shipout/backgroundshipout/foreground

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*[**;**].

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.[;].

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];
    }
}
```

Οι χρόνοι εκτέλεσης του αλγορίθμοι συναρτήσει του μεγέθους του προβλήματος παρατίθενται στον παρακάτων πίνακα. Το πρόγραμμα μεταγλωττίστηκε με επιλογή -O3 και -O0.

Πίνακας 1: Καταγραφή χρόνων εκτέλεσης

Μέγεθος προβλήματος	**Χρόνοι εκτέλεσης (sec)**	
	-O0	**-O2**
100000	0.0012	0.00011
1000000	0.0118	0.00171
10000000	0.1179	0.01631
100000000	1.1821	0.16124
200000000	2.3612	0.31867
300000000	3.5510	0.42806