {
      $$ = new gennode(PR_TYPE_NAME, $1, $2);
    }
;

// ISO/IEC 9899:1999 6.7.6
abstract_declarator:                   // *(char)
    pointer
    {
      $$ = $1;
    }
  | direct_abstract_declarator
    {
      $$ = $1;
    }
  | pointer direct_abstract_declarator
    {
```

```
        $$ = new gennode(PR_ABSTRACT_DECLARATOR, $1, $2);
      }
  ;

  // ISO/IEC 9899:1999 6.7.6
  direct_abstract_declarator:          // [3]    [][]
      '(' abstract_declarator ')'
      {
        $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_1, $1, $2, $3);
      }
    | '[' ']'
      {
        $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_2, $1, $2);
      }
    | direct_abstract_declarator '[' ']'
      {
        $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_3, $1, $2, $3);
      }
    | '[' assignment_expression ']'
      {
        $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_4, $1, $2, $3);
      }
    | direct_abstract_declarator '[' assignment_expression ']'
      {
        $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_5, $1, $2, $3,
$4);
      }
    | '[' '*' ']'
      {
        $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_6, $1, $2, $3);
      }
    | direct_abstract_declarator '[' '*' ']'
      {
        $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_7, $1, $2, $3,
$4);
      }
    | '(' ')'
      {
        $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_8, $1, $2);
      }
```

```
   | direct_abstract_declarator '(' ')'
     {
       $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_9, $1, $2, $3);
     }
   | '(' parameter_type_list ')'
     {
       $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_10, $1, $2, $3);
     }
   | direct_abstract_declarator '(' parameter_type_list ')'
     {
       $$ = new gennode(PR_DIRECT_ABSTRACT_DECLARATOR_11, $1, $2, $3,
$4);
     }
;

// ISO/IEC 9899:1999 6.7.7
typedef_name:
    TYPE_NAME
    {
      ((genident *)$1)->set_typename();
      $$ = $1;
    }
;

// ISO/IEC 9899:1999 6.7.8
initializer:
      // { a = 0, b = 3 }
    assignment_expression
    {
      $$ = $1;
    }
  | '{' initializer_list '}'
    {
      $$ = new gennode(PR_INITIALIZER_2, $1, $2, $3);
    }
  | '{' initializer_list ',' '}'
    {
      $$ = new gennode(PR_INITIALIZER_3, $1, $2, $3, $4);
    }
;
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
// ISO/IEC 9899:1999 6.7.8
initializer_list:                    // { a = 0 }, {b = 2 }
     initializer
     {
       $$ = $1;
     }
   | designation initializer
     {
       $$ = new gennode(PR_INITIALIZER_LIST_2, $1, $2);
     }
   | initializer_list ',' initializer
     {
       $$ = new gennode(PR_INITIALIZER_LIST_3, $1, $2, $3);
     }
   | initializer_list ',' designation initializer
     {
       $$ = new gennode(PR_INITIALIZER_LIST_4, $1, $2, $3, $4);
     }
;

// ISO/IEC 9899:1999 6.7.8
designation:
     designator_list '='
     {
       $$ = new gennode(PR_DESIGNATION, $1, $2);
     }
;

// ISO/IEC 9899:1999 6.7.8
designator_list:
     designator
     {
       $$ = new gennode(PR_DESIGNATOR_LIST, $1);
     }
   | designator_list designator
     {
       // flatten the designator_list tree (see translation unit rule)
       $$ = ((gennode *)$1)->add_parameter($2);
     }
```

```
;

// ISO/IEC 9899:1999 6.7.8
designator:
     '[' constant_expression ']'
     {
       $$ = new gennode(PR_DESIGNATOR_1, $1, $2, $3);
     }
   | '.' IDENTIFIER
     {
       $$ = new gennode(PR_DESIGNATOR_2, $1, $2);
     }
;


// ---------------------------------------------------------------------
// ------------- ISO/IEC 9899:1999 A.2.3 Statements -----------------
// ---------------------------------------------------------------------

// ISO/IEC 9899:1999 6.8
statement:
     labeled_statement
     {
       $$ = $1;
     }
   | compound_statement
     {
       $$ = $1;
     }
   | expression_statement
     {
       $$ = $1;
     }
   | selection_statement
     {
       $$ = $1;
     }
   | iteration_statement
     {
       $$ = $1;
```

```
      }
    | jump_statement
      {
        $$ = $1;
      }
    | openmp_construct // OpenMP Version 2.5 ISO/IEC 9899:1999 addition
      {
        $$ = $1;
      }
;

// ISO/IEC 9899:1999 6.8.1
labeled_statement:
      IDENTIFIER ':' statement
      {
        $$ = new gennode(PR_LABELED_STATEMENT_1, $1, $2, $3);
        }
    | CASE constant_expression ':' statement
      {
        $$ = new gennode(PR_LABELED_STATEMENT_2, $1, $2, $3, $4);
      }
    | DEFAULT ':' statement
      {
        $$ = new gennode(PR_LABELED_STATEMENT_3, $1, $2, $3);
      }
;

// ISO/IEC 9899:1999 6.8.2
//
// see notes in function_definition rule regarding the creation of
// the collections
compound_statement:
      '{' '}'
      {
        $$ = new gennode(PR_COMPOUND_STATEMENT_EMPTY, $1, $2);
      }
    | '{' block_item_list '}'
      {
        $$ = new gennode(PR_COMPOUND_STATEMENT_FULL, $1, $2, $3);
      }
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
;

