

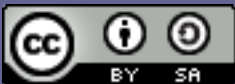
Λειτουργίες reduction σε GPU

(Εισαγωγή σε συγχρονισμό και shared memory)

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

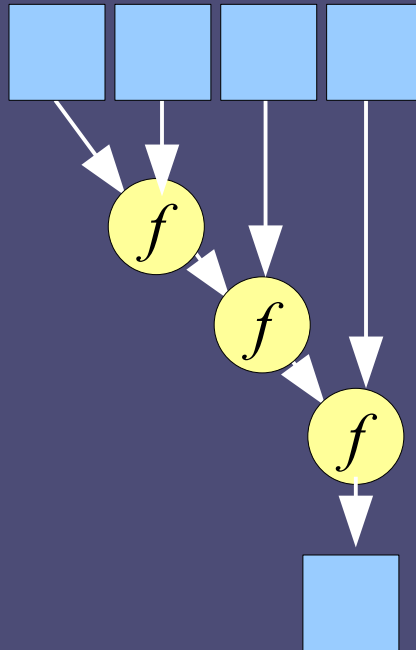
*Το εργαστήριο του μαθήματος χρησιμοποιεί υπολογιστικούς πόρους
AWS Cloud χρηματοδοτούμενους από το ΕΔΥΤΕ*

Μ.Στεφανιδάκης



Reduction

- Συνδυάζει όλα τα στοιχεία μιας συλλογής (collection) σε ένα μοναδικό στοιχείο μέσω τελεστή f



Reduction σε GPU

- **Απαιτείται κάποιου είδους συγχρονισμός**
 - Για την παραγωγή ενός και μοναδικού τελικού αποτελέσματος
- **Τι γνωρίζουμε ως τώρα;**
 - Το μόνο είδος συγχρονισμού που έχουμε συναντήσει είναι **έμμεσο**, μετά την ολοκλήρωση του kernel
 - Πριν τη μεταφορά των αποτελεσμάτων πίσω στο host
 - Εγγύηση ότι έχουν τελειώσει όλα τα threads, όλων των blocks, για το σύνολο του grid
 - Γνωρίζουμε επίσης ότι δεν μπορεί να υπάρξει συγχρονισμός μεταξύ των διαφορετικών blocks
 - Δεν γνωρίζουμε καν με ποια σειρά θα εκτελεστούν τα blocks

Reduction: πρώτες απόπειρες

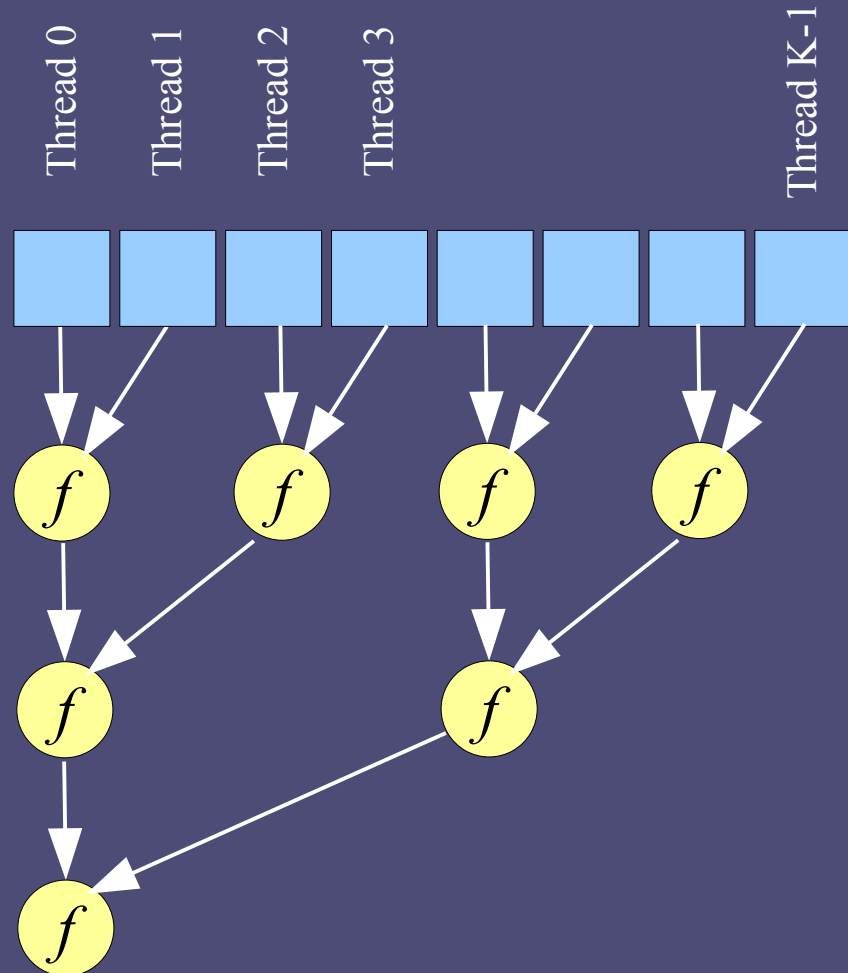
- **1 thread / 1 block**
 - Αναπαραγωγή του σειριακού προγράμματος
 - Αδικοιολόγητη σπατάλη υπολογιστικών πόρων της GPU
- **K threads / 1 block (π.χ. K=256)**
 - Κάθε thread αθροίζει μια ομάδα στοιχείων του array εισόδου
 - Το αποτέλεσμα είναι ένα array μερικών αποτελεσμάτων
 - Ο υπολογισμός του τελικού αποτελέσματος (συνδυασμός μερικών αθροισμάτων) πρέπει να γίνει
 - Είτε στο host (CPU) – ok γιατί τα μερικά αποτελέσματα είναι λίγα
 - Είτε από δεύτερο kernel που ακολουθεί – στη default λειτουργία οι kernels εκτελούνται ο ένας μετά τον άλλον

