Χαρακτηριστικά εκδόσεων OpenMP 3.0 - 4.5

Η έκδοση του *OpenMP* που ακολούθησε της 2.5, ήταν η 3.0. Η νέα έκδοση περιείχε σημαντικές προσθήκες και αλλαγές για τα δεδομένα του παράλληλου προγραμματισμού. Χαρακτηριστικά προηγούμενων εκδόσεων αναβαθμίστηκαν, ωστόσο τη σημαντικότερη αλλαγή αποτέλεσε η εισαγωγή της έννοειας των διεργασιών (*Tasking*). Η επόμενη έκδοση του *OpenMP* (4.0) κυκλοφόρησε τον Ιούλιο του 2013 και περιελάμβανε τα παρακάτω νέα χαρακτηριστικά:

- *Threads affinity*,
- Ετερογενείς προγραμματισμός,
- Διαχείριση σφαλμάτων,
- Διανυσματικοποίηση μέσω *SIMD*,
- *User-Defined Recuctions (UDRs)*
- Διεργασίας (*Tasking*

Τα τελευταία χρόνια, η ανάγκη για εφαρμογές κατασκευασμένες με υψηλά επίπεδα παραλληλισμού είναι αυξημένη. Κύρια αιτία αποτελεί η ευκολία πρόσβασης σε μεγάλο όγκο δεδομένων από το ευρύ κοινό, οι μικρές εξαρτήσεις ανάμεσά στα δεδομένα, και η ανάγκη για εντατικούς υπολογισμούς στα δεδομένα αυτά. Λύση στο πρόβλημα αυξημένου φόρτου υπολογισμών αποτελεί η χρήση ετερογενών συστημάτων προγραμματισμού. Ωστόσο, παρά τα οφέλη της υλοποίησης και χρήσης τέτοιων συστημάτων σε ότι αφορά την απόδοση, ο προγραμματισμός εφαρμογών σε τέτοια συστήματα, δρα ανασταλτικά για την ευρεία χρήση τους. Το *OpenMP* δημιούργησε τα κατάλληλα εργαλεία για την εξάλειψη προβλημάτων υλοποίησης. Με τη δημοσίευση της έκδοσης 4.0, προσέφερε την υποδομή για την υποστήριξη ετερογενών συστημάτων, καθώς η έκδοση περιλαμβάνει ένα σύνολο οδηγιών και φράσεων τα οποία χρησιμοποιούνται για τον προσδιορισμό ρουτίνων και δεδομένων, ικανών να μετακινηθούν σε μια συσκευή προορισμού (επιταχυντής), για να υπολογιστούν. Στόχος είναι η αύξηση των επιδόσεων υπολογισμού αλλά και η μείωση της κατανάλωσης ισχύος.

Εκτός από την υποστήριξη ετερογενών συστημάτων, το *OpenMP* την επεξεργασία δεδομένων μέσω διανυσματικοποίησης *SIMD*. Η επεξεργασία *SIMD (Simple Instruction Multiple Data)* εκμεταλλεύεται τον παραλληλισμό σε επίπεδο δεδομένων, πράγμα που σημαίνει ότι οι πράξεις που γίνονται σε ένα σύνολο στοιχείων διανύσματος, γίνονται ταυτόχρονα, μέσω απλών εντολών.

Στις ενότητες του κεφαλαίου που ακολουθούν, εκτός από την αναλυτικότερη περιγραφή των παραπάνω βασικών χαρακτηριστικών, γίνεται περιγραφή της έννοιας του *Thread Affinity*, των *User-Defined Reductions*, και των διεργασιών *(Tasking)*.

Διανυσματικοποίηση μέσω *SIMD*

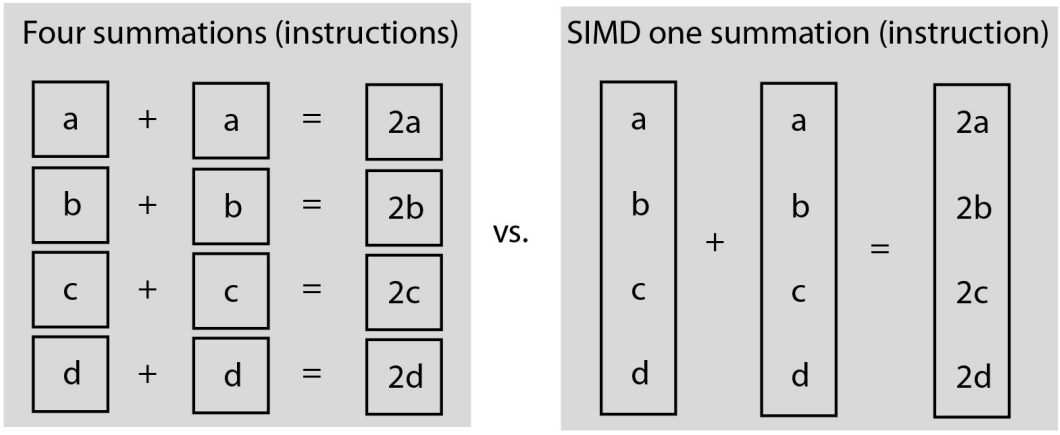
ΝΑ ΔΩ ΤΟ ΚΡΙΣΤΥ ΓΟΥΝΤΣ ΒΕΚΤΟΡΙΖΑΤΙΟΝ !

Κατά την διάρκεια εκτέλεσης ενός τυπικού προγράμματος από έναν μή παράλληλο υπολογιστή, οι εντολές που εκτελούνται είναι απλές, εφαρμοζόμενες σε απλά, μοναδικά δεδομένα. Το μοντέλο αυτό ονομάζεται (*SISD - Single Instruction Single Data*) και αποτελούσε για πολλά χρόνια το επικρατέστερο μοντέλο εκτέλεσης ενός προγράμματος. Ηταν ένα απο τα τέσσερα μοντέλα που αποτελούν την ταξινόμηση *Flynn* που προτάθηκε το 1966[6]. Τα άλλα τρία μοντέλα είναι:

- *SIMD - Single Instruction Multiple Data*
- *MISD - Multiple Instruction Single Data*
- *MIMD - Multiple Instructions Multiple Data*

Στο μοντέλο *Single Instruction Multiple Data (SIMD)*, εκτελούνται απλές διεργασίες για την τροποποίηση ενός συνόλου δεδομένων τοποθετημένων στη σειρά. Το σύνολο αυτών των δεδομένων ονομάζεται διάνυσμα.

Στο παρακάτω σχήμα παρουσιάζεται η πρόσθεση των στοιχείων δυο διανυσμάτων και η εκχώρηση των αποτελεσμάτων σε ένα τρίτο, με τον συμβατικό τρόπο, αλλά και με την χρήση *SIMD* μεθόδου. Το πλεονέκτημα είναι οτι οι *SIMD* οδηγίες εκτελούνται το ίδιο γρήγορα με τις αντίστοιχες βαθμωτές. Με άλλα λόγια η πράξη της παρακάτω εικόνας, θα γίνει 4 φορές που γρήγορα αν χρησιμοποιηθεί *SIMD* οδηγία.



Σχήμα 1: Πρόσθεση διανυσμάτων βαθμωτά και με SIMD

Στο κεφάλαιο αυτό, περιγράφονται οι λειτουργίες που είναι διαθέσιμες στο *OpenMP* και αφορούν λειτουργίες διανυσματικοποίησης μέσω *SIMD*.

Η οδηγία *simd*

Αν ο μεταγλωττιστής δεν εφαρμόζει διανυσματικοποίηση και δεν χρησιμοποιείται κάποια ειδική βιβλιοθήκη, τότε ο καλύτερος τρόπος διανυσμακοποίησης είναι με τη χρήση του *OpenMP*. Η διεπαφή εφοδιάζει με ένα σύνολο οδηγιών και φράσεων που σκοπό έχουν να ενημερώσουν το μεταγλωττιστή ωστε να παραλληλοποιήσει αυτόματα ή να διανυσματικοποιήσει τους βρόγχους που βρίσκονται στον κώδικα.

Βασικότερη οδηγία της διεπαφής αποτελεί η ***simd***. Εφαρμόζεται σε βρόγχους των οποίων η δομή είναι ίδια με την δομή των κοινών βρόγχων της *C++*.

Η εισαγωγή της οδηγίας *simd* δίνει εντολή στο μεταγλωττιστή να δημιουργήσει έναν *simd* βρόγχο. Η χρήση δεικτών μέσα στο βρόγχο μπορεί υπο συνθήκες να προκαλέσει απροσδιόριστη συμπεριφορα. Για παράδειγμα, στο παρακάτω τμήμα κώδικα, αν ο δείκτης *k* ή *m* είναι ταυτόσημος με τον δείκτη t, τότε αναμένονται λάθος αποτελέσματα.