// ISO/IEC 9899:1999 6.8.2
block_item_list:
    {
      if(!buffer_scanning_enabled)
      // 1st level block_item_list: this is a hack for scanning
      // in-memory buffers.
      // see the comments in treeproc.c
      {
        // The typenames collection is always created in this step
        active_typenames_collection = new typenames_collection();

        // However, the declarations collection may have also been
        // created in the function definition
        // rule. In this case, the active_declarations_collection
        // pointer is simply updated
        // to point to the already created copy.
        // If a function definition hasn't immediately preceded the
        // block_item_list, the collection
        // is created from scratch.
        if(function_declarations_collection != NULL)
        {
          active_declarations_collection =
function_declarations_collection;
          function_declarations_collection = NULL;
        }
        else
          active_declarations_collection = new
declarations_collection();
      }
      else
        buffer_scanning_enabled = 0;

      // the genblock has to be created in this mid-action rule so
      // the collections will be used in the block_item subtree.
      // The genblock is initially created with no parameters in the
      // vector. Parameters are added in the end action
      $$ = new genblock(PR_BLOCK_ITEM_LIST,
active_typenames_collection, active_declarations_collection);
```

```
    }
    block_item
    {
      $$ = ((gennode *)$1)->add_parameter($2);
    }
  | block_item_list
    {
      // reset the collection pointers to the block collections
      // since anything following will need to use the active
      // collections
      // see translation unit rule for comments
      active_typenames_collection = ((genblock *)$1)
        ->p_typenames_collection;
      active_declarations_collection = ((genblock *)$1)
        ->p_declarations_collection;
    }
    block_item
    {
      // flatten the block_item_list tree (see translation unit rule)
      $$ = ((gennode *)$1)->add_parameter($3);

    }
;

// ISO/IEC 9899:1999 6.8.2
block_item:
    declaration
    {
      $$ = $1;
    }
  | statement
    {
      $$ = $1;
    }
  | openmp_directive // OpenMP Version 2.5 ISvO/IEC 9899:1999
                     // addition
    {
      $$ = $1;
    }
;
```

```
// ISO/IEC 9899:1999 6.8.3
expression_statement:
    ';'
    {
      $$ = $1;
    }
  | expression ';'
    {
      int declaration_errors = 0;

      $$ = new gennode(PR_EXPRESSION_STATEMENT, $1, $2);

      if(create_tree_only != PF_FAKE_FUNCTION)
      {
        // update the genidents in the subtree starting from $$
        // to use the current declarations_collection node
        if((declaration_errors = $$->update_genident_pointers()) > 0)
        {
          fprintf(stderr, "expression_statement: %d errors
encountered\n", declaration_errors);
          return(1);
        }
      }
    }
;

// ISO/IEC 9899:1999 6.8.4
selection_statement:
    IF '(' expression ')' statement
    {
      $$ = new gennode(PR_SELECTION_STATEMENT_IF, $1, $2, $3, $4,
$5);
    }
  | IF '(' expression ')' statement ELSE statement
    {
      $$ = new gennode(PR_SELECTION_STATEMENT_IFELSE, $1, $2, $3, $4,
$5, $6, $7);
    }
  | SWITCH '(' expression ')' statement
```

```
    {
      $$ = new gennode(PR_SELECTION_STATEMENT_SWITCH, $1, $2, $3, $4,
$5);
    }
;

// ISO/IEC 9899:1999 6.8.5
iteration_statement:
    WHILE '(' expression ')' statement
    {
      $$ = new gennode(PR_ITERATION_STATEMENT_1, $1, $2, $3, $4, $5);
    }
  | DO statement WHILE '(' expression ')' ';'
    {
      $$ = new gennode(PR_ITERATION_STATEMENT_2, $1, $2, $3, $4, $5,
$6, $7);
    }
  | FOR '(' ';' ';' ')' statement
    {
      $$ = new gennode(PR_ITERATION_STATEMENT_3, $1, $2, $3, $4, $5,
$6);
    }
  | FOR '(' expression ';' ';' ')' statement
    {
      $$ = new gennode(PR_ITERATION_STATEMENT_4, $1, $2, $3, $4, $5,
$6, $7);
    }
  | FOR '(' ';' expression ';' ')' statement
    {
      $$ = new gennode(PR_ITERATION_STATEMENT_5, $1, $2, $3, $4, $5,
$6, $7);
    }
  | FOR '(' ';' ';' expression ')' statement
    {
      $$ = new gennode(PR_ITERATION_STATEMENT_6, $1, $2, $3, $4, $5,
$6, $7);
    }
  | FOR '(' expression ';' expression ';' ')' statement
    {
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
        $$ = new gennode(PR_ITERATION_STATEMENT_7, $1, $2, $3, $4, $5,
$6, $7, $8);
      }
    | FOR '(' expression ';' ';' expression ')' statement
      {
        $$ = new gennode(PR_ITERATION_STATEMENT_8, $1, $2, $3, $4, $5,
$6, $7, $8);
      }
    | FOR '(' ';' expression ';' expression ')' statement
      {
        $$ = new gennode(PR_ITERATION_STATEMENT_9, $1, $2, $3, $4, $5,
$6, $7, $8);
      }
    | FOR '(' expression ';' expression ';' expression ')' statement
      {
        // NB: assuming this is the only valid iteration rule for the
        // OMP_FOR
        // and OMP_PARALLEL_FOR directives
        $$ = new gennode(PR_ITERATION_STATEMENT_OMP_FOR, $1, $2, $3,
$4, $5, $6, $7, $8, $9);
      }
    | FOR '(' declaration ';' ')' statement
      {
        $$ = new gennode(PR_ITERATION_STATEMENT_11, $1, $2, $3, $4, $5,
$6);
      }
    | FOR '(' declaration expression ';' ')' statement
      {
        $$ = new gennode(PR_ITERATION_STATEMENT_12, $1, $2, $3, $4, $5,
$6, $7);
      }
    | FOR '(' declaration ';' expression ')' statement
      {
        $$ = new gennode(PR_ITERATION_STATEMENT_13, $1, $2, $3, $4, $5,
$6, $7);
      }
    | FOR '(' declaration expression ';' expression ')' statement
      {
        $$ = new gennode(PR_ITERATION_STATEMENT_14, $1, $2, $3, $4, $5,
$6, $7, $8);
```

