0.1 Πρόσθεση διανυσμάτων αριθμών μικρής ακρίβειας - *SAXPY*

Μια από τις λειτουργίες που κατέχει θεμελιώδη θέση σε εφαρμογές της γραμμικής άλγεβρας, αποτελεί η πρόσθεση διανυσμάτων δεκαδικών αριθμών μικρής ακρίβειας (*floats*), γνωστή ως **SAXPY**.

Στο παράδειγμα *SAXPY*, ο λόγος του μεγέθους υπολογισμών προς το μέγεθος των δεδομένων που τελούν υπό επεξεργασία είναι μικρός. Ως εκτότου, αποτελεί πρόβλημα περιορισμένης επεκτασιμότητας. Παρόλα αυτά, πρόκειται για ένα χρήσιμο παράδειγμα που ανήκει στην κατηγορία προβλημάτων παραλληλοποίησης τύπου *map* και των εννοιών *uniform* και *varying parameters*[1].

0.1.1 Περιγραφή προβλήματος

Η λειτουργία SAXPY δέχεται ως δεδομένα δυο διάνυσματα δεκαδικών αριθμών. Το πρώτο διάνυσμα πολλαπλασιάζεται με μια σταθερά a και το αποτέλεσμα προστίθεται στο δεύτερο διάνυσμα y. Τα διανύσματα x,y πρέπει να έχουν το ίδιο μέγεθος.

Ο υπολογισμός αυτός εμφανίζεται συχνά στη γραμμική άλγεβρα, όπως για παράδειγμα στη διαγραφή σειρών για την απαλοιφή **Gauss**. Το όνομα *SAXPY* δόθηκε από την βιβλιοθήκη **BLAS** (*"Basic Linear Algrbra Subprograms"*) για δεκαδικούς αρθμούς μικρής ακρίβειας (*floats*). Ο αντίστοιχος αλγόριθμος διπλής ακρίβειας ονομάζεται *DAXPY*, ενώ για μιγαδικούς αριθμούς ονομάζεται *CAXPY*. Η μαθηματική διατύπωση του *SAXPY* είναι:

$$\mathbf{y} = a * \mathbf{x} + \mathbf{y}$$

όπου το διάνυσμα x χρησιμοποιείται ως είσοδος, το y ως είσοδος και έξοδος. Δηλαδή το αρχικό διάνυσμα y τροποποιείται. Εναλλακτικά, η λειτουργία *SAXPY* μπορεί να περιγραφεί ως συνάρτηση που δρα σε μεμονωμένα στοιχεία, οπως φαίνεται παρακάτω:

$$f(t, p, q) = tp + q$$

$$\forall_i : y_i \leftarrow f(a, x_i, y_i)$$

Οι συναρτήσεις τύπου f δεχονται ως ορίσματα, δύο είδη παραμέτρων. Τις παραμέτρους όπως την a που παραμένουν σταθερές και ονομάζονται *uniform*, οι παράμετροι που είναι μεταβλητές σε κάθε κλίση της f ονομάζονται *varying*. Το μοτίβο *map* καλεί τη συνάρτηση f τόσες φορές όσες και ο αριθμός των στοιχείων του διανύσματος.[1].

0.2 Περιγραφή κεντρικού τμήματος προβλήματος SAXPY

Το πρόβλημα ξεκινάει δημιουργώντας ένα στοιχείο τύπου *Containers*, που περιέχει τα διανύσματα που εισάγονται στον αλγόριθμο SAXPY. Ο ρόλος του Containers είναι για την διαχείριση της *heap* μνήμης. Τα διανύσματα και η σταθερά cons αρχικοποιούνται με τυχαίους αριθμούς μικρής ακρίβειας. Το μέγεθος των διανυσμάτων (μέγεθος προβλήματος) ορίζεται από τον χρήστη μέσω της γραμμής εντολών. Στη συνέχεια, καλείται ο αλγόριθμος SAXPY μόλις τελειώσει γίνεται επαλήθευση των αποτελεσμάτων, όπου αν επαληθευτούν σωστά, γίνεται εκτύπωση του χρόνουν εκτέλεσης της παραλλαγής.

Συμβ. 1: Κεντρικός κώδικας προβλήματος SAXPY

```
int main(int argc, char **argv) {
    Opts o;
    parseArgs(argc, argv, o);
    Containers c(o.size);
    c.setRandomValues();
    float cons = float(rand()) / float(RAND_MAX);
    auto start = omp_get_wtime();
    saxpy(c.m_size, cons, c.m_a, c.m_b);
    auto end = omp_get_wtime();
    verify(c.m_size, cons, c.m_a, c.m_b, c.m_verification);
    std::cout << "Execution Time : " << std::fixed
         << end - start << std::setprecision(5);
    std::cout << " sec " << std::endl;
    return 0;
}
```

Συμβ. 2: Κλάση *Containers*

```
struct Containers {
    explicit Containers(size_t containers_size);
    ~Containers();

    size_t m_size;
    float *m_a;
    float *m_verification;
    float *m_b;
};

Containers::Containers(size_t containers_size)
    : m_size(containers_size) {
    srand(time(nullptr));
    m_a = new float[containers_size];
    m_verification = new float[containers_size];
    m_b = new float[containers_size];
}

Containers::~Containers() {
    delete []m_a;
    delete []m_b;
    delete []m_verification;
}

Containers::setRandomValues() {
    fill_random_arr(m_a, m_size);
    fill_random_arr(m_b, m_size);
}
```

Συμβ. 3: Συνάρτηση επαλήθευσης

```
static void verify(size_t size, float c, float *a, float *b,
                   float *verification) {
    for (size_t i = 0; i < size; ++i) {
        if (abs(c * a[i] + verification[i] - b[i]) >= 10e-6) {
            std::cout << "Failed index: " << i <<
             ". " << c * a[i] + verification[i] <<
             " =! " << b[i] << std::endl;
            exit(1);
        }
    }
}
```

Συμβ. 4: Συνάρτηση αρχικοποίησης τιμών

```
static void fill_random_arr(float *arr, size_t size) {
    for (size_t k = 0; k < size; ++k) {
        arr[k] = (float)(rand()) / RAND_MAX;
    }
}
```

0.3 Σειριακή εκτέλεση

Η υλοποίηση της σειριακής παραλλαγής της συνάρτησης saxpy περιλαμβάνει έναν επαναληπτικό βρόγχο στον οποίο γίνεται ο υπολογισμός για κάθε στοιχείο των διανυσμάτων.