Συμβ. 1: Παράδειγμα κώδικα με simd

```
void accumulate(int *t, int *k, int *m, int n) {
        #pragma omp simd
        for (int i = 0; i < n; ++i) {
                t[i] = k[i] + m[i];
        }
}
```

Στο παραπάνω παράδειγμα, η μεταβλητή *i* είναι ιδιωτική. Η διαφορά με την ιδιωτική μεταβλητή ενός παράλληλου βρόγχου είναι οτι η ιδιωτικότητα αναφέρεται σε ένα *SIMD lane*. Οι τιμές των διανυσμάτων *t, k, m* είναι κοινόχρηστες. Ο μεταγλωττιστής επιλέγει το κατάλληλο μήκος του διανύσματος για τη συγκεκριμένη αρχιτεκτονική.

Φράσεις οδηγίας *simd*

Οι φράσεις *private, lastprivate, reduction, collapse, ordered*, έχουν την ίδια χρησιμότητα που προαναφέρθηκε στα προηγούμενα κεφάλαια (πχ οδηγία διαμοιρασμού βρόγχου). Οι φράσεις που υποστηρίζονται από την οδηγία είναι οι παρακάτω:

Συμβ. 2: Φράσεις που υποστηρίζονται από simd

```
private (list)
lastprivate (list)
reduction (reduction-identifier : list)
collapse (n)
simdlen (length)
safelen (length)
linear (list[: linear-step J)
aligned (list{:alignment])
```

1.1.2.1 Φράση *simdlen*

Η φράση *simdlen* δέχεται ως όρισμα ένα θετικό ακέραιο αριθμό που καθορίζει τον προτιμώμενο αριθμό επαναλήψεων ενός βρόγχου που θα εκτελούνται ταυτόχρονα. Επηρεάζει το μήκος του διανύσματος που χρησιμοποιείται από τις παραγόμενες *simd* οδηγίες.

Η τιμή του ορίσματος είναι προτιμητέα αλλα όχι υποχρεωτική. Ο μεταγλωττιστής έχει την ελευθερία να αποκλίνει από αυτή την επιλογή και να επιλέξει διαφορετικό μήκος. Ελλείψη αυτής της φράσης ορίζεται μια προεπιλεγμένη τιμή που καθορίζεται από την υλοποίηση. Σκοπός της φράσης *simdlen* είναι να καθοδηγήσει τον μεταγλωττιστή. Χρησιμοποιείται από τον χρήστη όταν έχει καλή εικόνα των χαρακτηριστικών του βρόγχου και γνωρίζει ότι κάποιο συγκεκριμένο μήκος μπορεί να ωφελήσει στην απόδοση.

1.1.2.2 Η φράση *safelen*

Η φράση *safelen* δέχεται ως όρισμα ένα θετικό ακέραιο αριθμό. Η τιμή αυτή καθορίζει το ανώτερο όριο του μήκους διανύσματος. Είναι ο αριθμός που στο οποίο είναι ασφαλές για τον βρόγχο. Το τελικό μήκος διανύσματος επιλέγεται από τον μεταγλωττιστή, αλλά δεν υπερβαίνει την τιμή της φράσης *safelen*.

Στο παρακάτω παράδειγμα απαιτείται η φράση *safelen*. Πρόκειται για έναν βρόγχο που περιέχει δέσμευση στην προσπέλαση των στοιχείων του διανύσματος. Για παράδειγμα, το διάβασμα του *[i-10]* στην επανάληψη *i* δεν μπορεί να γίνει, μέχρι να ολοκληρωθεί ή εγγραφή στο *k[i]* της προηγούμενης επανάληψης.

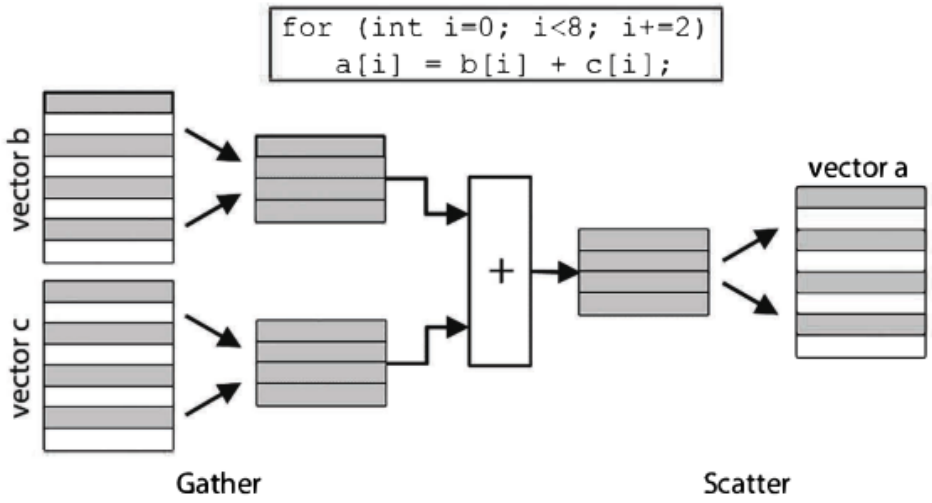
Συμβ. 3: Παράδειγμα κώδικα με simd

```
void dep_loop(float *k, float c, int n) {
        for (int i=10; i<n; i++) {
                k[i] = k[i-10] * c;
        }
}
```

1.1.2.3 Η φράση *linear*

Τα βαθμωτά στοιχεία εισέρχονται σε διανύσματα, στα οποία εκτελούνται οδηγίες *simd* και εξέρχονται. Οταν τα στοιχεία είναι προσβάσιμα με γραμμικό τρόπο, το διάνυσμα κατασκευάζεται εύκολα. Για παράδειγμα, δεδομένου ότι διατηρείται η ευθυγράμμιση των δεδομένων, μια απλή *simd* οδηγία φόρτωσης και αποθήκευσης, χρησιμοποιείται για να γράψει και να διαβάσει διαδοχικά δεδομένα στο διάνυσμα.

Σε περιπτώσεις που τα βαθμωτά παρατίθενται διαδοχικά στη μνήμη, η πρόσβαση στα στοιχεία γίνεται με *μοναδιαίο βήμα*. Αν τα δεδομένα δεν είναι διατεταγμένα σε διαδοχικές θέσεις μνήμης, αλλά με κάποια μετατόπιση μεταξύ τους, τότε η προσέγγιση του μπορεί να γίνει με μεγαλύτερο βήμα. Στην περίπτωση αυτή το βήμα είναι ίσο με τη μετατόπιση μεταξύ των στοιχείων.



Σχήμα 2: Σχηματική απεικόνιση φράσης *linear*

Για την κατασκευή του διανύσματος, μια λειτουργία συγκέντρωσης διαβάζει γραμμικά αλλά με βήμα μεγαλύτερο του ενός. Ομοίως, μια λειτουργία γράφει τα δεδομένα του διανύσματος πίσω στη μνήμη γραμμικά, αλλα με βήμα μεγαλύτερο του ενος.

1.1.2.4 Η φράση *aligned*

Η ευθυγράμμιση των δεδομένων είναι σημαντική για την καλή απόδοση του προγράμματος. Εαν ένα στοιχεία δεν είναι ευθυγραμμισμένο σε διευθυνση μνήμης που είναι πολλαπλάσιο του μεγέθους του στοιχείου σε βψτε, προκύπτει ένα επιπλέον κόστος κατά την πρόσβαση στο στοιχείο αυτό.

Για παράδειγμα, σε ορισμένες αρχιτεκτονικές ενδέχεται να μην είναι δυνατή η φόρτωση και εγγραφή απο μια διεύθυνση μνήμης που δεν ειναι ευθυγραμμισμένη με το μέγεθος του δεδομένου που χρησιμοποιείται. Αν ισχύει κάτι τέτοιο, οι λειτουργίες γίνονται κανονικά, με μεγαλύτερο κόστος. Σε περίπτωση διανυσματοκοποίησης μέσω της χρήσης οδηγίας *simd*, αλλά η προκύπτουσα απόδοση είναι κακή, η προσαρμογή ευθυγράμμισης δεδομένων μπορεί να βελτιώσει την εκτέλεση.

Η φράση ευθυγράμμισης υποστηρίζεται τόσο από την οδηγία *simd* όσο και από την οδηγία *declare simd*. Η φράση δέχεται ως όρισμα μια λίστα μεταβλητών. Η ευθυγράμμιση πρέπει να είναι ενας σταθερός ακεραιος αριθμός. Σε περίπτωση έλλειψης της φράσης, μια προεπιλεγμένη τιμή καθορίζεται από την υλοποίηση.

