

Προγραμματισμός σε GPUs

(το προγραμματιστικό μοντέλο CUDA)

Διασύνδεση CPU-GPU

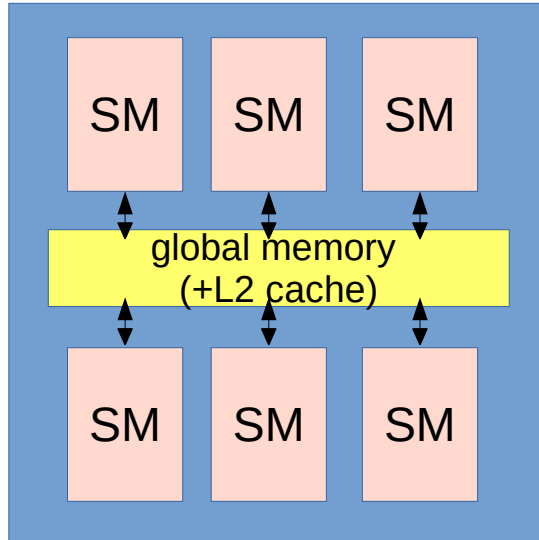
- Διαφορετικές αρχιτεκτονικές
 - Εάν το device είναι εξωτερικό: μέσω διαύλου E/E (π.χ. PCIe ή εξειδικευμένο link/bridge μεταξύ καρτών GPU)
 - Εάν το device βρίσκεται στο ίδιο chip με τη CPU: είτε μέσω εξειδικευμένου διαύλου είτε μέσω κοινής μνήμης (δηλ. το device είναι ένα διαφορετικό είδος core, παράλληλα με τα CPU cores).

Προγραμματισμός σε GPUs: ιδιαιτερότητες

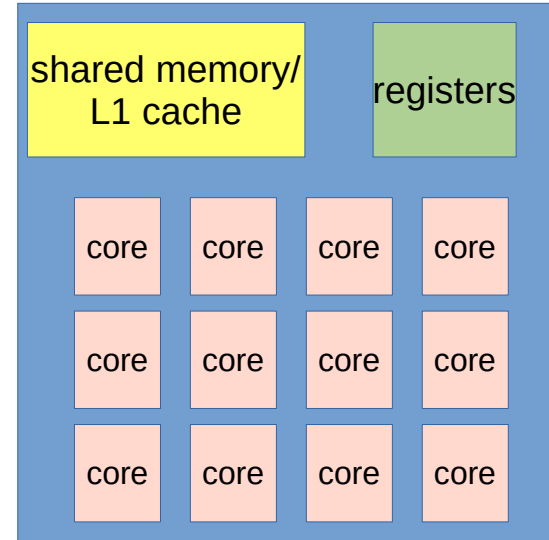
- Πού εκτελείται το πρόγραμμα;
 - ένα μέρος στη CPU (“host”) και ένα μέρος σε κάποια GPU/accelerator (“device”)
- Που βρίσκονται τα δεδομένα;
 - στην κύρια μνήμη του υπολογιστή
 - σε κάποια μνήμη της GPU
 - μια τυπική GPU έχει διάφορα είδη μνήμης
 - ο προγραμματιστής πρέπει να φροντίζει για την επιλογή της πιο κατάλληλης
 - συχνά προκύπτει η ανάγκη μεταφοράς δεδομένων μεταξύ διαφορετικών μνημών
 - πρέπει να συνυπολογίζεται το κόστος μεταφοράς στη συνολική απόδοση

GPUs/accelerators

GPU ή accelerator

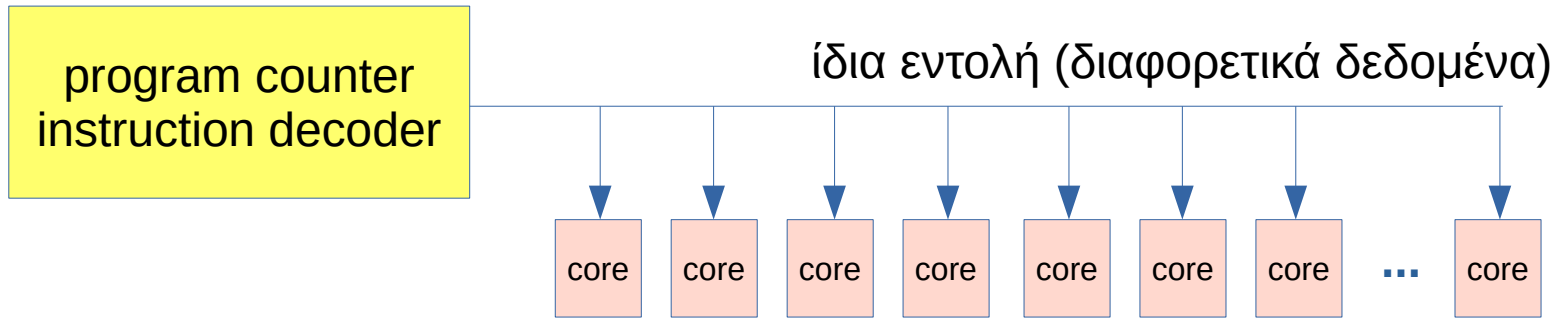


(SM)



