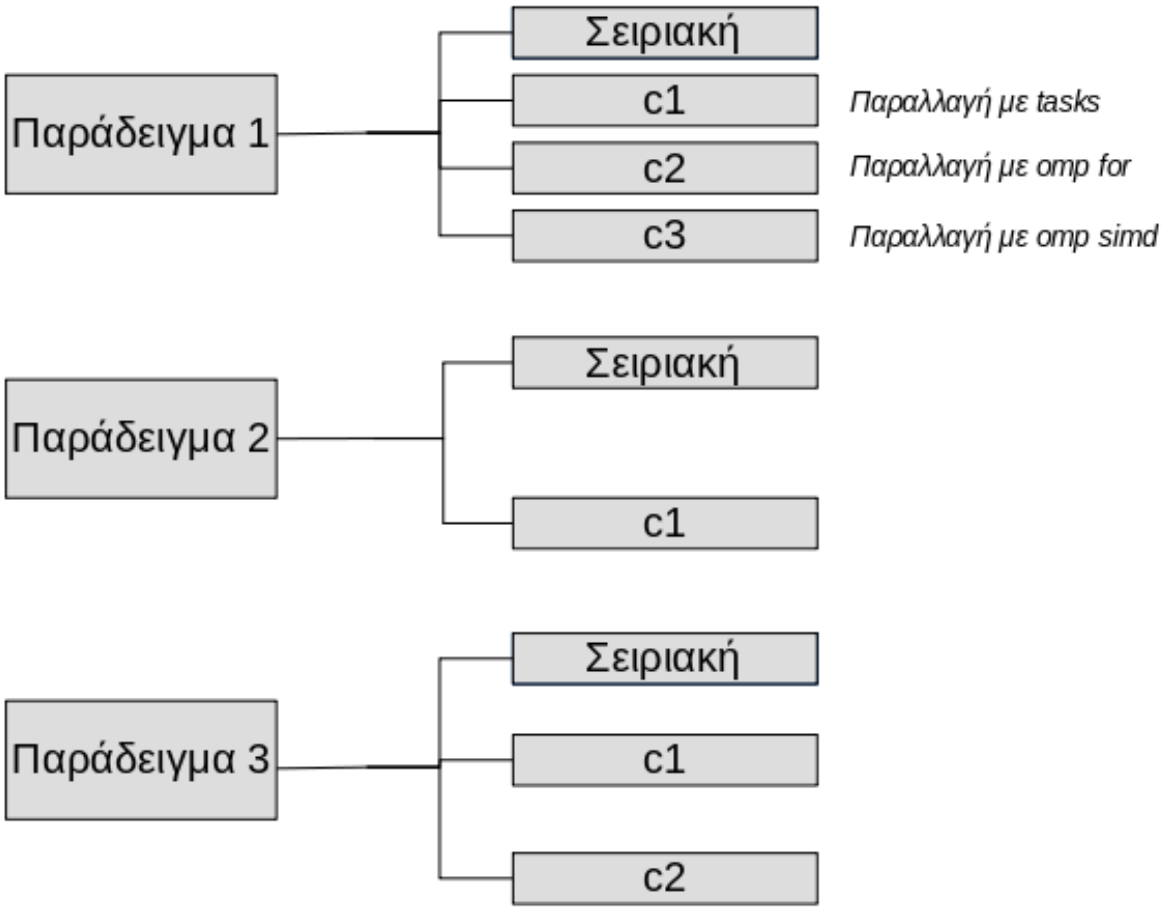
Υλοποιημένα παραδείγματα[1]

Οι ενότητες που ακολουθούν αφορούν την επίλυση προβλημάτων με διαφορετικές μεθόδους. Τα προβλήματα συγκεντρώθηκαν και επιλύθηκαν με στόχο την σύγκριση των αποτελεσμάτων και την εξαγωγή συμπερασμάτων. Ο πηγαίος των προβλημάτων βρίσκεται στον παρακάτω σύνδεσμο:

https://github.com/gkonto/openmp/

Κάθε πρόβλημα περιέχει υποφακέλους, και κάθε υποφάκελος αποτελεί μια παραλλαγή του προβλήματος.



