

Αλγοριθμικές Τεχνικές και Εφαρμογές

Φοιτητής: Παναγιώτης Κοντος

AM: mpsp2215

Η παρακάτω εργασία αφορά την υλοποίηση ενός αλγόριθμου που προσομοιώνει την διάδοση πυρκαγιάς σε ένα δάσος με χρήση του προγραμματιστικού μοντέλου των POSIX Threads.. Η εργασία έγινε με την χρήση της **Python**. Αξίζει να σημειωθεί ότι γράψαμε εκ νέου έναν fire starter αλγόριθμο απο C σε Python, τον οποίον χρησιμοποιήσαμε για να φτιάξουμε ένα fire starter Monte Carlo Simulation .

Βήματα

Αρχικά πρέπει να εξετάσουμε πως λειτουργεί ο αλγόριθμος με 1 thread, ο αλγόριθμος παίρνει σαν μεταβλητές το forest_size, prob_min, prob_max, n_trials, n_probs και για την κάθε πιθανότητα θα υπολογίσει το burn_output, δηλαδή το ποσοστό του δασους που καιγεται κάθε φορά.

```
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ ls
dynamic_threading.py single_thread.py static_threading.py
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ code .
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ pytho3 single_thread.py
Command 'pytho3' not found, did you mean:
  command 'python3' from deb python3 (3.10.6-1-22.04)
Try: sudo apt install <deb name>
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ python3 single_thread.py
0.0, 0.0
0.01, 5.0125313283208015e-05
0.02, 0.0003007518796992481
0.03, 0.0002506265664160401
0.04, 0.0004010025062656641
0.05, 0.0003007518796992481
0.06, 0.0006015037593984062
0.07, 0.0008020050125313282
0.08, 0.0016541353383458645
0.09, 0.0013533834586466164
0.1, 0.0011027568922305764
0.11, 0.0016040100250626565
0.12, 0.0022556390977443606
0.13, 0.0020551378446115286
0.14, 0.0019047619047619048
0.15, 0.002406015037593985
0.16, 0.004511278195488721
0.17, 0.003709273182957393
0.18, 0.002807017543859649
0.19, 0.003258145363408521
0.2, 0.0047619047619047615
0.21, 0.007619047619047619
0.22, 0.0071177944862155385
0.23, 0.006766917293233083
0.24, 0.008020050125313283
0.25, 0.009724310776942356
0.26, 0.00887218045112782
0.27, 0.01012531328320802
0.28, 0.012230576441102757
0.29, 0.016541353383458645
0.3, 0.013283208020050124
0.31, 0.014385964912280698
0.32, 0.02265664160401002
0.33, 0.024360902255639104
0.34, 0.02506265664160401
0.35, 0.03243107769423558
0.36, 0.029273182957393493
0.37, 0.040050125313283204
0.38, 0.056090225563909774
0.39, 0.05513784461152883
0.4, 0.05433583959899748
0.41, 0.08065162907268171
0.42, 0.08616541353383456
0.43, 0.06145363408521302
0.44, 0.12756892230576442
0.45, 0.15493734335839596
0.46, 0.17017543859649123
0.47, 0.2505764411027569
```

επισης μπορούμε να δουμε το δασος χρησιμοποιοντας την μεθοδο print_forest(forest_size, forest)

```
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo
XX.....XXXXX.....XXXXXX
XX.....XXX.....XXXXXX
XXX.....XXXXXX.....XXXXXX
XX.....XXXXXXXXXXXXXXXXX
0.48, 0.2770426065162907
XXXXXXXXXXXXXX.XX.X.
XXXXXXXXXXXXXX.X.....
XXXXXXXXX.....
XXXXXXXXXX.X.....
XXXXXXXXXX.X.X.X
XXXXXXXXX.....X
XXXXXXXXX.....XX
XXXXXXXXX.....X.X
XXXXXXXXX.....X
XXXXXXXXXX.....X
XXXXXXXXXX.....X
XXXXXXXXXX.XX.....XX
XXXXXXXXXXXXXX.....XX
XXXXXXXXXXXXXX.....
XXXXXXXXXXXXXX.....
XXXXXXXXXXXXXX.....X
XXXXXXXXXXXXXX.....X
XXXXXXXXXXXXXX.....X
XXXXXXXXXXXXXX.X.X.XX
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
0.49, 0.25679197994987474
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
```

αξιζει να σημειωθει οτι ο αλγοριθμος ξεκινει καθε φορα απο τυχαio σημειο και οτι τα X συμβολιζουν τα δεντρα ενω τα . τα καμμενα δεντρα.