1.1.2.5 Η σύνθετη οδηγία βρόγχου *SIMD*

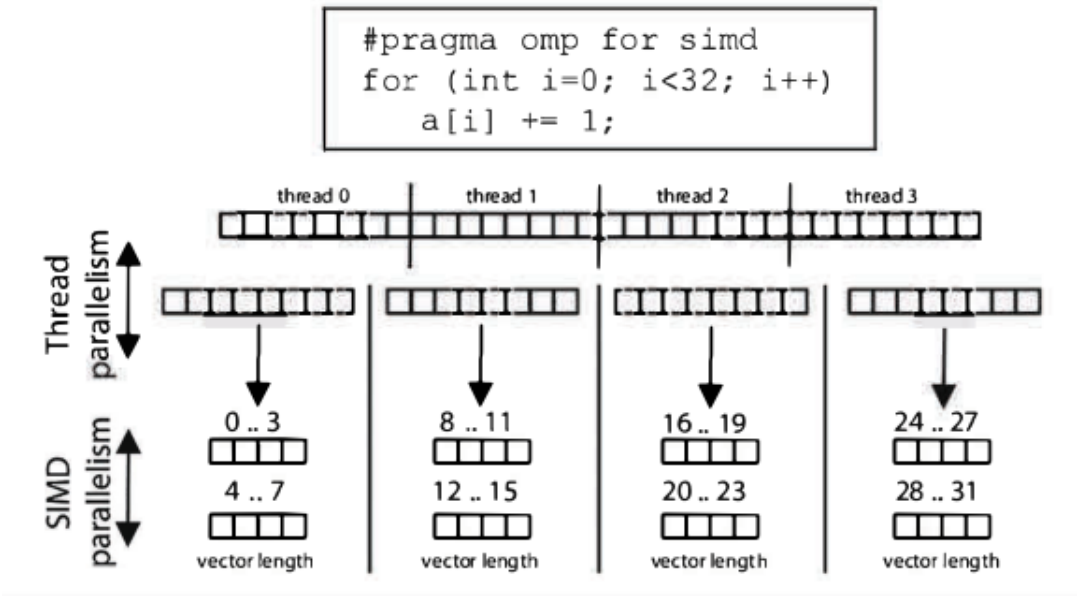
Συμβ. 4: Παράδειγμα κώδικα με simd

```
#pragma omp for simd [clause[[,] clause] ...] new-line
        for-loops
```

Η σύνθετη οδηγία βρόγχου *SIMD*, συνδυάζει παραλληλισμό νημάτων και *SIMD*. Κομμάτια επαναλήψεων βρόγχου διανέμονται στα νήματα σε μια ομάδα. Στη συνέχεια εκτελούνται τα κομμάτια των επαναλήψεων με βρόγχο *simd*. Μια φράση που μπορεί να εμφανιστεί στην οδηγία διαμοιρασμού βρόγχου είτε στην οδηγία *simd* μπορεί να εμφανιστεί και στο σύνθετο όρο.

Στη σύνθετη οδηγία, τμήματα επαναλήψεων του βρόγχου κατανέμονται σε μια ομάδα νημάτων με τη μέθοδο που ορίζεται από τις φράσεις που ορίζονται και για την οδηγία διαμοιρασμού βρόγχου. Στη συνέχεια, τα κομμάτια επαναλήψεων βρόγχου μπορούν να μετατραπούν σε οδηγίες *simd* με μέθοδο που καθορίζεται από της φράσης που ορίζεται και για την οδηγία *simd*. Τα παραπάνω φαίνονται σχηματικά στην επόμενη εικόνα:

Κάθε βρόγχος έχει ένα συγκεκριμένο ποσοστό εργασίας που πρέπει να εκτελέσει. Αν ο αριθμός των νημάτων αυξηθεί, το ποσοστό της εργασίας ανα νήμα θα μειωθεί. Προσθέτοντας παραλληλισμό *simd*, δεν ειναι απραίτητη η βελτίωση της απόδοσης, ειδικά αν ο σιμδ βρόγχος που ανήκει σε ένα νήμα, μειώσει το μήκος του.



Σχήμα 3: Βήματα διεργασιών οδηγίας *for simd*

Συναρτήσεις *SIMD*

Οι κλίσεις συναρτήσεων εντός βρόγχου *simd* εμποδίζουν τη δημιουργία αποτελεσματικών *simd* δομών. Στη χειρότερη περίπτωση η κλήση της συνάρτησης θα γίνει χρησιμοποιώντας βαθμωτά δεδομένα, και πιθανότατα θα επηρεάσει αρνητικά την αποτελεσματικότητα.

Για την πλήρη εκμετάλλευση του παραλληλισμού ΣΙΜΔ, μια συνάρτηση που καλείται μέσα από ένα βρόγχο *simd*, πρέπει να έιναι σε μια ισοδύναμη έκδοση της συνάρτησης *simd*. Ετσι, ο μεταγλωτιστής πρέπει να δημιουργήσει αυτή την ειδική έκδοση της συνάρτησης αυτής με *simd* παραμέτρους και οδηγίες.

Η οδηγία *simd directive* χρησιμοποιείται για να δημιουργήσει ο μεταγλωττιστής μια ή περισσότερες *simd* εκδόσεις μιας συνάρτησης. Αυτές οι εκδόσεις χρησιμοποιούνται απο βρόγχους *simd*.

1.1.3.1 Η οδηγία *declare simd*

Η οδηγία *declare simd* χρησιμοποιείται για να δηλώσει οτι μια *simd* παραλλαγή μιας συνάρτησης μπορεί να χρησιμοποιηθεί από μια περιοχή παραλληλισμού *simd*.

Συμβ. 5: Χρήση οδηγίας declare simd

```
#pragma omp declare simd [clause [[,] clause] ...] new-line
 function declaration definitions
```

Συμβ. 6: Φράσεις που υποστηρίζονται από simd

```
simdlen (length)
linear (list[: linear-step J)
aligned (list[:alignmentj)
uniform ( argument-list)
inbranch
notinbranch
```

Ο μεταγλωττιστής μπορεί να δημιουργήσει πολλές εκδόσεις μιας συνάρτησης *simd* και να επιλέξει την κατάλληλη για να κληθεί σε μια συγκεκριμένη τοποθεσία μιας κατασκευής *simd*. Ο χρήσης μπορεί να προσαρμόσει την λειτουργία μιας συνάρτησης με εξειδικευμένες φράσεις.

Υπάρχουν δύο περιορισμοί. Αν μια μεταβλητή αλλάζει ως αποτέλεσμα μιας τροποποίησης μιας φαινομενικά άσχετης μεταβλητής η συμπεριφορά είναι απροσδιόριστη. Επιπλέον, μια συνάρτηση που εμφανίζεται κάτω απο οδηγία *declare simd*, δεν επιτρέπονται τα *exceptions*.

Μια παρενέργεια στο πλαίσιο του προγραμματισμού λέγεται ότι συμβαίνει εάν μια μεταβλητή αλλάζει ως αποτέλεσμα μιας τροποποίησης σε μια φαινομενικά άσχετη μεταβλητή. Αυτό μπορεί να συμβεί λόγω του ψευδώνυμου του δείκτη για παράδειγμα. Σε μια τέτοια κατάσταση, δύο διαφορετικοί δείκτες δείχνουν στην ίδια θέση μνήμης. Αλλαγή ενός δείκτη επηρεάζει τους άλλους. Ένα άλλο παράδειγμα είναι μια συνάρτηση που τροποποιεί τα παγκόσμια δεδομένα. Δεδομένου ότι τα παγκόσμια δεδομένα χρησιμοποιούνται (δυνητικά) από άλλες λειτουργίες, αυτό μπορεί να προκαλέσει καταστροφή σε

ένα παράλληλο πρόγραμμα και πρέπει να ληφθεί μέριμνα για την αντιμετώπιση αυτής της κατάστασης.

1.1.3.2 Χαρακτηριστικά παραμέτρων συνάρτησης *simd*

Οι φράσεις *uniform, linear, simdlen, aligned* χρησιμοποιούνται για τον καθορισμό χαρακτηριστικών για τις παραμέτρους της συνάρτησης *simd*. Εκτός από τη ρήτρα *simdlen*, οι μεταβλητές που εμφανίζονται στις υπόλοιπες φράσεις πρέπει να είναι παράμετροι της συνάρτησης για την οποία ισχύει η οδηγία.

