

Threads και δυναμική κατανομή δεδομένων

(Εναλλακτικές λύσεις πέρα από το μοντέλο fork – join)

<https://mixstef.github.io/courses/parprog/>

Μ.Στεφανιδάκης



Στατική κατανομή δεδομένων (φορτίου)

- Μέχρι τώρα είδαμε σχήματα παραλληλισμού με στατική κατανομή δεδομένων
 - Χωρισμός σε μεγάλα μπλοκ και ανάθεση σε μικρό αριθμό threads με το μοντέλο fork – join
 - Ευνοϊκή εκμετάλλευση hardware threads και τοπικότητας κρυφής μνήμης
 - Αλγόριθμοι με «στατικά» loops
 - Γνωρίζουμε εκ των προτέρων τον αριθμό και τα όρια των επαναλήψεων
 - Δεν εξαρτώνται από τις τιμές των δεδομένων
 - Κυρίως λειτουργίες τύπου map
 - Ιδανικά με ισομερή κατανομή φόρτου
 - Δηλ. το κόστος του $f(i)$ είναι παρόμοιο για κάθε i

Αλγόριθμοι με δυναμική κατανομή δεδομένων (φορτίου)

- Τι συμβαίνει όμως όταν ένας αλγόριθμος διαμερίζει τα δεδομένα δυναμικά;
 - Η κατανομή της εργασίας εξαρτάται από τα δεδομένα εισόδου
 - Μόνο κατά την εκτέλεση μαθαίνουμε την κατανομή
 - Δεν υπάρχουν στατικά όρια των loops γνωστά εκ των προτέρων
 - Ακόμα και σε μετασχηματισμούς τύπου map ο φόρτος του $f(i)$ δεν είναι ίδιος/γνωστός εκ των προτέρων για κάθε i
 - Η παράλληλη επεξεργασία χωρίζεται δυναμικά σε «**πακέτα εργασίας**» που δεν αντιστοιχούν 1:1 με τα διαθέσιμα threads

Παράδειγμα: quicksort

- Διαμερισμός πίνακα σε δύο μέρη
 - Μικρότερα και μεγαλύτερα ενός επιλεγμένου στοιχείου (pivot)
- Και στη συνέχεια αναδρομική ταξινόμηση quicksort των δύο υποπινάκων
 - Στο τέλος όλος ο πίνακας έχει ταξινομηθεί
- Ο ακριβής χωρισμός των υποπινάκων εξαρτάται από το στοιχείο pivot
 - Συνεπώς εξαρτάται από το περιεχόμενο του αρχικού πίνακα
 - Το ίδιο και η συνολική απόδοση του αλγορίθμου:
 - $O(n \log_2 n)$ στην καλύτερη περίπτωση και κατά μέσο όρο
 - βέλτιστος αλγόριθμος ταξινόμησης
 - $O(n^2)$ στη χειρότερη περίπτωση (ήδη ταξινομημένα στοιχεία)

Σειριακή μορφή quicksort

- **In-place** ταξινόμηση
 - Χρήση **insertion sort** κάτω από ένα όριο μεγέθους πίνακα

```
void quicksort(double *a,int n) {
int i;
// check if below cutoff limit
if (n<=CUTOFF) {
    inssort(a,n);
    return;
}

// partition into two halves
i = partition(a,n);

// recursively sort halves
quicksort(a,i);
quicksort(a+i,n-i);
}
```

Πρώτη απόπειρα παραλληλισμού

- **Ας δοκιμάσουμε το εξής:**
 - Συγγραφή συνάρτησης `threaded_quicksort()`
 - Αμέσως μετά την κλήση της `partition()` δημιουργούμε ένα υπο-thread για να ταξινομήσει το αριστερό μισό του πίνακα
 - Ενώ συνεχίζουμε με κλήση της `threaded_quicksort()` για το δεξιό μισό
 - Στο τέλος αναμένουμε (με `join`) το υπο-thread να τελειώσει
 - Η συνάρτηση του thread απλά καλεί τη συνάρτηση `threaded_quicksort()`
 - Με τις παραμέτρους που δέχεται από το `pthread_create()`
 - Στη `main()`, ως φορτίο, καλούμε απευθείας τη `threaded_quicksort()`

Πρώτη απόπειρα παραλληλισμού

- Τι παρατηρείτε για διάφορες τιμές N ;
- Ποια η διαφορά απόδοσης με τον σειριακό κώδικα;

Πρώτη απόπειρα παραλληλισμού

- Τι παρατηρείτε για διάφορες τιμές N;
 - Για μεγάλα μεγέθη πίνακα και πάνω (π.χ. 1.000.000+), εμφανίζεται **αποτυχία δημιουργίας υπο-threads**
 - Ο αριθμός των δημιουργούμενων threads **δεν είναι ελεγχόμενος** και ξεπερνά το **επιτρεπτό όριο** του λειτουργικού συστήματος
- Ποια η διαφορά απόδοσης με τον σειριακό κώδικα;
 - Ακόμα και στην περίπτωση επιτυχίας δημιουργίας υπο-threads, η απόδοση σε σχέση με τον σειριακό κώδικα είναι **κατά πολύ χειρότερη**
 - Ο ανεξέλεγκτος αριθμός των δημιουργούμενων threads προσθέτει **πολύ μεγάλη επιβάρυνση**

Εναλλακτικές απόπειρες παραλληλισμού

- Να καλούμε το `serial_quicksort()` κάτω από ένα (σταθερό, με `#define`) μέγεθος υποπίνακα
 - Πώς θα το υπολογίσουμε;
 - Ένα αυθαίρετα ορισμένο μέγεθος (π.χ. ευριστικά) μπορεί να μειώσει τον παραλληλισμό αν είναι μεγάλο...
 - ...ή να μην επιτρέπει σε κάθε `thread` ικανό φόρτο εργασίας αν είναι πολύ μικρό
 - Με διαφορετικά δεδομένα εισόδου το όριο μπορεί να πάψει να είναι το ίδιο αποδοτικό

Εναλλακτικές απόπειρες παραλληλισμού (2)

- Να χρησιμοποιούμε έναν σταθερό αριθμό από μόνιμα “worker” threads (thread pool)
 - Θα αναλαμβάνουν διαδοχικά «πακέτα» εργασίας (tasks)
 - Απαιτείται ένας λεπτομερής μηχανισμός **κατανομής** και **εξισορρόπησης** του φόρτου εργασίας
 - Που θα λαμβάνει υπόψη και την **τοπικότητα** των δεδομένων στη μνήμη
 - Παρέχεται από τα διάφορα **threading frameworks**