Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής
Παράλληλος Προγραμματισμός
2024-25

Λειτουργίες reduction σε GPU

(Θέματα συγχρονισμού και shared memory)

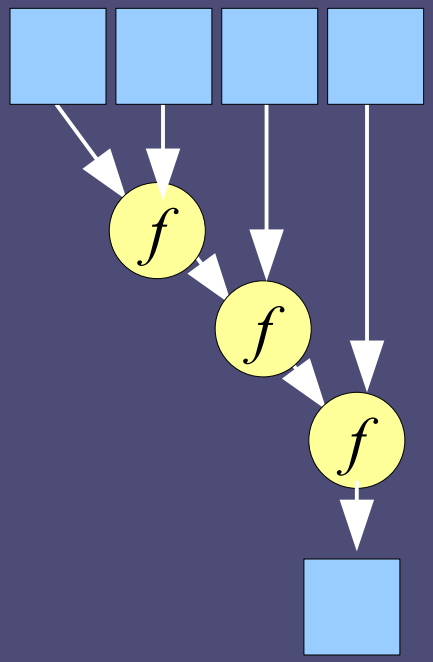
https://mixstef.github.io/courses/parprog/

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 του ενός και μοναδικού τελικού αποτελέσματος
- Τι γνωρίζουμε ως τώρα;
  - Το πρώτο είδος συγχρονισμού που έχουμε συναντήσει είναι έμμεσο, μετά την ολοκλήρωση του kernel
    - Πριν τη μεταφορά των αποτελεσμάτων πίσω στο host ή την εκτέλεση ενός νέου kernel
    - 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

Συγχρονισμός threads του ίδιου block

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

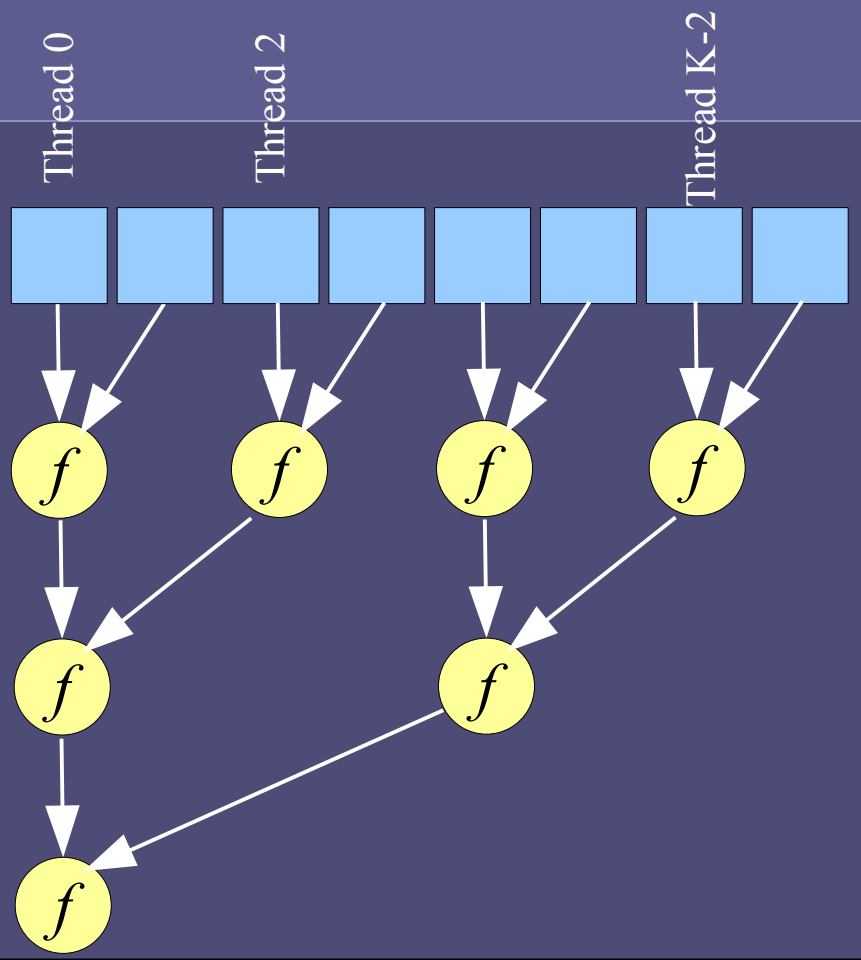
- Η πιο βασική λειτουργία συγχρονισμού
  - Πρέπει να εκτελεστεί από όλα τα threads ενός block για να συνεχιστεί η εκτέλεση μετά το σημείο αυτό (μορφή barrier)
    - Σε τμήματα κώδικα υπό συνθήκη, αν κάποια threads του block δεν εκτελέσουν το __syncthreads() τότε η εκτέλεση στο block αυτό θα «παγώσει»
  - Εγγυάται ότι όλες οι προσπελάσεις μνήμης (global και shared) από τα threads του block πριν το __syncthreads() έχουν ολοκληρωθεί
    - Τα threads του block θα δουν εγγυημένα τις αλλαγές στη μνήμη μετά το __syncthreads()