Συμβ. 5: Σειριακή υλοποίηση της SAXPY

```
void saxpy(size_t n, float a, const float *x, float *y) {
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

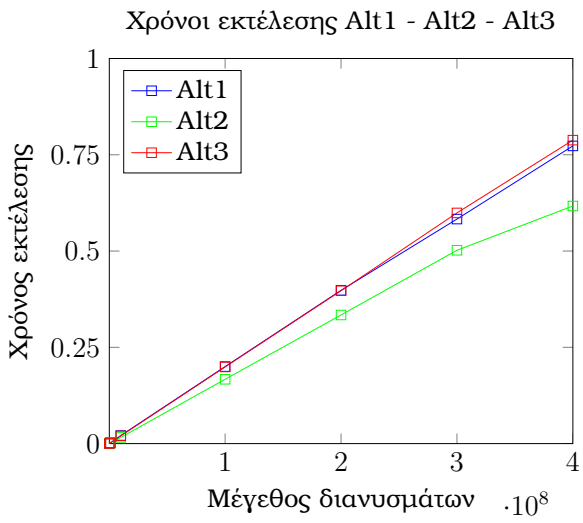
Η μεταγλώττιση έγινε με τους παρακάτω τρόπους και οι χρόνοι εκτέλεσης καταγράφονται στους πίνακες που ακολουθούν. Ο δεύτερος τρόπος μεταγλώττισης περιλαμβάνει διανυσματικοποίηση από τον μεταγλωττιστή ενώ η πρώτη όχι.

Πίνακας 1: Επιλογές μεταγλώττισης

Label	**Options**
Alt1	-fopt-info-vec=info.log -fno-inline -fno-tree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt2	-fopt-info-vec=info.log -fno-inline -ftree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt3	-fopt-info-vec=info.log -fno-inline -fopenmp -Wall -Wextra -std=c++14 -O2

Πίνακας 2: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**		
	Alt1	**Alt2**	**Alt3**
100000	0.0004	0.0001	0.0001
1000000	0.002	0.002	0.002
10000000	0.021	0.016	0.020
100000000	0.200	0.167	0.199
200000000	0.398	0.334	0.397
300000000	0.583	0.502	0.599
400000000	0.773	0.617	0.788
500000000	0.989	0.818	0.991



Σχήμα 1: Σύγκριση αποτελεσμάτων

Μέγεθος	Επιτάχυνση (%)
100000	-
1000000	-
10000000	23
100000000	17
200000000	16
300000000	13.9
400000000	20

Πίνακας 3: Ποσοστιαία σύγκριση μεταξύ Alt1 και Alt2

Η επιλογή για διανυσματικοποίηση κατά τη μεταγλώττιση επιφέρει περίπου 20% βελτίωση στις χρονικές επιδόσεις των εκτελέσεων με διαφορετικά μεγέθη διανύσματος. Ακόμη, η επιλογή μεταγλώττισης με *-fno-tree-vectorize* έχει τις ίδιες επιδόσεις με την παραλλαγή Alt3, πράγμα που επιβεβαιώνει ότι η -O2 δεν υπονοεί αυτόματη διανυσματικοποίηση. Τα αποτελέσματα αυτά θα χρησιμοποιηθούν για συγκρίσεις με τις παραλλαγές που θα ακολουθήσουν.

0.4 Παραλλαγή με οδηγία parallel for

Η πρώτη υλοποίηση παραλλαγής με παραλληλισμό της συνάρτησης saxpy περιλαμβάνει τον επαναληπτικό βρόγχο ενσωματωμένο στην οδηγία *parallel for* στον οποίο γίνεται ο υπολογισμός για κάθε στοιχείο των διανυσμάτων. Τα αποτελέσματα φαίνονται στον ακολουθούμενο πίνακα.

Συμβ. 6: Υλοποίηση παραλλαγής με parallel for

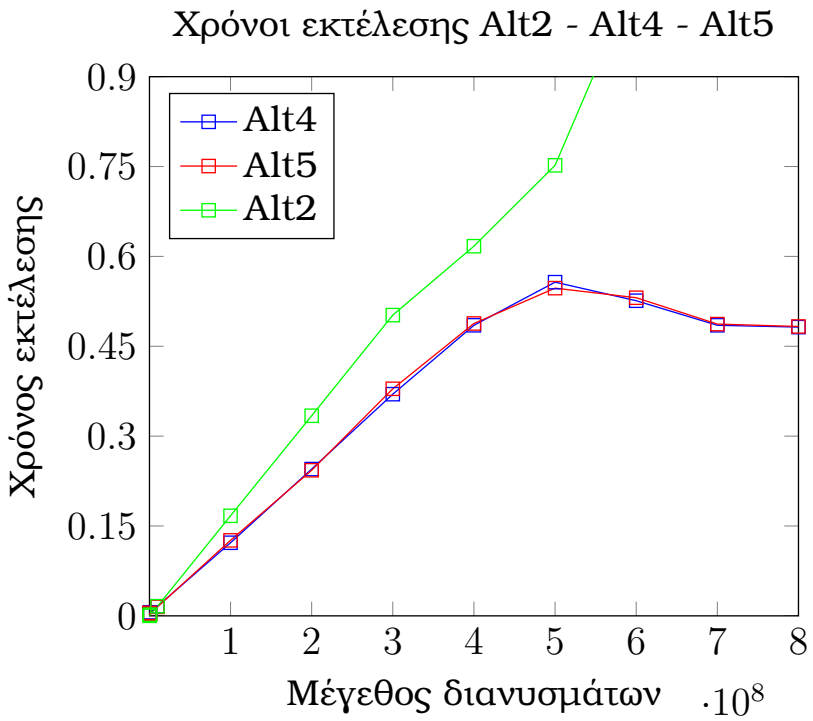
```
void saxpy(size_t n, float a, const float *x, float *y) {
    #pragma omp parallel for
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 4: Επιλογές μεταγλώττισης