Πως αυτος ο αλγοριθμος μπορει να πραγματοποιηθει παραλληλα με στατικη ανάθεση φόρτου εργασίας μεταξύ των νημάτων?

Αρχικα θα εχουμε μια νεα παραμετρο που θα ειναι ο αριθμος των νημάτων με ονομα THREADS, επειτα τα διαιρεσουμε τον αριθμο των πιθανοτητων (n_probs) με τον αριθμο των threads, και θα δοσουμε στο καθε thread το αναλογο φορτιο.

```

threads = list()
step = NUMS//THREADS
for i in range(THREADS):
    start = step*i
    if(i!=0):
        start = (step*i)+1
    end = step*(i+1)
    # If it is the last
    if(i==THREADS-1):
        end=NUMS
    t = threading.Thread(target=thread_function, args=(i,start,end))
    threads.append(t)
    t.start()
t.join()

```

Επομενως το καθε thread θα εκτελει το ιδιο method με πριν αλλα αυτην την φορα για διαφορετικο range πιθανοτητων.

```

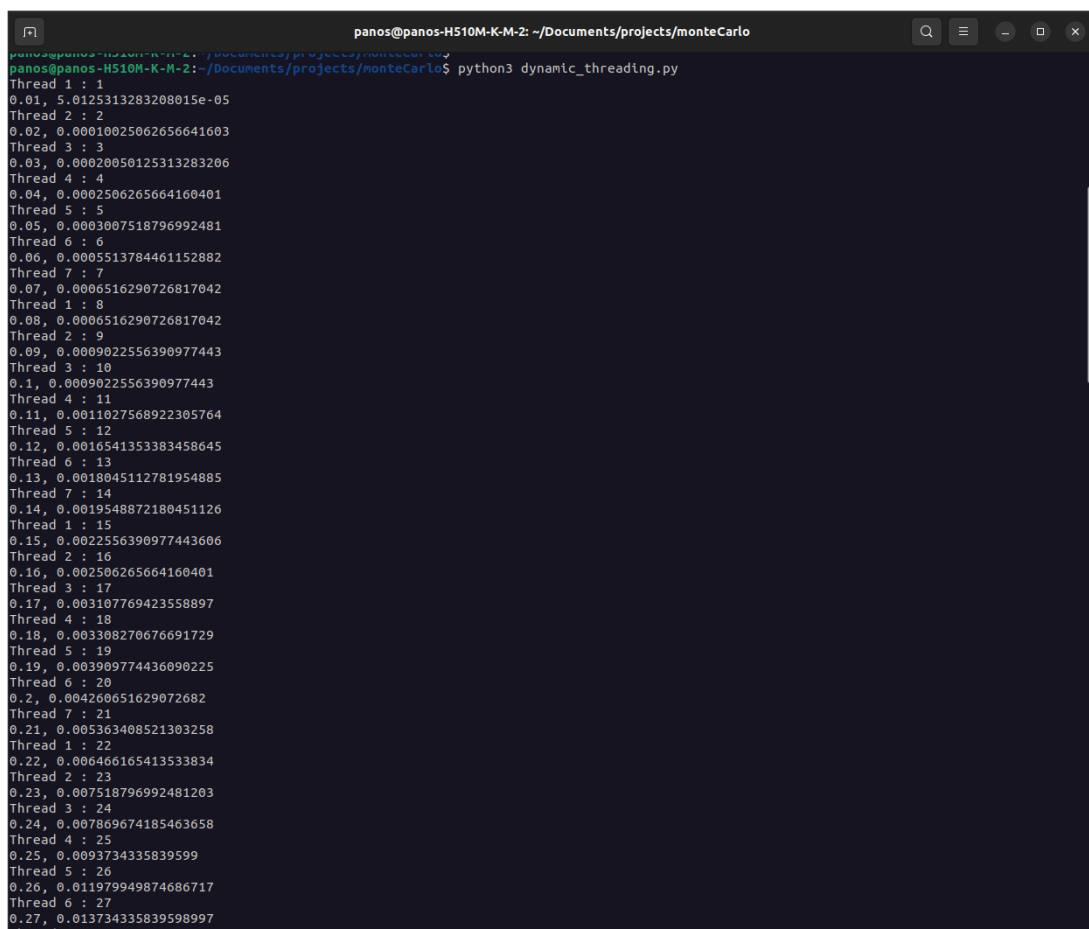
.....
Execution time: 5.698615312576294
panos@panos-H510M-K-M-2:~/Documents/projects/monteCarlo$ python3 static_threading.py
Thread 0: 0 - 5
0.0, 0.0
0.01, 0.00010025062656641603
0.03, 0.00045112781954887213
0.04, 0.0003007518796992481
0.05, 0.0007518796992481202
Thread 1: 6 - 10
0.06, 0.0006015037593984962
0.07, 0.0008521303258145363
0.09, 0.0011027568922305764
0.1, 0.0017543859649122805
Thread 2: 11 - 15
0.11, 0.0018045112781954885
0.12, 0.0017042606516290725