Σχήμα 1: Διάρθρωση παραδειγμάτων στο github

Αναφορά αρχιτεκτονικής μηχανήματος

Τα προβλήματα που ακολουθούν εκτελέστηκαν σε μηχάνημα λειτουργικό *linux* και μεταγλωττιστή *gcc*. Τα χαρακτηριστικά υλικού εμφανίζονται στον παρακάτω πίνακα:

Πίνακας 1: Χαρακτηριστικά Μηχανήματος Εκτέλεσης

Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	16
Thread(s) per core	1
Core(s) per socket	8
Socket(s)	2
NUMA node(s)	4
Model name	AMD Opteron(tm) Processor 6128 HE
L1d cache	64K
L2 cache	512K
L3 cache	5118K
Memory	16036

Παράδειγμα διπλασιασμού τιμών στοιχείων πίνακα

Σε αυτό το παράδειγμα γίνεται ανάλυση του προβλήματος τροποποίησης τιμών ενός διανύσματος. Μελετάται η επίδοση κάθε εναλλακτικής λύσης και σχολιάζονται εργαλεία που αναφέρθηκαν στα προηγούμενα κεφάλαια. Χρησιμοποιείται ένα διάνυσμα μεταβαλλόμενου μεγέθους για κάθε επίλυση, το οποίο αρχικοποιείται στον εξυπηρετητή με τον παρακάτω τρόπο.

Συμβ. 1: Αρχικοποίηση τιμών διανύσματος

```
void fill_array(int *arr, size_t size) {
    for (size_t k = 0; k < size; ++k) {
            arr[k] = static_cast<int>(k);
    }
}
```

Για την επαλήθευση σωστού αποτελέσματος, χρησιμοποιείται η παρακάτω ρουτίνα, που καλείται μετά την εκτέλεση του διπλασιασμού:

Συμβ. 2: Αρχικοποίηση τιμών διανύσματος

```
void verify(int *arr, size_t size) {
        for (size_t k = 0; k < size; ++k) {
                if (arr[k] != k * 2) {
                printf('Error␣in␣position␣%ld.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Got␣%d,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣expected␣%ld\n',
                                k,
                                arr[k],
                                k * 2);
                                exit(1);
                }
        }
}
```

Η μεταγλώττιση του κώδικα έγινε με μεταγλωττιστή *g++-7* και τις επιλογές:$-Wall -oexec -O0$. Οι παραλλαγές εκτέλεσης του προβλήματος χωρίζονται σε δύο κατηγορίες, σε αυτές που απαιτείται αντιγραφή δεδομένων από τον εξυπηρετητή στον επιταχυντή, και από αυτές που δεσμεύουν μνήμη απευθείας στον επιταχυντή.

Σειριακή εκτέλεση

Στο σειριακό υπολογισμό, το πρόγραμμα εκτελείται από ένα μοναδικό νήμα, χωρίς βελτιστοποίηση παραλληλισμού. Στη συγκεκριμένη περίπτωση, καλείται μια ρουτίνα που δέχεται ως όρισμα ένα μοναδιαίο πίνακα με ακέραιους αριθμούς και ένας αριθμός που υποδηλώνει το μέγεθος αυτού του πίνακα. Η ρουτίνα είναι η εξής:

Συμβ. 3: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
    for (size_t i = 0; i < size; ++i) {
            A[i] = A[i] * 2;
    }
}
```

Οι χρόνοι εκτέλεσης που καταγράφηκαν εμφανίζονται στον παρακάτω πίνακα:

Πίνακας 2: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.002
1000000	0.0097
10000000	0.098
100000000	0.980
200000000	1.978
300000000	2.968

Υλοποιήσεις που απαιτούν αντιγραφή μνήμης

Τα παραδείγματα που ακολουθούν αφορούν υλοποίηση του προβλήματος με μεθόδους που απαιτούν αντιγραφή δεδομένων ανάμεσα στον εξυπηρετητή και στον επιταχυντή. Για την αντιγραφή των δεδομένων χρησιμοποιούνται οι φράσεις που υποστηρίζονται από την οδηγία *map* και αναφέρθηκαν στα προηγούμενα κεφάλαια.

1.2.2.1 Παραλλαγή 1η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 4: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target map(A[:size] )
    for (size_t i = 0; i < size; ++i) {
            A[i] = A[i] * 2;
    }
}
```

Πίνακας 3: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.971
1000000	1.838
10000000	10.177