Label	Options
Alt4	-fopt-info-vec=info.log -fno-inline -fno-tree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt5	-fopt-info-vec=info.log -fno-inline -ftree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2

Πίνακας 5: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	Alt4	**Alt5**
100000	0.005	0.005
1000000	0.006	0.003
10000000	0.016	0.015
100000000	0.122	0.126
200000000	0.245	0.243
300000000	0.370	0.379
400000000	0.485	0.488
500000000	0.557	0.547
600000000	0.526	0.531
700000000	0.485	0.487
800000000	0.482	0.483



0.4.1 Παρατηρήσεις

Με τη χρήση της οδηγίας *pragma omp parallel for* επετέφχθει μείωση του χρόνου εκτέλεσης του αλγορίθμου σε σύγκριση με την αντίστοιχη σειριακή παραλλαγή. Δεν προκύπτει καμία διαφοροποίηση της μεταγλώττισης με εντολή διανυσματικοποίησης ή χωρίς. Σύμφωνα ωστόσο με τον ορισμό του false sharing, υπάρχει πιθανότητα περαιτέρω βελτίωσης της παραλλαγής με οδηγία parallel for, κάτι που εξετάζεται στην επόμενη ενότητα.

0.5 Παραλλαγή με *parallel for* και padding

Σε αυτή την περίπτωση, ο αλγόριθμος παραμένει ίδιος, χρησιμοποιείται δηλαδή η οδηγία *pragma omp parallel for*. Ωστόσο στη συνάρτηση εισάγονται ως ορίσματα δομές που εμπεριέχουν μία μεταβλητή αριθμού μικρής ακρίβειας και ένα τεχνητό κενό *padding*. Το μέγεθος της είναι 64bytes και έχει ως στόχο την αποφυγή του φαινομένου **false sharing**.

Συμβ. 7: Υλοποίηση παραλλαγής με parallel for

```
void saxpy(size_t n, float a, const float64 *x, float64 *y) {
    #pragma omp parallel for
    for (size_t i = 0; i < n; ++i) {
        y[i].val = a * x[i].val + y[i].val;
    }
}
```

Πίνακας 6: Επιλογές μεταγλώττισης

Label	Options
Alt6	-fopt-info-vec=info.log -fno-inline -fno-tree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt7	-fopt-info-vec=info.log -fno-inline -ftree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2

0.5.1 Παρατηρήσεις

Το πρόβλημα *false sharing* δεν ήταν δυνατό να εντοπισθεί στη συγκεκριμένη παραλλαγή. Μάλιστα, ο έλεγχος για μεγέθη διανυσμάτων μεγαλύτερων των 1e8 στοιχείων, ήταν ανεπιτυχής λόγω έλλειψης υπολογιστικών πόρων.

Πίνακας 7: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	Alt6	**Alt7**
100000	0.006	0.006
1000000	0.023	0.025
10000000	0.193	0.198
100000000	killed	killed

0.6 Παραλλαγές με χρήση οδηγίας *SIMD*

Η συγκεκριμένη ενότητα καθώς και ορισμένες που ακολουθούν αφορούν την επίλυση του προβλήματος *SAXPY* με χρήση της οδηγίας διανυσματικοποίησης **SIMD**. Οπως προαναφέρθηκε, η οδηγία δεν έχει ως στόχο την παραλληλοποίηση τμήματος κώδικα, αλλά την ταυτόχρονη εκτέλεση εντολών ως μία εντολή **SIMD**. Στη περίπτωση του g++, γίνεται αυτόματη προσπάθεια διανυσματικοποίησης χρησιμοποιείται η επιλογή -O3. Αυτός ειναι και ο λόγος που στα προηγούμενα παραδείγματα χρησιμοποιήθηκε επιλογή μεταγλώττισης -O2 με ταυτόχρονη χρήση των εντολών **-fno-tree-vectorize** και **ftree-vectorize** που επιτρέπουν στο χρήστη να επιλέγει μεταγλώττιση με διανυσματικοποίηση ή χωρίς. Παράλληλα, χρησιμοποιούνται οι εντολές **-fno-inline** και **-fopt-info-vec**. Η πρώτη απογορεύει στον μεταγγλωτιστή το loop-unrolling ενώ η δεύτερη δημιουργεί ένα αρχείο με χρήσιμα μηνύματα κατά τη διάρκεια της μεταγλώττισης.

0.6.1 Παραλλαγή με *omp simd*

Η εντολή *pragma omp simd* αποσκοπεί στη διανυσματικοποίηση του τμήματος κώδικα που ακολουθεί, χωρίς ωστόσο να γίνεται διαμοιρασμός των επαναλήψεων του βρόγχου σε διαφορετικά νήματα όπως θα γίνόταν με την οδηγία *pragma omp parallel for simd*.

