Κατανεμημένα και Πολυεπεξεργαστικά Υπολογιστικά Συστήματα

Πολυνημάτωση, πολυπύρηνοι επεξεργαστές κοινόχρηστης μνήμης, πρωτόκολλα συνοχής και Graphics Processing Units (GPUs)

Μιχάλης Ψαράκης

Παραλληλία επιπέδου νήματος

- ILP - εκμεταλλεύεται την ύπαρξη πιθανής παραλληλίας εκτέλεσης σε ένα βρόχο ή σε ένα τμήμα βασικού κώδικα
- Σε μερικές εφαρμογές μπορεί να υπάρχει υψηλή εγγενής παραλληλία (π.χ. βάση δεδομένων)
- Ρητή παραλληλία επιπέδου νήματος (Thread Level Parallelism – TLP)
- Thread (νήμα): διεργασία με δικές της εντολές και δεδομένα
  - Κάθε νήμα διατηρεί όλη την πληροφορία (εντολές, δεδομένα, PC, καταχωρητές, κτλ.) που του επιτρέπει να εκτελεστεί ανεξάρτητα
  - Μπορεί να είναι τμήμα ενός παράλληλου προγράμματος με πολλαπλές διεργασίες ή ένα ανεξάρτητο πρόγραμμα
- Στόχος: Πολλαπλές ροές εντολών για να βελτιωθεί
  - Διεκπεραιωτική ικανότητα (throughput) των επεξεργαστών που τρέχουν πολλά προγράμματα
  - Χρόνος εκτέλεσης των πολυνηματικών προγραμμάτων

Πολυνημάτωση (multithreading)

- Πολυνημάτωση: Εκτέλεση πολλαπλών νημάτων παράλληλα
  - πολλαπλά νήματα μοιράζονται εναλλάξ τις λειτουργικές μονάδες ενός επεξεργαστή
- Ο επεξεργαστής πρέπει να διατηρεί αντίγραφα της κατάστασης κάθε νήματος
  - αντίγραφα καταχωρητών , PC, ξεχωριστός πίνακας σελίδων, κτλ
- Γρήγορη εναλλαγή μεταξύ των νημάτων
  - Με υποστήριξη υλικού
- Πότε γίνεται εναλλαγή νημάτων?
  - Σε κάθε κύκλο ρολογιού (fine grain)
  - Μετά από μεγάλη καθυστέρηση (coarse grain)

Λεπτή πολυνημάτωση

- Λεπτή πολυνημάτωση (fine-grain multithreading):
  - Εναλλαγή νημάτων εκ περιτροπής (round-robin) μετά από κάθε κύκλο
  - Διάπλεξη της εκτέλεσης των εντολών
  - Εάν ένα νήμα σταματήσει (stall), τα υπόλοιπα συνεχίζουν να εκτελούνται
- Ο επεξεργαστής θα πρέπει να υποστηρίζει την εναλλαγή νήματος σε κάθε κύκλο
- Πλεονεκτήματα: μπορεί να κρύψει μικρές και μεγάλες καθυστερήσεις
  - Όταν ένα νήμα σταματήσει, εκτελούνται εντολές από άλλα νήματα
- Μειονεκτήματα: καθυστερεί την εκτέλεση των ξεχωριστών νημάτων
  - Ενώ ένα νήμα μπορεί να ολοκληρώσει την εκτέλεσή του χωρίς stalls θα πρέπει να περιμένει να εκτελεστούν εντολές άλλων νημάτων
- Παράδειγμα: Sun UltraSPARC T1

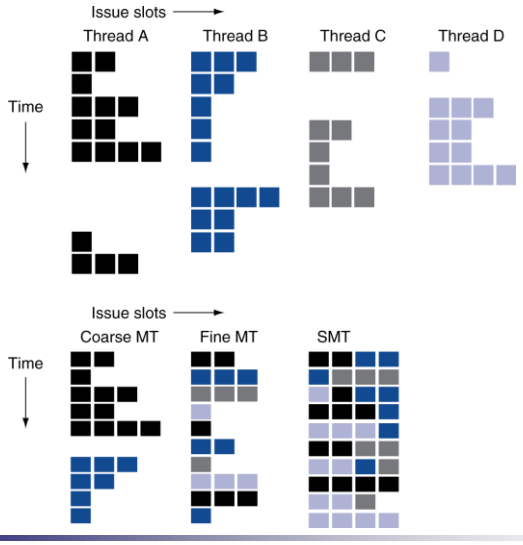
Χονδρή πολυνημάτωση

- Χονδρή πολυνημάτωση (coarse-grain multithreading)
  - Εναλλαγή μόνο μετά από μεγάλη καθυστέρηση (π.χ., αστοχία L2-cache)
- Πλεονεκτήματα:
  - Δεν υπάρχει ανάγκη για γρήγορη εναλλαγή νήματος
  - Δεν καθυστερεί στην εκτέλεση των νημάτων
    - Εντολές από άλλα νήματα εκτελούνται μόνο όταν το ενεργό νήμηα αντιμετωπίσει ένα χρονοβόρο stall
- Μειονεκτήματα:
  - Δεν καλύπτει το χάσιμο χρόνου από τις μικρές καθυστερήσεις (λόγω του χρόνου για να γεμίσει το pipeline)
    - Αφού ο επεξεργαστής εκτελεί εντολές μόνο από ένα νήμα, όταν συμβεί stall, ο μηχανισμός διοχέτευσης θα πρέπει να αδειάσει
    - Το επόμενο νήμα θα πρέπει να γεμίσει το μηχανισμό προτού αρχίσουν να ολοκληρώνονται οι εντολές

