Ετερογενής Αρχιτεκτονική

Η ετερογενής αρχιτεκτονική είναι ένα σύνολο προδιαγραφών που επιτρέπουν την ενσωμάτωση κεντρικών μονάδων επεξεργασίας *(CPU)* και μονάδων επεξεργασίας γραφικών στον ίδιο δίαυλο, με κοινόχρηστη μνήμη και διεργασίες[3].

Οι μονάδες επεξεργασίας γραφικών βελτιώνουν σημαντικά την απόδοση των προγραμμάτων, με αποτέλεσμα τον πολλαπλασιάσμό της ζήτησης τους. Η αύξηση της δημοτικότητας των ετερογενών αρχιτεκτονικών σε όλους τους τύπους υπολογιστών είχε αξιοσημείωτο αντίκτυπο στην ανάπτυξη λογισμικών υψηλών προδιαγραφών.

Για την εκμετάλλευση των ετερογενών συστημάτων, οι χρήστες πρέπει να κατασκευάζουν λογισμικό που εκτελεί τμήματα κώδικα σε διαφορετικές συσκευές. Τέτοια τμήματα αποτελούν οι βρόγχοι επανάληψης, που χρησιμοποιούνται ευρέως για την δημιουργία λογισμικών.

Ωστόσο, τα μοντέλα προγραμματισμού για ετερογενή συστήματα είναι δύσκολο να χρησιμοποιηθούν λόγω της αυξημένης δυσκολίας διαχείρισής τους. Συνήθως, τμήματα κώδικα γράφονται σε δύο εκδόσεις, μία φορά για τον επεξεργαστή γενικού σκοπού και μια για τον επιταχυντή. Η έκδοση του επιταχυντή γράφεται γλώσσα χαμηλότερου επιπέδου. Η συντήρηση και ανάπτυξη διπλού κώδικα, αποτελεί ένα από τα μεγαλύτερα προβλήματα στην έννοια του προγραμματισμού.

Για τις ανάγκες διευκόλυνσης, το *OpenMP* επέκτεινε τις λειτουργίες του με σκοπό την υποστήριξη και ευρεία χρήση τέτοιων τύπων συστημάτων[2]. Τα αποτελέσματα της εργασίας αρχικά δημοσιεύτηκαν στην έκδοση 4.0 και εξελίχθηκαν περαιτέρω στην έκδοση 4.5. Οι χρήστες μπορούν πλέον να χρησιμοποιήσουν τη διεπαφή για τη δημιουργία λογισμικών γραμμένων με γλώσσες προγραμματισμού υψηλότερου επιπέδου και εκτελέσιμων από συσκευές επεξεργασίας γραφικών. Ετσι επιτυγχάνεται η διατήρηση μίας μόνο έκδοσης του κώδικα, η οποία μπορεί να τρέξει είτε σε μονάδα επεξεργασίας γραφικών ή σε επεξεργαστή γενικής χρήσης*CPU*.

Με τον όρο συσκευή στόχου, εννοείται ένας υπολογιστικός πόρος, στον οποίο μπορεί να εκτελεστεί μια περιοχή κώδικα. Παραδείγματα τέτοιων συσκευών είναι *GPU, CPU, DSP, FPGA* κ.α. Οι συσκευές στόχου 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 (συσκευή στόχου).

Συμβ. 1: Παράδειγμα εκτέλεσης κώδικα στη συσκευή στόχου

```
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];
    }
}
```