Συμβ. 8: Υλοποίηση παραλλαγής με omp simd

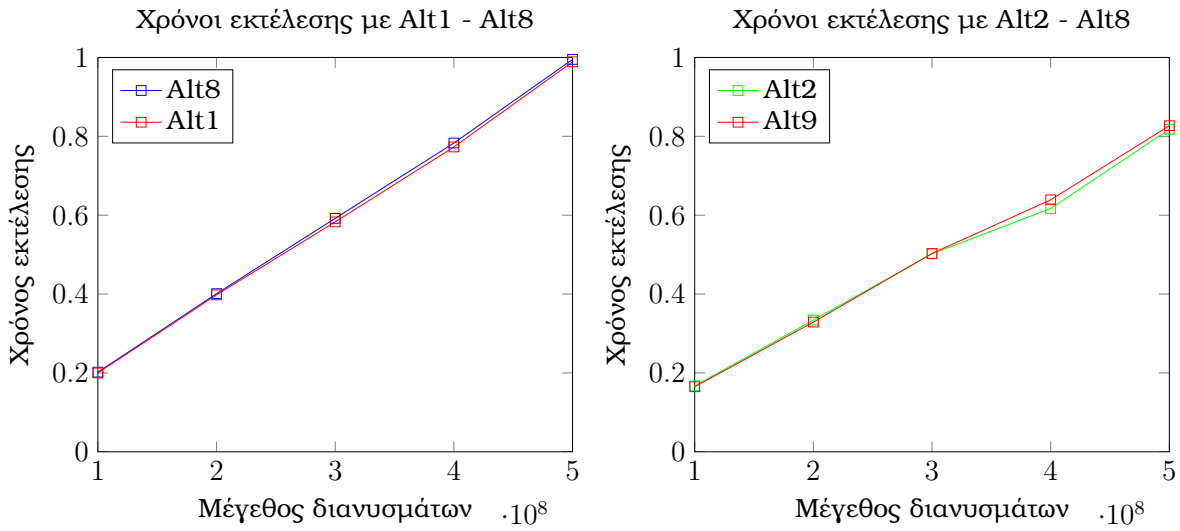
```
void saxpy(size_t n, float a, const float *x, float *y) {
    #pragma omp simd
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 8: Επιλογές μεταγλώττισης

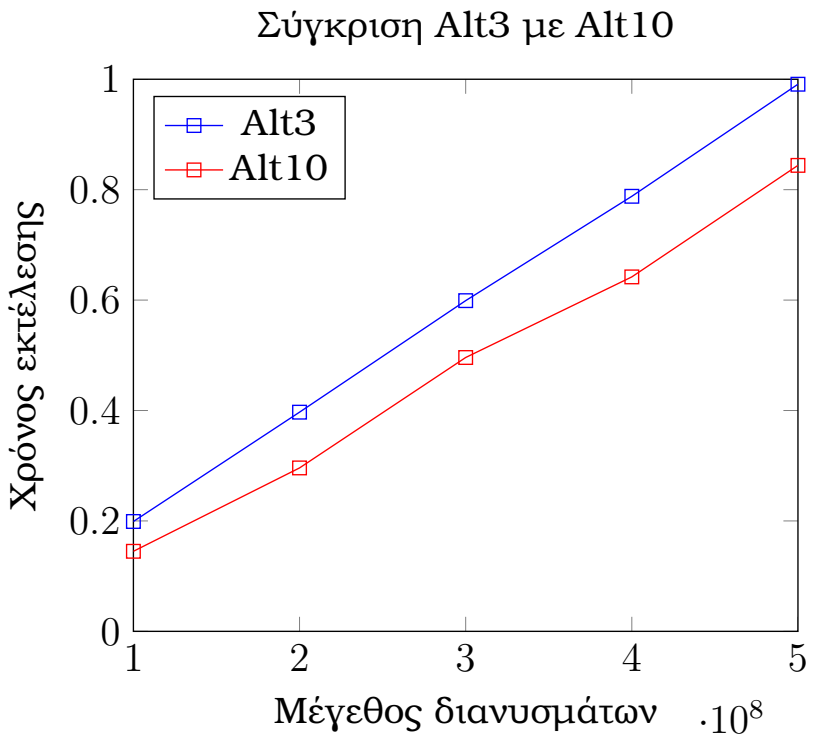
Label	**Options**
Alt8	-fopt-info-vec=info.log -fno-inline -fno-tree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt9	-fopt-info-vec=info.log -fno-inline -ftree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt10	-fopt-info-vec=info.log -fno-inline -fopenmp -Wall -Wextra -std=c++14 -O2

Πίνακας 9: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**		
	Alt8	**Alt9**	**Alt10**
100000	0.001	0.001	0.001
1000000	0.002	0.002	0.002
10000000	0.021	0.017	0.017
100000000	0.202	0.165	0.145
200000000	0.401	0.329	0.296
300000000	0.592	0.503	0.496
400000000	0.783	0.639	0.642
500000000	0.995	0.827	0.844



Σχήμα 2: Λεφτ: Νο Ιντεραστιον. Ριγητ: Ιντεραστιον



Από τα παραπάνω διαγράμματα διαφαίνεται ότι η μεταγλώττιση με επιλογή *-fno-tree-vectorize* και *-ftree-vectorize* παρακάμπτει την οδηγία omp simd και εκτελείται ως σειριακή, με διανυσματικοποίηση ή χωρίς, ανάλογα με την επιλογή. Στη περίπτωση που δε δοθεί η επιλογή ωστόσο, τότε διανυσματικοποίηση εφαρμόζεται μέσω του *OpenMP* και της οδηγίας *omp simd* αν υπάρχει

Πίνακας 10: Συνοπτικός πίνακας εφαρμογών διανθσματικοποίησης

Επιλογή μεταγλώττισης	**Σειριακή**	**OpenMP - omp simd**
-fno-tree-vectorize	Οχι	Οχι
-ftree-vectorize	Ναι	Ναι
None	Οχι	Ναι

0.6.2 Παραλλαγή με *omp parallel for simd*

Στην παραλλαγή αυτής της ενότητας χρησιμοποιείται ο συνδυασμός παραλληλοποίησης μέσω της οδηγίας *parallel for* με διανυσματικοποίηση μέσω *simd*.

Συμβ. 9: Υλοποίηση παραλλαγής με omp parallel for simd

```
void saxpy(size_t n, float a, const float *x, float *y) {
    #pragma omp parallel for simd
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