Ταυτόχρονη πολυνημάτωση

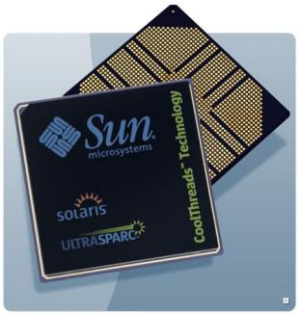
- Ταυτόχρονη πολυνημάτωση (simultaneous multithreading – SMT)
- Σε δυναμικά χρονοπρογραμματιζόμενο επεξεργαστή με εκκίνηση πολλών εντολών
  - Χρονοπρογραμματίζει εντολές από πολλαπλά νήματα
  - Εντολές από ανεξάρτητα νήματα εκτελούνται όταν είναι διαθέσιμες οι λειτουργικές μονάδες
  - Μέσα στα νήματα, οι εξαρτήσεις λύνονται με το χρονο-προγραμματισμό και τη μετονομασία καταχωρητών
- Παράδειγμα: Intel Pentium-4 HT
  - Δύο νήματα: διπλοί καταχωρητές, κοινόχρηστες λειτουργικές μονάδες και κρυφές μνήμες

Παράδειγμα πολυνημάτωσης

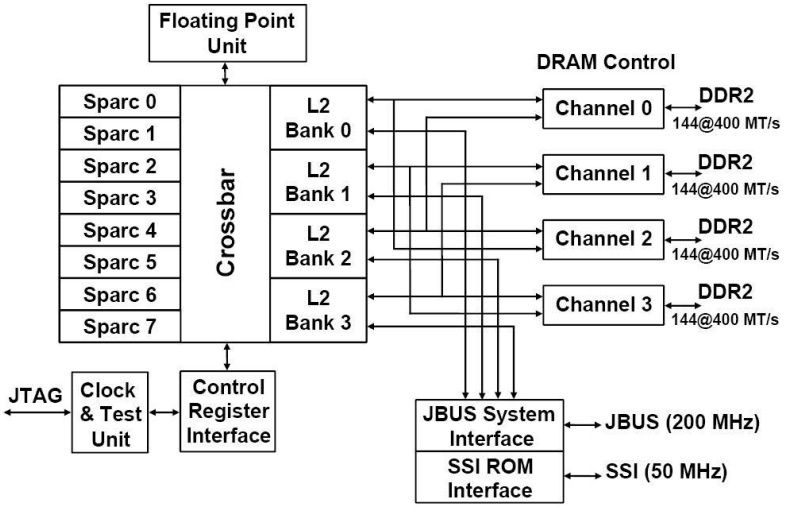


Sun UltraSPARC T1

- Στόχος: server applications
  - Υψηλό TLP
    - Μεγάλος αριθμός από παράλληλες αιτήσεις πελατών
  - Low ILP
    - Υψηλός ρυθμός αστοχίας cache
    - Πολλές εξαρτήσεις δεδομένων
- Παράγοντες που ενδιαφέρουν τα data centers: power, cooling, space
- Μέτρο επίδοσης: Performance/Watt/Sq. Ft.
- Προσέγγιση: Πολλαπλοί πυρήνες, Λεπτή πολυνημάτωση, απλή διοχέτευση, μικρές L1 caches, κοινόχρηστη L2

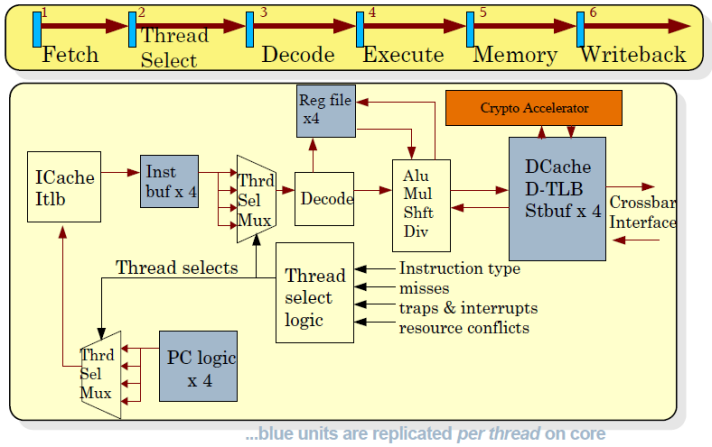


Αρχιτεκτονική του UltraSPARC T1



Διοχέτευση του T1

- Single issue, in-order, 6-stage pipeline
- Κοινόχρηστες μονάδες:
  - L1, L2
  - TLB
  - functional units
  - pipeline registers
- Hazards:
  - Data
  - Structural



