Παράλληλος Προγραμματισμός σε Συστήματα Μηχανικής Μάθησης

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Εργαστήριο 3:

Το πρόβλημα του περιοδεύοντα πωλητή, τυχαία αναζήτηση και ο αλγόριθμος ant colony

Δασούλας Ιωάννης – 1053711 – 5ο Έτος

Εισαγωγή

Η αναφορά αφορά την υλοποίηση τριών αλγορίθμου για το πρόβλημα του περιοδεύοντα πωλητή. Για κάθε εργασία έχει δημιουργηθεί ένα ξεχωριστό αρχείο c (tsp1.c για την Εργασία 1, tsp2.c για την Εργασία 2 και ούτω καθ' εξής). Για κάποιες εργασίες έχουν δημιουργηθεί δύο εκδόσεις που χαρακτηρίζονται ως tsp_slow_edition.c και tsp_fast_edition.c αντίστοιχα. Αναλυτικά εξηγήσεις και σχόλια για τα βήματα που ακολουθούνται υπάρχουν σε κάθε αρχείο. Η μέτρηση του μεγέθους της μνήμης που χρησιμοποιείται έγινε με την εντολή env time -v ./exe, όπου exe το εκτελέσιμο.

ΕΝΟΤΗΤΑ 1 – ΤΥΧΑΙΑ ΑΝΑΖΗΤΗΣΗ

Εργασία 1

Α) Αργή Έκδοση

Για την εργασία 1, αφού μελετήθηκε ο αλγόριθμος, δημιουργήθηκαν αρχικά 4 παράμετροι: το πλήθος των πόλεων 'N', το μέγεθος κάθε διάστασης του χάρτη 'MAP_SIZE', το πλήθος των διαστάσεων κάθε πόλης 'DIMS' και ο αριθμός των επαναλήψεων που θα τρέξει ο αλγόριθμος 'ITERATIONS'. Έπειτα, δημιουργήθηκαν οι 3 βασικοί πίνακες του αλγορίθμου: ο πίνακας cities[N][DIMS] που περιέχει τις συντεταγμένες όλων των πόλεων, ο πίνακας travel[N+1] που περιέχει την διαδρομή, χαρακτηρίζοντας κάθε πόλη με έναν ακέραιο αριθμό από το 0 έως και το N-1, και ο

πίνακας swapped[2] στον οποίο αποθηκεύονται οι πόλεις που εναλλάσσονται στην διαδρομή, ώστε να μπορεί στην συνέχεια να αναιρεθεί η αλλαγή αυτή.

Στην πρώτη επίλυση του προβλήματος, στην αργή έκδοση, αρχικά δημιουργούνται οι τυχαίες πόλεις με συντεταγμένες στο πεδίο [0,1000] στην συνάρτηση CreateCities(), δημιουργείται μια αρχική διαδρομή από την πόλη 0 στην πόλη 0 ξανά στην συνάρτηση CreateRandomTravel(), διασχίζοντας όλες τις πόλεις και υπολογίζεται η συνολική απόσταση της αρχικής αυτής διαδρομής στην συνάρτηση CalcDistance().

Στην συνέχεια, για κάθε μία από τις επαναλήψεις που ορίζονται στην αρχή, εναλλάσσονται τυχαία 2 πόλεις της διαδρομής στην συνάρτηση SwapCities() και υπολογίζεται ξανά η συνολική απόσταση της διαδρομής. Αν η απόσταση αυτή είναι μικρότερη από την προηγούμενη αποθηκευμένη απόσταση, αποθηκεύεται αυτή ως η νέα μικρότερη απόσταση και αντίστοιχα η τρέχουσα διαδρομή ως η συντομότερη διαδρομή. Αν δεν είναι μικρότερη, καλείται η συνάρτηση RestoreCities() για να επαναφέρει στην προηγούμενη κατάσταση την διαδρομή, αφού η αλλαγή που έγινε έχει χειρότερο αποτέλεσμα.

Για 10.000 επαναλήψεις τα αποτελέσματα είναι τα εξής:

```
Ο αλγόριθμος ολοκληρώθηκε μετά από 10000 επαναλήψεις!
Η μικρότερη διαδρομή έχει μήκος 2105175388.718711 χιλιόμετρα.

real    0m1,277s
user    0m1,276s
sys     0m0,000s
```

Αντίστοιχα, για 100.000 επαναλήψεις:

```
Ο αλγόριθμος ολοκληρώθηκε μετά από 100000 επαναλήψεις!
Η μικρότερη διαδρομή έχει μήκος 810967940.989328 χιλιόμετρα.

real    0m12,506s
user    0m12,497s
sys     0m0,004s
```

Τέλος, για 1.000.000 επαναλήψεις:

```
Ο αλγόριθμος ολοκληρώθηκε μετά από 1000000 επαναλήψεις!
Η μικρότερη διαδρομή έχει μήκος 243310843.267604 χιλιόμετρα.

real    2m9,719s
user    2m9,640s
sys     0m0,024s
```

Μέγιστο μέγεθος μνήμης που χρησιμοποιήθηκε:

```
Maximum resident set size (kbytes): 1816
```