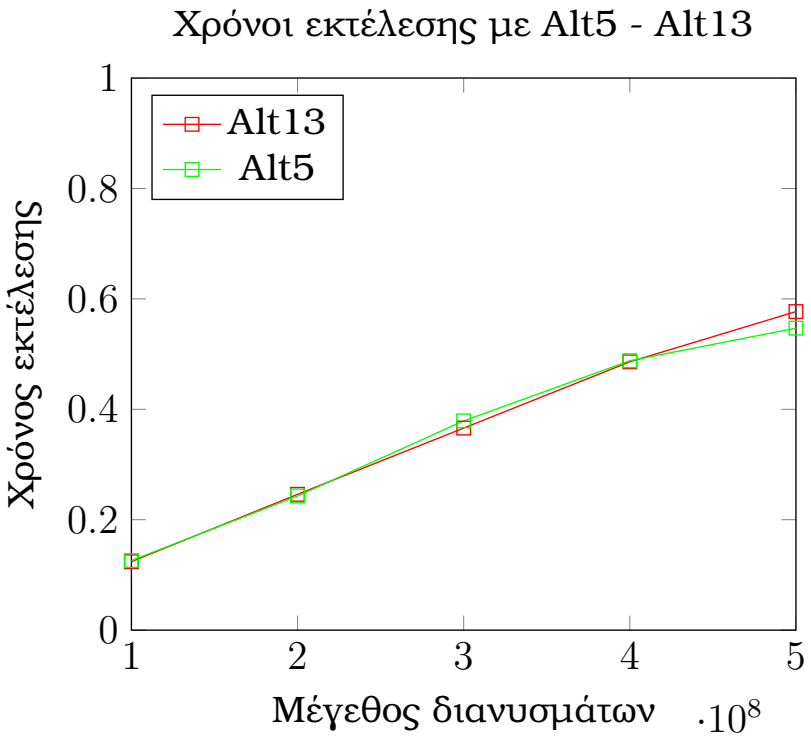
Πίνακας 11: Επιλογές μεταγλώττισης

Label	Options
Alt11	-fopt-info-vec=info.log -fno-inline -fno-tree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt12	-fopt-info-vec=info.log -fno-inline -ftree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt13	-fopt-info-vec=info.log -fno-inline -fopenmp -Wall -Wextra -std=c++14 -O2

Πίνακας 12: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**		
	Alt11	**Alt12**	**Alt13**
100000	0.006	0.002	0.005
1000000	0.006	0.002	0.004
10000000	0.015	0.016	0.016
100000000	0.129	0.123	0.124
200000000	0.247	0.251	0.246
300000000	0.368	0.368	0.366
400000000	0.489	0.486	0.486
500000000	0.578	0.576	0.577
500000000	0.578	0.576	0.577
600000000	0.515	0.458	0.525
700000000	0.496	0.496	0.460
800000000	0.487	0.500	0.483

Από τις παραπάνω εκτελέσεις του αλγόριθμου προκύπτει το συμπέρασμα ότι η οδηγία διανυσματικοποίησης μέσω simd δεν λαμβάνεται υπόψη όταν εφαρμόζεται σε συνδυασμό με την οδηγία parallel for.


Χρόνοι εκτέλεσης με Alt5 - Alt13
Alt13
Alt5
Χρόνος εκτέλεσης
0
0.2
0.4
0.6
0.8
1
1
2
3
4
5
Μέγεθος διανυσμάτων
$\cdot 10^8$

0.6.3 Παραλλαγή με *omp declare simd uniform*

Υλοποίηση παραλλαγής με χρήση της φράσης *uniform*. Θεωρητικά δεν υπάρχει κέρδος στις επιδόσεις του αλγορίθμου σε σχέση με της προηγούμενες παραλλαγές. Η εναλλακτική αυτη χρησιμοποιείται για την επαλήθευση της διανυσματικοποίησης μέσω *OpenMP*.

Συμβ. 10: Υλοποίηση παραλλαγής με omp declare simd uniform

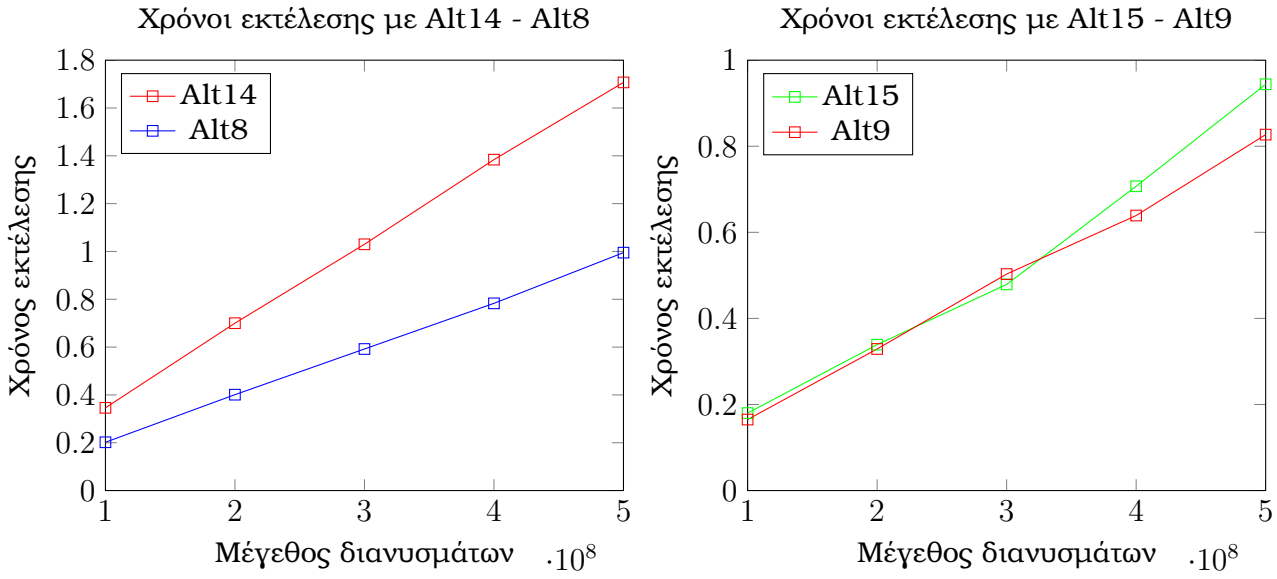
```
#pragma omp declare simd uniform(a)
float do_work(float a, float b, float c)
{
    return a * b + c;
}

void saxpy(size_t n, float a, const float *x, float *y) {
    #pragma omp simd
    for (size_t i = 0; i < n; ++i) {
        y[i] = do_work(a, x[i], y[i]);
    }
}
```