Οταν μια παράμετρος βρίσκεται στη φράση *uniform*, υποδεικνύει οτι η τιμή της παραμέτρου έχει την ίδια τιμή για όλες τις ταυτόχρονες κλήσεις κατά την εκτέλεση μιας οδηγίας *simd* βρόχου. Η φράση *linear* έχει διαφορετική σημασία όταν εμφανίζεται σε οδηγία *simd*. Δείχνει οτι ένα όρισμα που μεταβιβάζεται σε μια συνάρτηση έχει γραμμική σχέση μεταξύ των παράλληλων επικλήσεων μιας συνάρτησης.

Κάθε *simd* λωρίδα παρατηρεί την τιμή του ορίσματος στην πρώτη λωρίδα και προσθέτει την μετατόπιση της *simd* λωρίδας από την πρώτη, επί το γραμμική βήμα.

$$val_{curr} = val_1 + offset * step$$

Η φράση *uniform(arg1, arg2)* ενημερώνει τον *compiler* να δημιουργήσει μια *simd* συνάρτηση που προυποθέτει οτι αυτές οι δύο μεταβλητές είναι ανεξάρτητες από τον βρόγχο.

Η φράση *inbranch* υποστηρίζει οτι μια συνάρτηση καλείται πάντα μέσα σε ένα βρόγχο *simd* που περιέχει *if condition*. Ο μεταγλωττιστής πρέπει να αναδιαρθρώσει τον κώδικα για να χειριστεί την πιθανότητα ότι μια λωρίδα *simd* μπορεί να μην εκτελέσει τον κώδικα μιας συνάρτησης.

Η φράση *notinbranch* χρησιμοποιείται όταν η συνάρτηση δεν εκτελείται ποτέ μέσα από ένα *if condition* σε ένα *simd* βρόγχο. Επιτρέπει τον μεταγλωττιστή κάνει μεγαλύτερες βελτιστοποιήσεις στην απόδοση του κώδικα μιας συνάρτησης που χρησιμοποιεί *simd* οδηγίες.

Συμβ. 7: Παράδειγμα χρήσης φράσεων inbranch - notinbranch

```
#pragma omp declare simd inbranch
float pow(float x) {
        return (x * x);
}

#pragma omp declare simd notinbranch
extern float div(float);

void simd_loop(float *a, float *b, int n)
{
        #pragma omp simd
        for (int i=0; i<n; i++) {
                if (a[i] < 0.0 )
                        b[i] = pow(a[i]);
```

```
                b[i] = div(b[i]);
        }
}
```

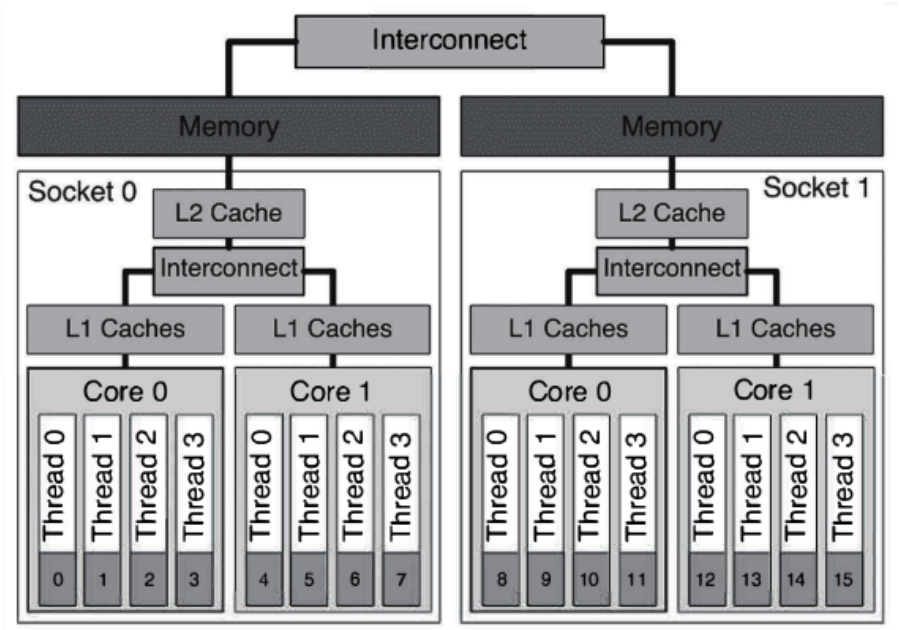
Thread Affinity

Thread Affinity είναι μια ευρύτερη έννοια που περιλαμβάνει την βελτιστοποίηση του χρόνου εκτέλεσης ενός προγράμματος, μέσω βελτιστοποιήσεων στο εύρος ζώνης μνήμης, τη ν αποφυγή καθυστέρησης μνήμης ή της καθυστέρησης χρήσης προσωρινής μνήμης.

Το *OpenMP 4.0* εισάγει ένα σύνολο οδηγιών για την υποστήριξη του *thread affinity*[2]. Η πλειοψηφία πλέον των μηχανημάτων βασίζονται στην *cc-NUMA* αρχιτεκτονική. Ο λόγος που αυτό το σύστημα μνήμης έγινε κυρίαρχο, είναι η συνεχής αύξηση του αριθμού των επεξεργαστών. Η μονολιθική διασύνδεση μνήμης με σταθερό εύρος ζώνης μνήμης θα αποτελούσε πρόβλημα στην ραγδαία αύξηση των επεξεργαστών.

Στη *cc-NUMA* αρχιτεκτονική κάθε υποδοχή συνδέεται με ένα υποσύνολο της συνολικής μνήμης του συστήματος. Μία διασύνδεση ενώνει τα υποσύνολα μεταξύ τους και δημιουργεί την εικόνα ενιαιας μνήμης στον χρήστη. Ενα τέτοιο σύστημα είναι ευκολότερο να επεκταθεί.

Το πλεονέκτημα της διασύνδεσης είναι ότι η εφαρμογή έχει πρόσβαση σε όλη την μνήμη του συστήματος, ανεξάρτητα από το που βρίσκονται τα δεδομένα. Ωστόσο, πλεον ο χρόνος πρόσβασης σε αυτά δεν ειναι ο σταθερός καθώς εξαρτάται από τη θέση τους στη μνήμη.



Σχήμα 4: Αρχιτεκτονική cc-NUMA[8]

Tasking

Η έννοια των διεργασιών (*Tasking*) εισήχθει στο OpenMP το 2008 με την έκδοση 3.0[5]. Οι διεργασίες παρέχουν τη δυνατότητα, οι αλγόριθμοι με ακανόνιστη και εξαρτώμενη από το χρόνο ροή εκτέλεσης να μπορούν να παραλληλιστούν. Ένας μηχανισμός ουράς διαχειρίζεται δυναμικά την εκχώρηση νημάτων στη διεργασία που πρέπει να εκτελεστεί. Τα νήματα παραλαμβάνουν εργασίες από την ουρά εως ότου αυτή αδειάσει. Διεργασία

(*task*) είναι ενα μπλοκ κώδικα σε μια παράλληλη περιοχή που εκτελειται ταυτόχρονα με μια αλλη διεργασία στην ίδια περιοχή.

Οι διεργασίες είναι τμήμα της παράλληλης περιοχής. Χωρίς ειδική μέριμνα, η ίδια διεργασία μπορεί να εκτελεστεί από διαφορετικά νήματα. Αυτό αποτελεί ασάφεια, καθώς οι οδηγίες διαμοιρασμού μνήμης, καθορίζουν με σαφήνεια τον τρόπο που οι εργασίες διανέμονται στα νήματα. Για να εγγυηθεί το OpenMP ότι κάθε διεργασία εκτελείται μόνο μια φορά, η κατασκευή τους θα πρέπει να ενσωματωθούν σε μια οδηγία *single* ή *master*.

Συμβ. 8: Παράδειγμα κώδικα με διεργασίες

```
#include <omp.h>

int main(void) {
    #pragma omp parallel          // Begin of parallel region
    {
        #pragma omp single
        {
            #pragma omp task
                func_task_1();    // First task creation
            #pragma omp task
                func_task_2();    // Second task creation
        }// end of single region  // Implicit barrier
    } //end of parallel region    // Implicit barrier
}
```

Το παραπάνω παράδειγμα αποτελεί επεξήγηση της λειτουργίας των διεργασιών. Υπάρχουν δύο νήματα, ένα νήμα συναντά την οδηγία *single* και αρχίζει να εκτελεί το μπλοκ κώδικα. Δύο διεργασίες δημιουργούνται, αλλά δεν έχουν ακόμη εκτελεστεί. Το άλλο νήμα συναντά και περιμένει στο *barrier* στο τέλος της οδηγίας *single*. Στην περίπτωση την διεργασιών, τα αδρανή νήματα δεν περιμένουν στο *barrier*. Αντι αυτου, αυτά τα νήματα είναι διαθέσιμα για την εκτέλεση των διεργασιών. Το νήμα που δημιουργεί τις διεργασίες καταλήγει επίσης στο φράγμα, μόλις ολοκληρωθεί η φάση παραγωγής και μπορεί να εκτελεί και αυτό διεργασίες. Η σειρά με την οποία θα εκτελεστούν οι διεργασίες δεν καθορίζεται.