Τα αποτελέσματα του profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
100.68     13.16    13.16   100001   131.59   131.59  CalcDistance
  0.08     13.17     0.01                             main
  0.00     13.17     0.00   100000     0.00     0.00  SwapCities
  0.00     13.17     0.00    87363     0.00     0.00  RestoreCities
  0.00     13.17     0.00        1     0.00     0.00  CreateCities
  0.00     13.17     0.00        1     0.00     0.00  CreateRandomTravel
```

Όπως αναμενόταν, η συνολική απόσταση είναι πολύ μικρότερη όσο αυξάνονται οι επαναλήψεις του αλγορίθμου. Παρόλα αυτά οι χρόνοι του αλγορίθμου είναι απαγορευτικοί για δοκιμές πολλών περισσότερων επαναλήψεων, λόγω της χρονοβόρας λειτουργείας της συνάρτησης υπολογισμού της απόστασης, κάτι που φαίνεται και στον profiler. Οι χρόνοι αυτοί οδήγησαν στην δημιουργία της δεύτερης έκδοσης της εργασίας.

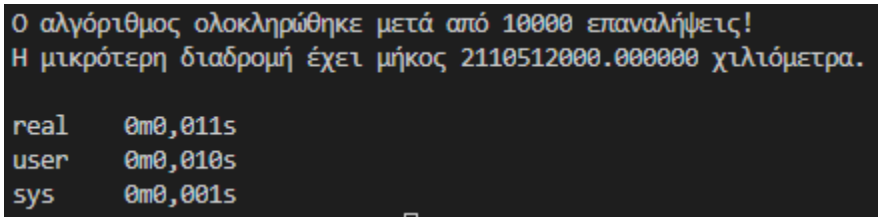
Β) Γρήγορη Έκδοση

Στην δεύτερη έκδοση, προστέθηκε ο πίνακας distances[N] στον οποίο αποθηκεύονται οι αποστάσεις κάθε πόλης από την επόμενη στην διαδρομή και ο πίνακα save_distances[4] όπου αποθηκεύονται οι 2 αποστάσεις σχετικές με την κάθε μία από τις 2 πόλεις που εναλλάσσονται, συνολικά 4 αποστάσεις.

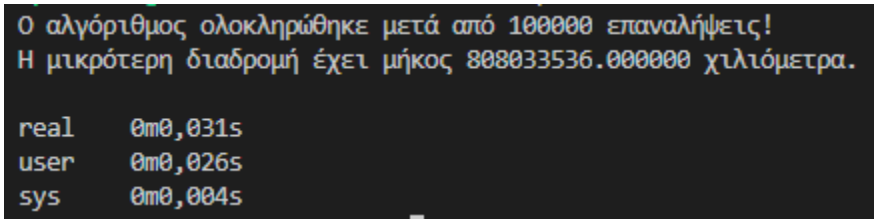
Σε αντίθεση με την προηγούμενη έκδοση, η συνολική απόσταση υπολογίζεται μία φορά στην αρχή του προγράμματος. Έπειτα, εναλλάσσονται 2 πόλεις και αφαιρούνται από την συνολική απόσταση οι 4 αποστάσεις που αφορούσαν τις πόλεις

αυτές, 2 για την κάθε πόλη (προς και από την κάθε πόλη). Στην συνέχεια, στην συνολική απόσταση προστίθενται οι 4 νέες αποστάσεις μετά την εναλλαγή των πόλεων και συγκρίνεται το αποτέλεσμα με την έως τώρα μικρότερη συνολική απόσταση, όπως και προηγουμένως.

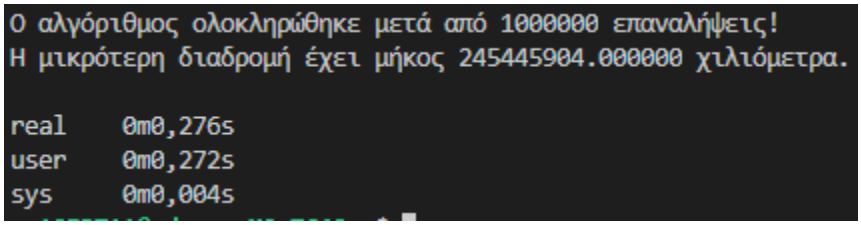
Για 10.000 επαναλήψεις, τα αποτελέσματα είναι τα εξής:



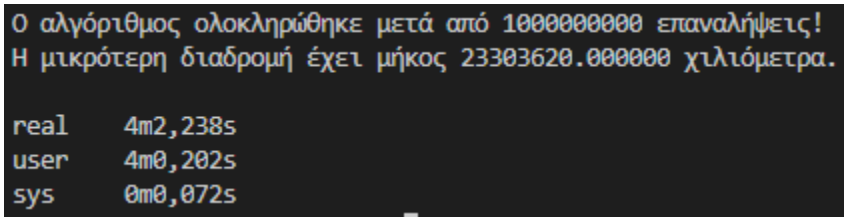
Αντίστοιχα, για 100.000 επαναλήψεις:



Αντίστοιχα, για 1.000.000 επαναλήψεις:



Τέλος, για 1.000.000.000 επαναλήψεις:



Μέγιστο μέγεθος μνήμης που χρησιμοποιήθηκε:

```
Maximum resident set size (kbytes): 1892
```

Αποτελέσματα profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
100.52      0.01     0.01   100000   100.52   100.52  SwapCities
  0.00      0.01     0.00   200001     0.00     0.00  CalcDistances
  0.00      0.01     0.00    87387     0.00     0.00  RestoreCities
  0.00      0.01     0.00        1     0.00     0.00  CreateCities
  0.00      0.01     0.00        1     0.00     0.00  CreateRandomTravel
```

Παρατηρείται ότι, με λίγο παραπάνω χρησιμοποιούμενη διεύθυνση, ο αλγόριθμος δίνει πολύ καλύτερα αποτελέσματα χρόνου και προφανώς είναι προτιμότερος.

Εργασία 2

Στην διαδικασία παραλληλοποίησης του αλγορίθμου, όλες οι προσπάθειες παραλληλοποίησης του γρήγορου αλγορίθμου δεν είχαν επιτυχία, αντ' αυτού είχαν ως αποτέλεσμα μεγαλύτερους χρόνους από το σειριακό πρόγραμμα.