- Μια GPU αποτελείται από Streaming Multiprocessors (SMs), πρόσβαση όλων σε global memory
- Κάθε SM διαθέτει έναν αριθμό υπολογιστικών cores, καθώς και ένα ξεχωριστό set καταχωρητών και μια μοιραζόμενη μνήμη (shared memory)
 - Τα cores έχουν απλή λογική λειτουργίας, σχεδιασμένα να εκτελούν **μαζικά** πράξεις (ίδια πράξη σε μεγάλες ομάδες cores)

GPUs και threads



- Στις GPUs η έννοια του thread είναι διαφορετική απ' ό,τι σε μια CPU
 - Διαθέσιμος (μαζικά) μεγάλος αριθμός hardware threads (cores)
 - Παράλληλη εκτέλεση της ίδιας εντολής από πολλά threads την ίδια στιγμή (μοντέλο SIMT), **χωρίς επιβάρυνση**: όλοι οι πόροι (state) των εκτελούμενων threads βρίσκονται **συνεχώς** στους καταχωρητές του SM
 - Κάθε SM χρονοδρομολογεί διαδοχικά ομάδες threads με **τον ίδιο program counter** (PC) στα cores

Ιδιαιτερότητες εκτέλεσης SIMT

- Τι συμβαίνει στις διακλαδώσεις;

```
if cond {  
    A;  
}  
else {  
    B;  
}
```

- Η ομάδα των threads με ίδιο PC πρέπει να εκτελεστεί **δύο φορές**: την 1η φορά με ενεργοποιημένα μόνο τα threads που εκτελούν το A και τη 2η φορά με ενεργοποιημένα τα threads που εκτελούν το B

Ιδιαιτερότητες εκτέλεσης SIMT

- Αν ένα thread της ομάδας δεν μπορεί να προχωρήσει (π.χ. αναμονή για πόρους);
 - Όλα η ομάδα threads πρέπει να περιμένει
 - Το SM επιλέγει άλλη ομάδα threads που είναι έτοιμη να εκτελεστεί
 - Προσοχή στα barriers: πρέπει να εκτελούνται εκτός διακλαδώσεων
- Τα παραπάνω προσπαθούν να διορθώσουν νεώτερες αρχιτεκτονικές με ανεξάρτητο PC ανά thread

Το προγραμματιστικό μοντέλο CUDA

- Ένας τρόπος «αφαιρετικής» περιγραφής των threads, cores, SMs σε ένα προγραμματιστικό interface
 - Και οι βιβλιοθήκες και εργαλεία που το υλοποιούν
- Για να καλυφθούν με ενιαίο τρόπο GPUs με διαφορετικά χαρακτηριστικά και δυνατότητες
 - Κάποια γνώση των ιδιοτεροτήτων του hardware είναι παρόλα αυτά αναγκαία...

Ορολογία CUDA

- **Kernel**: μια συνάρτηση, περιγράφει τι θα εκτελεστεί στη GPU από κάθε ένα thread
- **Thread**: η μικρότερη μονάδα εκτέλεσης του κώδικα του kernel
- **Block**: μια ομάδα threads που εκτελούν τον ίδιο kernel, εκτελούνται στο ίδιο SM
 - Αν υπάρχουν πόροι, το SM μπορεί να εκτελεί και άλλο block παράλληλα
- **Grid**: το σύνολο των blocks που εκτελούν τον kernel

CUDA block

- Μια ομάδα από threads που εκτελούνται ταυτόχρονα σε ένα SM
- Τα threads αυτά μοιράζονται τους καταχωρητές (registers) και την κοινή μνήμη (shared memory) του SM
 - Κάθε thread έχει δεσμευμένο το δικό του μέρος από τα παραπάνω όσο διαρκεί η εκτέλεση του block
- Τα threads ενός block μπορούν να συγχρονιστούν μεταξύ τους
 - Αντιθέτως, δεν υπάρχει τρόπος να συγχρονιστούν διαφορετικά blocks μεταξύ τους
 - Ούτε μπορούμε να υποθέσουμε με ποια σειρά θα εκτελεστούν τα blocks