Αν αυξήσουμε τον αριθμό των blocks;

- Η λύση δεν είναι επεκτάσιμη
 - Τα μερικά αθροίσματα συνεχώς θα μεγαλώνουν σε μέγεθος (blocks * threads)
- Αν κάθε block συνδύαζε όλα τα μερικά αποτελέσματα των threads του σε έναν αριθμό;
 - Τα μερικά αθροίσματα θα έχουν μέγεθος ίσο με τον αριθμό των blocks
 - Διαχειρίσιμο μέγεθος, ακόμα κι από CPU
 - Το τελικό αποτέλεσμα πρέπει όπως και πριν να υπολογιστεί **μετά τη λήξη** του συνολικού kernel
 - Θυμηθείτε: δεν μπορούμε να συγχρονίσουμε τα blocks μεταξύ τους!

Reduction σε ένα block

- Η γενική ιδέα
 - Reduction σε βήματα
 - Σε κάθε βήμα ο αριθμός των threads που είναι ενεργά **υποδιπλασιάζεται**
 - Το τελευταίο thread παράγει το μερικό αποτέλεσμα για το block



Reduction σε ένα block: προβλήματα

- **Συγχρονισμός μεταξύ threads**
 - Σε κάθε βήμα πρέπει να είμαστε σίγουροι ότι τα threads που συμμετείχαν στο προηγούμενο βήμα έχουν τελειώσει
- **Χώρος αποθήκευσης**
 - Απαιτείται η προσπέλαση δεδομένων από άλλα threads
 - Δεν μπορούμε να χρησιμοποιήσουμε καταχωρητές (ξεχωριστοί ανά thread)
 - Χρειαζόμαστε έναν κοινό χώρο (shared memory) **πιο γρήγορο** από την κύρια μνήμη της GPU
- **Σειρά υπολογισμού**
 - Το σχήμα στην προηγούμενη διαφάνεια δεν είναι βέλτιστο
 - Μεγάλος βαθμός απόκλισης εκτέλεσης των threads στα warps
 - Μη αποδοτική προσπέλαση μνήμης

Συγχρονισμός threads του ίδιου block

```
// synchronize all threads of block  
__syncthreads();
```

- Πρέπει να εκτελεστεί από όλα τα threads του block
 - Για να συνεχιστεί η εκτέλεση μετά το σημείο αυτό
- Προσοχή! Δεν πρέπει να μπαίνει σε τμήματα κώδικα υπό συνθήκη (if..)
 - Αν κάποια threads του block δεν εκτελέσουν το `__syncthreads()` τότε η εκτέλεση στο block αυτό θα «παγώσει»