```
    }
;

// ISO/IEC 9899:1999 6.8.6
jump_statement:
    GOTO IDENTIFIER ';'
    {
      $$ = new gendecl(PR_JUMP_STATEMENT_1, $1, $2, $3);
      active_declarations_collection->add($2);
    }
  | CONTINUE ';'
    {
      $$ = new gennode(PR_JUMP_STATEMENT_2, $1, $2);
    }
  | BREAK ';'
    {
      $$ = new gennode(PR_JUMP_STATEMENT_3, $1, $2);
    }
  | RETURN ';'
    {
      $$ = new gennode(PR_JUMP_STATEMENT_4, $1, $2);
    }
  | RETURN expression ';'
    {
      $$ = new gennode(PR_JUMP_STATEMENT_5, $1, $2, $3);
    }
;


// -------------------------------------------------------------------
// ------ ISO/IEC 9899:1999 A.2.4 External definitions --------------
// -------------------------------------------------------------------

// ISO/IEC 9899:1999 6.9
translation_unit:
    {
      if(create_tree_only == PF_INITIAL_PARSING)
      {
        active_typenames_collection = new typenames_collection();
```

```
          active_typenames_collection->add(new
genident("__builtin_va_list"));

          active_declarations_collection = new
declarations_collection();
        }
        else
        {
          buffer_scanning_enabled = 1;

          // add a level of parallelism for subsequent parsers
          parallel_level = reparsing_parallel_level;
        }

        // create a new typename collection in the mid-rule action so
        // it can be
        // populated in the external_declaration rule
        $$ = new genblock(PR_TRANSLATION_UNIT,
          active_typenames_collection, active_declarations_collection);
      }
      external_declaration
      {
        // update the root object
        $$ = ((gennode *)$1)->add_parameter($2);

        // keep an object pointer to the root object
        // (there must be a smarter way to do this!)
        if(create_tree_only == PF_INITIAL_PARSING)
        {
          p_start = $$;

          // if flag is set, update the program_start_element counter
          // to the current number of parameters
          // this will later be used to add declarations to the tree
          // for ompi
          if(program_start)
          {
            program_start_element = ((gennode *)$$)->numparams - 1;
            // initial value for inserting elements
            program_put_element = program_start_element;
```

```
            program_start = 0;
          }
        }
        else
          t_start = $$;

      }
    | translation_unit
      {
        // reset the collection pointers to the original collections
        // since anything following is either a new global declaration
        // or a function definition.
        // This has to be a mid-action rule so the following block will
        // use the correct collections.

        active_typenames_collection = ((genblock *)$1)
          ->p_typenames_collection;
        active_declarations_collection = ((genblock *)$1)
          ->p_declarations_collection;
      }
      external_declaration
      {
        // add the external_declaration as a new node in the
        // translation_unit
        // vector (translation_unit tree flattening)
        $$ = ((gennode *)$1)->add_parameter($3);

        // if flag is set, update the program_start_element counter
        // to the current number of parameters
        // this will later be used to add declarations to the tree for
ompi
        if((create_tree_only == PF_INITIAL_PARSING) && program_start)
        {
          program_start_element = ((gennode *)$$)->numparams - 1;
          // initial value for inserting elements
          program_put_element = program_start_element;
          program_start = 0;
        }
      }
;
```

```
// ISO/IEC 9899:1999 6.9
external_declaration:
    function_definition
    {
      $$ = $1;
    }
  | declaration
    {
      $$ = $1;
    }
;

// ISO/IEC 9899:1999 6.9.1
//
// The typename and declarations collections are inserted just before
// the compound statement rule is called. This way, the function
// identifier can be treated as a variable in the parent scope and
// the function parameters
// as local variables in the function.
function_definition:
    declarator // NON-ISO function definition without a return type,
               // defaults to "int" (Spyros Melissovas)
    {
      if(create_tree_only != PF_FAKE_FUNCTION)
      {
        int numvars;

        //
        // get the function identifier and add it to the preceding
        // declarations collection
        if(numvars = active_declarations_collection
          ->traverse_and_add_function_ident($1))
        {
          fprintf(stderr, "%s: error: %d variable%s redefined\n",
            orig_fname, numvars, (numvars - 1)?"s":"");
          exit(1);
        }

        // update the active_declarations_collection pointer
```

```
          function_declarations_collection = new
            declarations_collection();