Πίνακας 13: Επιλογές μεταγλώττισης

Label	Options
Alt14	-fopt-info-vec=info.log -fno-inline -fno-tree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt15	-fopt-info-vec=info.log -fno-inline -ftree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt16	-fopt-info-vec=info.log -fno-inline -fopenmp -Wall -Wextra -std=c++14 -O2

Πίνακας 14: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**		
	Alt14	**Alt15**	**Alt16**
100000	0.0003	0.0001	0.0003
1000000	0.003	0.002	0.002
10000000	0.035	0.018	0.018
100000000	0.346	0.180	0.179
200000000	0.700	0.339	0.318
300000000	1.030	0.479	0.469
400000000	1.384	0.707	0.637
500000000	1.707	0.944	0.911



Από τα διαγράμματα προκύπτει ότι ο υπολογισμός SAXPY μέσω συνάρτησης, προσδίδει εξτρά καθυστέρηση στην συνολική απόδοση, που πιθανόν οφείλεται στη μετακίνηση της διεύθυνσης μνήμης καθώς κατά τη μεταγλώττιση απενεργοποιείται το *loop unrolling*. Στη περίπτωση της explicit εντολής διανυσματικοποίησης κατά τη μεταγλώττιση, οι επιδόσεις δείχνουν να μοιάζουν, με λίγο καλύτερες επιδόσεις για την εκτέλεση χωρίς ενδιάμεση κλήση συνάρτησης.

0.6.4 Παραλλαγή με *omp declare simd uniform notinbranch*

Στη περίπτωση που ακολουθεί, γίνεται προσπάθεια εξαγωγής συμπερασμάτων σχετικών με τη φράση *notinbranch* και το κέρδος που μπορεί να επιφέρει σε σύγκριση με τη προηγούμενη παραλλαγή.

Συμβ. 11: Υλοποίηση παραλλαγής με omp declare simd uniform notinbranch

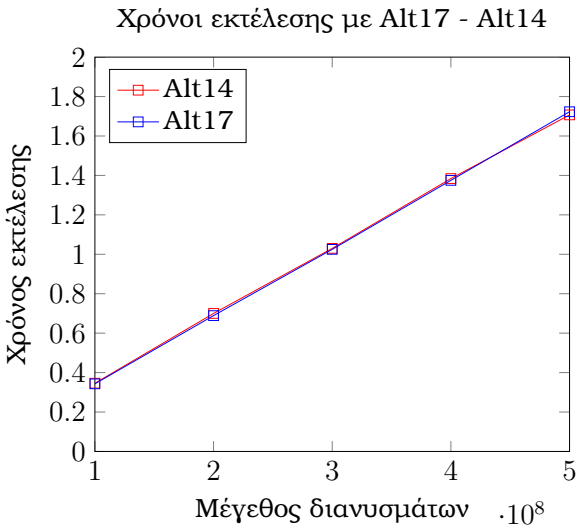
```
#pragma omp declare simd uniform(a) notinbranch
float do_work(float a, float b, float c)
{
    return a * b + c;
}

void saxpy(size_t n, float a, const float *x, float *y) {
    #pragma omp simd
    for (size_t i = 0; i < n; ++i) {
        y[i] = do_work(a, x[i], y[i]);
    }
}
```

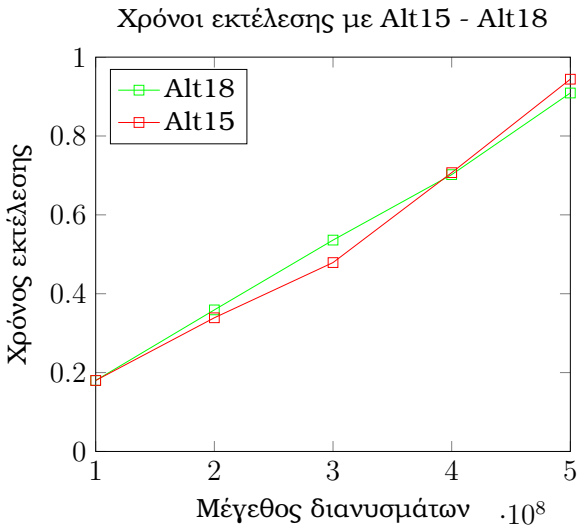
Πίνακας 15: Επιλογές μεταγλώττισης

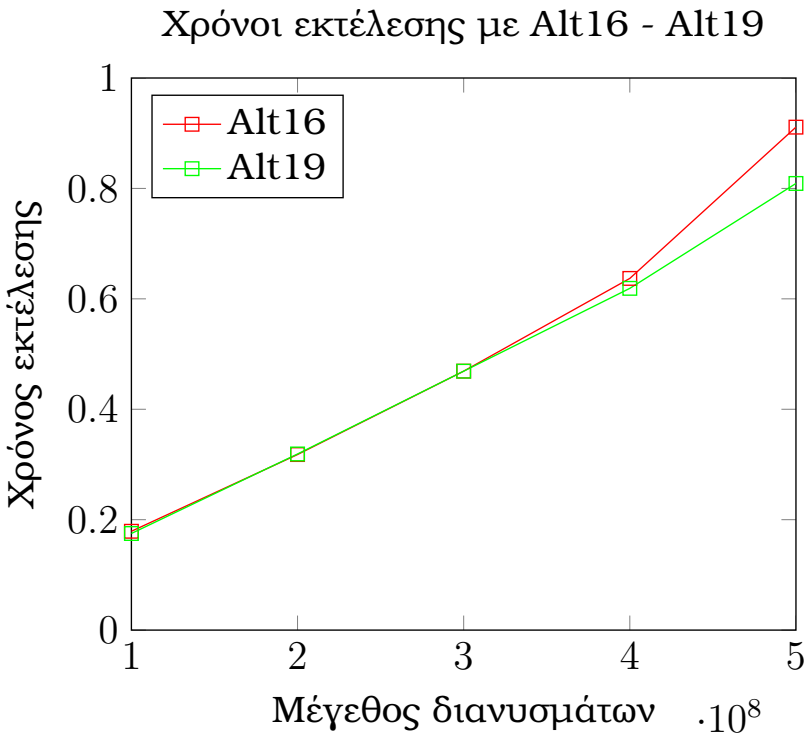
Label	**Options**
Alt17	-fopt-info-vec=info.log -fno-inline -fno-tree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt18	-fopt-info-vec=info.log -fno-inline -ftree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt19	-fopt-info-vec=info.log -fno-inline -fopenmp -Wall -Wextra -std=c++14 -O2

