Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής
ΠΜΣ «Ερευνητικές Κατευθύνσεις στην Πληροφορική»
Παράλληλη και Κατανεμημένη Υπολογιστική Επεξεργασία
2019-20

Σχεδιασμός Παράλληλων Προγραμμάτων (Ι)

(με τη βοήθεια της παραλληλίας δεδομένων)

http://mixstef.github.io/courses/pms-parcomp/

Μ.Στεφανιδάκης

Στρατηγικές παραλληλοποίησης

- Για την επεκτάσιμη (scalable) συγγραφή παράλληλων προγραμμάτων
- Παραλληλία δεδομένων
  - Data parallelism
  - Η πιο «εύκολη» και επεκτάσιμη στρατηγική παραλληλοποίησης
- Παραλληλία λειτουργιών
  - Functional decomposition (“task parallelism”)
  - Συνήθως σε συνδυασμό με την παραλληλία δεδομένων

Παραλληλία δεδομένων

- Εφαρμογή της ίδια λειτουργίας σε διαφορετικά σύνολα δεδομένων
- Προσοχή: «ίδια λειτουργία» δεν σημαίνει κατ' ανάγκη «ακριβώς ίδιες εντολές»
  - Ο όρος καλύπτει διάφορες μορφές, από την απλή εφαρμογή ενός μετασχηματισμού σε κάθε στοιχείο ενός πίνακα έως πιο σύνθετες μορφές λειτουργιών
- Η παραλληλία αυξάνεται όσο αυξάνονται τα δεδομένα
  - Επεκτασιμότητα (scalability)

Διαθέσιμοι μηχανισμοί

- Παραλληλισμός σε επίπεδο εντολών
  - Η εκτέλεση της ίδιας εντολής σε ομάδες δεδομένων (vector parallelism, π.χ. streaming instructions)
- Παραλληλισμός σε GPU (SIMT)
  - Η μαζικά παράλληλη εκτέλεση των ίδιων λειτουργιών σε διαφορετικά δεδομένα
- Παραλληλισμός σε επίπεδο threads
  - Πολλές διεργασίες (διαφορετικός program counter) εκτελούνται παράλληλα
  - Κατάλληλο για παραλληλία δεδομένων και λειτουργιών

Σειριακή σημασιολογία των προγραμμάτων

- Παράδειγμα loop:

```
for (i=0;i<N;i++) {
  a[i] = func(a[i]);
}
```

- Κλασσικό παράδειγμα παραλληλίας δεδομένων;
  - Θεωρητικά η παραλληλοποίηση μοιάζει πολύ εύκολη
  - Αρκεί να επιτύχουμε παράλληλη εκτέλεση π.χ. κατά ομάδες δεδομένων
- Κατά πόσο αυτό όμως ισχύει;

Σειριακή σημασιολογία των προγραμμάτων

- Παράδειγμα loop:

```
for (i=0;i<N;i++) {
  a[i] = func(a[i]);
}
```

- Τι εννοούμε σε ένα σειριακό πρόγραμμα
  - «Θα επεξεργαστούμε τα στοιχεία του a[i] **το ένα μετά το άλλο**»
- Τι ισχύει για ένα παράλληλο πρόγραμμα
  - «Θα επεξεργαστούμε τα στοιχεία του a[i] **το ένα ανεξάρτητα από το άλλο**»

Παράλληλη σημασιολογία

- Παράδειγμα loop:

```
for (i=0;i<N;i++) {
  a[i] = func(a[i]);
}
```

- Μήπως ο μετασχηματισμός του a[i] επηρεάζει global μεταβλητές;
- Τι θα γινόταν αν π.χ. για τον υπολογισμό του a[i] χρειαζόταν και το a[i - 1];
  - Πολλοί αλγόριθμοι χρησιμοποιούν γειτονικά στοιχεία