Λεπτή πολυνημάτωση του T1

- Κάθε πυρήνας υποστηρίζει 4 νήματα και έχει τοπική L1 cache
  - 16KB I-cache και 8 KB D-cache
- Λεπτή πολυνημάτωση
  - Εναλλαγή νημάτων σε κάθε κύκλο ρολογιού
- Τα ανενεργά (idle) νήματα παρακάμπτονται με τον χρονοπρογραμματισμό
  - Αναμονή λόγω καθυστέρησης του pipeline ή αστοχίας cache
  - Ο επεξεργαστής είναι ανενεργός μόνο όταν και τα 4 νήματα είναι idle ή stalled
- Μια μονάδα κινητής υποδιαστολής μοιράζεται μεταξύ των 8 πυρήνων
  - η απόδοση της αριθμητικής κινητής υποδιαστολής δεν ήταν πρωταρχικός στόχος στον T1

Επιλογή νήματος

- Εναλλαγή νημάτων σε κάθε κύκλο ρολογιού
- Εναλλαγή σε ένα από τα διαθέσιμα (ready to run) νήματα
  - Προτεραιότητα στο least-recently-executed νήμα
- Ένα νήμα γίνεται μη-διαθέσιμο λόγω:
  - Long latency operation (load, branch, mul, or div)
  - Pipeline stall (cache miss, or resource conflict)

UltraSPARC T2: στόχοι

- Διπλασιασμός του throughput έναντι του T1
  - Αύξηση των πυρήνων ή των νημάτων ανά πυρήνα?
  - Χρησιμοποίηση των μονάδων εκτέλεσης
- Βελτίωση της απόδοσης του ενός νήματος
- Βελτίωση της απόδοσης της αριθμητικής κινητής υποδιαστολής

Αλλαγές από τον T1 στον T2

- Αύξηση των νημάτων από 4 σε 8 σε κάθε πυρήνα
  - 8 * 8 = 64 νήματα
- Αύξηση των μονάδων εκτέλεσης από 1 σε 2 σε κάθε πυρήνα
- Μονάδα κινητής υποδιαστολής σε κάθε πυρήνα
- Προσθήκη νέου σταδίου στο μηχανισμό διοχέτευσης
  - Επιλέγει 2 νήματα από τα 8 για εκτέλεση σε κάθε κύκλο
- Αλλαγές στην ιεραρχία μνήμης
  - Αύξηση του set associativity της L1 I-cache σε 8
  - Αύξηση των L2 banks από 4 σε 8

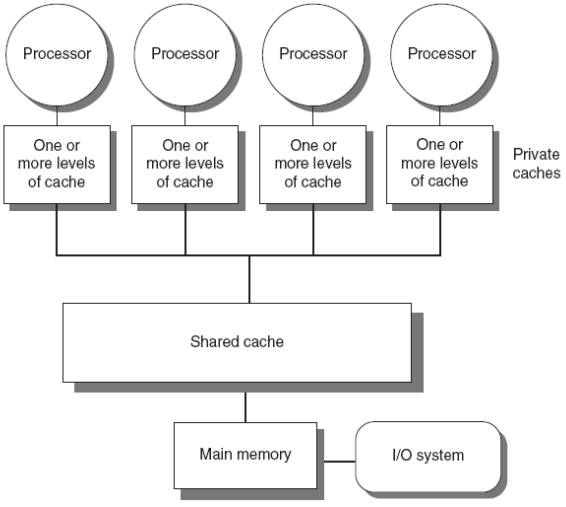
Πολυεπεξεργαστές

- Πολυεπεξεργαστής κοινόχρηστης μνήμης (SMP: shared memory multiprocessor)
  - Το υλικό παρέχει ένα μοναδικό χώρο φυσικών διευθύνσεων για όλους τους επεξεργαστές
- Συγκεντρωμένη (concentrated) ή κατανεμημένη (distributed) κοινόχρηστη μνήμη
  - Ανάλογα με τον αριθμό των επεξεργαστών
  - Χρόνος προσπέλασης μνήμης (memory access time)
    - UMA (uniform) vs. NUMA (nonuniform)

Αρχιτεκτονικές κοινόχρηστης μνήμης

- Συγκεντρωμένη κοινόχρηστη μνήμη (concentrated shared-memory)
  - Μικρός αριθμός επεξεργαστών
  - Μία κοινόχρηστη κεντρική μνήμη
- Κατανεμημένη κοινόχρηστη μνήμη (distributed shared-memory)
  - Μεγαλύτερος αριθμός επεξεργαστών
  - Μνήμη κατανεμημένη στους επεξεργαστές