0.14, 0.003007518796992481
0.15, 0.0036591478696741853
Thread 3: 16 - 20
0.16, 0.003007518796992481
0.17, 0.003057644110275689
0.19, 0.004661654135338346
0.2, 0.004711779448621553
Thread 4: 21 - 25
0.21, 0.004611528822055138
0.22, 0.007017543859649122
0.24, 0.008020050125313283
0.25, 0.007167919799498747
Thread 5: 26 - 30
0.26, 0.00857142857142857
0.27, 0.014536340852130318
0.29, 0.01769423558897243
0.3, 0.014285714285714282
Thread 6: 31 - 35
0.31, 0.022706766917293227
0.32, 0.02050125313283208
0.34, 0.029874686716791974
0.35, 0.027669172932330822
Thread 7: 36 - 40
0.36, 0.04250626566416038
0.37, 0.04726817042606516
0.39, 0.07182957393483706
0.4, 0.059548872180451094
Thread 8: 41 - 45
0.41, 0.08641604010025063
0.42, 0.11899749373433582
0.44, 0.1174360902255638
0.45, 0.1876691729323308
Thread 9: 46 - 50
0.46, 0.2328822055137844
0.47, 0.2937844611528822
0.49, 0.28385964912280703
0.5, 0.3916791979949874

```

Το πρόβλημα με αυτήν την υλοποίηση είναι το load balancing καθώς τα threads που θα παρουν τις τελευταίες πιθανότητες θα κανουν και την περισσότερη ώρα. Για να μην γίνεται αυτό θα πρέπει να βρούμε έναν τρόπο να φορτίο να ισομοιραζεται.

Πως αυτός ο αλγόριθμος μπορεί να πραγματοποιηθεί παράλληλα με δυναμική ανάθεση φόρτου εργασίας μεταξύ των νημάτων?