Έπειτα, παραλληλοποιήθηκε η αργή έκδοση, που επιδεχόταν βελτίωση χρόνου. Στην αργή έκδοση παραλληλοποιήθηκε η συνάρτηση υπολογισμού απόστασης όπως φαίνεται παρακάτω:

```
double CalcDistance(){                                          //Συνάρτηση υπολογισμού συνολικής διαδρομής

    int i,j;
    double dist = 0, temp;

    #pragma omp parallel for private(temp,i,j) reduction(+: dist)
    for(i=0; i<N; i++){
        for(j=0; j<DIMS; j++){
            temp = cities[travel[i]][j] - cities[travel[i+1]][j];
            dist += temp*temp;
        }
    }

    return dist;
}
```

Όλες οι μεταβλητές ορίζονται ως private, εκτός από την μεταβλητή απόστασης που ορίζεται με reduction.

Τα αποτελέσματα για 10.000 επαναλήψεις και 2 νήματα:

```
Χρησιμοποιήθηκαν 2 νήματα.
Ο αλγόριθμος ολοκληρώθηκε μετά από 10000 επαναλήψεις!
Η μικρότερη διαδρομή έχει μήκος 2124994724.627563 χιλιόμετρα.

real    0m0,861s
user    0m1,512s
sys     0m0,008s
```

Τα αποτελέσματα για 10.000 επαναλήψεις και 4 νήματα:

```
Χρησιμοποιήθηκαν 4 νήματα.
Ο αλγόριθμος ολοκληρώθηκε μετά από 10000 επαναλήψεις!
Η μικρότερη διαδρομή έχει μήκος 2104022869.621390 χιλιόμετρα.

real    0m0,356s
user    0m1,400s
sys     0m0,013s
```

Τα αποτελέσματα για 10.000 επαναλήψεις και 8 νήματα:

```
Χρησιμοποιήθηκαν 8 νήματα.
Ο αλγόριθμος ολοκληρώθηκε μετά από 10000 επαναλήψεις!
Η μικρότερη διαδρομή έχει μήκος 2146363511.949424 χιλιόμετρα.

real    0m0,233s
user    0m1,824s
sys     0m0,005s
```

Το μέγιστο μέγεθος μνήμης που χρησιμοποιήθηκε:

```
Maximum resident set size (kbytes): 2204
```

Τα αποτελέσματα του profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
100.73      1.35     1.35    10001   134.96   134.96  CalcDistance
  0.00      1.35     0.00    10000     0.00     0.00  SwapCities
  0.00      1.35     0.00     6602     0.00     0.00  RestoreCities
  0.00      1.35     0.00        1     0.00     0.00  CreateCities
  0.00      1.35     0.00        1     0.00     0.00  CreateRandomTravel
```

Όπως φαίνεται στα αποτελέσματα, στην αργή έκδοση υπήρχε ιδιαίτερη διαφορά με την σειριακή έκδοση μετά την παραλληλοποίηση. Η μνήμη που χρησιμοποιείται είναι λίγο μεγαλύτερη από την σειριακή έκδοση, λόγω της χρήσης νημάτων.

ΕΝΟΤΗΤΑ 2 - Αλγόριθμος των Heinritz-Hsiao

Εργασία 3

Α) Αργή Έκδοση

Στην πρώτη έκδοση που δημιουργήθηκε για τον αλγόριθμο Heinritz-Hsiao, δημιουργήθηκαν οι ίδιοι πίνακες με τον αλγόριθμο τυχαίας αναζήτησης. Σε μία επαναληπτική δομή, για κάθε πόλη βρίσκονται όλες οι αποστάσεις με όλες τις άλλες και διατηρείται η μικρότερη, η οποία προστίθεται στο ταξίδι. Το χρονοβόρο κομμάτι του αλγορίθμου είναι ότι κάθε φορά πρέπει να ελέγχεται αν η υποψήφια πόλη για να γίνει ο επόμενος προορισμός είναι ήδη μέσα στο ταξίδι, διότι σε αυτήν την περίπτωση θα υπήρχε λάθος. Αυτό γίνεται στην συνάρτηση CheckIfVisited(). Σε κάθε επανάληψη αρχικοποιείται η μικρότερη απόσταση 2 προορισμών ως 2*1000*1000 βάσει του πυθαγόρειου θεωρήματος, διότι αυτή είναι η μέγιστη απόσταση που θα μπορούσαν να έχουν δύο πόλεις σε έναν χάρτη 1000*1000.

Αποτελέσματα:

```
Ο αλγόριθμος ολοκληρώθηκε επιτυχώς!
Η απόσταση της μικρότερης διαδρομής είναι 2650839.657271.

real    12m58,612s
user    12m52,368s
sys     0m0,404s
```

Μέγιστο μέγεθος μνήμης που χρησιμοποιήθηκε:

```
Maximum resident set size (kbytes): 1380
```

Αποτελέσματα profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 93.68    719.76   719.76 99980001     0.00     0.00  CheckIfVisited
  0.15    720.91     1.15        1     1.15   721.38  HeinritzHsiaoAlgorithm
  0.06    721.38     0.47 49995001     0.00     0.00  CalcDistance
  0.00    721.38     0.00        1     0.00     0.00  CreateCities
```

Όπως φαίνεται και στα αποτελέσματα, ενώ η απόσταση που υπολογίζεται είναι πολύ μικρή σε σχέση με τον αλγόριθμο τυχαίας αναζήτησης, ο χρόνος εκτέλεσης είναι πάρα πολύ μεγάλος, λόγω των συνεχών ελέγχων (99.980.001 για την ακρίβεια) για το αν μια πόλη είναι ήδη στην διαδρομή.