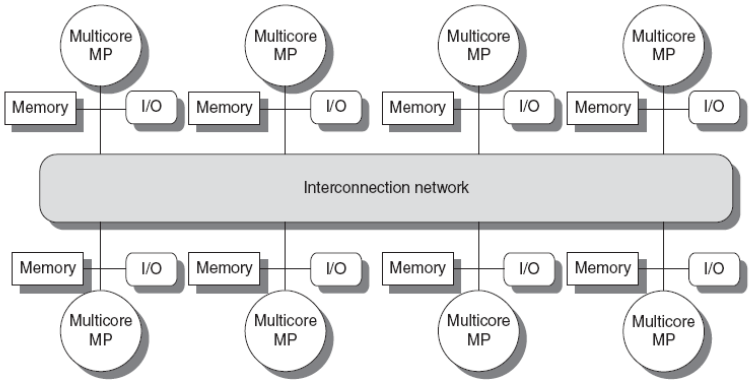
Concentrated shared memory



Concentrated shared memory

- Με μεγάλες κρυφές μνήμες ικανοποιούνται οι απαιτήσεις μνήμης ενός μικρού αριθμού επεξεργαστών
  - Λιγότερο αποδοτική όσο αυξάνει ο αριθμός των πυρήνων
- Αντικατάσταση του μοναδικού διαύλου (bus) με πολλαπλούς διαύλους ή μεταγωγέα (switch matrix)
  - επέκταση σε μεγαλύτερο αριθμό επεξεργαστών
- Symmetric multiprocessors (SMP)
  - ομοιόμορφη προσπέλαση μνήμης (uniform memory access, UMA)

Distributed shared-memory



Distributed shared-memory

- Εύκολος τρόπος για να επεκτείνουμε το εύρος ζώνης της μνήμης
  - εάν οι περισσότερες προσπελάσεις είναι στην τοπική μνήμη
- Μειώνει τον χρόνο προσπέλασης στη μνήμη
- Μειονέκτημα:
  - Πιο πολύπλοκη και χρονοβόρα η ανταλλαγή δεδομένων μεταξύ επεξεργαστών
- Κάθε κόμβος μπορεί να είναι ένας μικρός SMP

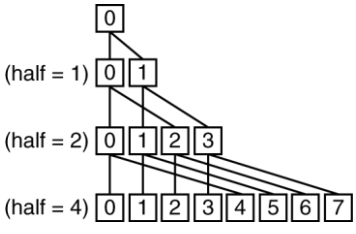
Παράδειγμα: Μείωση αθροισμάτων

- Άθροισμα 100,000 αριθμών σε έναν SMP με 100 πυρήνες
  - Κάθε επεξεργαστής έχει ID: 0 ≤ Pn ≤ 99
  - Διαίρεση των αριθμών: 1000 αριθμοί ανά επεξεργαστή
  - Αρχικό άθροισμα σε κάθε επεξεργαστή

```
sum[Pn] = 0;
  for (i = 1000*Pn; i < 1000*(Pn+1); i = i + 1)
    sum[Pn] = sum[Pn] + A[i];
```

- Τώρα πρέπει να προσθέσουμε αυτά τα μερικά αθροίσματα
  - Μείωση (reduction): διαίρει και βασίλευε
  - Οι μισοί επεξεργαστές προσθέτουν ζεύγη αθροισμάτων, μετά το 1/4, …
  - Απαιτείται συγχρονισμός ανάμεσα στα βήματα μείωσης

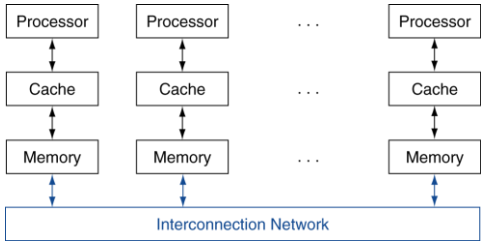
Παράδειγμα: Μείωση αθροισμάτων




```
half = 100;
repeat
  synch();
  if (half%2 != 0 && Pn == 0)
    sum[0] = sum[0] + sum[half-1];
    /* Conditional sum needed when half is odd;
       Processor0 gets missing element */
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
until (half == 1);
```

Μεταβίβαση Μηνυμάτων

- Κάθε επεξεργαστής έχει το δικό του ιδιωτικό χώρο φυσικών διευθύνσεων
- Το υλικό στέλνει/λαμβάνει μηνύματα μεταξύ των επεξεργαστών



Μείωση Αθροισμάτων (ξανά)

- Άθροισμα 100,000 σε 100 επεξεργαστές με μεταβίβαση μηνυμάτων
- Πρώτα, κατανομή 1000 αριθμών στον καθένα
  - Έπειτα, τα μερικά αθροίσματα

```
sum = 0;
for (i = 0; i<1000; i = i + 1)
  sum = sum + AN[i];
```

- Μείωση
  - Οι μισοί επεξεργαστές στέλνουν, οι άλλοι μισοί λαμβάνουν και προσθέτουν
  - Το ¼ στέλνει, το ¼ λαμβάνει και προσθέτει, …

Μείωση Αθροισμάτων (ξανά)