          // add the function arguments to the function's declaration
collection
         numvars = function_declarations_collection
           ->traverse_and_add_function_params($1);

         if(debugflag)
           fprintf(stderr, "%s: %d function parameters added\n",
orig_fname, numvars);
       }
     }
     compound_statement
     {
       // also add the implied "int" return type
       $$ = new gennode(PR_FUNCTION_DEFINITION_NORETURNTYPE, new
genterm("int"), $1, $3);
     }
   | declaration_specifiers declarator
     {
       if(create_tree_only != PF_FAKE_FUNCTION)
       {
         int numvars;

         //
         // get the function identifier and add it to the preceding
         // declarations collection
         if(numvars = active_declarations_collection-
>traverse_and_add_function_ident($2))
         {
           fprintf(stderr, "%s: error: %d variable%s redefined\n",
orig_fname, numvars, (numvars - 1)?"s":"");
           exit(1);
         }

         // update the active_declarations_collection pointer
         function_declarations_collection          =           new
declarations_collection();
```

```
        // add the function arguments to the function's declaration
        // collection
        numvars = function_declarations_collection-
>traverse_and_add_function_params($2);

        if(debugflag)
          fprintf(stderr, "%s: %d function parameters added\n",
orig_fname, numvars);
      }
    }
    compound_statement
    {
      $$ = new gennode(PR_FUNCTION_DEFINITION_NORMAL, $1, $2, $4);
    }
  | declaration_specifiers declarator declaration_list
    {
      if(create_tree_only != PF_FAKE_FUNCTION)
      {
        int numvars;

        //
        // get the function identifier and add it to the preceding
        // declarations collection
        if(numvars = active_declarations_collection-
>traverse_and_add_function_ident($2))
        {
          fprintf(stderr, "%s: error: %d variable%s redefined\n",
orig_fname, numvars, (numvars - 1)?"s":"");
          exit(1);
        }

        // update the active_declarations_collection pointer
        function_declarations_collection = new
declarations_collection();

        // add the function arguments to the function's declaration
        // collection
        numvars = function_declarations_collection-
>traverse_and_add_function_params($2);
        if(debugflag)
```

```
            fprintf(stderr, "%s: %d function parameters added\n",
orig_fname, numvars);
        }
      }
      compound_statement
      {
        $$ = new gennode(PR_FUNCTION_DEFINITION_OLDSTYLE, $1, $2, $3,
$5);
      }
;

// ISQ/IEC 9899:1999 6.9.1
declaration_list:
      declaration
      {
        $$ = new gennode(PR_DECLARATION_LIST, $1);
      }
    | declaration_list declaration
      {
        // flatten the declaration_list tree (see translation unit
        // rule)
        $$ = ((gennode *)$1)->add_parameter($2);
      }
;


// ----------------------------------------------------------------------
// --- OpenMP Version 2.5 ISO/IEC 9899:1999 additions begin ---------
// ----------------------------------------------------------------------
openmp_construct:
      parallel_construct
      {
        $$ = $1;
      }
    | for_construct
      {
        $$ = $1;
      }
    | sections_construct
      {
```

```
      $$ = $1;
    }
  | single_construct
    {
      $$ = $1;
    }
  | parallel_for_construct
    {
      $$ = $1;
    }
  | parallel_sections_construct
    {
      $$ = $1;
    }
  | master_construct
    {
      $$ = $1;
    }
  | critical_construct
    {
      $$ = $1;
    }
  | atomic_construct
    {
      $$ = $1;
    }
  | ordered_construct
    {
      $$ = $1;
    }
;

openmp_directive:
    pomp_construct
    {
      $$ = $1;
    }
  | barrier_directive
    {
      $$ = $1;
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
    }
  | flush_directive
    {
      $$ = $1;
    }
;

structured_block:
    statement
    {
      $$ = $1;
    }
;

parallel_construct:
    parallel_directive { parallel_level++; } structured_block
    {
      parallel_level--;
      $$ = new genomp(PR_PARALLEL_CONSTRUCT, parallel_level, $1, $3);
    }
;

parallel_directive:
    PRAGMA_OMP OMP_PARALLEL parallel_clause_optseq '\n'
    {
      $$ = new gennode(PR_PARALLEL_DIRECTIVE, $1, $2, $3, new
genterm("\n"));
    }
;

parallel_clause_optseq:
    // empty
    {
      $$ = NULL;
    }
  | parallel_clause_optseq parallel_clause
    {
      // parallel_clause_optseq tree flattening
      // see translation_unit rule
      if($1 == NULL)
```

```
          $$ = new gennode(PR_PARALLEL_CLAUSE_OPTSEQ, $2);
        else
          ((gennode *)$1)->add_parameter($2);
      }
    | parallel_clause_optseq ',' parallel_clause
      {
        // parallel_clause_optseq tree flattening
        // see translation_unit rule
        if($1 == NULL)
          $$ = new gennode(PR_PARALLEL_CLAUSE_OPTSEQ, $2, $3);
        else
          ((gennode *)$1)->add_parameter($2, $3);
      }
;

parallel_clause:
    unique_parallel_clause
    {
      $$ = $1;
    }
  | data_clause
    {
      $$ = $1;
    }
;

unique_parallel_clause:
    OMP_IF '(' expression ')'
    {
      $3->update_genident_pointers();
      $$ = new gennode(PR_UNIQUE_PARALLEL_CLAUSE_IF, $1, $2, $3, $4);
    }
  | OMP_NUMTHREADS '(' expression ')'
    {
      $$ = new gennode(PR_UNIQUE_PARALLEL_CLAUSE_NUMTHREADS, $1, $2,
$3, $4);
    }
;