Υπολογισμός αποτελέσματος ανά block

- Κάθε block συνδυάζει όλα τα μερικά αποτελέσματα των threads του σε ένα μερικό άθροισμα
  - Τα μερικά αθροίσματα θα έχουν μέγεθος ίσο με τον αριθμό των blocks
    - Διαχειρίσιμο μέγεθος, ακόμα κι από CPU
  - Το τελικό αποτέλεσμα πρέπει να υπολογιστεί μετά τη λήξη του συνολικού kernel
    - Θυμηθείτε: δεν μπορούμε να συγχρονίσουμε τα blocks μεταξύ τους!

Reduction σε ένα block

- Η αρχική 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




```
for (int stride=1;stride<THREADS;stride*=2) {
        if (threadIdx.x%(2*stride) == 0) {
            buffer[threadIdx.x] += buffer[threadIdx.x + stride];
        }
        // synchronize all threads of block
        __syncthreads();
}
```

Reduction σε ένα block: απαιτήσεις

- Συγχρονισμός μεταξύ threads
  - Σε κάθε βήμα μείωσης πρέπει να είμαστε σίγουροι ότι τα threads που συμμετείχαν στο προηγούμενο βήμα έχουν τελειώσει → syncthreads()
- Χώρος αποθήκευσης
  - Απαιτείται η προσπέλαση δεδομένων από άλλα threads του ίδιου block
    - Δεν μπορούμε να χρησιμοποιήσουμε καταχωρητές (είναι ξεχωριστοί ανά thread)
    - Χρειαζόμαστε έναν κοινό χώρο πιο γρήγορο από την κύρια μνήμη της GPU για ανταλλαγή δεδομένων μεταξύ των threads του block → shared memory

Δήλωση χώρου από το shared memory

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

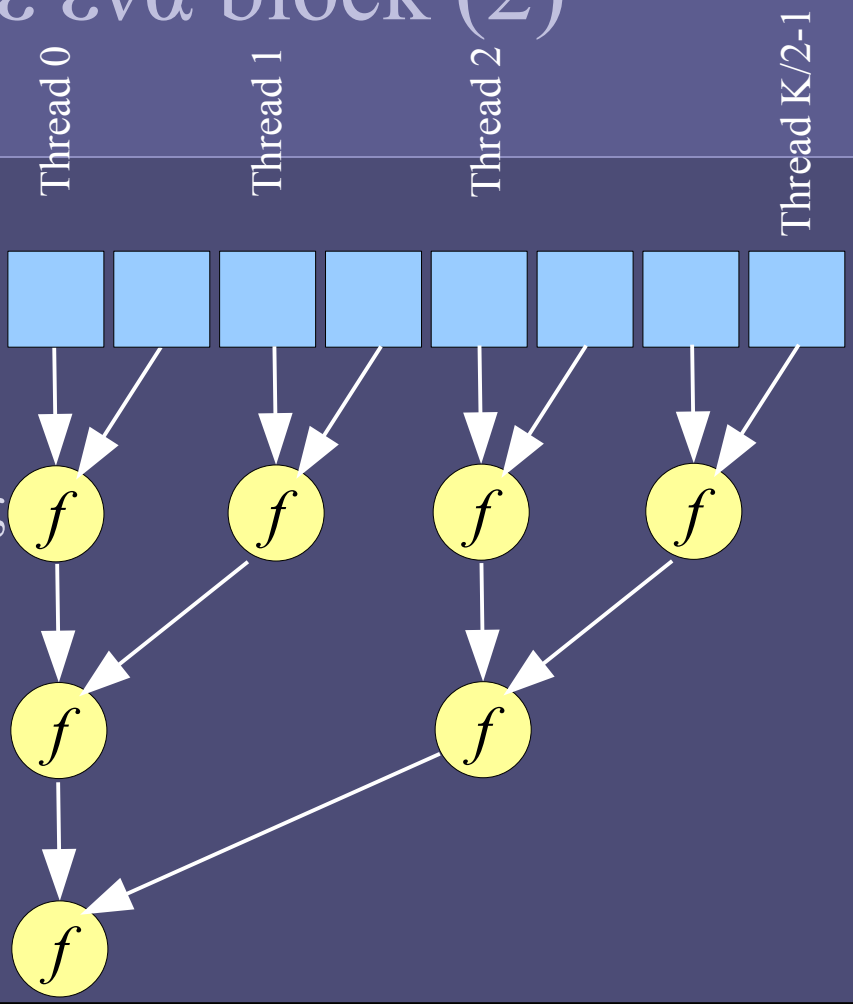
- Χώρος για ανταλλαγή δεδομένων μεταξύ threads του block
  - Με μέγεθος ίσο με τον αριθμό των threads του block
  - Κάθε ένα από τα ενεργά threads συνδυάζει δύο αποτελέσματα από προηγούμενο βήμα (2 αναγνώσεις) και γράφει το αποτέλεσμα πίσω στο buffer (1 εγγραφή)