Η οδηγία διεργασιών

Ορίζει μια ρητή διεργασία. Το περιβάλλον δεδομένων της διεργασίας δημιουργείται σύμφωνα με τις φράσεις χαρακτηριστικών κοινής χρήσης δεδομένων κατασκευή διεργασίων και τυχόν προεπιλογές που ισχύουν[9].

Συμβ. 9: Σύνταξη διεργασίας

```
#pragma omp task [clause[ [, ]clause] ...]
        structured-block
```

Συμβ. 10: Αποδεκτές φράσεις οδηγίας sections

```
if([ task :] scalar-expression)
final(scalar-expression)
untied
default(shared | none)
mergeable
private(list)
firstprivate(list)
shared(list)
in_reduction(reduction-identifier : list)
depend([depend-modifier,] dependence-type : locator-list)
priority(priority-value)
allocate([allocator :] list)
affinity([aff-modifier :] locator-list)
detach(event-handle)
```

1.3.1.1 Φράση if

Η έκφραση της συνθήκης *if* μιας διεργασίας, θα πρέπει να αξιολογείται ώς ψευδής ή αληθής. Η οδηγία των διεργασιών παρέχει μια φράση *if*, που δέχεται μια έκραση ως όρισμα. Αν η έκφραση αξιολογείται ώς ψευδής, απαγορεύει η αναβολή της διεργασίας. Ανεξαρήτου αποτελέσματος της έκφρασης, δημιουργείται πάντα ένα νέο περιβάλλον δεδομένων εργίας. Αν η έκφραση αξιολογείται ως ψευδής, δεν γίνονται όλοι οι υπολογισμοί που απετούνται για μια διεργασία αν δεν υπήρχε η φράση ιφ, επομένως θα μπορούσαν να αποφευχθούν ορισμένα κόστη επιδόσεων.

Ως εκ τούτου η φράση *if* αποτελεί λύση σε καταστάσεις, όπως οι αναδρομικοί αλγόριθμοι, όπου το υπολογιστικό κόστος μειώνεται καθώς αυξάνεται το βάθος και το όφελος από τη δημιουργία μιας νέας διεργασίας μειώνεται λόγω του γενικού κόστους. Ωστόσο, για λόγους ασφαλείας [5], οι μεταβλητές που αναφέρονται σε μια οδηγία κατασκευής διεργασιών είναι, στις περισσότερες περιπτώσεις, από προεπιλογή *firstprivate*, επομένως το κόστος της ρύθμισης περιβάλλοντος δεδομένων μπορεί να είναι το κυρίαρχο συστατικό της δημιουργίας διεργασιών.

1.3.1.2 Φράση final

Η φράση χρησιμοποιείται για να ελέγχεται με ευκρίνεια η ευαισθησία των διεργασιών. Οταν χρησιμοποιείται μέσα στην οδηγεία *task*, η έκφραση αξιολογείται κατά τη διάρκεια δημιουργίας αυτού. Αν είναι αληθής, η διεργασία θεωρείται τελική. Όλες οι διεργασίες που δημιουργούνται μέσα σε αυτή τη διεργασία, αγνοούνται και εκτελούνται στο πλαίσιο της.

Υπάρχουν δυο διαφορές ανάμεσα στη φράση *if* και στη *final*: την κατασκευή διεργασιών που επηρεάζει και τον τρόπο με τον οποίο οι κατασκευές αγνοούνται[1].

Πίνακας 1: Διαφορές ανάμεσα στις φράσεις *if* και *final* οταν εισάγονται σε κατασκευή διεργασίας.

***if* clause**	***final* clause**
Επηρεάζει την διεργασία που κατασκευάζεται από τη συγκεκριμένη οδηγιά κατασκευής διεργασιών	Επηρεάζει όλες τις διεργασίες ¨απογόνους¨ δηλαδή όλες τις διεργασίες που πρόκειται να δημιουργηθούν μέσα από αυτή που περιείχε τη φράση *final*
Η οδηγία κατασκευής διεργασίας αγνοείται εν μέρει. Η διεργασία και το περιβάλλον μεταβλητών δημιουργείται ακόμα και αν η έκφραση αξιολογηθεί ως *false*	Αγνοείται εντελώς η κατασκευή διεργασιών δηλαδή δε θα δημιουργηθεί νέα εργασία ούτε νέο περιβάλλον δεδομένων.

1.3.1.3 Φράση mergeable

Μια συγχωνευμένη διεργασία είναι η διεργασία της οποίας το περιβάλλον δεδομένων είναι ίδιο με το περιβάλλον που δημιούργησε την διεργασία. Οταν εισάγεται η φράση *mergeable* σε μια οδηγία δημιουργίας διεργασίας τότε η υλοποίηση μπορεί να δημιουργήσει μια συγχωνευμένη διεργασία.

1.3.1.4 Φράση depend

Η φράση *depend* επιβάλλει πρόσθετους περιορισμούς στη δημιουργία διεργασιών. Αυτοί οι περιορισμοί δημιουργούν εξαρτήσεις μόνο μεταξύ συγκενικών διεργασιών.

1.3.1.5 Φράση untied

Κατά την επανάληψη μιας περιοχής διεργασίας που έχει τεθεί σε αναστολή, μια δεσμευμένη διεργασία θα πρέπει να εκτελεστεί ξανά από το ίδιο νήμα. Με τη φράση *untied*, δεν υπάρχει τέτοιος περιορισμός και η διεργασία συνεχίζεται από οποιοδήποτε νήμα.

Συγχρονισμός διεργασιών

Οι διεργασίες δημιουργούνται όταν υπάρχει μια οδηγία δημιουργίας διεργασιών. Η στιγμή εκτέλεσης τους δεν είναι καθορισμένη. Το *OpenMP* εγγυάται οτι θα έχουν ολοκληρωθεί όταν ολοκληρωθεί η εκτέλεση του προγράμματος, ή όταν προκύψει κάποια οδηγία συγχρονισμού διεργασίας.

Πίνακας 2: Οδηγιες συγχρονισμού διεργασιών.

Οδηγία	**Περιγραφή**
#pragma omp barrier	Μπορεί είτε να υπάρχει υπονοούμενη είτε να δηλωθεί ρητά.
#pragma omp taskwait	Περιμένει μέχρι να ολοκληρωθούν όλες οι διεργασίες παιδιά της συγκεκριμένης διεργασίας.
#pragma omp taskgroup	Περιμένει μέχρι να ολοκληρωθούν όλες οι διεργασίες παιδιά της συγκεκριμένης διεργασίας αλλά και οι απόγονοι τους.

Στην περίπτωση της δημιουργίας διεργασιών μέσω οδηγίας σινγλε, υπάρχει υπονοούμενο φράγμα εκτέλεσης, σε αντίθεση με την οδηγία μαστερ που δεν υπονοείται.

Συμβ. 11: Παράδειγμα taskwait

```
#pragma omp parallel {
        #pragma omp single
        {
                #pragma omp task
                {
                        function1();
                }//Task #1
                #pragma omp task
                {
                        function2();
                } // Task #2

                #pragma omp taskwait
                        last_to_be_executed();
        } // End of single region
} // End of parallel region
```

Ετερογενής Αρχιτεκτονική

Η ετερογενής αρχιτεκτονική είναι ένα σύνολο προδιαγραφών που επιτρέπουν την ενσωμάτωση κεντρικών μονάδων επεξεργασίας και επεξεργαστών γραφικών στον ίδιο δίαυλο, με κοινόχρηστη μνήμη και διεργασίες[7].

Εξειδικευμένη επεξεργαστές επιταχυντών που βελτιώνουν δραματικά την απόδοση υπολογισμών, πολλαπλασιάζονται και οι επεξεργαστές γενικής χρήσης συνδέονται σε κάποιον επιταχυντή. Η αύξηση της δημοτικότητας αυτών των ετερογενών αρχιτεκτονικών σε όλους τους τύπους υπολογιστών είχε αξιοσημείωτο αντίκτυπο στην ανάπτυξη λογισμικού.

Για την εκμετάλλευση αυτών των συστημάτων, οι χρήστες πρέπει να κατασκευάζουν λογισμικό που εκτελεί διάφορες περιοχές κώδικα σε διαφορετικούς τύπους συσκευών. Το μεγαλύτερο κίνητρο αποτελεί η επιτάχυνση των υπολογισμών στους βρόγχους επανάληψης.