for_construct:
```

```
    for_directive { parallel_level++; } iteration_statement
    {
      parallel_level--;
      $$ = new genomp(PR_FOR_CONSTRUCT, parallel_level, $1, $3);
    }
;

for_directive:
    PRAGMA_OMP OMP_FOR for_clause_optseq '\n'
    {
      $$ = new gennode(PR_FOR_DIRECTIVE, $1, $2, $3, new
genterm("\n"));
    }
;

for_clause_optseq:
    // empty
    {
      $$ = NULL;
    }
  | for_clause_optseq for_clause
    {
      // tree flattening
      // see translation_unit rule
      if($1 == NULL)
        $$ = new gennode(PR_FOR_CLAUSE_OPTSEQ, $2);
      else
        ((gennode *)$1)->add_parameter($2);
    }
  | for_clause_optseq ',' for_clause
    {
      // tree flattening
      // see translation_unit rule
      if($1 == NULL)
        $$ = new gennode(PR_FOR_CLAUSE_OPTSEQ, $2, $3);
      else
        ((gennode *)$1)->add_parameter($2, $3);
    }
;
```

```
for_clause:
     unique_for_clause
     {
       $$ = $1;
     }
   | data_clause
     {
       $$ = $1;
     }
   | OMP_NOWAIT
     {
       $$ = $1;
     }
;

unique_for_clause:
     OMP_ORDERED
     {
       $$ = $1;
     }
   | OMP_SCHEDULE '(' schedule_kind ')'
     {
       $$ = new gennode(PR_UNIQUE_FOR_CLAUSE_SCHEDULE, $1, $2, $3,
$4);
     }
   | OMP_SCHEDULE '(' schedule_kind ',' expression ')'
     {
       $$ = new gennode(PR_UNIQUE_FOR_CLAUSE_SCHEDULE_CHUNKSIZE, $1,
$2, $3, $4, $5, $6);
     }
;

schedule_kind:
     OMP_STATIC
     {
       $$ = $1;
     }
   | OMP_DYNAMIC
     {
       $$ = $1;
```

```
        }
      | OMP_GUIDED
        {
          $$ = $1;
        }
      | OMP_RUNTIME
        {
          $$ = $1;
        }
    ;

sections_construct:
        sections_directive { parallel_level++; } section_scope
        {
          parallel_level--;
          $$ = new genomp(PR_SECTIONS_CONSTRUCT, parallel_level, $1, $3);
        }
    ;

sections_directive:
        PRAGMA_OMP OMP_SECTIONS sections_clause_optseq '\n'
        {
          $$ = new gennode(PR_SECTIONS_DIRECTIVE, $1, $2, $3, new
genterm("\n"));
        }
    ;

sections_clause_optseq:
        // empty
        {
          $$ = NULL;
        }
      | sections_clause_optseq sections_clause
        {
          // tree flattening
          // see translation_unit rule
          if($1 == NULL)
            $$ = new gennode(PR_SECTIONS_CLAUSE_OPTSEQ, $2);
          else
            ((gennode *)$1)->add_parameter($2);
```

```
        }
      | sections_clause_optseq ',' sections_clause
        {
          // tree flattening
          // see translation_unit rule
          if($1 == NULL)
            $$ = new gennode(PR_SECTIONS_CLAUSE_OPTSEQ, $2, $3);
          else
            ((gennode *)$1)->add_parameter($2, $3);
        }
  ;

  sections_clause:
        data_clause
        {
          $$ = $1;
        }
      | OMP_NOWAIT
        {
          $$ = $1;
        }
  ;

  section_scope:
        '{' section_sequence '}'
        {
          $$ = new gennode(PR_SECTION_SCOPE, $1, $2, $3);
        }
  ;

  section_sequence:
        { parallel_level++; } structured_block  // 1 shift/reduce
                                                // conflict here
        {
          parallel_level--;
          $$ = new gennode(PR_SECTION_SEQUENCE, $2);
        }
      | section_directive { parallel_level++; } structured_block
        {
          parallel_level--;
```

```
        $$ = new gennode(PR_SECTION_SEQUENCE, $1, $3);
      }
    | section_sequence section_directive { parallel_level++; }
structured_block
      {
        parallel_level--;
        // add new params to the already created node in previous rules
        ((gennode *)$1)->add_parameter($2, $4);
      }
;

section_directive:
      PRAGMA_OMP OMP_SECTION '\n'
      {
        $$ = new gennode(PR_SECTION_DIRECTIVE, $1, $2, new
genterm("\n"));
      }
;

single_construct:
      single_directive { parallel_level++; } structured_block
      {
        parallel_level--;
        $$ = new genomp(PR_SINGLE_CONSTRUCT, parallel_level, $1, $3);
      }
;

single_directive:
      PRAGMA_OMP OMP_SINGLE single_clause_optseq '\n'
      {
        $$ = new gennode(PR_SINGLE_DIRECTIVE, $1, $2, $3, new
genterm("\n"));
      }
;