Shared Memory

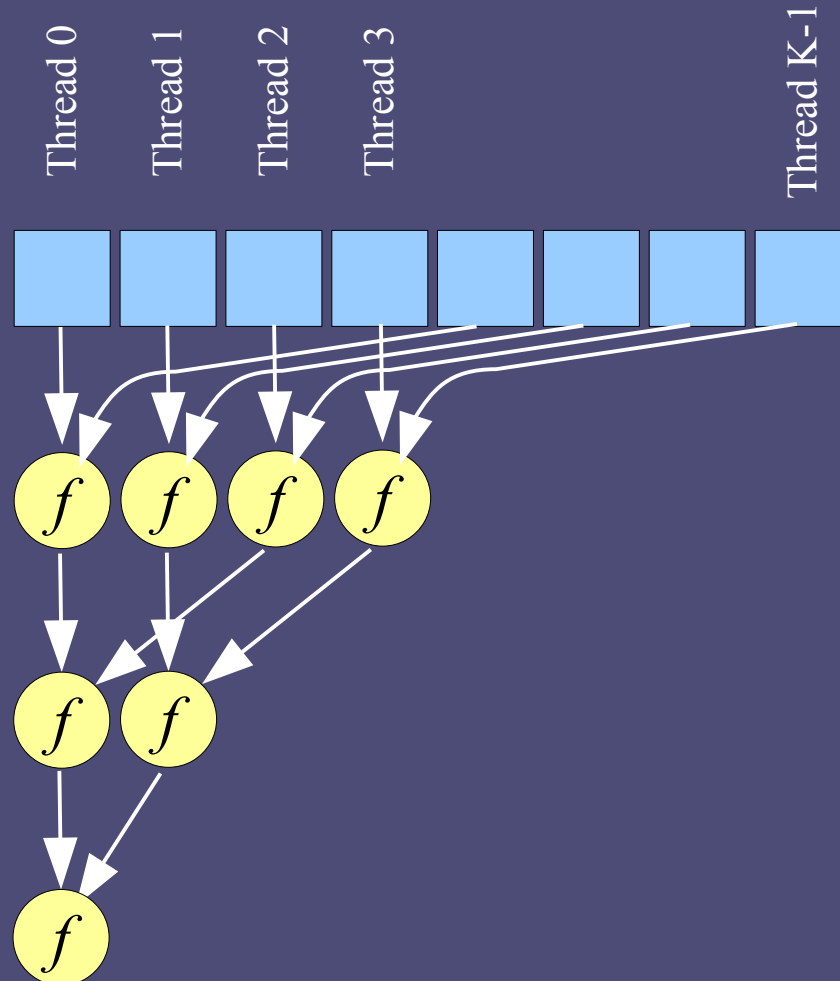
- **Μικρή γρήγορη μνήμη μέσα σε κάθε SM**
 - Σε ιδανικές συνθήκες σχεδόν τόσο γρήγορη όσο οι καταχωρητές
 - Τυπικά μεγέθη 64KB έως 96KB
 - Μοιράζεται τον χώρο με L1 Cache/Texture memory
- **Δέσμευση ατομικού χώρου ανά block**
 - Για όλη τη διάρκεια εκτέλεσης του block
 - Διαθέσιμη σε όλα τα threads του block

Δήλωση χώρου από το shared memory

```
__global__ void sumReduction(float *a, float *psums) {  
    __shared__ float buffer[THREADS];
```

- Στο παράδειγμα φαίνεται η δήλωση με στατικό μέγεθος, μέσα στον κώδικα του kernel
 - Υπάρχει και η δυνατότητα δήλωσης με δυναμικό μέγεθος (ως τρίτη παράμετρος στην εκκίνηση του kernel) αλλά δεν θα τη χρησιμοποιήσουμε εδώ

Βελτιωμένο σχήμα reduction ενός block



Παράδειγμα κώδικα για το reduction στο block

```
// reduce block results, number of threads must be a power of 2
for (unsigned int stride = blockDim.x/2; stride>0; stride /= 2)
{
    if (tid<stride) {
        buffer[tid] += buffer[tid+stride];
    }

    // synchronize all threads of block before next step
    __syncthreads();
}
```

- Σε κάθε βήμα τα threads που βρίσκονται στο «κάτω μισό» προσθέτουν την αντίστοιχη τιμή από το «πάνω μισό»
 - tid είναι το threadIdx.x
- Ο χώρος μειώνεται στο μισό και επαναλαμβάνουμε στο επόμενο βήμα

Άλλες χρήσεις Shared Memory

- **Βελτίωση προσπέλασης κύριας μνήμης GPU**
 - Το σύστημα μνήμης (global memory) της GPU επιτυγχάνει τη μέγιστη απόδοση όταν οι ζητούμενες λέξεις βρίσκονται σε γειτονικές (συνεχόμενες) διευθύνσεις (coalesced memory accesses)
 - Πάντως, οι μοντέρνες GPU έχουν ιεραρχίες μνήμης ικανές να αμβλύνουν σε μεγάλο βαθμό τις επιπτώσεις από την προσπέλαση μη συνεχόμενων διευθύνσεων
 - Εάν ένας αλγόριθμος απαιτεί την προσπέλαση μη συνεχόμενων διευθύνσεων (π.χ. striding σε προσπέλαση πινάκων) μπορεί να χρησιμοποιηθεί ο χώρος του shared memory για την πιο ευνοϊκή αναδιάταξη των δεδομένων
 - Το κόστος προσπέλασης του shared memory είναι πολύ μικρό, ακόμα και με τη χρήση μη συνεχόμενων διευθύνσεων

Συγκρούσεις προσπέλασης Shared Memory

- Η αρχιτεκτονική του shared memory επιτρέπει την ταυτόχρονη προσπέλαση από όλα τα threads ενός warp
 - 32 αναγνώσεις σε γειτονικές διευθύνσεις μπορούν να εκτελεστούν ταυτόχρονα
 - Το shared memory είναι χωρισμένο σε 32 μέρη (banks), τα οποία μπορούν να προσπελαστούν την ίδια στιγμή
 - Εάν οι ζητούμενες διευθύνσεις από τα threads πέφτουν στο ίδιο bank, τότε η προσπέλαση γίνεται σειριακά (bank conflict)
 - Αριθμός bank = διεύθυνση shared memory % 32
 - Για τη μέγιστη δυνατή απόδοση θα πρέπει να αποφεύγονται τα bank conflicts