- Έστω οι λειτουργίες send() και receive()

```
limit = 100; half = 100;/* 100 processors */
repeat
  half = (half+1)/2; /* send vs. receive
                        dividing line */
  if (Pn >= half && Pn < limit)
    send(Pn - half, sum);
  if (Pn < (limit/2))
    sum = sum + receive();
  limit = half; /* upper limit of senders */
until (half == 1); /* exit with final sum */
```

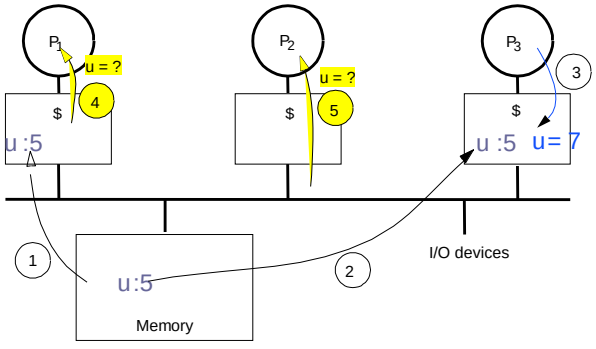
  - Οι εντολές send/receive παρέχουν επίσης συγχρονισμό
  - Υποθέτουμε ότι οι λειτουργίες send/receive διαρκούν περίπου ίδιο χρόνο με την πρόσθεση

Συνοχή κρυφής μνήμης

- Cache coherence problem: διαφορετικοί επεξεργαστές μπορεί να βλέπουν διαφορετικές τιμές για την ίδια θέση μνήμης
  - Παράδειγμα για κρυφή μνήμη ταυτόχρονης εγγραφής (write-through)

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Παράδειγμα: write back

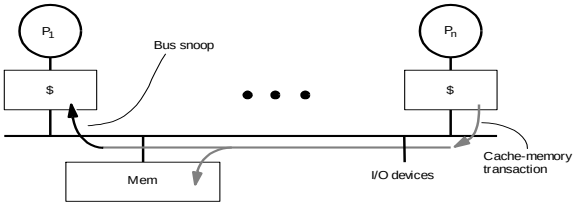


- Οι επεξεργαστές βλέπουν διαφορετική τιμή για την u μετά το βήμα 3

Πρωτόκολλα συνοχής

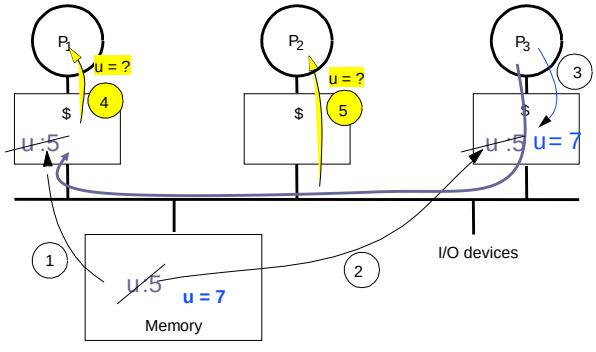
- Οι πολυεπεξεργαστές χρησιμοποιούν ένα πρωτόκολλο υλικού για να διατηρήσουν τη συνοχή των κρυφών μνημών
  - **Κλειδί**: παρακολουθούν την κατάσταση όλων των κοινών χρήσεων των μπλοκ δεδομένων
- 2 τύποι πρωτοκόλλων συνοχής
- Snooping (κατασκοπεία): κάθε κρυφή μνήμη που έχει ένα αντίγραφο των δεδομένων έχει επίσης ένα αντίγραφο της κατάστασης κοινής χρήση του μπλοκ, χωρίς να διατηρείται η κατάσταση κάπου κεντρικά
  - Όλες οι κρυφές μνήμες είναι προσπελάσιμες μέσω κάποιου μέσου μετάδοσης (bus or switch)
  - Όλοι οι ελεγκτές κρυφής μνήμης παρακολουθούν ή κατασκοπεύουν (snoop) το μέσο μετάδοσης για να προσδιορίσουν εάν έχουν κάποιο αντίγραφο του μπλοκ που ζητείται σε μια προσπέλαση στο δίαυλο
- Directory based (βασισμένο σε κατάλογο): η κατάσταση κοινής χρήσης των μπλοκ φυσικής μνήμης διατηρείται μόνο σε μια θέση, στον κατάλογο (directory)

Snooping protocol



- Ο ελεγκτής της κρυφής μνήμης "κατασκοπεύει" όλες τις συναλλαγές στο κοινόχρηστο μέσο μετάδοσης (bus or switch)
  - Σχετική συναλλαγή εάν αφορά μπλοκ που περιέχει
  - Ενεργεί ώστε να εξασφαλίσει συνοχή
    - Ακυρώνει (invalidate) ή ενημερώνει (update) τα υπόλοιπα αντίγραφα
  - Εξαρτάται από την κατάσταση του μπλοκ και το πρωτόκολλο

Παράδειγμα: write-through invalidate



- Η τεχνική write update χρησιμοποιεί περισσότερο εύρος ζώνης του διαύλου
  - Όλοι οι σύγχρονοι SMPs χρησιμοποιούν write invalidate