single_clause_optseq:
      // empty
      {
        $$ = NULL;
      }
```

```
    | single_clause_optseq single_clause
      {
        // tree flattening
        // see translation_unit rule
        if($1 == NULL)
          $$ = new gennode(PR_SINGLE_CLAUSE_OPTSEQ, $2);
        else
          ((gennode *)$1)->add_parameter($2);
      }
    | single_clause_optseq ',' single_clause
      {
        // tree flattening
        // see translation_unit rule
        if($1 == NULL)
          $$ = new gennode(PR_SINGLE_CLAUSE_OPTSEQ, $2, $3);
        else
          ((gennode *)$1)->add_parameter($2, $3);
      }
;

single_clause:
    data_clause
    {
      $$ = $1;
    }
  | OMP_NOWAIT
    {
      $$ = $1;
    }
;

parallel_for_construct:
    parallel_for_directive { parallel_level++; } iteration_statement
    {
      parallel_level--;
      $$ = new genomp(PR_PARALLEL_FOR_CONSTRUCT, parallel_level, $1,
$3);
    }
;
```

```
parallel_for_directive:
    PRAGMA_OMP OMP_PARALLEL OMP_FOR parallel_for_clause_optseq '\n'
    {
      $$ = new gennode(PR_PARALLEL_FOR_DIRECTIVE, $1, $2, $3, $4, new
genterm("\n"));
    }
;

parallel_for_clause_optseq:
    // empty
    {
     $$ = NULL;
    }
  | parallel_for_clause_optseq parallel_for_clause
    {
      // tree flattening
      // see translation_unit rule
      if($1 == NULL)
        $$ = new gennode(PR_PARALLEL_FOR_CLAUSE_OPTSEQ, $2);
      else
        ((gennode *)$1)->add_parameter($2);
    }
  | parallel_for_clause_optseq ',' parallel_for_clause
    {
      // tree flattening
      // see translation_unit rule
      if($1 == NULL)
        $$ = new gennode(PR_PARALLEL_FOR_CLAUSE_OPTSEQ, $2, $3);
      else
        ((gennode *)$1)->add_parameter($2, $3);
    }
;

parallel_for_clause:
    unique_parallel_clause
    {
      $$ = $1;
    }
  | unique_for_clause
    {
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
        $$ = $1;
      }
    | data_clause
      {
        $$ = $1;
      }
;

parallel_sections_construct:
    parallel_sections_directive { parallel_level++; } section_scope
    {
      parallel_level--;
      $$ = new genomp(PR_PARALLEL_SECTIONS_CONSTRUCT, parallel_level,
$1, $3);
    }
;

parallel_sections_directive:
    PRAGMA_OMP OMP_PARALLEL OMP_SECTIONS
parallel_sections_clause_optseq '\n'
    {
      $$ = new gennode(PR_PARALLEL_SECTIONS_DIRECTIVE, $1, $2, $3,
$4, new genterm("\n"));
    }
;

parallel_sections_clause_optseq:
    // empty
    {
      $$ = NULL;
    }
  | parallel_sections_clause_optseq parallel_sections_clause
    {
      // tree flattening
      // see translation_unit rule
      if($1 == NULL)
        $$ = new gennode(PR_PARALLEL_SECTIONS_CLAUSE_OPTSEQ, $2);
      else
        ((gennode *)$1)->add_parameter($2);
    }
```

```
   | parallel_sections_clause_optseq ',' parallel_sections_clause
     {
       // tree flattening
       // see translation_unit rule
       if($1 == NULL)
         $$ = new gennode(PR_PARALLEL_SECTIONS_CLAUSE_OPTSEQ, $2, $3);
       else
         ((gennode *)$1)->add_parameter($2, $3);
     }
;

parallel_sections_clause:
    unique_parallel_clause
    {
      $$ = $1;
    }
   | data_clause
    {
      $$ = $1;
    }
;

master_construct:
    master_directive { parallel_level++; } structured_block
    {
      parallel_level--;
      $$ = new genomp(PR_MASTER_CONSTRUCT, parallel_level, $1, $3);
    }
;

master_directive:
    PRAGMA_OMP OMP_MASTER '\n'
    {
      $$ = new gennode(PR_MASTER_DIRECTIVE, $1, $2, new
genterm("\n"));
    }
;

critical_construct:
    critical_directive { parallel_level++; } structured_block
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
    {
      parallel_level--;
      $$ = new genomp(PR_CRITICAL_CONSTRUCT, parallel_level, $1, $3);
    }
;

critical_directive:
    PRAGMA_OMP OMP_CRITICAL '\n'
    {
      $$ = new gennode(PR_CRITICAL_DIRECTIVE, $1, $2, new
genterm("\n"));
    }
  | PRAGMA_OMP OMP_CRITICAL region_phrase '\n'
    {
      $$ = new gennode(PR_CRITICAL_DIRECTIVE_REGION_PHRASE, $1, $2,
$3, new genterm("\n"));
    }
;

region_phrase:
    '(' IDENTIFIER ')'
    {
      $$ = new gennode(PR_REGION_PHRASE, $1, $2, $3);
    }
;

barrier_directive:
    PRAGMA_OMP OMP_BARRIER '\n'
    {
      $$ = new genomp(PR_BARRIER_DIRECTIVE, parallel_level, $1, $2,
new genterm("\n"));
    }
;

atomic_construct:
    atomic_directive expression_statement
    {
      $$ = new genomp(PR_ATOMIC_CONSTRUCT, parallel_level, $1, $2);
    }
;
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
atomic_directive:
     PRAGMA_OMP OMP_ATOMIC '\n'
     {
       $$ = new gennode(PR_ATOMIC_DIRECTIVE, $1, $2, new
genterm("\n"));
     }
;