Ωστόσο, τα μοντέλα προγραμματισμού για αυτά τα συστήματα είναι δύσκολο να χρησιμοποιηθούν. Συχνά, οι ενότητες κώδικα γράφονται δύο φορές, μία φορά για τον επεξεργαστή γενικού σκοπού και μια για τον επιταχυντή. Η έκδοση του επιταχυντή γράφεται γλώσσα χαμηλότερου επιπέδου. Τα παραπάνω προβλήματα οδηγούν σε δυσκολία συντήρησης λογισμικού και ανάγκη διατήρησης δύο εκδόσεων ίδιου κώδικα.

Για τις ανάγκες διευκόλυνσης, το *OpenMP* επέκτεινε τις λειτουργίες του με σκοπό να υποστηρίξει αυτούς τους τύπους συστημάτων[4]. Τα αποτελέσματα της εργασίας αρχικά δημοσιεύτηκαν στην έκδοση 4.0 και στη ανανεώθηκαν στην έκδοση 4.5.

Ο χρήστης μπορεί να χρησιμοποιήσει την διεπαφή *OpenMP*, για να προγραμματίσει επιταχυντές σε υψηλότερο επίπεδο γλώσσας και για τη διατήρηση μίας μόνο έκδοσης του κώδικα, η οποία μπορεί να τρέξει είτε σε επιταχυντή είτε σε επεξεργαστή γενικής χρήσης.

Κίνητρο για την εκτέλεση κώδικα σε μια ετερογενή αρχιτεκτονική είναι να εκτελεστεί ένα κομμάτι κώδικα σε έναν επιταχυντή. Ο ρόλος των επιταχυντών είναι η δραματική βελτίωση της απόδοσης ενός προγράμματος αξιοποιώντας τις δυνατότητες υλικού των συσκευών αυτών. Το *OpenMP* παρέχει τα μέσα για τη διανομή εκτέλεσης ενός προγράμματος σε διαφορετικές συσκευές με ετερογενή αρχιτεκτονική. Συσκευή είναι ένας υπολογιστικός πόρος όπου μια περιοχή κώδικα μπορεί να εκτελεστεί. Παραδείγματα τέτοιων συσκευών είναι ΓΠΥ, ΨΠΥ, ΔΣΠ, ΦΠΓΑ κ.α.

Οι συσκευές έχουν τα δικά τους νήματα, των οποίων η μετεγκατάσταση σε άλλες συσκευές δεν είναι δυνατή. Η εκτέλεση του προγράμματος ξεκινάει από την κεντρική συσκευή (*host device*. Η κεντρική συσκευή είναι υπεύθυνη για την μεταφορά του κώδικα και των δεδομένων στον επιταχυντή.

Συμβ. 12: Παράδειγμα taskwait

```
void add_arrays(double *A, double *B, double *C, size_t size) {
    size_t i = 0;
#pragma omp target map(A, B, C)
    for (i = 0; i < size; ++i) {
            C[i] = A[i] + B[i];
    }
}
```

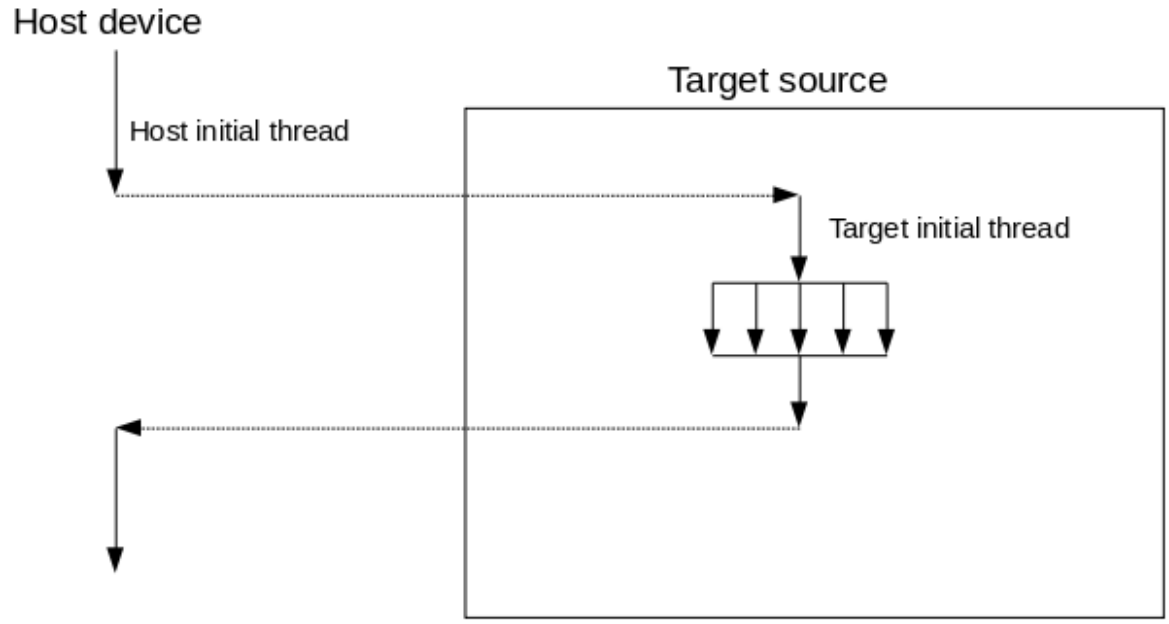
Οταν ένα νήμα του εξυπηρετητή συναντάει την οδηγία *target*, η περιοχή που ακολουθεί εκτελείται σε έναν επιταχυντή. Απο προεπιλογή, το νήμα που συναντά την οδηγία περιμένει την ολοκλήρωση της εκτέλεσης της παράλληλης περιοχής, προτού συνεχίσει.

Πριν το νέο νήμα αρχίσει να εκτελεί την παράλληλη περιοχή, οι μεταβλητές *A, B, C* αντιστοιχίζονται στον επιταχυντή. Η φράση *mapped* είναι η έννοια που χρησιμοποιεί το *OpenMP* για να περιγράψει τον τρόπο κοινής χρήσης των μεταβλητών σε όλες τις συσκευές.

Το αρχικό νήμα της συσκευής προορισμού

Το νήμα που ξεκινάει την εκτέλεση ενός προγράμματος και όλου του σειριακού τμήματος του κώδικα, ονομάζεται κύριο νήμα. Στην περίπτωση του ετερογενούς προγραμματισμού, το κύριο νήμα ανήκει στην κεντρική συσκευή, δηλαδή το πρόγραμμα ξεκινάει να εκτελείται στην κεντρική συσκευή.

Με την εισαγωγή της οδηγίας *target* στο *OpenMP* 4.0, πολλαπλά αρχικά νήματα μπορούν να δημιουργηθούν κατά τη διάρκεια εκτέλεσης ενός προγράμματος. Την εκτέλεση του τμήματος κώδικα στη συσκευή προορισμού, την αναλαμβάνει ένα νέο αρχικό νήμα και όχι το νήμα που συνάντησε το *target*. Το νήμα αυτό μπορεί να συναντήσει οδηγίες παραλληλισμού και να δημιουργήσει υποομάδες νημάτων.



Σχήμα 5: Διάγραμμα ομάδων νημάτων σε ετερογενή αρχιτεκτονική

Η οδηγία *target teams*

Η οδηγία *target teams* κατασκευάζει μια ομάδα (*league* που λειτουργούν σε έναν επιταχυντή. Κάθε μία από αυτές τις ομάδες είναι ένα αρχικό νήμα που εκτελεί παράλληλα την επόμενη δήλωση κώδικα. Η λειτουργία αυτή είναι παρόμοια με μια οδηγία *parallel* με τη διαφορά ότι τώρα κάθε νήμα είναι μια ομάδα. Τα νήματα σε διαφορετικές ομάδες δεν μπορούν να συγχρονιστούν μεταξύ τους.

Όταν μια παράλληλη περιοχή συναντάται από μια ομάδα, κάθε αρχικό τηρεαδ ομάδας γίνεται κύριο σε μια νέα υποομάδα. Το αποτέλεσμα είναι ένα σύνολο υποομάδων, οπου κάθε υποομάδα αποτελείται από ένα ή περισσότερο νήματα.

Αυτή η δομή χρησιμοποιείται για να εκφράζεται ένας τύπος χαλαρού παραλληλισμού, όπου ομάδες νημάτων εκτελούν παράλληλα, αλλά με μικρή αλληλεπίδραση μεταξύ των υποομάδων.