Β)Γρήγορη Έκδοση

Η γρήγορη έκδοση είχε ως σκοπό την εξάλειψη της ανάγκης ελέγχου αν μια πόλη είναι ήδη στην διαδρομή. Για τον λόγο αυτό δημιουργήθηκε ο πίνακας cities_indexes[N-1] που περιέχει όλες τις πόλεις με την μορφή ακεραίου αριθμού από το 0 έως το N-1. Κάθε φορά που μία πόλη υπολογίζεται ως η κοντινότερη της τρέχουσας, καλείται η συνάρτηση RemoveIndex(), η οποία μεταφέρει αυτή την πόλη στο τέλος του πίνακα. Στην επόμενη επανάληψη, μειώνεται το πάνω όριο αναζήτησης πόλεων κι έτσι αναζητούνται πόλεις μέχρι την προτελευταία πόλη, με την τελευταία να είναι αυτή που μόλις επιλέχτηκε. Με λίγα λόγια, αφαιρούνται από το πεδίο αναζήτησης οι πόλεις που είναι ήδη στην διαδρομή, όχι μέσω διαγραφής τους από την λίστα διότι θα ήταν λίγο πιο χρονοβόρο, αλλά μέσω μεταφοράς τους στο τέλος της λίστας. Η πρώτη πόλη επίσκεψης πηγαίνει στο τέλος, η δεύτερη στην δεύτερη από το

τέλος θέση και ούτω καθ’ εξής. Με αυτόν τον τρόπο δεν χρειάζεται κάθε φορά έλεγχος για το αν μια πόλη είναι στην διαδρομή.

Αποτελέσματα:

```
Ο αλγόριθμος ολοκληρώθηκε επιτυχώς!
Η απόσταση της μικρότερης διαδρομής είναι 2440763.235692.

real    0m0,890s
user    0m0,875s
sys     0m0,000s
```

Μέγιστο μέγεθος μνήμης που χρησιμοποιήθηκε:

```
Maximum resident set size (kbytes): 1784
```

Αποτελέσματα profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 54.80      0.37     0.37 49995001     0.00     0.00  CalcDistance
 44.43      0.67     0.30        1   302.14   674.77  HeinritzHsiaoAlgorithm
  1.48      0.68     0.01                             PrintVec
  0.00      0.68     0.00     9999     0.00     0.00  RemoveIndex
  0.00      0.68     0.00        1     0.00     0.00  CreateCities
  0.00      0.68     0.00        1     0.00     0.00  CreateCitiesIndexes
```

Τα αποτελέσματα φανερώνουν την τεράστια διαφορά του αλγορίθμου, που από χρόνο 13 λεπτών πήγε σε χρόνο κάτω του ενός δευτερολέπτου, παρουσιάζοντας τα ίδια αποτελέσματα, με την χρήση λίγου χώρου μνήμης παραπάνω για την αποθήκευση της λίστας των πόλεων.

Εργασία 4

Και η εργασία 4 υλοποιήθηκε σε 2 εκδόσεις, μετατρέποντας τις αντίστοιχες 2 από την εργασία 3. Και στις 2 εκδόσεις χρησιμοποιήθηκε η ίδια λογική. Ορίστηκε ως παράμετρος η πιθανότητα επί τοις εκατό επιλογής της μικρότερης απόστασης, με το υπόλοιπο ποσοστό να είναι η πιθανότητα επιλογής της δεύτερης μικρότερης απόστασης (POSSIBILITY_OF_SHORTEST). Και οι δύο αλγόριθμοι λειτουργούν παρόμοια με την εργασία 3, μόνο που πλέον αποθηκεύονται οι 2 μικρότερες αποστάσεις κάθε φορά. Έπειτα, μέσω της συνάρτησης rand(), δημιουργείται ένας αριθμός από το 0 έως το 100. Αν αυτός ο αριθμός είναι μικρότερος ή ίσος από την παράμετρο που έχει τεθεί, επιλέγεται η μικρότερη διαδρομή, ενώ αν είναι μεγαλύτερος επιλέγεται η δεύτερη μικρότερη (συνάρτηση HeinritzHsiaoAlgorithm()).

Ενδεικτικά, αποτελέσματα από την γρήγορη έκδοση:

Για πιθανότητα 50% επιλογής της κοντινότερης απόστασης:

```
Ο αλγόριθμος ολοκληρώθηκε επιτυχώς!
Η μικρότερη διαδρομή έχει μήκος 11028068.326924 χιλιόμετρα.

real    0m0,812s
user    0m0,811s
sys     0m0,001s
```

Για πιθανότητα 70% επιλογής της κοντινότερης απόστασης:

```
Ο αλγόριθμος ολοκληρώθηκε επιτυχώς!
Η μικρότερη διαδρομή έχει μήκος 9090527.554034 χιλιόμετρα.

real    0m0,810s
user    0m0,806s
sys     0m0,005s
```

Για πιθανότητα 90% επιλογής της κοντινότερης απόστασης:

```
Ο αλγόριθμος ολοκληρώθηκε επιτυχώς!
Η μικρότερη διαδρομή έχει μήκος 4126333.793259 χιλιόμετρα.

real    0m0,812s
user    0m0,811s
sys     0m0,001s
```

Μέγιστο μέγεθος μνήμης που χρησιμοποιήθηκε:

```
Maximum resident set size (kbytes): 1848
```

Αποτελέσματα profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 60.72      0.41     0.41        1   412.92   412.92  HeinritzHsiaoAlgorithm
 39.99      0.68     0.27                             RemoveIndex
  0.00      0.68     0.00        1     0.00     0.00  CreateCities
  0.00      0.68     0.00        1     0.00     0.00  CreateCitiesIndexes
```

Όπως είναι λογικό, παρατηρείται ότι όσο μεγαλύτερη η πιθανότητα επιλογής της κοντινότερης πόλης, τόσο μικρότερη είναι η συνολική απόσταση. Η παρατήρηση αυτή ισχύει φυσικά για τα τυχαία δεδομένα που χρησιμοποιούνται στην άσκηση και όχι για όλα τα ήδη δεδομένων και τοποθεσιών.