flush_directive:
     PRAGMA_OMP OMP_FLUSH '\n'
     {
       $$ = new genomp(PR_FLUSH_DIRECTIVE_1, parallel_level, $1, $2,
new genterm("\n"));
     }
   | PRAGMA_OMP OMP_FLUSH flush_vars '\n'
     {
       $$ = new genomp(PR_FLUSH_DIRECTIVE_2, parallel_level, $1, $2,
$3, new genterm("\n"));
     }
;

flush_vars:
     '(' variable_list ')'
     {
       $$ = new gennode(PR_FLUSH_VARS_OPT, $1, $2, $3);
     }
;

ordered_construct:
     ordered_directive { parallel_level++; } structured_block
     {
       parallel_level--;
       $$ = new genomp(PR_ORDERED_CONSTRUCT, parallel_level, $1, $3);
     }
;

ordered_directive:
     PRAGMA_OMP OMP_ORDERED '\n'
     {
```

```
        $$ = new genomp(PR_ORDERED_DIRECTIVE, parallel_level, $1, $2,
new genterm("\n"));
    }
;

threadprivate_directive:
    PRAGMA_OMP_THREADPRIVATE '(' variable_list ')' '\n'
    {
      $$ = new genomp(PR_THREADPRIVATE_DIRECTIVE, parallel_level, $1,
$2, $3, $4, new genterm("\n"));
    }
;

data_clause:
    OMP_PRIVATE '(' variable_list ')'
    {
      $$ = new gennode(PR_DATA_CLAUSE_PRIVATE, $1, $2, $3, $4);
    }
  | OMP_COPYPRIVATE '(' variable_list ')'
    {
      $$ = new gennode(PR_DATA_CLAUSE_COPYPRIVATE, $1, $2, $3, $4);
    }
  | OMP_FIRSTPRIVATE '(' variable_list ')'
    {
      $$ = new gennode(PR_DATA_CLAUSE_FIRSTPRIVATE, $1, $2, $3, $4);
    }
  | OMP_LASTPRIVATE '(' variable_list ')'
    {
      $$ = new gennode(PR_DATA_CLAUSE_LASTPRIVATE, $1, $2, $3, $4);
    }
  | OMP_SHARED '(' variable_list ')'
    {
      $$ = new gennode(PR_DATA_CLAUSE_SHARED, $1, $2, $3, $4);
    }
  | OMP_DEFAULT '(' OMP_SHARED ')'
    {
      $$ = new gennode(PR_DATA_CLAUSE_DEFAULTSHARED, $1, $2, $3, $4);
    }
  | OMP_DEFAULT '(' OMP_NONE ')'
    {
```

```
        $$ = new gennode(PR_DATA_CLAUSE_DEFAULTNONE, $1, $2, $3, $4);
      }
    | OMP_REDUCTION '(' reduction_operator ':' variable_list ')'
      {
        $$ = new gennode(PR_DATA_CLAUSE_REDUCTION, $1, $2, $3, $4, $5,
$6);
      }
    | OMP_COPYIN '(' variable_list ')'
      {
        $$ = new gennode(PR_DATA_CLAUSE_COPYIN, $1, $2, $3, $4);
      }
;

reduction_operator:
      '+'
      {
        $$ = $1;
      }
    | '*'
      {
        $$ = $1;
      }
    | '-'
      {
        $$ = $1;
      }
    | '&'
      {
        $$ = $1;
      }
    | '^'
      {
        $$ = $1;
      }
    | '|'
      {
        $$ = $1;
      }
    | AND_OP
      {
```

```
        $$ = $1;
      }
    | OR_OP
      {
        $$ = $1;
      }
;

variable_list:
      IDENTIFIER
      {
        $$ = new gennode(PR_VARIABLE_LIST, $1);
        $$->update_genident_pointers();
      }
    | variable_list ',' IDENTIFIER
      {
        // flatten the variable_list tree (see translation unit rule)
        $$ = ((gennode *)$1)->add_parameter($2, $3);
        $3->update_genident_pointers();

      }
;
// ------------------------------------------------------------------
// OpenMP Version 2.5 ISO/IEC 9899:1999 additions end --------------
// ------------------------------------------------------------------




// ------------------------------------------------------------------
// --------------- Additional POMP construct type: ------------------
// ------------------------------------------------------------------
pomp_construct:
      PRAGMA_OMP POMP_INST POMP_INIT '\n'
      {
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_INIT, $1, $2, $3, new
genterm("\n"));
      }
    | PRAGMA_OMP POMP_INST POMP_FINALIZE '\n'
      {
```