Μοντέλο μνήμης ετερογενούς αρχιτεκτονικής

1.4.3.1 Η φράση *map*

Τα νήματα που εκτελούνται σε έναν επιταχυντή μπορούν να έχουν ιδιωτικές μεταβλητές. Το κύριο νήμα που ξεκινά την εκτέλεση μιας παράλληλης περιοχής λαμβάνει μια ιδιωτική μεταβλήτη που εμφανίζεται στη φράση *private* ή *firstprivate* στην οδηγία *target*.

Η φράση *map* χρησιμοποιείται για τον διαμοιρασμό κοινής μνήμης από τον *host* στον επιταχυντή. Οταν οι δυο συσκευές δεν έχουν κοινόχρηστη μνήμη, η μεταβλητή αντιγράφεται στον επιταχυντή. Η φράση *en* αποκρύπτει αν η μεταβλητή μοιράζεται ή αντιγράφεται στη συσκευή στόχου. Το *OpenMP* ενεργεί ανάλογα με την αρχιτεκτονική που χρησιμοποιείται.

Πίνακας 3: Ενέργειες που απαιτούνται στο *map* ανάμεσα σε διαμοιραζόμενη και κοινόχρηστη μνήμη

Τύπος Μνήμης	**memory allocation**	**copy**	**flush**
Διαμοιρασμένη	Ναι	Ναι	Ναι
Κοινόχρηστη	Οχι	Οχι	Ναι

1.4.3.2 Περιβάλλον δεδομένων συσκευής

Ο επιταχυντής έχει ένα περιβάλλον μνήμης που περιέχει το σύνολο των μεταβλητών που είναι προσβάσιμες από νήματα που εκτελούνται σε αυτή τη συσκευή. Η αντιστοίχιση των δεδομένων διασφαλίζει ότι η μεταβλητή βρίσκεται στο περιβάλλον δεδομένων του επιταχυντή.

Μία μεταβλητής του εξυπηρετητή, αντιστοιχίζεται στην αντίστοιχη μεταβλητή του περιβάλλοντος δεδομένων του επιταχυντή. Ανάλογα με τη διαθεσιμότητα της κοινόχρηστης μνήμης μεταξύ του εξυπηρετητή *host* και της συσκευής προορισμού, η πρωτότυπη μεταβλητή του εξυπηρετητή και η αντίστοιχη μεγαβλητή της συσκευής προορισμού είναι είτε η ίδια μεταβλητή που βρίσκεται στη κοινόχρηστη μνήμη ή βρίσκεται σε διαφορετικές θέσεις, με αποτέλεσμα να απαιτούνται εργασίες αντιγραφής και ενημέρωσης για να διατηρηθεί η συνέπεια μεταξύ των δυο θέσεων.

Η ελαχιστοποίηση της μεταφοράς δεδομένων ανάμεσα στον εξυπηρετητή και τον επιταχυντή, αποτελεί κρίσιμο σημείο για την επίτευξη καλύτερης επίδοσης στις ετερογενείς αρχιτεκτονικές. Η επαναληπτική αντιστοίχιση μεταβλητών που επαναχρησιμοποιούνται είναι αναποτελεσματική.

1.4.3.3 Δείκτες μεταβλητών συσκευής

Αν ο εξυπηρετητής και ο επιταχυντής δεν μοιράζονται τη κοινόχρηστη μνήμη, οι τοπικές μεταβλητές τους βρίσκονται σε διαφορετικές θέσεις μνήμης. Οταν μια μεταβλητή αντιστοιχίζεται στο περιβάλλον δεδομένων ενός επιταχυντή, γίνεται μια αντιγραφή και η καινούργια μεταβλήτή ειναι διαφορετική από την μεταβλητή του εξυπηρετητή.

Οι διευθύνσεις μνήμης αποθηκεύονται σε μεταβλητές που ονομάζονται δείκτες (*pointers*. Ένα νήμα του εξυπηρετητή δε μπορεί να έχει πρόσβαση σε μνήμη μέσω ενός δείκτη που περιέχει διεύθυνση μνήμης του επιταχυντή. Ακόμη, ο επιταχυντής και ο εξυπηρετητής μπορεί να έχουν διαφορετική αρχιτεκτονική, δηλαδή ένας τύπος μεταβλητής μπορεί να είναι διαφορετικού μεγέθους ανάμεσα στις δύο συσκευές.

Ο δείκτης συσκευής *(device pointer)* είναι ένας δείκτης που αποθηκεύεται στον εξυπηρετητή και περιέχει την διεύθυνση μνήμης στο περιβάλλον δεδομένων του επιταχυντή.

Συμβ. 13: Παράδειγμα taskwait

```
int device = omp_get_default_device();
char *device_ptr = omp_target_alloc(n, device);
#pragma omp target is_device_ptr (device_ptr)
for (int j=0; j<n ; j++)
        *device_ptr++ = 0;
```

Εχω θέμα το add vector doubles c10.

Η οδηγία target

Σκοπός της οδηγίας target είναι η μεταφορά και εκτέλεση ενός τμήματος κώδικα στον επιταχυντή. Η εκτέλεση γίνεται από ένα αρχικό νήμα στη συσκευή. Σε περίπτωση έλλειψης επιταχυντή στο σύστημα, ο κώδικας που προορίζεται να εκτελεστεί εκεί μέσω της οδηγίας *target* θα εκτελεστεί στον εξυπηρετητή.

Συμβ. 14: Σύνταξη οδηγίας target

```
#pragma omp target [clause[[,] clause]...]
```

Συμβ. 15: Παράδειγμα εκτέλεσης στον επιταχυντή

```
void test() {
        int flag = 0;
        #pragma omp target map(flag)
        {
                flag = !omp_is_initial_device() ? 1 : 2;
        }
        if (flag == 1) {
                printf("Running on accelerator\n");
        } else if (flag == 2) {
                printf("Running on host\n");
        }
}
```

Η οδηγία *target* δημιουργεί μια διεργασία που εκτελείται στον επιταχυντή. Η διεργασία για τον εξυπηρετητή ολοκληρώνεται όταν ολοκληρωθεί η εκτέλεση στον επιταχυντή. Οι φράσεις *nowait* και *depend* επηρεάζουν τον τύπο και την ασύγχρονη συμπεριφορά της διεργασίας. Από προεπιλογή, η διεργασίες στόχου είναι συγχρονισμένη. Το νήμα που τη συναντά περιμένει μέχρι την ολοκλήρωση της εκτέλεσής της.

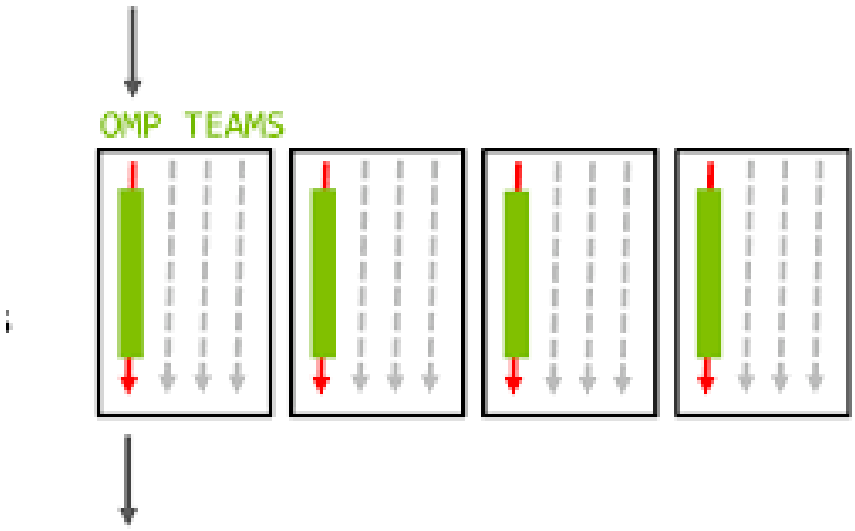
Οι δείκτης μεταβλητών που εισάγονται στη φράση *map*, είναι ιδιωτικές (*private*) μέσα στη συσκευή στόχου. Οι ιδιωτικές μεταβλητές δείκτη διεύθυνσης αρχικοποιούνται με την τιμή της διεύθυνσης του επιταχυντή.

Συμβ. 16: Φράσεις οδηγίας *target*

```
if (/target:] scalar-expression)
map ([[map-type-modifier[,JJ map-type:] list]
device (integer-expression)
private (list)
firstprivate (list)
is_device_ptr (list)
defaultmap( tofrom:scalar)
nowait
depend ( dependence-type: list)
```

Η οδηγία target teams

Η οδηγία *target teams* καθορίζει την δημιουργία μια συστάδα αρχικών νημάτων όπου κάθε αρχικό νήμα αποτελεί και μια ομάδα. Κάθε αρχικό νήμα εκτελεί την περιοχή παράλληλα.