Εργασία 5

Για την εργασία 5 παραλληλοποίηθηκε ο γρήγορος αλγόριθμος της εργασίας 4. Αυτό έγινε στην συνάρτηση HeinritzHsiaoAlgorithm() με τον παρακάτω τρόπο:

```
double HeinritzHsiaoAlgorithm(){

    int i, j, min_index, cities_left = N-1;
    int index_to_remove, index_to_remove2, min_index2, possibility;
    double dist, min_dist, min_dist2, total_dist=0;

    min_dist = 2*MAP_SIZE*MAP_SIZE;
    min_dist2 = 2*MAP_SIZE*MAP_SIZE;

    #pragma omp parallel for\
        private(i,j,min_index,min_index2,possibility,dist)\
        firstprivate(min_dist,min_dist2)\
        reduction(+: total_dist)\
        shared(travel, cities_indexes, cities_left)\
        schedule(static)

    for(i=0; i<N-1; i++){
        for(j=0; j<cities_left; j++){

            dist = CalcDistance(travel[i],cities_indexes[j]);
            if(dist<=min_dist){
                min_dist = dist;
                min_index = cities_indexes[j];
                index_to_remove = j;
            }
            else if(dist<min_dist2){
                min_dist2 = dist;
                min_index2 = cities_indexes[j];
                index_to_remove2 = j;
            }

        }
```

Ως private δηλώθηκαν οι μετρητές των επαναλήψεων των βρόγχων i,j, οι μεταβλητές min_index και min_index2 στις οποίες αποθηκεύονται οι αριθμοί των πιο κοντινών πόλεων, ο τυχαίος αριθμός possibility που χρησιμοποιείται για την επιλογή της μίας από τις δύο αποστάσεις και η μεταβλητή dist που αποθηκεύεται η απόσταση

που υπολογίζεται κάθε φορά. Ως firstprivate δηλώνονται οι μεταβλητές min_dist και min_dist2 ώστε να πάρουν την αρχική τιμή που τους δίνεται και να μπορεί να γίνει σύγκριση στην συνέχεια με τις αποστάσεις που υπολογίζονται. Με reduction δηλώνεται η μεταβλητή total_dist ώστε όλα τα νήματα να γράφουν την συνολική απόσταση στην ίδια μεταβλητή. Ως shared δηλώνεται οι πίνακες travel και cities_indexes που είναι κοινοί καθώς και η μεταβλητή cities_left που εκφράζει το άνω όριο της αναζήτησης πόλεων και πρέπει να είναι ίδιο για όλες τις επεκεργασίες.

Επειδή, σε κάθε επανάληψη μειώνεται το άνω όριο της αναζήτησης, η πράξη της μείωσης αυτή δηλώνεται ως atomic για να εξασφαλιστεί ο συγχρονισμός των νημάτων.

```
#pragma omp atomic
cities_left --;

RemoveIndex(index_to_remove, cities_left);
min_dist = 2*MAP_SIZE*MAP_SIZE;
min_dist2 = 2*MAP_SIZE*MAP_SIZE;
```

Αποτελέσματα για 2 νήματα και πιθανότητα 80% επιλογής της κοντινότερης απόστασης:

```
Χρησιμοποιήθηκαν 2 νήματα.
Ο αλγόριθμος ολοκληρώθηκε επιτυχώς!
Η μικρότερη διαδρομή έχει μήκος 5737171.159818 χιλιόμετρα.

real    0m0,440s
user    0m0,875s
sys     0m0,001s
```

Αποτελέσματα για 4 νήματα και πιθανότητα 80% επιλογής της κοντινότερης απόστασης:

```
Χρησιμοποιήθηκαν 4 νήματα.
Ο αλγόριθμος ολοκληρώθηκε επιτυχώς!
Η μικρότερη διαδρομή έχει μήκος 5073873.844041 χιλιόμετρα.

real    0m0,236s
user    0m0,931s
sys     0m0,001s
```

Αποτελέσματα για 8 νήματα και πιθανότητα 80% επιλογής της κοντινότερης απόστασης:

```
Χρησιμοποιήθηκαν 8 νήματα.
Ο αλγόριθμος ολοκληρώθηκε επιτυχώς!
Η μικρότερη διαδρομή έχει μήκος 4608988.672638 χιλιόμετρα.

real    0m0,136s
user    0m0,995s
sys     0m0,008s
```

Μέγιστο μέγεθος μνήμης που χρησιμοποιήθηκε:

```
Maximum resident set size (kbytes): 2296
```