```
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_FINALIZE, $1, $2, $3,
new genterm("\n"));
      }
    | PRAGMA_OMP POMP_INST POMP_ON '\n'
      {
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_ON, $1, $2, $3, new
genterm("\n"));
      }
    | PRAGMA_OMP POMP_INST POMP_OFF '\n'
      {
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_OFF, $1, $2, $3, new
genterm("\n"));
      }
    | PRAGMA_OMP POMP_INST POMP_BEGIN '\n'
// region_phrase_opt is the same [optional name] in omp critical or
// user regions
      {
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_BEGIN, $1, $2, $3, new
genterm("\n"));
      }
    | PRAGMA_OMP POMP_INST POMP_BEGIN region_phrase '\n'
// region_phrase_opt is the same [optional name] in omp critical or
// user regions
      {
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_BEGIN_REGION_PHRASE,
$1, $2, $3, $4, new genterm("\n"));
      }
    | PRAGMA_OMP POMP_INST POMP_END '\n'
      {
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_END, $1, $2, $3, new
genterm("\n"));
      }
    | PRAGMA_OMP POMP_INST POMP_END region_phrase '\n'
      {
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_END_REGION_PHRASE, $1,
$2, $3, $4, new genterm("\n"));
      }
    | PRAGMA_OMP POMP_NOINSTRUMENT '\n'
      {
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_NOINSTRUMENT, $1, $2,
new genterm("\n"));
      }
   | PRAGMA_OMP POMP_INSTRUMENT '\n'
      {
        $$ = new gennode(PR_POMP_CONSTRUCT_POMP_INSTRUMENT, $1, $2, new
genterm("\n"));
      }
;

%%


// Called by yyparse on error
//
// yyerror normally takes only a char* argument. After delcaring a
// parameter to yyparse,
// the same parameter has to be declared to yyerror too.
void yyerror (int create_tree_only, char *s)
{
  fprintf(stderr, "%s: error: %s (%s: %s) in line %d,%d\n",
orig_fname, s, class_names[yylval->classname], yylval->value, line_no
- new_line + orig_line, column);
}




int main(int argc, char *argv[])
{
  FILE *dumpfile;
  int parse_ret = 0;

  if(argc > 1)
  {
    if(!strcmp(argv[1], "-v"))  // verbose
    {
      debugflag = 1;
    }
    else
```

```
        if(!strcmp(argv[1], "-vv")) // even more verbose
        {
          debugflag = 1;
          yydebug = 1;
        }
    }


    switch(argc)
    {
        case 2:
        if((yyin = fopen(argv[1], "r")) == NULL)
        {
          fprintf(stderr, "Error opening file %s for reading!\n",
argv[1]);
          exit(1);
        }
        break;

        case 3:
        if((yyin = fopen(argv[2], "r")) == NULL)
        {
          fprintf(stderr, "Error opening file %s for reading!\n",
argv[2]);
          exit(1);
        }
        break;

      default:
        break;
        }

    // NB: parameter passed to yyparse
    parse_ret = yyparse(0);

    fclose(yyin);

    if(parse_ret == 0) // parsing successful
    {
      assert(p_start != NULL);
```

```
      process_tree(p_start);

      if(SRCOUTFILE != stdout)
      {
        if((dumpfile = fopen("source.c", "wt")) != NULL)
        {
          fprintf(stderr, "Dumping source code to \"source.c\"\n");
          p_start->source(dumpfile);
          fclose(dumpfile);
        }
        else
          fprintf(stderr, "Error opening %s for writing!\n",
SRCOUTFILE);
      }
      else
        p_start->source(stdout);

      if(debugflag)
        p_start->traverse();

      if((dumpfile = fopen(XMLOUTFILE, "wt")) != NULL)
      {
        fprintf(stderr, "Dumping parse tree in xml format to file
%s\n", XMLOUTFILE);
        p_start->dumpxml(dumpfile);
        fclose(dumpfile);
      }
      else
        fprintf(stderr, "Error opening %s for writing!\n", XMLOUTFILE);

      fflush(stdout);

      // recover allocated memory
      delete p_start;

      fprintf(stderr, "parser ended successfully\n");
    }
    else
      // yynerrs is declared globally by the parser and contains the
```

```
    // number of errors encountered. Since this parser does not
    // implement error recovery the value should be 1 at this point
    fprintf(stderr, "%d error%s encountered\n", yynerrs, (yynerrs -
1)?"s":"");

 return 0;

}
```

ΠΑΡΑΡΤΗΜΑ Β

Οι κανόνες στους οποίους εφαρμόζεται η τεχνική απλοποίησης της δενδρικής δομής είναι οι:

- argument_expression_list
- init_declarator_list
- struct_declaration_list
- specifier_qualifier_list
- struct_declarator_list
- enumerator_list
- type_qualifier_list
- parameter_list
- identifier_list
- designator_list
- block_item_list
- translation_unit
- declaration_list
- parallel_clause_optseq
- for_clause_optseq
- sections_clause_optseq
- single_clause_optseq
- parallel_for_clause_optseq
- parallel_sections_clause_optseq
- variable_list

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Σπύρος Μελισσόβας του Βασιλείου και της Παρασκευής γεννήθηκε στα Ιωάννινα το 1975.
Σπούδασε στο Τμήμα Πληροφορικής του Πανεπιστημίου Ιωαννίνων την περίοδο 1993-1999 από όπου αποφοίτησε με το βαθμό 6.25 «καλώς».
Τα έτη 1999-2000 εργάστηκε στο Τμήμα Πληροφορικής στη θέση «Ειδικός λογισμικού» με καθήκοντα διαχείρισης συστημάτων.
Το έτος 2000 έγινε δεκτός από το Τμήμα Πληροφορικής στο Πρόγραμμα Μεταπτυχιακών Σπουδών.
Από το έτος 2003 εργάζεται στο ίδιο Τμήμα ως Ε.Τ.Ε.Π. με τα καθήκοντα του διαχειριστή συστημάτων.
Στα ενδιαφέροντά του περιλαμβάνονται η περιοχή των ενσύρματων και ασύρματων δικτύων και των λειτουργικών συστημάτων.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