Εκτέλεση kernel

- Ο προγραμματιστής ζητά την εκτέλεση ενός kernel με συγκεκριμένο αριθμό **blocks/grid** και **threads/block**
- Τα blocks που αποτελούν το grid του kernel κατανέμονται στα SM της GPU
- Τα SM εκτελούν τα blocks που τους αντιστοιχούν, το ένα μετά το άλλο ή παράλληλα (ανάλογα με τους διαθέσιμους πόρους)
- Κάθε SM χρονοδρομολογεί ομάδες από threads του ίδιου block για εκτέλεση από τα cores του
 - Σε ομάδες με τον ίδιο PC (“warps” στην ορολογία CUDA, 32 threads με την τρέχουσα τεχνολογία)

Παραμετρική εκτέλεση

- Πώς γνωρίζει κάθε thread ποιο μέρος της συνολικής εργασίας θα εκτελέσει
- Κάθε thread έχει διαθέσιμες κατά την εκτέλεση του kernel μια σειρά από μεταβλητές index
- Οι μεταβλητές αυτές μπορούν να οργανωθούν σε 1, 2 ή 3 διαστάσεις
 - Προγραμματιστική ευκολία, για να ταιριάζουν με το είδος της εφαρμογής
 - Δεν αλλάζει η υποκείμενη οργάνωση σε threads/blocks/grid

Οργάνωση σε μία διάσταση

- Μεταβλητές:
 - threadIdx.x = «ποιο thread του block είμαι»
 - blockIdx.x = «σε ποιο block ανήκω»
 - blockDim.x = «πόσα threads υπάρχουν σε κάθε block»
 - gridDim.x = «πόσα blocks υπάρχουν στο grid»
- Σε οργανώσεις με 2 ή 3 διαστάσεις υπάρχουν επιπλέον τα .y και .z των παραπάνω μεταβλητών
- Για όλα τα παραπάνω κάθε GPU έχει περιορισμούς στον μέγιστο αριθμό τους, όπως και στο πόσα threads/block ή blocks/grid μπορούν να εκτελεστούν

Άσκηση #1: Το 1ο πρόγραμμα CUDA

- Υπόδειγμα: **one-addition.cu**, πρόσθεση σε 1 thread / 1 block
 - Τυπική ροή:
 1. Δέσμευση μνήμης στη συσκευή GPU (device)
 2. Μεταφορά δεδομένων στη GPU
 3. Εκτέλεση ενός kernel
 4. Μεταφορά των αποτελεσμάτων πίσω στη μνήμη του υπολογιστή (host)
 5. Αποδέσμευση μνήμης στη συσκευή
 - **nvcc** compiler, στέλνει την κλασσική C/C++ στον gcc (ή αντίστοιχο) ενώ χειρίζεται διαφορετικά τις επεκτάσεις (extensions) της CUDA

Άσκηση #1: Το 1ο πρόγραμμα CUDA

- Συνδεθείτε στο σύστημα δοκιμών
- Μεταφέρετε το πρόγραμμα .cu στον φάκελό σας
- Χρησιμοποιήστε το nvcc για τη μεταγλώττιση
- Εκτελέστε το πρόγραμμα

Άσκηση #2: Πρόσθεση διανυσμάτων

- Υπόδειγμα: **vectoradd-threads-only.cu**
 - Πρόσθεση διανυσμάτων float μεγέθους $N = 100$
 - 1 block / N threads
 - Κάθε thread εκτελεί την πράξη $c[i] = a[i] + b[i]$
 - Παρατηρήστε πώς ορίζεται το i
 - Δοκιμάστε με μεγαλύτερο N
 - Μέχρι ποιο μέγεθος μπορείτε να φτάσετε;
 - Χρησιμοποιήστε το **deviceQuery.cpp** για να δείτε τα όρια της συγκεκριμένης GPU

Άσκηση #2: Πρόσθεση διανυσμάτων

- Δοκιμάστε να αλλάξετε το πρόγραμμα έτσι ώστε να εκτελεί την πρόσθεση των διανυσμάτων με N blocks / 1 thread
 - Πώς θα ορίσετε το i αυτή τη φορά;
 - Ξεκινήστε με $N = 10000$
 - Ποιο είναι το όριο που μπορείτε να φτάσετε στο μέγεθος του N ;

Άσκηση #2: Πρόσθεση διανυσμάτων

- Δοκιμάστε με συνδυασμό αριθμού threads και blocks
 - threads = 256 (σταθερά)
 - blocks = $(N + \text{threads} - 1) / \text{threads}$ (η παλιά καλή φόρμουλα για να βρούμε τον αριθμό blocks που καλύπτει το N)
 - Πώς υπολογίζεται τώρα το i;
 - **Προσοχή**: τώρα μπορεί να εμφανιστεί $i \geq N$, πρέπει να προβλεφθεί στον κώδικα!
 - Δοκιμάστε με $N = 10.000.000$
 - Μέχρι ποιο μέγεθος N μπορείτε να φτάσετε;
 - Εναλλακτικά: **grid-striding loop**
 - Επανάληψη, σε κάθε βήμα το μέγεθος του grid ($= \text{blocks} * \text{threads}$)