Αποτελέσματα profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 37.03      0.25     0.25                             CreateCitiesIndexes
 32.58      0.47     0.22                             CalcDistance
 23.70      0.63     0.16 49995001     0.00     0.00  RemoveIndex
  7.41      0.68     0.05        2    25.18   105.75  CreateCities
  0.00      0.68     0.00        1     0.00     0.00  __do_global_dtors_aux
```

Τα αποτελέσματα επαληθεύουν τη σωστή λειτουργεία σε πολύ μικρό χρόνο. Ενδεικτικό είναι ότι μετά την παραλληλοποίηση, το 44% του χρόνου εκτέλεσης αφορά την δημιουργία των δεδομένων (συναρτήσεις CreateCitiesIndexes() και CreateCities()) και μόνο το 56% την εκτέλεση του αλγορίθμου.

ΕΝΟΤΗΤΑ 3 - Αποικία Μυρμηγκιών (ant colony)

Εργασία 6

Για την εργασία 6, μελετήθηκε ο γενετικός αλγόριθμος της αποικίας μυρμηγκιών και λογισμικό από το διαδίκτυο και δημιουργήθηκε το αντίστοιχο πρόγραμμα. Αρχικά ορίζονται οι κατάλληλες παράμετροι και στην συνέχεια η δομή ant που περιέχει όλες τις απαραίτητες πληροφορίες για το ταξίδι ενός μυρμηγκιού, όπως το που βρίσκεται, ποιες πόλεις έχει επισκεφτεί και ποια είναι η διαδρομή του. Επίσης, δημιουργήθηκαν οι πίνακες distances[CITIES][CITIES] και hormone[CITIES][CITIES] που περιέχουν τις αποστάσεις και τις ποσότητες φερομόνης, αντίστοιχα, μεταξύ όλων των πόλεων, καθώς και ο πίνακας ants[ANTS] που περιέχει όλα τα μυρμήγκια και ο πίνακας citis[CITIES][DIMS] που περιέχει τις συντεταγμένες όλων των πόλεων.

Ο αλγόριθμος ξεκινάει με την δημιουργία των τυχαίων πόλεων στην συνάρτηση CreateCities(), τον υπολογισμό όλων των μεταξύ τους αποστάσεων και την αρχικοποίηση του πίνακα της φερομόνης στην συνάρτηση CreateDistancesMatrix() και την αρχικοποίηση των δεδομένων των μυρμηγκιών στην συνάρτηση InitializeAnts(). Στην αρχή όλες οι τιμές στον πίνακα της φερομόνης ορίστηκαν ως ίδιες (όλες με μη μηδενική απόσταση). Ο υπολογισμός των αποστάσεων στην αρχή του αλγορίθμου συμφέρει από άποψη χρόνου στον συγκεκριμένο αλγόριθμο, διότι οι αποστάσεις χρησιμοποιούνται πολλές φορές (ανάλογα και με τον αριθμό των μυρμηγκιών) στην διάρκεια του αλγορίθμου και θα ήταν πολύ πιο χρονοβόρο να υπολογίζονται συνέχεια. Βέβαια, για την ταχύτητα αυτή, αναγκαστικά χρησιμοποιείται μεγαλύτερο μέρος της μνήμης.

Στην συνέχεια με μία επαναληπτική διαδικασία ξεκινάει η κίνηση των μυρμηγκιών στην συνάρτηση SimulateAnts(). Για κάθε μυρμήγκι, υπολογίζεται ο επόμενος προορισμός του βάσει πιθανοτήτων στην συνάρτηση NextCity(). Ο τύπος που χρησιμοποιήθηκε για την εύρεση των πιθανοτήτων επίσκεψης μιας πόλης είναι ο παρακάτω, όπου:

i: Η τρέχουσα πόλη
j: Η υποψήφια προς επίσκεψη πόλη
τ: Η ποσότητα φερομόνης στην μεταξύ τους απόσταση
η: Η ποσότητα 1/d, όπου d η απόσταση των πόλεων
α: Το βάρος της φερομόνης
β: Το βάρος της απόστασης

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{k \in allowed_k} [\tau_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}} & \text{if } j \in \text{allowed}_k \\ 0 & \text{otherwise} \end{cases}$$

Το ‘τ’, με την σειρά του υπολογίζεται με τους παρακάτω τύπους, όπου:

ρ: Ο βαθμός εξάτμισης της φερομόνης
Q: Σταθερά που μεγαλώνει το βάρος της φερομόνης (απαραίτητη για μεγάλες αποστάσεις)
L_k: Απόσταση συνολικής διαδρομής μυρμηγκιού

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$$

$$\Delta\tau_{i,j}^{k} = \begin{cases} \dfrac{Q}{L_k} & if\,(i,j) \in \text{tour}_k \\ 0 & \text{otherwise} \end{cases}$$

Στην υλοποίηση αυτή η σταθερά ‘ρ’ τέθηκε ίση με το 0 διότι η εκφώνηση δεν έκανε λόγο για εξάτμιση της φερομόνης, αλλά με μερικές μικρές αλλαγές μπορεί να ληφθεί υπόψη από τον αλγόριθμο. Η ενημέρωση του πίνακα της φερομόνης γίνεται στην συνάρτηση UpdateFeromones().

Επίσης, στον αλγόριθμο, για κέρδος χρόνου δεν υπολογίζεται ο παρονομαστής της πιθανότητας διότι είναι κοινός για όλες τις πιθανότητες και δεν επηρεάζει την απόφαση του μυρμηγκιού. Έτσι, υπολογίζεται μόνο ο αριθμητής. Για τον υπολογισμό του αριθμητή στην συνάρτηση AntProduct() χρησιμοποιείται η συνάρτηση pow της

βιβλιοθήκης math.h, αν και έχει αργές επιδόσεις, ώστε το πρόβλημα να είναι παραμετροποιήσιμο ως προς τις σταθερές ‘α’ και ‘β’.

Αφού, όλα τα μυρμήγκια ολοκληρώσουν την διαδρομή τους, αποθηκεύεται ο αριθμός του μυρμηγκιού με την μικρότερη συνολική διαδρομή και η διαδρομή που έκανε στην συνάρτηση RestartAnts(). Επίσης, εκεί αρχικοποιούνται ξανά τα μυρμήγκια για την επόμενη επανάληψη του αλγορίθμου. Ο λόγος που αυτές οι 2 ενέργειες γίνονται στην ίδια συνάρτηση είναι πάλι ο χρόνος εκτέλεσης, αφού οι 2 αυτές λειτουργείες μπορούν να γίνουν στον ίδιο βρόγχο (loop jamming).

Αφού ενημερωθεί ο πίνακας της φερομόνης και επαναρυθμιστούν τα μυρμήγκια, η διαδικασία ξεκινάει από την αρχή, τα μυρμήγκια κάνουν την διαδρομή τους και κάθε φορά αποθηκεύεται και τυπώνεται η καλύτερη.

Για 20 μυρμήγκια, 10 επαναλήψεις, α = 1 και β = 1 τα αποτελέσματα είναι τα εξής:

```
Καλύτερη απόσταση έως την επανάληψη 1: 2289011.500000
Καλύτερη απόσταση έως την επανάληψη 2: 2249439.000000
Καλύτερη απόσταση έως την επανάληψη 3: 2249439.000000
Καλύτερη απόσταση έως την επανάληψη 4: 2127871.000000
Καλύτερη απόσταση έως την επανάληψη 5: 2077320.750000
Καλύτερη απόσταση έως την επανάληψη 6: 2077320.750000
Καλύτερη απόσταση έως την επανάληψη 7: 2077320.750000
Καλύτερη απόσταση έως την επανάληψη 8: 2077320.750000
Καλύτερη απόσταση έως την επανάληψη 9: 2077320.750000
Καλύτερη απόσταση έως την επανάληψη 10: 2077320.750000

real    3m9,704s
user    3m8,481s
sys     0m0,808s
```

Για 20 μυρμήγκια, 10 επαναλήψεις, α = 3 και β = 1 τα αποτελέσματα είναι τα εξής:

```
Καλύτερη απόσταση έως την επανάληψη 1: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 2: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 3: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 4: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 5: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 6: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 7: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 8: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 9: 2284052.000000
Καλύτερη απόσταση έως την επανάληψη 10: 2284052.000000

real    7m37,274s
user    7m34,846s
sys     0m0,876s
```

Για 20 μυρμήγκια, 10 επαναλήψεις, α = 1 και β = 3 τα αποτελέσματα είναι τα εξής:

```
Καλύτερη απόσταση έως την επανάληψη 1: 2445365.250000
Καλύτερη απόσταση έως την επανάληψη 2: 2337912.500000
Καλύτερη απόσταση έως την επανάληψη 3: 2337912.500000
Καλύτερη απόσταση έως την επανάληψη 4: 2209375.000000
Καλύτερη απόσταση έως την επανάληψη 5: 2073315.750000
Καλύτερη απόσταση έως την επανάληψη 6: 2073315.750000
Καλύτερη απόσταση έως την επανάληψη 7: 1833166.875000
Καλύτερη απόσταση έως την επανάληψη 8: 1833166.875000
Καλύτερη απόσταση έως την επανάληψη 9: 1833166.875000
Καλύτερη απόσταση έως την επανάληψη 10: 1833166.875000

real    7m56,936s
user    7m55,570s
sys     0m0,972s
```

Μέγιστο μέγεθος μνήμης που χρησιμοποιείται:

```
Maximum resident set size (kbytes): 1566400
```

Αποτελέσματα profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 55.09     61.64    61.64       10     6.16    10.96  NextCity
 28.86     93.94    32.29  1999800     0.00     0.00  AntProduct
 14.00    109.60    15.66 1409065408     0.00     0.00  RestartAnts
  1.98    111.81     2.22        2     1.11     1.25  CalcDistancesMatrix
  0.25    112.09     0.28 49995000     0.00     0.00  CalcDistances
  0.07    112.17     0.08       10     0.01     0.01  SimulateAnts
  0.06    112.24     0.07                             UpdateFeromones
  0.00    112.24     0.00       10     0.00     0.00  InitializeAnts
  0.00    112.24     0.00        1     0.00     0.00  CreateCities
```

Παρατηρείται ότι όσο μεγαλύτερη είναι η σταθερά ‘β’ σε σχέση με την σταθερά ‘α’, τόσο καλύτερα είναι τα αποτελέσματα, που είναι λογικό, αφού η σταθερά ΄β΄ αφορά την απόσταση. Επίσης, παρατηρείται μεγάλη διαφορά χρόνου όταν οι σταθερές ‘α’, ’β’ είναι διάφορα του 1, κάτι για το οποίο ευθύνεται η συνάρτηση pow() της “math.h”. Επίσης, παρατηρείται πολύ μεγάλη χρήση της μνήμης, το οποίο συμβαίνει λόγω των πολλών δεδομένων που αφορούν τα μυρμήγκια, όπως η διαδρομή για το καθένα, ποιες πόλεις έχει επισκεφτεί το καθένα και ούτω καθ’ εξής. Όσα περισσότερα μυρμήγκια, τόσο πιο πολύ μνήμη είναι απαραίτητη.

Εργασία 7

Στην εργασία 7, αρχικά αφαιρέθηκε η συνάρτηση pow() και το πρόγραμμα φτιάχτηκε με τις σταθερές α = 1 και β = 3 για να βελτιστοποιηθεί ο χρόνος. Βάσει των αποτελεσμάτων του profiler, έγινε προσπάθεια παραλληλοποίησης των συναρτήσεων. Οι συναρτήσεις που παραλληλοποιήθηκαν και παρουσιάστηκε βελτίωση χρόνου ήταν οι RestartAnts(), η NextCity() και η SimulateAnts() όπως φαίνεται παρακάτω:

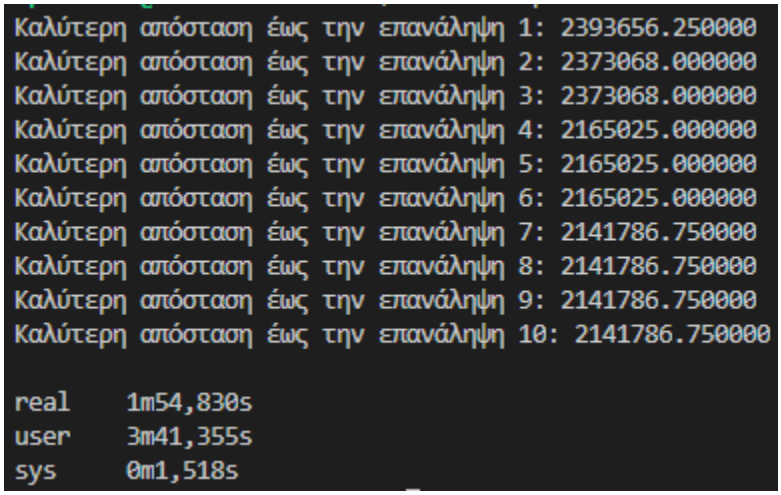
```
void RestartAnts(){

    int ant,i,to=0;
    float best = bestdistance;

    for(ant = 0; ant < ANTS; ant++){
        if(ants[ant].tourLength < best){            //Απο
            best = ants[ant].tourLength;
            bestIndex = ant;
        }

        #pragma omp parallel for private(i) shared(ants)
        for(i = 0; i < CITIES; i++){
            ants[ant].visited[i] = 0;
            ants[ant].path[i] = -1;
        }

        ants[ant].nextCity = -1;
        ants[ant].tourLength = 0.0;
        ants[ant].curCity = rand()%CITIES;
        ants[ant].pathIndex = 1;
        ants[ant].path[0] = ants[ant].curCity;
        ants[ant].visited[ants[ant].curCity] = 1;

        bestdistance = best;
    }
}
```

```
int NextCity( int pos ){                                        //Εύρεση επόμε
    int from, to, destination;
    double denom = 0.0, p, max_p = 0;

    from = ants[pos].curCity;

    #pragma omp parallel for private(to,p) shared(ants, max_p, destination)
    for(to = 0; to < CITIES; to++){                             //Για κάθε πόλ
        if(ants[pos].visited[to] == 0){                         //Αν το μυρμήγ
            //denom += antProduct( from, to );                  //Για καλύτερο
            p = (double) AntProduct(from,to);                   //Υπολογισμός
            if(p>max_p){                                        //Αποθήκευση μ
                max_p = p;
                destination = to;                               //Και του προο
            }
        }
    }

    //assert(denom != 0.0);

    return destination;
}
```

