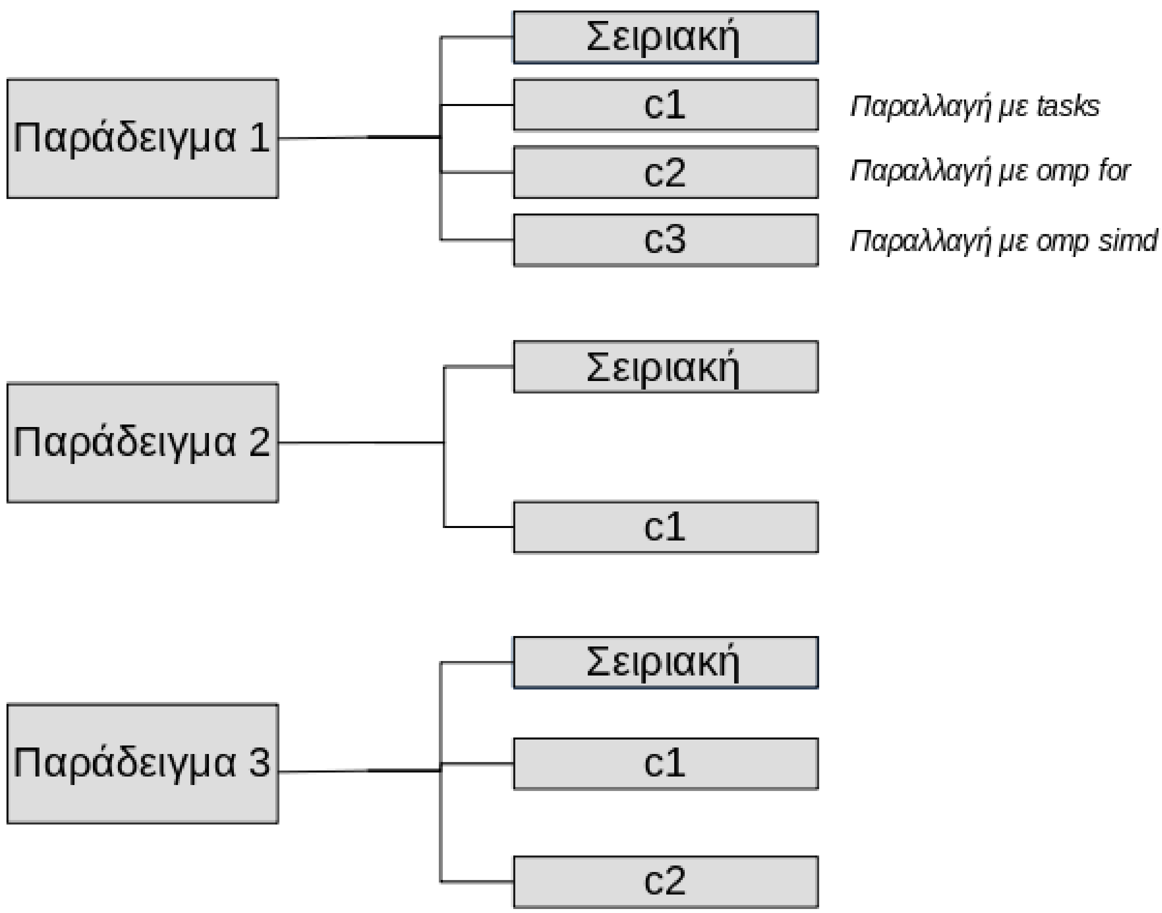
Υλοποιημένα παραδείγματα[1]

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 σε επίπεδο χρονικής επίδοσης. Παράλληλα θα σχολιαστούν επιμέρους οι διάφορες παραλλαγές επίλυσης του κάθε προβλήματος. Ο πηγαίος βρίσκεται συγκεντρωμένος 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.com*

Αναφορά αρχιτεκτονικής μηχανήματος

Τα προβλήματα που ακολουθούν εκτελέστηκαν σε μηχάνημα λειτουργικό *linux* και μεταγλωττιστή *gcc*. Οι προδιαγραφές υλικού του μηχανήματος που εκτελέστηκαν τα προβλήματα, αναφέρονται στο παρακάτω παράδειγμα:

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 υπολογισμού π

Στο επόμενο παράδειγμα ακολουθεί ο υπολογισμός του αριθμού π. Το πρόβλημα ανάγεται στον υπολογισμό του παρακάτω ολοκληρώματος, με τη χρήση αριθμητικών μεθόδων:

$$\pi = \int_0^1 \frac{4.0}{(1+x^2)}\,dx$$

που υπολογίζεται αριθμητικά ως:

$$\pi \approx \sum_{k=1}^{N} F(xi)\Delta x$$

Το πρόβλημα δέχεται ως παράμετρο τον αριθμό των βημάτων της αριθμητικής ολοκλήρωσης. Όσο πιο μεγάλος είναι ο αριθμός των βημάτων, τόσο πιο ακριβής είναι και ο υπολογισμός του π.

Σειριακή εκτέλεση

Η σειριακή υλοποίηση του υπολογισμού π με χρήση αριθμητικών μεθόδων, αποτελείται από ένα βρόγχο επανάληψης. Σε κάθε επανάληψη του οποίο υπολογίζεται ένα μικρό τμήμα του συνολικού ολοκληρώματος, το ίχνος του οποίο είναι ίσο με $1/num\_steps$, όπως φαίνεται παρακάτω:

Συμβ. 1: Υλοποίηση σειριακής έκδοσης υπολογισμού π

```
double pi(long num_steps) {
    int upper_limit = 1;
    double step = upper_limit/(double)num_steps;
    double sum = .0, pi = .0;

    for (int i = 0; i < num_steps; ++i)
    {
        double x = (i + 0.5) * step;
        sum += 4.0 / (1.0 + x*x);
    }
    pi = step * sum;

    return pi;
}
```

Πίνακας 2: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*)**
100000000	2.812
200000000	5.633
300000000	8.469
400000000	11.592

1.2.1.1 Σχόλιο:

Ο συγκεκριμένος αλγόριθμος λόγω του βρόγχου επανάληψης και του αθροίσματος των δεδομένων σε μια μεταβλητή, επιδέχεται πολλών παραλλαγών που υλοποιούνται στις επόμενες παραγράφους.

Παραλλαγή 1η

Στη συγκεκριμένη περίπτωση, ο αλγόριθμος παραλληλοποιείται με τη χρήση της οδηγίας *pragma omp parallel*. Η επαναλήψεις του βρόγχου διαμοιράζονται στα νήματα και τα απότελεσματα των υπολογισμών του κάθε νήματος αποθηκεύονται στη σχετική θέση ενός διανύσματος. Ο τελικός υπολογισμός γίνεται σειριακά, με τη χρήση του διανύσματος αυτού.

Συμβ. 2: Υπολογισμός παραλλαγής 1

```
double pi(long num_steps) {
    double pi = .0;
    int num_threads = 0;
#pragma omp parallel shared(num_threads)
    {
            int id = omp_get_thread_num();
            if (id == 0) {
                    num_threads = omp_get_num_threads();
            }
    }
    particle *sum = new particle[num_threads];
    for (int i = 0; i < num_threads; ++i) {
        sum[i].val = 0.0;
    }
    double step= 1.0/(double)num_steps;
#pragma omp parallel
    {
        int thread_num = omp_get_thread_num();
        int numthreads = omp_get_num_threads();

        int low = num_steps * thread_num / numthreads;
        int high = num_steps * (thread_num + 1)/ numthreads;

        for (int i = low; i < high; ++i) {
            double x = (i + 0.5)*step;
            sum[thread_num].val += 4.0/(1.0 + x*x);
        }
    }

    for (int i = 0; i < num_threads; ++i) {
        pi += sum[i].val * step;
    }
    delete []sum;
    return pi;
}
```

Συμβ. 3: Ορισμός δομής particle

```
        struct particle
{
        double val;
};
‘
```

Πίνακας 3: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*)**
100000000	2.18
200000000	4.24
300000000	6.44
400000000	8.5

1.2.2.1 Σχόλιο:

Παρατηρείται κακή επίδοση του αλγόριθμου οφείλεται στο φαινόμενο που ονομάζεται ***false sharing*** και αναφέρθηκε στα προηγούμενα κεφάλαιο. Για την αποτροπή του, θα πρέπει η δομή *particle* να είναι μεγαλύτερου μεγέθους από 64 byte, όσο δηλαδή είναι και το μέγεθος της μνήμης *cache*.