Απόδοση block reduction

- Απόκλιση εκτέλεσης των threads κάθε warp
  - Το σχήμα στην προηγούμενη διαφάνεια δεν είναι βέλτιστο
  - Τα threads εκτελούν δύο διαφορετικά μονοπάτια
    - Συμμετέχουν στη μείωση ή όχι
  - Χωρίς να υπάρχει ομοιομορφία στην απόκλιση εκτέλεσης
    - Στο ίδιο warp υπάρχουν εναλλάξ ενεργά και μη threads
    - Κάποια warps έχουν ένα ενεργό thread μέχρι το τέλος του υπολογισμού

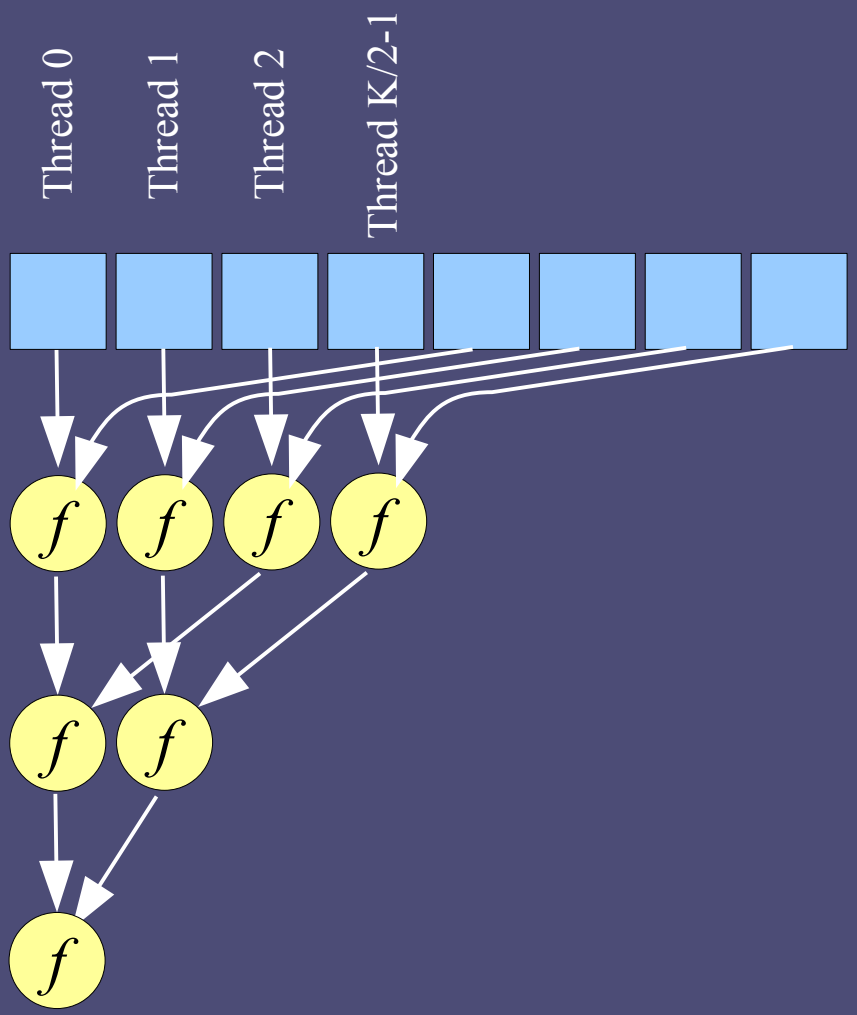
Reduction σε ένα block (2)

- Βελτίωση απόκλισης εκτέλεσης
  - Ταυτόχρονα όμως αυξάνονται οι συγκρούσεις (bank conflicts) σε shared memory (>50x)
  - Τα threads του ίδιου warp προσπελαύνουν το ίδιο bank ταυτόχρονα




```
for (int stride=1;stride<THREADS;stride*=2) {
        int index = 2*stride*threadIdx.x;
        if (index<THREADS) {
            buffer[index] += buffer[index + stride];
        }
        // synchronize all threads of block
        __syncthreads();
}
```

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