Υλοποίηση του πρωτοκόλλου

- Διάγραμμα μετάβασης καταστάσεων για τα μπλοκ της κρυφής μνήμης (Cache block state transition diagram)
  - FSM που καθορίζει πως αλλάζει η κατάσταση του μπλοκ
    - Invalid (άκυρο), shared (κοινόχρηστο), exclusive (αποκλειστικό)
- Μηχανισμός κατασκοπείας (snooping):
  - Κατασκοπεύει κάθε διεύθυνση που τοποθετείται στον δίαυλο
  - Εάν ένας επεξεργαστής έχει τροποποιημένο (dirty) αντίγραφο του μπλοκ που ζητείται, το παρέχει ως απόκριση στην αίτηση ανάγνωσης και ματαιώνει την προσπέλαση της μνήμης
    - Η ανάκτηση του μπλοκ από την κρυφή μνήμη είναι πιο πολύπλοκη και παίρνει περισσότερο χρόνο από την ανάκτηση του μπλοκ από τη μνήμη

Κρυφή μνήμη για snooping

- Τα πεδία των μπλοκ της κρυφής μνήμης χρησιμοποιούνται για την υλοποίηση του πρωτοκόλλου
  - Οι ετικέτες (tags) χρησιμοποιούνται για την κατασκοπεία
  - Το έγκυρο (valid) bit χρησιμοποιείται για την υλοποίηση της ακύρωσης (invalidation)
- Αναγνώσεις:
  - Οι αστοχίες ανάγνωσης προωθούνται στον δίαυλο και εξυπηρετούνται είτε από την κοινόχρηστη μνήμη είτε από κάποια κρυφή μνήμη
- Εγγραφές: πρέπει να γνωρίζει εάν υπάρχουν αντίγραφα του μπλοκ σε άλλες κρυφές μνήμες
  - Όχι άλλα αντίγραφα: δεν χρειάζεται να τοποθετήσει την εγγραφή στο δίαυλο
  - Άλλα αντίγραφα: πρέπει να τοποθετήσει την εντολή ακύρωσης στο δίαυλο
  - Τα μπλοκ της κρυφής μνήμης σημειώνονται ως κοινόχρηστα (shared) ή αποκλειστικά (exclusive/modified)
    - Μόνο εγγραφές σε κοινόχρηστα μπλοκ προκαλούν τη μετάδοση εντολής ακύρωσης
      - Μετά από αυτό, το μπλοκ σημειώνεται ως αποκλειστικό

Απόδοση των SMPs

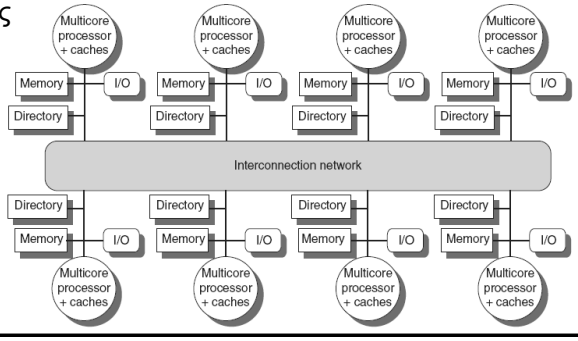
- Η απόδοση της κρυφής μνήμης είναι συνδυασμός των
  - ρυθμού αστοχίας της κρυφής μνήμης του μονού επεξεργαστή
  - επιβάρυνση του διαύλου λόγω επικοινωνίας
  - αποτελέσματα των ακυρώσεων (invalidations) και των αστοχιών κρυφής μνήμης που προκαλούνται από αυτές
  - 4ο C: coherence miss
    - μαζί με τα Compulsory, Capacity, Conflict misses

Πρωτόκολλα καταλόγου

- Τα πρωτόκολλα κατασκοπείας (snooping) απαιτούν επικοινωνία με όλες τις κρυφές μνήμες σε κάθε cache miss (read or write) και write σε κοινόχρηστα δεδομένα (invalidation)
  - Αύξηση του εύρους ζώνης της μνήμης με κατανομή της μνήμης (distributed shared memory architectures - DSM)
  - Δεν αρκεί εάν δεν περιορίσουμε την ανάγκη για broadcast σε όλες τις κρυφές μνήμης
- Εναλλακτική λύση: πρωτόκολλα καταλόγου (directory protocols)
  - Η πληροφορία για την κατάσταση των μπλοκ συγκεντρώνεται σε έναν κατάλογο
- Ο κατάλογος (directory) διατηρεί πληροφορίες για κάθε μπλοκ
  - Ποιες κρυφές μνήμες έχουν το κάθε μπλοκ
  - Κατάσταση τροποποίησης (dirty) για κάθε μπλοκ