Πίνακας 16: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**		
	Alt17	**Alt18**	**Alt19**
100000	0.001	0.001	0.001
1000000	0.003	0.002	0.002
10000000	0.035	0.018	0.018
100000000	0.343	0.180	0.175
200000000	0.689	0.359	0.319
300000000	1.025	0.536	0.469
400000000	1.375	0.702	0.619
500000000	1.723	0.909	0.809






Χρόνοι εκτέλεσης με Alt16 - Alt19
Alt16
Alt19
Χρόνος εκτέλεσης
0
0.2
0.4
0.6
0.8
1
1
2
3
4
5
Μέγεθος διανυσμάτων
.10^8

0.7 Παραλλαγές με *offloading*

Η ομάδα παραλλαγών αυτής της ενότητας, αφορά τη μεταφορά του αλγορίθμου σε άλλο μέσο για την επίλυση του. Το βασικότερο τμήμα του *SAXPY*, εκτελείται στη μονάδα επεξεργασίας κάρτας γραφικών - *GPU*. Στα πλαίσια της μεταφοράς, συμπεριλαμβάνεται και η μεταφορά των μεταβλητών από τη μνήμη της κεντρικής μονάδας στη μνήμη της κάρτας γραφικών. Αυτή η μεταφορά γίνεται με διάφορους τρόπου και τεχνικές που αναφέρονται στις επόμενες παραγράφους.

0.7.1 Παραλλαγή με *target map*

Στη συγκεκριμένη παραλλαγή γίνεται απλή μεταφορά του κώδικα και των μεταβλητών στην κάρτα γραφικών για την εκτέλεση του *SAXPY* σε αυτή.

Συμβ. 12: Υλοποίηση παραλλαγής με target map

```
void saxpy(size_t n, float a, const float *x, float *y) {
    #pragma omp target map(tofrom: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 17: Επιλογές μεταγλώττισης

Label	**Options**
Alt20	-fopt-info-vec=info.log -fno-inline -fno-tree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2
Alt21	-fopt-info-vec=info.log -fno-inline -ftree-vectorize -fopenmp -Wall -Wextra -std=c++14 -O2

Πίνακας 18: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	Alt20	**Alt21**
100000	0.881	0.873
1000000	1.200	1.169
10000000	3.848	3.752
100000000	29.984	29.937
200000000	59.055	59.894

Παρόλο που η συγκρεκριμένη παραλλαγή προσομοιώνει σειριακή εκτέλεση στη μονάδα επεξεργασίας της κάρτας γραφικών, είναι προφανές ότι η δημιουργία ενός νέου περιβάλλοντος δεδομένων, με τα απαραίτητα δεδομένα για την εκτέλεση του προβλήματος και η εκτέλεσή του στη μονάδα επεξεργασίας γραφικών μέσω *OpenMP*, αποτελεί μια χρονοβόρα διαδικασία, πολύ πιο χρονοβόρα από τη σειριακή εκτέλεση στη κεντρική μονάδα επεξεργασίας *CPU*.

0.8 Παραλλαγή με *target simd map*

Συμβ. 13: Υλοποίηση παραλλαγής με target simd map

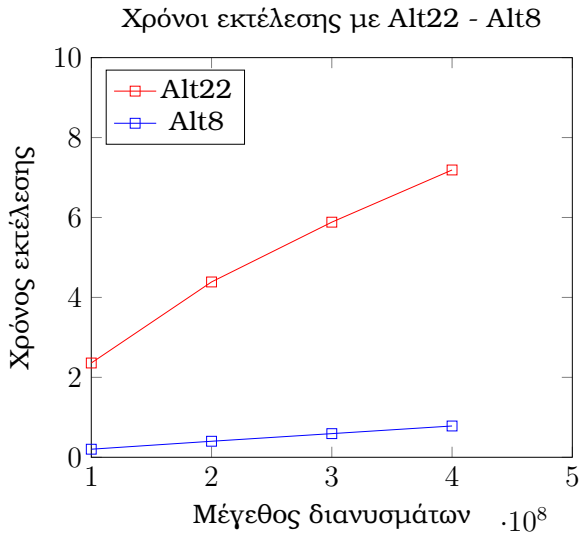
```
void saxpy(size_t n, float a, const float *x, float *y) {
    #pragma omp target simd map(tofrom: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