Σχήμα 6: Ομάδες νημάτων με την οδηγία *target teams [3]*

Συμβ. 17: Φράσεις οδηγίας *target*

```
num_teams (integer-expression)
threadJimit (integer-expression)
default(shared I none)
private (list)
firstprivate (list)
shared (list)
reduction (reduction-identifier : list)
```

Η οδηγία distribute

Συμβ. 18: Σύνταξη οδηγίας *distribute*

```
#pragma omp distribute {clause[[,] clause}. . . j
        for–loops
```

Συμβ. 19: Φράσεις υποστηριζόμενες από την οδηγία *distribute*

```
private {list)
firstprivate {list)
lastprivate {list)
collapse (n)
dist_schedule {kind[, chunk_sizej)
```

Η οδηγία *distribute* καθορίζει τον διαμοιρασμό επαναλήψεων ενός βρόγχου στα αρχικά νήματα των ομάδων που δημιουργήθηκαν από την οδηγία *target teams*. Οι επαναλήψεις του βρόγχου χωρίζονται σε τμήματα και μοιράζονται στα κύρια νήματα των ομάδων. Η οδηγία *distribute* δεν εχει υπονοούμενο φράγμα εργασιών στο τέλος της, πράγμα που σημαίνει οτι τα κύρια νήματα των ομάδων δε συγχρονίζονται στο τέλος της οδηγίας.

Η φράση *distschedule* καθορίζει τον τρόπο που διαμοιράζονται οι επαναλήψεις σε τμήματα. Συγκριτικά με την οδηγία *for* η *distribute* έχει δυνατότητες για καλύτερη απόδοση. Ο μεταγλωττιστής μπορεί να πετύχει μεγαλύτερη βελτιστοποίηση.

Σύνθετες οδηγίες επιταχυντών

Οι συνδυασμένες οδηγίες είναι ισοδύναμες με τις επιμέρους. Για παράδειγμα η οδηγία *parallel for* έχει την ιδια σημασία με την *parallel* ακολουθούμενη από την οδηγία *for*. Παρόλα αυτά, ορισμένες φορές, οι συνδυασμένες οδηγίες μπορούν να επιτύχουν καλύτερες επιδόσεις. Σε αυτή την παράγραφο, οι οδηγίες χωρίζονται σε δύο κατηγορίες, τις συνδυασμένες με *target* και αυτές που συνδυάζονται με *target teams*.

Συμβ. 20: Συνδυασμένες οδηγίας επιταχυντή

```
#pragma omp target parallel [clause[[,] clause]...]
        structured block

#pragma omp target parallel for [clause[[,] clause]...]
        for–loops

#pragma omp target parallel for simd [clause[[,] clause]...]
        for–loops

#pragma omp target simd [clause[[,] clause]...]
        for–loops
```

Συμβ. 21: Συνδυασμένες οδηγίας επιταχυντή

```
#pragma omp distribute parallel for [clause[[,] clause]...]
        for-loops
#pragma omp distribute simd [clause[[,] clause]...]
        for-loops
#pragma omp distribute parallel for simd [clause[[,] clause]...]
        for-loops
```

Φράσεις οδηγίας *map*

Συμβ. 22: Σύνταξη οδηγίας map

```
map ([[map-type-modifier[,}} map-type:} list)
```

Συμβ. 23: Αποδεκτές τιμές για το map-type

```
        alloc
        to
        from
        tofrom -> default
        release
        delete
```

Υπάρχουν τρεις φάσεις στην αντιστοίχιση μεταβλητών στον επιταχυντή:

1. Η φάση *map-enter* στην αρχή της εκτέλεσης της οδηγίας *target*, όπου οι μεταβλητή αντιστοιχίζεται στον επιταχυντή. Σε αυτή τη φάση δεσμεύεται μνήμη του επιταχυντή για την αποθήκευση της μεταβλητής, και αντιγράφεται από τον εξυπηρετητή.

2. Η φάση υπολογισμού που προκύπτει όταν, κατά τη διάρκεια εκτέλεσης της παράλληλης περιοχής, τα νήματα που εκτελούν το πρόγραμμα αποκτούν πρόσβαση στην αντιστοιχισμένη μεταβλητή.

3. Η φάση εξόδου όπου ολοκληρώνεται η αντιστοίχιση των μεταβλητών στον επιταχυντή. Η τιμή της μεταβλητής στον επιταχυντή αντιγράφεται στην αντίστοιχη θέση του εξυπηρετητή. Η δεσμευμένη μνήμη του επιταχυντή ελευθερώνεται.

Οι φάσεις 1 και 3 διαχειρίζονται την αποθήκευση και αντιγραφή των μεταβλητών ανάμεσα σε δυο συσκευές. Ο τύπος της αντιστοίχισης επηρεάζει την αντιγραφή μεταβλητών στον επιταχυντή ή τον εξυπηρετητή. Ο καθορισμός του τύπου αντιστοίχισης επηρεάζει την απόδοση του κώδικα.

Πίνακας 4: Απαιτούμενη αντιγραφή για κάθε τύπο μεταβλητής κατά τις φάσεις εισόδου-εξόδου

map-type	**Είσοδος**	**Έξοδος**
alloc	Οχι	Οχι
to	Ναι	Οχι
from	Οχι	Ναι
tofrom	Ναι	Ναι
release	-	Οχι
delete	-	Οχι

Συμβ. 24: Παράδειγμα χρήσης τύπου αντιστοίχισης μεταβλητών

```
void foo(double A[1024], double B[1024], double C[1024) {
        #pragma omp target map(from : A) map(to: B)
                        map(alloc: C) // map enter
        {
                //CODE
        } // map exit
}
```

Στο προηγούμενο παράδειγμα:
Η μεταβλητή **A**:

- Δεν αρχικοποιείται στον επιταχυντή
- Οι τιμή της αντιγράφεται στον εξυπηρετητή
- Η μνήμη αποδεσμεύεται κατά την επιστροφή στον εξυπηρετητή.

Η μεταβλητή **B**:

- Οι τιμή της αντιγράφεται στον επιταχυντή.
- Η μνήμη αποδεσμεύεται κατά την επιστροφή στον εξυπηρετητή.

Η μεταβλητή **C**:

- Οι τιμή της αντιγράφεται στον επιταχυντή.
- Η μνήμη αποδεσμεύεται κατά την επιστροφή στον εξυπηρετητή.

Οδηγία *declare target*

Η οδηγία *declare target* χρησιμοποιείται για συναρτήσεις και μεταβλητής. Μια συνάρτηση που καλείται μέσα στο τμήμα του *target* κώδικα, θα πρέπει να δηλώνεται στην οδηγία *declare target*. Ακόμη, η οδηγία χρησιμοποιείται για την αντιστοίχιση *global* μεταβλητών στο περιβάλλον δεδομένων του επιταχυντή.

Συμβ. 25: Συνδυασμένες οδηγίας επιταχυντή

```
#pragma omp declare target
        declarations-definitions-seq
#pragma omp end declare target
#pragma omp declare target(extended-list)
#pragma omp declare target clause[[,] clause]...]

CLAUSE:
to (extended-list)
link (list)
```

TODO έχει και αλλο.

References

[1] *An Extension to Improve OpenMP Tasking Control (Tsukuba, Japan, June 2010)*, volume 6132 of *Lecture Notes in Computer Science*, Berlin, Germany, 2010. Springer.

[2] *The Design of OpenMP Thread Affinity (Heidelberg, Berlin, June 2012)*, volume 7312 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2012. Springer.

[3] N. Aeronautics and S. Administration. Using openmp 4.5 target offload for programming heterogeneous systems. http://cacs.usc.edu/education/cs653/OpenMP4.5_3-20-19.pdf. Accessed: 2020-06-20.

[4] R. v. d. P. Barbara Chapman, Gabriele Jost. *Using OpenMP-Portable Shared Memory Programming*. The MIT Press, 2007.

[5] A. D. Eduard Ayguade, Nawal Copty. *The Design of OpenMP Tasks*, pages 404–418. IEEE Trans, 2009.

[6] M. Flynn. *Some Computer Organizations and Their Effectiveness*, pages 948–960. IEEE, 1972.

[7] T. Harware. Amd unveils its heterogeneous uniform memory access (huma) technology. https://www.tomshardware.com/news/AMD-HSA-hUMA-APU,22324.html. Accessed: 2020-06-13.

[8] C. T. Ruud van der Pas, E. Stotzer. *Using OpenMP. The Next Step: affinity, accelerators, tasking, SIMD*, page 152. The MIT Press, 2017.

[9] C. T. Ruud van der Pas, E. Stotzer. *Using OpenMP. The Next Step: affinity, accelerators, tasking, SIMD*, page 20. The MIT Press, 2017.