Κατάλογος

- Ο κατάλογος αποθηκεύεται σε κοινόχρηστη L2 ή L3 cache
  - Συσχετίζει ένα διάνυσμα (καταχώρηση στον κατάλογο) για κάθε μπλοκ στην L2 ή L3
    - Μέγεθος διανύσματος σε bit = # πυρήνων
- Υλοποιείται με έναν κατανεμημένο τρόπο
  - Κατανέμεται μαζί με τη μνήμη σε DSM αρχιτεκτονικές ή αποθηκεύεται στις L2 caches σε μια SMP αρχιτεκτονικές με σειρές μνήμης



Πληροφορίες καταλόγου

- Δύο βασικές περιπτώσεις πρέπει να χειριστεί το πρωτόκολλο:
  - read miss & write to a shared block
- Ο κατάλογος διατηρεί μια κατάσταση για κάθε μπλοκ:
  - Shared (κοινόχρηστο)
    - Ένας η περισσότεροι κόμβοι έχουν το μπλοκ στην κρυφή μνήμη, και η τιμή στην μνήμη είναι ενημερωμένη
    - Σύνολο από ID κόμβων
  - Uncached (εκτός κρυφής μνήμης)
    - Κανένας επεξεργαστής δεν έχει αντίγραφο του μπλοκ στην κρυφή μνήμη
  - Modified (τροποποιημένο)
    - Ακριβώς ένας κόμβος έχει ένα αντίγραφο του μπλοκ στην κρυφή μνήμη, και η τιμή στην μνήμη δεν είναι ενημερωμένη
    - ID του κόμβου-κατόχου
- Ο κατάλογος διατηρεί τις καταστάσεις των μπλοκ και στέλνει μηνύματα ακύρωσης (invalidation messages)

GPU (graphics processing unit)

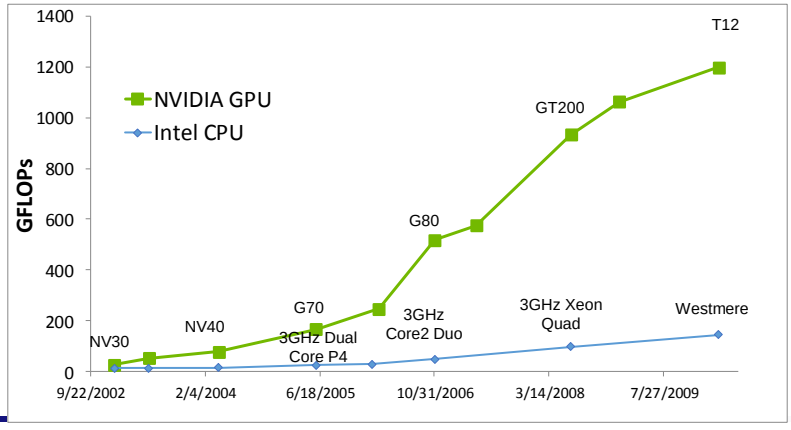
- GPU (graphics processing unit)
  - Πλέον υπάρχει σε κάθε φορητό, προσωπικό υπολογιστή και σταθμό εργασίας
- Μια GPU διαθέτει μια ενοποιημένη αρχιτεκτονική για γραφικά και υπολογιστική που μπορεί να χρησιμοποιηθεί ως
  - Προγραμματιζόμενος επεξεργαστής γραφικών
  - Επεκτάσιμη πλατφόρμα παράλληλης υπολογιστικής
- Ετερογενή συστήματα CPU-GPU
  - Προσωπικοί υπολογιστές, παιχνιδομηχανές

Ιστορία των GPU

- Αρχικά ένα μεγάλο μέρος του υπολογιστικού χρόνου των επεξεργαστών χρησιμοποιούνταν για επεξεργασία γραφικών
  - Πρώιμες κάρτες βίντεο:
    - Προσωρινές μνήμες για την αποθήκευση των καρέ και δυνατότητα για έξοδο βίντεο
- Βελτίωση της επεξεργασίας γραφικών
  - Νόμος του Moore ⇒ χαμηλότερο κόστος, υψηλότερη πυκνότητα
    - Προσθήκη λειτουργιών στα τσιπ ελεγκτών γραφικών (VGA controller)
    - Υπολογιστές υψηλών επιδόσεων (π.χ. SGI) με ακριβές κάρτες γραφικών
  - Κινητήρια δύναμη: βιομηχανία παιχνιδιών για υπολογιστές και κονσόλες παιχνιδιών (π.χ. Sony Playstation)
    - Ταχύτερη βελτίωση της υπολογιστικής γραφικών από την γενικού τύπου υπολογιστική
- Μονάδες Επεξεργασίας Γραφικών (Graphics Processing Units - GPUs)
  - Ενσωματώνουν σχεδόν κάθε λεπτομέρεια της διοχέτευσης γραφικών
  - Διαφορετικό όνομα από αυτό του VGA controller
    - Είναι πλέον ένας επεξεργαστής