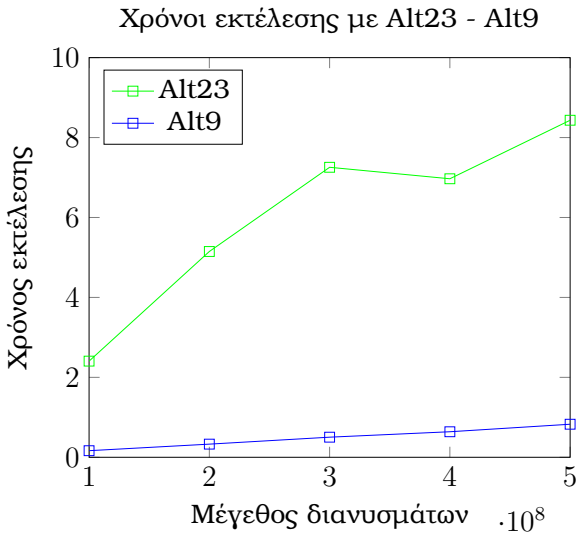
Πίνακας 19: Επιλογές μεταγλώττισης

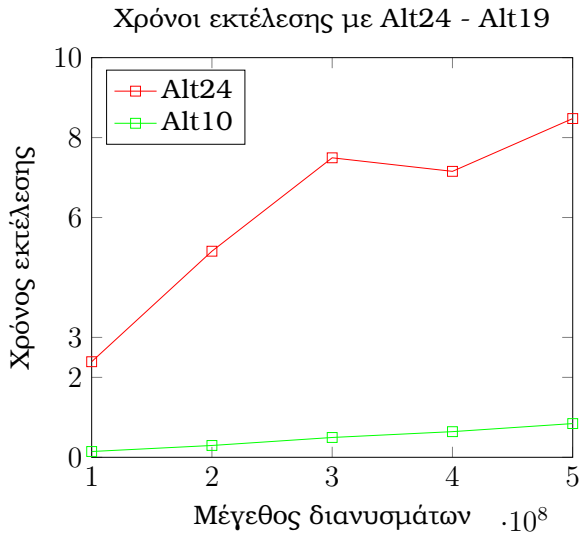
Label	Options
Alt22	-fopt-info-vec=builds/alt22.log -O2 -fno-tree-vectorize -fno-inline -fno-stack-protector -foffload=nvptx-none="-O2 -fno-tree-vectorize -fno-inline" -fopenmp -o ./builds/Alt22
Alt23	-fopt-info-vec=builds/alt23.log -O2 -ftree-vectorize -fno-inline -fno-stack-protector -foffload=nvptx-none="-O2 -ftree-vectorize -fno-inline" -fopenmp -o ./builds/Alt23
Alt24	-fopt-info-vec=builds/alt24.log -O2 -fno-inline -fno-stack-protector -foffload=nvptx-none="-O2 -fno-inline" -fopenmp -o ./builds/Alt24

Πίνακας 20: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**		
	Alt22	**Alt23**	**Alt24**
100000	0.888	0.809	0.837
1000000	0.841	0.837	0.833
10000000	1.018	1.018	0.997
100000000	2.359	2.406	2.392
200000000	4.387	5.149	5.157
300000000	5.883	7.256	7.494
400000000	7.189	6.969	7.155







0.9 Παραλλαγή με *target parallel for*

Συμβ. 14: Υλοποίηση παραλλαγής με target parallel for

```
void saxpy(size_t n, float a, const float *x, float *y) {
#pragma omp target parallel for map(tofrom: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 21: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	-O0	**-O3**
100000	0.011	0.011
1000000	0.009	0.013
10000000	0.035	0.021
100000000	0.150	0.127
200000000	0.264	0.251
300000000	0.387	0.369
400000000	0.500	0.490

0.10 Παραλλαγή με *target parallel for simd*

Συμβ. 15: Υλοποίηση παραλλαγής με target parallel for simd

```
void saxpy(size_t n, float a, const float *x, float *y) {
#pragma omp target parallel for simd map(tofrom: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 22: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**			
	-O0	**-O0 -fopenmp-simd**	**-O3**	**-O3 -fopenmp-simd**
100000	0.010	0.010	0.011	0.011
1000000	0.012	0.014	0.011	0.009
10000000	0.027	0.026	0.020	0.020
100000000	0.140	0.154	0.128	0.130
200000000	0.257	0.271	0.249	0.247
300000000	0.378	0.385	0.370	0.365
400000000	0.513	0.505	0.450	0.489

0.11 Παραλλαγή με *target teams map*

Συμβ. 16: Υλοποίηση παραλλαγής με target teams map

```
void saxpy(size_t n, float a, const float *x, float *y) {
    #pragma omp target teams map(tofrom: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 23: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	-O0	**-O3**
100000	0.005	0.004
1000000	0.016	0.006
10000000	0.132	0.025
100000000	1.186	0.221
200000000	2.374	0.420
300000000	3.543	0.652
400000000	4.729	0.821

0.12 Παραλλαγή με *target teams distribute map*

Συμβ. 17: Υλοποίηση παραλλαγής με target teams distribute map

```
void saxpy(size_t n, float a, const float *x, float *y) {
#pragma omp target teams distribute map(from: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 24: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	-O0	**-O3**
100000	0.005	0.004
1000000	0.015	0.006
10000000	0.117	0.025
100000000	1.142	0.214
200000000	2.278	0.428
300000000	3.433	0.625
400000000	4.575	0.851

0.13 Παραλλαγή με *target teams distribute parallel for map*

Συμβ. 18: Υλοποίηση παραλλαγής με target teams distribute parallel for

```
void saxpy(size_t n, float a, const float *x, float *y) {
#pragma omp target teams distribute parallel for map(from: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 25: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	-O0	**-O3**
100000	0.010	0.011
1000000	0.016	0.011
10000000	0.023	0.020
100000000	0.139	0.127
200000000	0.257	0.250
300000000	0.389	0.369
400000000	0.511	0.490

0.14 Παραλλαγή με *target teams distribute simd map*

Συμβ. 19: Υλοποίηση παραλλαγής με target teams distribute simd map

```
void saxpy(size_t n, float a, const float *x, float *y) {
#pragma omp target teams distribute simd map(from: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 26: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**			
	-O0	**-O0 -fopenmp-simd**	**-O3**	**-O3 -fopenmp-simd**
100000	0.005	0.050	0.004	0.004
1000000	0.015	0.015	0.006	0.006
10000000	0.120	0.119	0.021	0.021
100000000	1.159	1.153	0.159	0.168
200000000	2.311	2.305	0.326	0.327
300000000	3.487	3.472	0.482	0.495
400000000	4.620	4.610	0.647	0.644

0.15 Παραλλαγή με *target teams distribute parallel for simd map*

Συμβ. 20: Υλοποίηση παραλλαγής με teams distribute parallel for simd map

```
void saxpy(size_t n, float a, const float *x, float *y) {
#pragma omp target teams distribute parallel for simd\
                        map(from: y[0:n]) map(to: x[0:n])
    for (size_t i = 0; i < n; ++i) {
        y[i] = a * x[i] + y[i];
    }
}
```

Πίνακας 27: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	-O0	**-O3**
100000	0.010	0.010
1000000	0.012	0.010
10000000	0.025	0.021
100000000	0.146	0.127
200000000	0.263	0.246
300000000	0.389	0.371
400000000	0.519	0.489

References

[1] J. R. Michael McCool, Arch D.Robison. *Structural Parallel Programming*, pages 124–125. Morgan Kaufmann, 2012.