1.2.2.2 Παραλλαγή 2η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 5: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target map(A[:size], flag)
        {
                #pragma omp parallel for
                for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
            }
    }
}
```

Πίνακας 4: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.885
1000000	0.971
10000000	2.153
100000000	13.566

1.2.2.3 Παραλλαγή 3η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 6: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target parallel map(A[:size], flag)
        {
                for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
            }
    }
}
```

Πίνακας 5: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.885
1000000	0.971
10000000	2.153
100000000	13.566

1.2.2.4 Παραλλαγή 4η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 7: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target parallel for simd map(A[:size], flag)
        {
                for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
            }
    }
}
```

Πίνακας 6: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.885
1000000	0.971
10000000	2.153
100000000	13.566

1.2.2.5 Παραλλαγή 5η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 8: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target parallel simd map(A[:size], flag)
        {
                for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
            }
    }
}
```

Πίνακας 7: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.885
1000000	0.971
10000000	2.153
100000000	13.566

1.2.2.6 Παραλλαγή 6η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 9: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target teams distribute map(A[:size], flag)
        {
                for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
            }
    }
}
```

Πίνακας 8: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.885
1000000	0.971
10000000	2.153
100000000	13.566

1.2.2.7 Παραλλαγή 7η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 10: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target teams distribute parallel for map(A[:size], flag)
        {
                for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
            }
    }
}
```

Πίνακας 9: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.885
1000000	0.971
10000000	2.153
100000000	13.566

1.2.2.8 Παραλλαγή 8η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 11: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target teams distribute simd map(A[:size], flag)
        {
                for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
            }
    }
}
```

Πίνακας 10: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	
1000000	
10000000	
100000000	

1.2.2.9 Παραλλαγή 9η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 12: Αρχικοποίηση τιμών διανύσματος

```
void double_elements(int *A, size_t size) {
#pragma omp target teams distribute parallel for simd map(A[:size], flag)
        {
                for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
            }
    }
}
```

Πίνακας 11: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.885
1000000	0.971
10000000	2.153
100000000	13.566

Υλοποιήσεις που δεσμεύουν μνήμη απευθείας στον επιταχυντή

Στις υλοποιήσεις αυτής της ενότητας δεν απαιτείται αντιγραφή δεδομένων μνήμης ανάμεσα στα περιβάλλοντα δεδομένων του επιταχυντή και του εξυπηρετητή, καθώς η μνήμη δεσμεύεται απευθείας στον επιταχυντή μέσω της οδηγίας *omp_target_alloc*. Σε αυτή την εκτέλεση, δεσμεύεται μνήμη για το διάνυσμα απευθείας στη συσκευή στόχου. Με αυτό τον τρόπο αποφεύγονται εργασίες αντιγραφής ανάμεσα στα δύο μέσα.

Συμβ. 13: Κώδικας αρχικοποίησης διανύσματος στη συσκευή στόχου και επαλήθευση ομαλής εκτέλεσης

```
    std::cout << "Device: " << device << std::endl;
    int *a = (int *)omp_target_alloc(sizeof(int) * o.size, device);
    if (!a) {
            std::cout << "Could not allocate memory for array"
             << std::endl;
            exit(1);
    } else {
            std::cout << "Successful allocation" << std::endl;
    }
#pragma omp target is_device_ptr(a)
    for (size_t i = 0; i < o.size; ++i) {
            a[i] = static_cast<int>(i);
    }
    //Count time
    auto start = omp_get_wtime();
    double_elements(a, o.size);
    auto end = omp_get_wtime();
    std::cout << "Starting verification" << std::endl;
#pragma omp target is_device_ptr(a)
    for (size_t i = 0; i < o.size; ++i) {
            if (a[i] != i * 2) {
                    exit(1);
            }
    }
    std::cout << "Successful verification" << std::endl;
    omp_target_free(a, device);
```

1.2.3.1 Παραλλαγή 11η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 14: Εκτέλεση υπολογισμών

```
void double_elements(int *A, size_t size) {
#pragma omp target is_device_ptr(A)
        {
        #pragma omp parallel for
                for (size_t i = 0; i < size; ++i) {
                        A[i] = A[i] * 2;
                }
        }
}
```

Πίνακας 12: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	0.016400
1000000	0.118455
10000000	1.179038
100000000	11.780315

1.2.3.2 Παραλλαγή 12η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 15: Εκτέλεση υπολογισμών