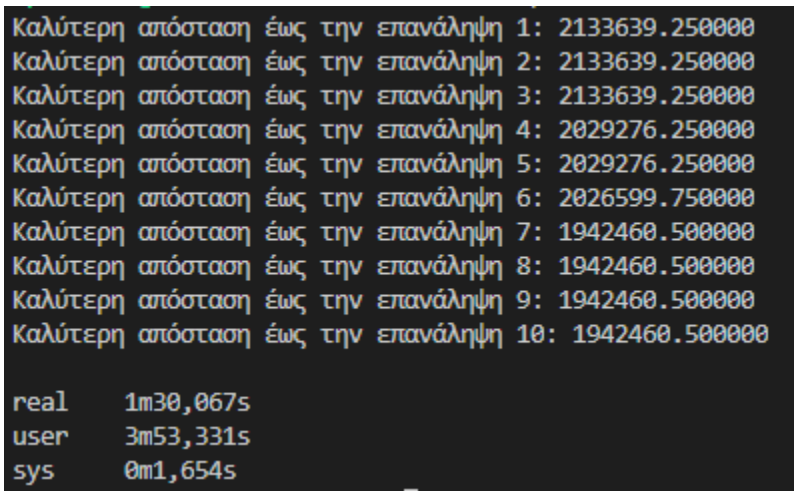
```
void SimulateAnts(){

    int k;

    #pragma omp parallel for private(k) shared(ants)
    for(k = 0; k < ANTS; k++){
        while( ants[k].pathIndex < CITIES ){
            ants[k].nextCity = NextCity(k);
            ants[k].visited[ants[k].nextCity] = 1;
            ants[k].path[ants[k].pathIndex++] = ants[k].nextCity;
            ants[k].tourLength += distances[ants[k].curCity][ants[k].nextCity];
            ants[k].curCity = ants[k].nextCity;
        }
        ants[k].tourLength += distances[ants[k].path[CITIES -1]][ants[k].path[0]];
    }
}
```

Η πιο αποτελεσματική ήταν η παραλληλοποίηση της NextCity(), κάτι λογικό αφού όπως φάνηκε στον profiler, είναι η πιο χρονοβόρα. Οι προσπάθειες για παραλληλοποίηση των υπόλοιπων συναρτήσεων δεν είχαν ουσιαστικό αντίκτυπο. Τα αποτελέσματα επιβεβαιώνουν την σωστή λειτουργεία και το κέρδος χρόνου.

Για 20 μυρμήγκια, 10 επαναλήψεις, α = 1, β = 3 και 2 νήματα τα αποτελέσματα είναι τα εξής:



Για 20 μυρμήγκια, 10 επαναλήψεις, α = 1, β = 3 και 4 νήματα:



Μέγιστο μέγεθος μνήμης που χρησιμοποιείται:

```
Maximum resident set size (kbytes): 1566348
```

Αποτελέσματα profiler:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 67.04     75.01    75.01       10     7.50    10.95  AntProduct
 30.89    109.57    34.56 1411065208     0.00     0.00  RestartAnts
  1.74    111.51     1.95        1     1.95     1.95  CalcDistancesMatrix
  0.25    111.79     0.28 49995000     0.00     0.00  CreateCities
  0.24    112.06     0.27        1     0.27     0.55  CalcDistances
  0.11    112.18     0.12       10     0.01     0.01  UpdateFeromones
  0.04    112.23     0.05                             NextCity
  0.01    112.24     0.01       10     0.00     0.00  InitializeAnts
  0.00    112.24     0.00        1     0.00     0.00  frame_dummy
```

Όπως φαίνεται από τα αποτελέσματα, η παραλληλοποίηση απέφερε κέρδος χρόνου παρουσιάζοντας σωστά αποτελέσματα.

ΕΝΟΤΗΤΑ 4 – ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ

Όσον αφορά τα αποτελέσματα, καλύτερα αποτελέσματα παρουσίασε ο αλγόριθμος αποικίας μυρμηγκιών. Παρόλα αυτά, ο αλγόριθμος αυτός απαιτεί πολύ χρόνο και πάρα πολλή περισσότερη μνήμη σε σχέση με τους άλλους δύο. Ο αλγόριθμος με τα καλύτερα δυνατά αποτελέσματα στον μικρότερο δυνατό χρόνο και με την μικρότερη χρήση μνήμης ήταν ο αλγόριθμος των Heinritz-Hsiao. Ο λιγότερο αποδοτικός αλγόριθμος φάνηκε πως είναι αυτός της τυχαίας αναζήτησης.