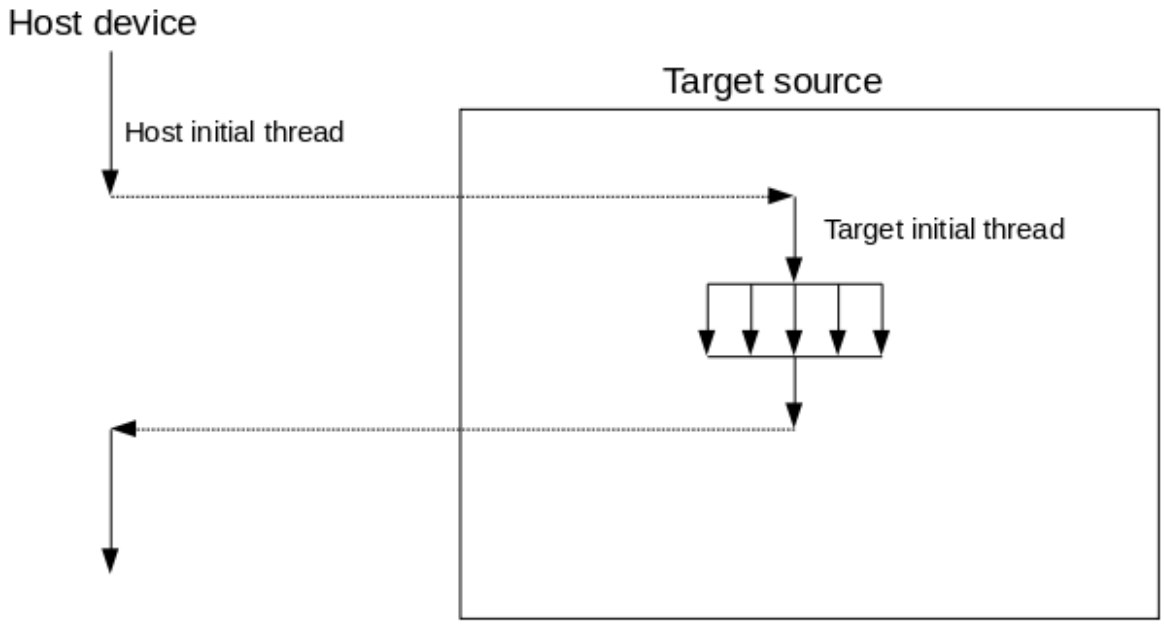
Οπως δείχνει το παραπάνω παράδειγμα, ένα νήμα του εξυπηρετητή (κύρια συσκευή - *host*) συναντάει την οδηγία ***target***. Το τμήμα κώδικα που ακολουθεί μεταφέρεται και εκτελείται στη συσκευή του επιταχυντή, αν αυτός υπάρχει. Απο προεπιλογή, το νήμα που συναντά την οδηγία περιμένει την ολοκλήρωση της εκτέλεσης της παράλληλης περιοχής, προτού συνεχίσει.

Πριν ένα καινούργιο νήμα που βρίσκεται στη συσκευή στόχου αρχίσει να εκτελεί την περιοχή που περικλείεται στην οδηγία *target*, οι μεταβλητές *A, B, C ¨αντιστοιχίζονται¨* στον επιταχυντή. Η φράση *mapped* είναι το εργαλείο που χρησιμοποιεί το *OpenMP* για να εξασφαλίσει τον πρόσβαση του της συσκευής στόχου, στις μεταβλητές αυτές.

Το αρχικό νήμα της συσκευής προορισμού

Το νήμα που ξεκινάει την εκτέλεση ενός προγράμματος, ονομάζεται κύριο νήμα και ανήκει πάντα στην κεντρική συσκευή. Με άλλα λόγια, ένα πρόγραμμα σε μια αρχιτεκτονική ετερογενούς προγραμματισμού, δε ξεκινάει ποτέ από τη συσκευή στόχου.

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 την οδηγία *target*. Το νήμα αυτό μπορεί να συναντήσει οδηγίες παραλληλισμού και να δημιουργήσει υποομάδες νημάτων.



Σχήμα 1: Διάγραμμα ομάδων νημάτων σε ετερογενή αρχιτεκτονική

Η οδηγία *target teams*

Η οδηγία *target teams* κατασκευάζει ομάδες νημάτων (*league* που λειτουργούν σε έναν επιταχυντή. Κάθε μία από τις ομάδες αποτελείται από ένα αρχικό νήμα. 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, τότε το αρχικό νήμα 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 τους.

Μοντέλο μνήμης ετερογενούς αρχιτεκτονικής

Στις επόμενες παραγράφους, ακολουθεί μια περιγραφή του μοντέλου μνήμης της ετερογενούς αρχιτεκτονικής, περιγράφονται έννοιες που σχετίζονται με τις μεταβλητές και η γνώση του θεωρητικού υπόβαθρου καθίσταται απαραίτητη για την ορθή υλοποίηση προγραμμάτων σε τέτοιες αρχιτεκτονικές.