Παραλλαγή 2η

Σε συνέχεια της προηγούμενης παραλλαγής, χρησιμοποιείται ένα τεχνητό κενό ανάμεσα στα στοιχεία *val* του διανύσματος που αποθηκεύουν τους υπολογισμούς του κάθε νήματος, για την αποφυγή του φαινομένου *false sharing*. Ο κώδικας υπολογισμού του π παραμένει ο ίδιος, όμως η δομή που θα χρησιμοποιηθεί είναι η παρακάτω.

Συμβ. 4: Δομή particle με τεχνητό κενό

```
struct particle
{
        double val;
        double pad1;
        double pad2;
        double pad3;
        double pad4;
        double pad5;
        double pad6;
        double pad7;
};
```

Πίνακας 4: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*)**
100000000	0.19
200000000	0.36
300000000	0.53
400000000	0.7

1.2.3.1 Σχόλιο:

Παρατηρείται σημαντική βελτίωση σε σύγκριση με την παραλλαγή 1. Το μέγεθος της δομής *particle* είναι ίσο με 64 bytes, όσο δηλαδή είναι και το μέγεθος της μνήμης *cache* στη συγκεκριμένη αρχιτεκτονική.

Παραλλαγή 3η

Στη συγκεκριμένη παραλλαγή γίνεται χρήση της οδηγίας ***pragma omp atomic*** που εξασφαλίζει την αποφυγή του φαινομένου *race condition*, που προκαλείται όταν πολλά νήματα τροποποιούν την ίδια θέση διεύθυνση μνήμης ταυτόχρονα.

Συμβ. 5: Υλοποίηση παραλλαγής 3

```
double pi(long num_steps) {
    int nthreads = 0;
    double pi = .0;
    double step= 1.0/(double)num_steps;
#pragma omp parallel
    {
        int id = omp_get_thread_num();
        int nthrds = omp_get_num_threads();
        double sum = 0.0, x = 0.0;

        if (id == 0) nthreads = nthrds;

        for (int i = id; i < num_steps; i += nthreads) {
            x = (i + 0.5)*step;
            sum += 4.0/(1.0 + x*x);
        }
        sum *= step;
#pragma omp atomic
        pi += sum;
    }

    return pi;
}
```

1.2.4.1 Σχόλιο:

Η χρήση της οδηγίας *atomic* διευκολύνει την υλοποίηση του αλγορίθμου καθώς δεν απαιτείται η δημιουργία διανύσματος μεταβλητών, όπου κάθε νήμα θα αποθηκεύει τα αποτελέσματα των υπολογισμών σε συγκεκριμένη θέση μνήμης.

Πίνακας 5: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*)**
100000000	0.207
200000000	0.387
300000000	0.579
400000000	0.761

Παραλλαγή 4η

Η παραλλαγή 4 υλοποιήθηκε με τη χρήση της φράσης ***reduction***.

Συμβ. 6: Υλοποίηση παραλλαγής 4

```
    double pi(long num_steps, int num_threads) {
    double pi = .0;
    double step= 1.0/(double)num_steps;
    double sum = 0.0;
    omp_set_num_threads(num_threads);

#pragma omp parallel
    {
        double x = 0.0;

#pragma omp for reduction(+:sum)
        for (int i = 0; i < num_steps; i++) {
            x = (i + 0.5)*step;
            sum += 4.0/(1.0 + x*x);
        }
    }
    pi = step * sum;

    return pi;
}
```

Πίνακας 6: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*)**
100000000	0.027
200000000	0.052
300000000	0.071
400000000	0.0901

Παραλλαγή 5η

Η παραλλαγή 5 υλοποιήθηκε με τη χρήση διεργασιών. Οπως αναφέρθηκε στα προηγούμενα κεφάλαια, η κατασκευή των διεργασιών γίνεται από ένα μοναδικό νήμα, ενώ η εκτέλεση τους από πολλαπλά νήματα ταυτόχρονα.

