Παράλληλος και Κατανεμημένος Υπολογισμός

Εργαστήριο 6

1. Μελετήσετε τον κώδικα Code/MapReduce/BruteForceStringMatch.java

Προσθέστε εντολές μέτρησης χρόνου.
Χρησιμοποιείστε το αρχείο E.coli (πλήρες DNA του βακτηριδίου Escherichia coli)από το Large Corpus στο σύνδεσμο https://corpus.canterbury.ac.nz/descriptions/#large (περίπου 4,6 Gbyets)

Μπορείτε για παράδειγμα να τετραπλασιάσετε το μέγεθος του αρχείου ως εξής:
Windows: type file.txt file.txt file.txt file.txt > fourtimes.txt
Linux: cat file.txt file.txt file.txt file.txt > fourtimes.txt

Διαλέξτε ένα pattern, για παράδειγμα “tacccagattatcgccatcaacgg” .
Χρησιμοποιείστε τις τεχνικές απεικόνισης και αναγωγής για να παραλληλίσετε τον κώδικα.
Μετρήστε το χρόνο εκτέλεσης της παράλληλης έκδοσης. Για να δείτε αξιόπιστους χρόνους χρειάζεται πολύ μεγάλο αρχείο, πχ 16 φορές το E.coli.

Η ταύτιση προτύπων (pattern matching) – εδώ ταυτίζουμε strings αλλά θα μπορούσαμε να το γενικεύσουμε και σε άλλα δεδομένα-- αποτελεί μια από τις θεμελιώδεις λειτουργίες στην ανάκτηση πληροφοριών (πχ grep), ασφάλεια υπολογιστών (πχ snort), υπολογιστική βιολογία (γονιδίωμα, πρωτεΐνες κλπ) και σε πολλές άλλες εφαρμογές. Παρακάτω δίνονται ορισμένα ενδεικτικά links για όσους ενδιαφέρονται.

https://www-igm.univ-mlv.fr/~lecroq/string/
https://algs4.cs.princeton.edu/50strings/
https://smart-tool.github.io/smart/

2. Μελετήστε τους κώδικες Code/LoadBalancing

Γράψτε παράλληλο κώδικα για το πρόγραμμα Sieve με τις εξής κατανομές φόρτου:

2.1. Στατική κατανομή: όπως έχουμε υλοποιήσει μέχρι τώρα τη κατανομή φόρτου.
2.2. Απλή κυκλική κατανομή: όπως στο κώδικα Code/Cyclic.java
2.3. Δυναμική κατανομή: όπως στο κὠδικα Code/MasterWorkerLock.java

Συγκρίνετε τις επιδόσεις τους για όσο μπορείτε μεγάλο πλήθος πρώτων αριθμών, για παράδειγμα 1000000000.

Οι πρώτοι αριθμοί και ο υπολογισμός μεγάλων πρώτων αριθμών είναι ιδιαίτερα χρήσιμοι στη κρυπτογραφία, στα κρυπτονομίσματα, το blockchain και αλλού. Παρακάτω δίνονται ορισμένα ενδεικτικά links για όσους ενδιαφέρονται.

https://en.wikipedia.org/wiki/Prime_number
https://en.wikipedia.org/wiki/Formula_for_primes
https://en.wikipedia.org/wiki/Prime_number_theorem

3. Μελετήστε τους κώδικες Code/LoadBalancing Recursive.. και τον κώδικα NumInt.java που ήδη έχετε από το Εργαστήριο 5.

Γράψτε παράλληλο κώδικα για το πρόγραμμα NumInt με τη χρήση αναδρομικής κατανομής φόρτου. Βρείτε τις καλύτερες τιμές για το μέγεθος block και για το τερματισμό της αναδρομής. Συγκρίνετε τους χρόνους εκτέλεσης με προηγούμενες παράλληλες λύσεις του ίδιου προβλήματος. Δοκιμάστε για μεγάλο αριθμό βημάτων, πχ 100000000.

Η θεωρία πίσω από την αναδρομική κατανομή φόρτου βασίζεται στο Cilk Plus, βιβλιοθήκη του MIT.
https://www.opencilk.org/.

Η Java έχει υλοποιήσει ειδικό πλαίσιο εκτέλεσης ForkJoinExecutor
https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html

Επίσης η αναδρομική κατανομή φορτίου χρησιμοποιείται και στα parallel streams της Java.
https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html