

Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής  
Παράλληλος Προγραμματισμός  
2023-24

# Παραλληλία Δεδομένων

(και η λειτουργία map)

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

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 SIMD 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()

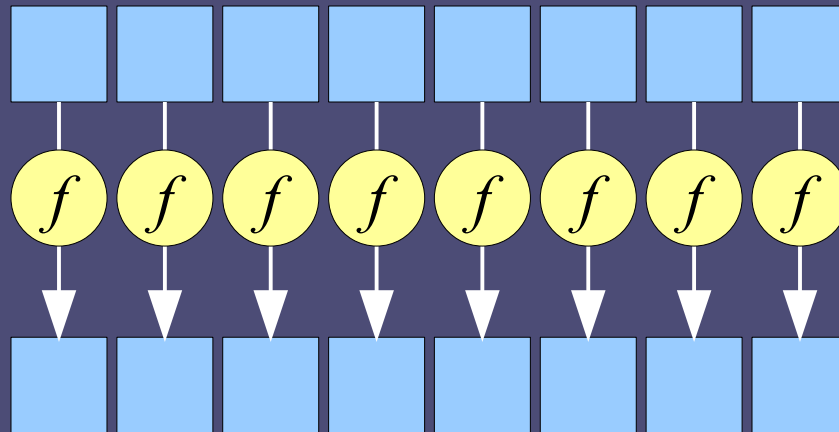
- Και ποτέ πριν τον υπολογισμό της συνθήκης

# 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
- **Streaming SIMD Instructions**
  - Single Instruction Multiple Data
  - Θα εξετάσουμε αρχικά την υλοποίηση του map μέσω εντολών SIMD στην αρχιτεκτονική x86
    - Εντολές που εκτελούν την ίδια πράξη σε μια ομάδα «πακεταρισμένων» (packed) δεδομένων
    - ALU και καταχωρητές μεγάλου εύρους (128 – 512 bits)

# Προγραμματισμός εντολών SSE/AVX

- **Include headers**
  - `#include <immintrin.h>`
  - Ορίζονται οι τύποι δεδομένων και οι διαθέσιμες λειτουργίες
- **Compiler flags**
  - Προσδιορίζουμε ποιο/ποια σετ εντολών χρησιμοποιούμε
  - Π.χ. `-mavx -mavx2 -mfma`
  - Ο εκτελέσιμος κώδικας που παράγεται πρέπει να υποστηρίζεται από το σύστημα εκτέλεσης

# Προγραμματισμός εντολών SSE/AVX

- **Τύποι δεδομένων**

- Θα ανατεθούν σε 16 καταχωρητές
  - xmm (128-bit) ή ymm (256-bit)
- `__m128` 4 πακεταρισμένοι floats
- `__m128d` 2 πακεταρισμένοι doubles
- `__m128i` πακεταρισμένοι ints
  - 8, 16, 32, 64 ή 128 bits ο καθένας, το ακριβές σχήμα εξαρτάται από την εντολή
- `__m256` 8 πακεταρισμένοι floats
- `__m256d` 4 πακεταρισμένοι doubles
- `__m256i` πακεταρισμένοι ints
  - 8, 16, 32, 64 ή 128 bits ο καθένας, το ακριβές σχήμα εξαρτάται από την εντολή

# Προγραμματισμός εντολών SSE/AVX

- **Intrinsics**

- Μοιάζουν με κλήσεις συναρτήσεων και χρησιμοποιούν μεταβλητές για ορίσματα αλλά στην πραγματικότητα είναι οδηγίες προς τον μεταγλωττιστή να εισάγει συγκεκριμένες εντολές sse/avx
  - Παράδειγμα: `*pc = _mm256_add_ps(*pa,*pb);`
  - `_mm256` επιστρέφεται 256-bit ποσότητα (`_mm` για 128-bit)
  - `add` κάνει παράλληλη πρόσθεση
  - `ps` σε πακεταρισμένους floats (`pd` για doubles, `epi8`, `epi16`, `epi32`, `epi64`, `epi128` για ints, `epu..` για unsigned ints)

# Προγραμματισμός εντολών SSE/AVX

- Χρήση

- Μεταβλητές τύπου `__mm256` κλπ: θα ανατεθούν σε καταχωρητές `xmm` ή `ymm`
  - Υπάρχουν «ψευδο»intrinsic που αρχικοποιούν στις επιθυμητές τιμές
  - Υπάρχουν intrinsic που φορτώνουν ένα «πακέτο» αριθμών από τη μνήμη και τα αντίστοιχα intrinsic για εγγραφή
  - Για να γίνονται αποδοτικά αναγνώσεις και εγγραφές θα πρέπει τα δεδομένα να είναι ευθυγραμμισμένα (aligned) ως προς 16 (για 128-bit) ή 32 (για 256-bit): η διεύθυνση του πρώτου byte του «πακέτου» θα πρέπει να είναι πολλαπλάσιο το 16 (ή 32)



# Προγραμματισμός εντολών SSE/AVX

- Χρήση

- Μεταβλητές τύπου δείκτη σε `__m256` κλπ:  
χρησιμοποιούνται για να δείξουν σε θέσεις στη μνήμη για ανάγνωση ή εγγραφή «πακέτων» δεδομένων
  - Η μνήμη αυτή θα πρέπει να έχει τη σωστή ευθυγράμμιση  
`float *a;`  
`i = posix_memalign((void **)&a, 32, 8*sizeof(float));`
  - Οι μεταβλητές τύπου δείκτη αρχικοποιούνται να δείχνουν το δεσμευμένο χώρο και χρησιμοποιούνται στα intrinsics  
`__m256 *pa;`  
`pa = (__m256 *)a;`  
`*pc = _mm256_add_ps(*pa, *pb);`

# 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.