Συμβ. 7: Δημιουργία διεργασιών

```
int main(int argc, char **argv) {
    Opts o;
    parseArgs(argc, argv, o);
    auto seconds = omp_get_wtime();
    double step = 1.0/(double)o.num_steps;
    double sum = 0.0;
    double p = 0.0;
#pragma omp parallel
    {
#pragma omp single
        sum = pi_comp(0, o.num_steps, step);
    }
    p = step * sum;
        double time_elapsed = omp_get_wtime() - seconds;
    std::cout << "Elapsed Time: " << time_elapsed << std::endl;
    std::cout << "pi Value: " << p << std::endl;
    return 0;
}
```

Πίνακας 7: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*) (*MINBLK: 10000000*)**
100000000	0.372117
200000000	0.736442
300000000	1.09589
400000000	1.46092

Συμβ. 8: Υλοποίηση υπολογισμού π με διεργασίες

```
#define MIN_BLK 1000000

double pi_comp(int Nstart, int Nfinish, double step) {
    double x = 0.0;
    double sum = 0.0, sum1 = 0.0, sum2 = 0.0;

    if (Nfinish - Nstart < MIN_BLK) {
        for (int i = Nstart; i < Nfinish; ++i) {
            x = (i + 0.5) * step;
            sum += 4.0/(1.0 + x*x);
        }
    } else {
        int iblk = Nfinish-Nstart;
#pragma omp task shared(sum1)
        sum1 = pi_comp(Nstart, Nfinish-iblk/2, step);
#pragma omp task shared(sum2)
        sum2 = pi_comp(Nfinish-iblk/2, Nfinish, step);
#pragma omp taskwait
        sum = sum1 + sum2;
    }

    return sum;
}
```

Πίνακας 8: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*) (*MINBLK: 50000000*)**
100000000	0.278347
200000000	0.65436
300000000	0.973535
400000000	1.45033

Παραλλαγή 6η

Συμβ. 9: Υλοποίηση παραλλαγής 6

```
double pi(long num_steps) {
    double dH = 1.0/(double)num_steps;
    double dX, dSum = 0.0;

#pragma omp parallel for simd private(dX) \
    reduction(+:dSum) schedule(simd:static)
    for (int i = 0; i < num_steps; i++) {
        dX = dH * ((double) i  + 0.5);
        dSum += (4.0 / (1.0 + dX * dX));
    } // End parallel for simd region

    return dH * dSum;
}
```

Πίνακας 9: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*) (*MINBLK: 50000000*)**
100000000	0.207
200000000	0.371
300000000	0.546
400000000	1.722

Παραλλαγή 7η

Συμβ. 10: Υλοποίηση παραλλαγής 7

```
double pi(long num_steps) {
    double dH = 1.0/(double)num_steps;
    double dX = 0.0, dSum = 0.0;

#pragma omp target teams distribute map(tofrom: dSum),\
                                    map(to:dX,  dH, num_steps)\
                                    reduction(+:dSum)
    for (int i = 0; i < num_steps; i++) {
        dX = dH * ((double) i  + 0.5);
        dSum += (4.0 / (1.0 + dX * dX));
    } // End parallel for simd region

    return dH * dSum;
}
```

Πίνακας 10: *Καταγραφή χρόνων εκτέλεσης παραδειγμάτων*

Αριθμός Βημάτων	**Χρόνος Υπολογισμού (*sec*)**	**Αποτέλεσμα Υπολογισμού (*sec*)**
100000000	5.540	3.100
200000000	10.178	3.099
300000000	14.757	3.098
400000000	19.432	3.098

References

[1] . 0.