0.1.3.1 Η φράση *map*

Οπως τα νήματα των εξυπηρετητών, έτσι και τα νήματα που δημιουργούνται στις συσκευές στόχου μπορούν να έχουν ιδιωτικές μεταβλητές. Αντίγραφα μεταβλητών στη συσκευή στόχου με χαρακτηριστικό ιδιωτικής μνήμης, δημιουργούνται όταν η οδηγία *target* ακολουθείται από τη φράση *private ή firstprivate*.

Στις αρχιτεκτονικές ετερογενούς προγραμματισμού και σε επίπεδο υλικού, ο εξυπηρετητής με τον επιταχυντή μπορεί να μοιράζονται κοινόχρηστη φυσική μνήμη. Η φράση *map* χρησιμοποιείται για την αντιστοίχιση δεδομένων ανάμεσα στις δύο συσκευές, αποκρύπτοντας παράλληλα χαρακτηριστικά της φυσικής υλοποίησης. Για παράδειγμα, όταν οι δυο συσκευές δεν έχουν κοινόχρηστη φυσική μνήμη, η μεταβλητή αντιγράφεται στον επιταχυντή. Αντίθετα, στην περίπτωση της υλοποίησης με κοινόχρηστη μνήμη, δεν απαιτείται δημιουργία αντίγραφου. Η φράση *map* απαλλάσει τον χρήστη από τον έλεγχο των χαρακτηριστικών της υλοποίησης σε επίπεδο υλικού, και η διεπαφή ενεργεί ανάλογα με την αρχιτεκτονική που χρησιμοποιείται.

Πίνακας 1: Ενέργειες που εκτελούνται από την οδηγία*map* ανάλογα με το είδος της αρχιτεκτονικής μνήμης

	memory allocation	**copy**	**flush**
Διαμοιρασμένη Μνήμη	Ναι	Ναι	Ναι
Κοινόχρηστη Μνήμη	Οχι	Οχι	Ναι

0.1.3.2 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.

0.1.3.3 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 μεταβλητή 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. 2: 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. 3: Σύνταξη οδηγίας target

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

Συμβ. 4: 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.

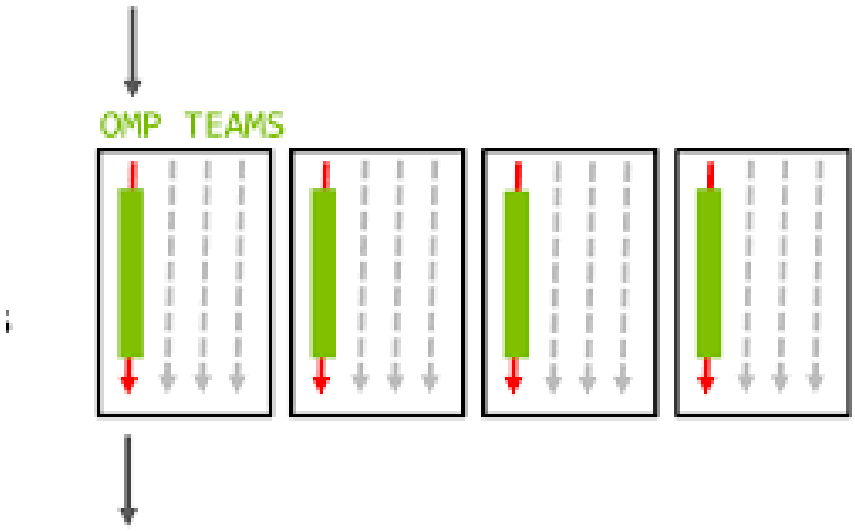
Οι δείκτης μεταβλητών που εισάγονται στη φράση *map*, είναι ιδιωτικές (*private*) μέσα στη συσκευή στόχου. Οι ιδιωτικές μεταβλητές δείκτη διεύθυνσης αρχικοποιούνται με την τιμή της διεύθυνσης του επιταχυντή.

Συμβ. 5: 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 2: Ομάδες νημάτων με την οδηγία *target teams [1]*

Συμβ. 6: 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. 7: Σύνταξη οδηγίας *distribute*

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

Συμβ. 8: 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. 9: 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. 10: 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. 11: Σύνταξη οδηγίας map

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

Συμβ. 12: 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 2: 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. 13: 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. 14: 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] 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.

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

[3] 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.