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();
```

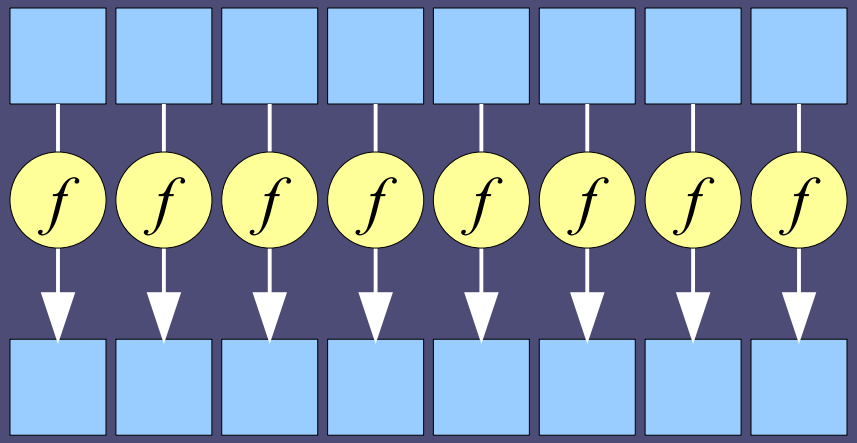
- Υπονοείται ότι, ανάλογα με τη συνθήκη, θα εκτελεστεί ή το f() ή το g()
  - Και ποτέ πριν τον υπολογισμό της συνθήκης

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»intrinsics που αρχικοποιούν στις επιθυμητές τιμές
    - Υπάρχουν intrinsics που φορτώνουν ένα «πακέτο» αριθμών από τη μνήμη και τα αντίστοιχα intrinsics 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 __mm256 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

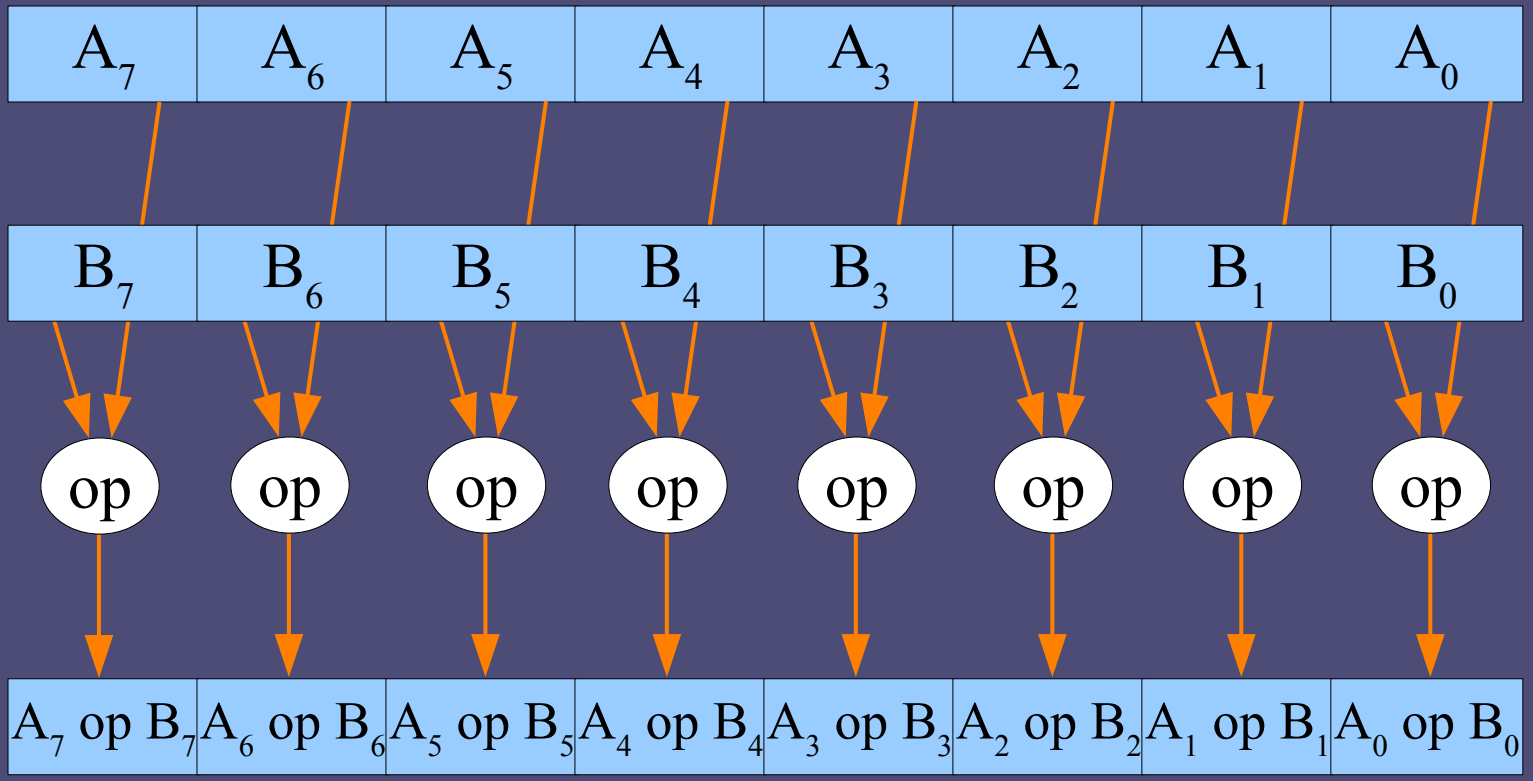
```
float *a;
i = posix_memalign((void **)&a,32,8*sizeof(float));
```

    - 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);
```