Σειριακές αλγοριθμικές δομές

- Ακολουθία (sequence)

```
f();
g();
h();
```

- Υπονοείται μια σειρά που πρέπει να τηρηθεί, ακόμα κι αν δεν υπάρχουν αλληλεξαρτήσεις μεταξύ των f(), g() και h()
  - Για να διατηρηθούν στη σωστή σειρά αλλαγές σε global μεταβλητές (side effects)

Σειριακές αλγοριθμικές δομές

- Επιλογή (selection)

```
if (condition)
  f();
else
  g();
```

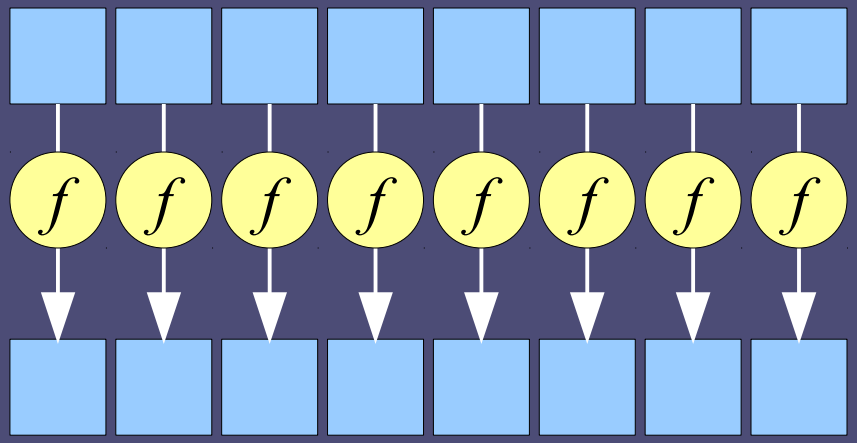
- Υπονοείται ότι, ανάλογα με τη συνθήκη, θα εκτελεστεί ή το f() ή το g()
  - Και ποτέ πριν τον υπολογισμό της συνθήκης

Σειριακές αλγοριθμικές δομές

- Επανάληψη (iteration)
- Όπως είδαμε, είναι δύσκολο στη γενική μορφή να ξέρουμε αν μπορεί να παραλληλοποιηθεί
  - Εξάρτηση από προηγούμενες επαναλήψεις
  - Μη γνωστά/σταθερά όρια εκτέλεσης
  - Επικάλυψη στοιχείων, π.χ. μέσω δεικτών
- Το κυριότερο εμπόδιο είναι ότι μία και μόνο σειριακή αλγοριθμική δομή (η επανάληψη) αντιστοιχεί κατά περίπτωση σε διαφορετικές παράλληλες υλοποιήσεις

Map: μια απλή περίπτωση παραλληλίας δεδομένων

- Εφαρμογή μιας συνάρτησης σε κάθε στοιχείο μιας ακολουθίας δεδομένων
  - Ανεξάρτητες επαναλήψεις
  - Γνωστά όρια εκτέλεσης
  - Εξαρτάται μόνο από το *i* (index) και τα *data[i]*
  - Δεν επηρεάζει global μεταβλητές



Map: μια απλή περίπτωση παραλληλίας δεδομένων

- Σημαντική μορφή παραλληλίας
  - "Embarrassing parallelism"
  - Μπορεί να εκτελεστεί τόσο με vectors, GPUs όσο και με threads
- Προσοχή στην επιβάρυνση των threads
  - Δεν δικαιολογείται το σχήμα «ένα στοιχείο ανά thread»
  - Συνήθως σε ομάδες στοιχείων (blocks) – καλύτερη εκμετάλλευση και της κρυφής μνήμης
- Τι συμβαίνει όταν η συνάρτηση δεν είναι ισοβαρής;
  - Π.χ. όταν η δουλειά του *f(i)* είναι ανάλογη του *i*

Βιβλιογραφία

- Michael McCool, James Reinders, and Arch Robison. 2012. *Structured Parallel Programming: Patterns for Efficient Computation* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.