Αυτή την φορά αντί να παίρνει το κάθε thread από μια ομάδα από πιθανότητες, θα παίρνει μια πιθανότητα και ύστερα θα συνεχίζει το επόμενο.



```
panos@panos-H510M-K-M-2: ~/Documents/projects/monteCarlo$ python3 dynamic_threading.py
Thread 1 : 1
0.01, 5.0125313283208015e-05
Thread 2 : 2
0.02, 0.00010025062656641603
Thread 3 : 3
0.03, 0.00020050125313283206
Thread 4 : 4
0.04, 0.0002506265664160401
Thread 5 : 5
0.05, 0.0003007518796992481
Thread 6 : 6
0.06, 0.0005513784461152882
Thread 7 : 7
0.07, 0.0006516290726817042
Thread 1 : 8
0.08, 0.0006516290726817042
Thread 2 : 9
0.09, 0.0009022556390977443
Thread 3 : 10
0.1, 0.0009022556390977443
Thread 4 : 11
0.11, 0.0011027568922305764
Thread 5 : 12
0.12, 0.0016541353383458645
Thread 6 : 13
0.13, 0.0018045112781954885
Thread 7 : 14
0.14, 0.0019548872180451126
Thread 1 : 15
0.15, 0.0022556390977443606
Thread 2 : 16
0.16, 0.002506265664160401
Thread 3 : 17
0.17, 0.003107769423558897
Thread 4 : 18
0.18, 0.003308270676691729
Thread 5 : 19
0.19, 0.003909774436090225
Thread 6 : 20
0.2, 0.004260651629072682
Thread 7 : 21
0.21, 0.005363408521303258
Thread 1 : 22
0.22, 0.006466165413533834
Thread 2 : 23
0.23, 0.007518796992481203
Thread 3 : 24
0.24, 0.007869674185463658
Thread 4 : 25
0.25, 0.0093734335839599
Thread 5 : 26
0.26, 0.011979949874686717
Thread 6 : 27
0.27, 0.013734335839598997
```

Μέχρι να απασχοληθεί και το τελευταίο thread, αλλά όταν τελειώσει αυτό θα συνεχίσει την δουλειά πάλι το πρώτο thread και θα συνεχίζει να γίνεται αυτό μέχρι να καλυφθούν όλες οι πιθανότητες από τα threads.

```

i = 0
stop_loop = False
while(i<NUMS and not(stop_loop)):
    step = 2
    for j in range(1, THREADS):
        start = i+j
        end = start+1
        globals()['thread' + str(j)] =
        threading.Thread(target=thread_function, args=(i, start, end, j) )
        if(start>=NUMS):
            stop_loop=True
            break
        else:
            globals()['thread' + str(j)].start()
            globals()['thread' + str(j)].join()
    i+=THREADS-1

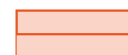
```

Με αυτό τον τρόπο έχουμε μια πιο δίκαιη μοίραση σε όλα τα threads καθώς σχεδόν όλα έχουν παρόμοιο φορτίο.

Μετρήσεις και Αποτελέσματα

Number of Threads	4	8	20	30
Forest Size	20	20	20	20
Single Thread	5.75	5.75	5.75	5.75
Static Threading	5.9299	5.52	4.91	4.21
Dynamic Threading	5.82	5.83	5.77	5.84

Number of Threads	4	8	20	30
Forest Size	50	50	50	50
Single Thread	61	61	61	61
Static Threading	62.2	59.49	50.18	46.3
Dynamic Threading	62.2	61.9	61.9	61.4



Τεσταρούμε την χρονική εκτέλεση και στα 3 implementation για διαφορετικά forest size και αριθμό threads. Από τις μετρήσεις φαίνεται ότι όσο αυξάνονται γενικά τόσο αυξάνονται τα threads ανεβαίνει και ταχύτητα εκτέλεσης και στα 2 threading implementation.

Αξίζει να σημειωθεί ότι φαίνεται ότι έχει γίνει πολύ καλή εκτέλεση στο static threading καθώς σε thread της τάξης 20-30 φαίνεται ότι έχουμε έως και 30% αύξηση στην ταχύτητα εκτέλεσης.

Αντιθετα, οι μετρησεις στο dynamic threading δειχνουν το αντιθετο καθως οπως φαινεται ακομα και οταν χρησιμοποιούμε πολλα threads δεν καταφερνουμε να ξεπερασουμε την ταχυτητα εκτελεσης του single thread. Αυτο μπορεί να οφειλεται στο overhead καθως η λειτουργια και διαχειριση νηματων απαιτεί επιπλέον λειτουργική επιβάρυνσ σε σχέση με την εκτέλεση ενός προγράμματος απο ενα μόνο νήμα.