«Κάθετες» εντολές SSE/AVX

- dst[i] = a[i] op b[i]



«Κάθετες» εντολές SSE/AVX

- Αριθμητικές πράξεις
  - add, sub, addsub, mul, div, max, min
    - π.χ. __m256 _mm256_add_ps (__m256 a, __m256 b)
    - π.χ. __m256i _mm256_max_epi32 (__m256i a, __m256i b) αλλά και __m256i _mm256_max_epu32 (__m256i a, __m256i b)
  - sqrt, rsqrt, rcp, ceil, floor, round
    - π.χ. __m256d _mm256_sqrt_pd (__m256d a)
  - fmadd, fmsub, fmaddsub, fnmadd, fnmsub (fma flag)
    - π.χ. __m256 _mm256_fmadd_ps (__m256 a, __m256 b, __m256 c)
- Λογικές πράξεις
  - and, or, xor, andnot
    - π.χ. __m256 _mm256_and_ps (__m256 a, __m256 b)
    - π.χ. __m256i _mm256_and_si256 (__m256i a, __m256i b)

Αρχικοποιήσεις «πακέτων» 256 (και 128) bits

- Πρόκειται για «ψευδο»-intrinsics
  - Δεν αντιστοιχούν σε μία μοναδική εντολή μηχανής, ο μεταγλωττιστής θα προσθέσει μια ακολουθία εντολών για την αρχικοποίηση ενός «πακέτου» δεδομένων 256/128 bits
  - Πρέπει να χρησιμοποιούνται προσεκτικά σε σημεία κώδικα κρίσιμα για την απόδοση
    - π.χ. __m256 _mm256_set1_ps (float a)
    - π.χ. __m256 _mm256_set_ps (float e7, float e6, float e5, float e4, float e3, float e2, float e1, float e0)
    - π.χ. __m256 _mm256_setr_ps (float e7, float e6, float e5, float e4, float e3, float e2, float e1, float e0) (ανάποδη σειρά αρχικοποίησης)
- Μοναδική εξαίρεση: η ανάθεση του 0
  - Υλοποιείται πολύ αποδοτικά με εντολή xor
    - π.χ. __m256 _mm256_setzero_ps (void)

Intrinsics για ανάγνωση/εγγραφή στη μνήμη

- Μεταφορά ενός «πακέτου» από/προς τη μνήμη
  - Μη ευθυγραμμισμένες μεταφορές
    - Οποιαδήποτε διεύθυνση μνήμης
    - π.χ. __m256 _mm256_loadu_ps (float const * mem_addr)
    - π.χ. void _mm256_storeu_ps (float * mem_addr, __m256 a)
  - Ευθυγραμμισμένες μεταφορές (πιο αποδοτικές!)
    - Στα 32 bytes (για 256 bits) και 16 bytes (για 128 bits)
    - Πιθανό σφάλμα αν η διεύθυνση που θα δοθεί δεν είναι ευθυγραμμισμένη
    - π.χ. __m256 _mm256_load_ps (float const * mem_addr)
    - π.χ. void _mm256_store_ps (float * mem_addr, __m256 a)
- Πρακτικά δεν χρειάζονται αν χρησιμοποιούμε δείκτες
  - Ο μεταγλωττιστής θα προσθέσει αυτόματα τις κατάλληλες εντολές load/store π.χ. στο `*pc = _mm256_add_ps(*pa,*pb);`

Συγκρίσεις SSE/AVX

- Είναι «οριζόντιες» πράξεις επίσης
  - Ανάλογα με το αποτέλεσμα αποθηκεύουν όλο άσσους (αληθές) ή όλο μηδενικά (ψευδές) στο αντίστοιχο τμήμα
    - π.χ. __m256d _mm256_cmp_pd (__m256d a, __m256d b, const int imm8) με τα 5 λιγότερο σημαντικά bits του imm8 να προσδιορίζουν τη συνθήκη
    - π.χ. __m256i _mm256_cmpgt_epi64 (__m256i a, __m256i b)
  - Προσοχή: η ποσότητα «όλο άσσοι» σε floats αντιστοιχεί σε NaN (Not a Number) – δεν μπορεί να χρησιμοποιηθεί ως «κανονικός» αριθμός float!

Συγκρίσεις SSE/AVX

- Χρήση του αποτελέσματος της σύγκρισης
  - Είναι μια «μάσκα» για εκτέλεση επόμενων πράξεων υπό συνθήκη (if …)
  - Μπορεί να χρησιμοποιηθούν λογικές πράξεις (and/andnot, or) για να βάλουμε σε επιλεγμένα τμήματα μιας 256- (ή 128-)bit ποσότητας κάποια τιμή
  - Ή να χρησιμοποιήσουμε το intrinsic blendv
    - π.χ. __m256 _mm256_blendv_ps (__m256 a, __m256 b, __m256 mask) που επιστρέφει ένα «πακέτο» 256-bit με 8 floats: όπου η μάσκα είναι 0 (ψευδής) η έξοδος έχει το αντίστοιχο τμήμα του a, αλλιώς το αντίστοιχο τμήμα του b

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.
- Intel® Intrinsics Guide (https://software.intel.com/sites/landingpage/IntrinsicsGuide/)