```
void double_elements(int *A, size_t size) {
#pragma omp target parallel for is_device_ptr(A)
        for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
        }
}
```

Πίνακας 13: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**

1.2.3.3 Παραλλαγή 13η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 16: Εκτέλεση υπολογισμών

```
void double_elements(int *A, size_t size) {
#pragma omp target teams distribute is_device_ptr(A)
        for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
        }
}
```

Πίνακας 14: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	
1000000	
10000000	
100000000	
200000000	

1.2.3.4 Παραλλαγή 14η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 17: Εκτέλεση υπολογισμών

```
void double_elements(int *A, size_t size) {
#pragma omp target teams distribute parallel for is_device_ptr(A)
        for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
        }
}
```

Πίνακας 15: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	
1000000	
10000000	
100000000	
200000000	
300000000	

1.2.3.5 Παραλλαγή 15η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

Συμβ. 18: Εκτέλεση υπολογισμών

```
void double_elements(int *A, size_t size) {
#pragma omp target teams distribute simd is_device_ptr(A)
        for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
        }
}
```

Πίνακας 16: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	
1000000	
10000000	
100000000	
200000000	
300000000	

1.2.3.6 Παραλλαγή 16η

Για τον υπολογισμό, η ρουτίνα που εκτελέστηκε ήταν η εξής:

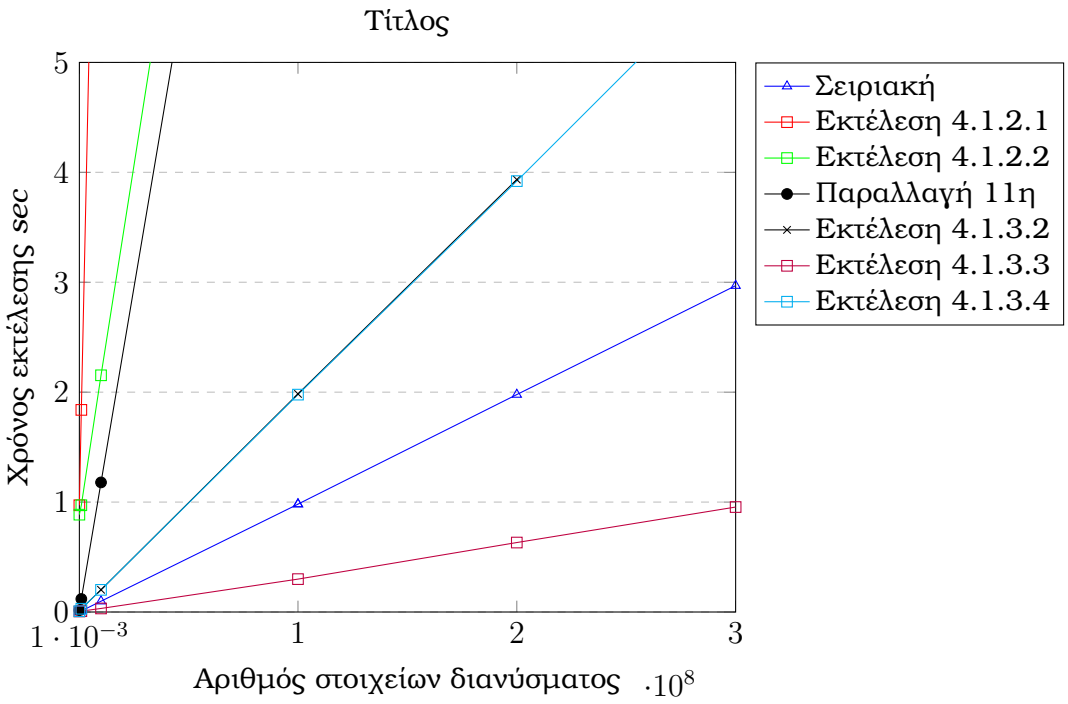
Συμβ. 19: Εκτέλεση υπολογισμών

```
void double_elements(int *A, size_t size) {
#pragma omp target teams distribute parallel for simd is_device_ptr(A)
        for (size_t i = 0; i < size; ++i) {
                A[i] = A[i] * 2;
        }
}
```

Πίνακας 17: Καταγραφή χρόνων εκτέλεσης παραδειγμάτων

Αριθμός στοιχείων πίνακα	**Χρόνος εκτέλεσης (*sec*)**
100000	
1000000	
10000000	
100000000	
200000000	
300000000	

Αποτελέσματα και σχόλια



References

[1] . 0.