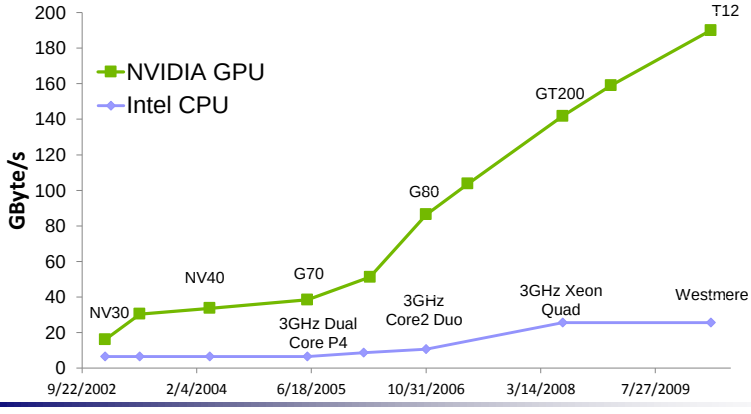
Γιατί μαζική υπολογιστική παραλληλία (massively parallel computing)?

- Σύγκριση GPU, CPI
  - TFLOPs vs. 100 GFLOPs



Γιατί μαζική υπολογιστική παραλληλία (massively parallel computing)?

- Σύγκριση GPU, CPI
  - Εύρος ζώνης



GPUs vs. CPUs

- Είναι επιταχυντές (accelerators) που συμπληρώνουν τη CPU
  - Δεν χρειάζεται να εκτελούν όλες τις λειτουργίες μιας CPU
  - Συνδυασμός CPU-GPU: παράδειγμα ετερογενούς πολυεπεξεργαστικής αρχιτεκτονικής
- Βασίζονται σε υψηλή παραλληλία δεδομένων για να πετύχουν υψηλή απόδοση
  - Υλοποιούν πολλούς επεξεργαστές και πολλά ταυτόχρονα νήματα
- Χρησιμοποιούν εναλλαγή νημάτων για να κρύψουν το λανθάνοντα χρόνο της μνήμης
  - Δεν βασίζονται σε πολυεπίπεδες κρυφές μνήμες
  - Είναι έντονα πολυνηματικοί επεξεργαστές
- Η μνήμη τους είναι προσανατολισμένη στο εύρος ζώνης και όχι στο χρόνο προσπέλασης

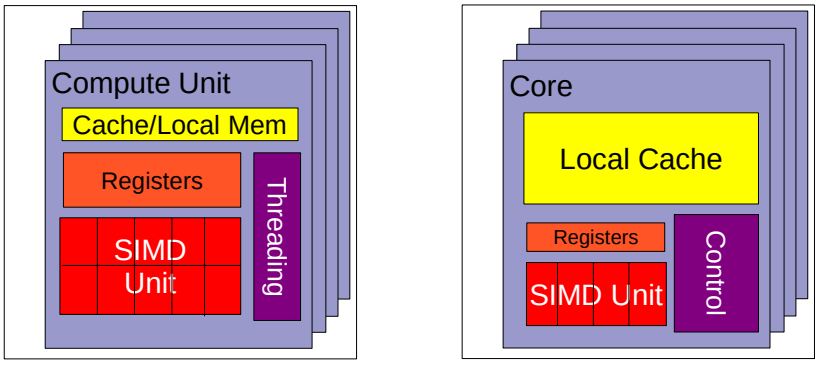
Υπολογιστική με GPU

- GPU computing
  - Χρήση της GPU μέσω μιας γλώσσας προγραμματισμού
    - Μπορούν να γράψουμε προγράμματα απευθείας για τις GPU χωρίς τη χρήση των παραδοσιακών API
- Compute Unified Device Architecture (CUDA)
  - Μοντέλο παράλληλου προγραμματισμού και πλατφόρμα λογισμικού για τις GPU
  - Βασίζεται σε C, C++
  - Δυνατότητα προγραμματισμού και πολυπύρηνων CPU
    - Περιβάλλον προγραμματισμού για το συνολικό ετερογενές σύστημα CPU-GPU
- Όσο αυξάνεται η ευκολία προγραμματισμού των GPU, τόσο θα αυξάνεται και η χρήση τους στην παράλληλη υπολογιστική

CPU και GPU έχουν διαφορετική σχεδιαστική φιλοσοφία

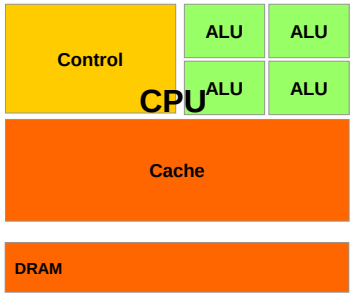
GPU
Throughput Oriented Cores

CPU
Latency Oriented Cores



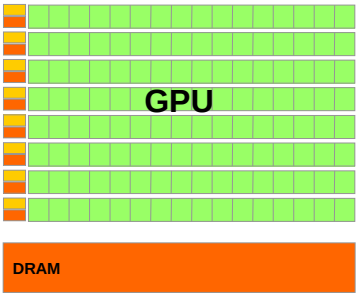
CPUs: Έμφαση στην καθυστέρηση (latency)

- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency
- Powerful ALU
  - Reduced operation latency



GPUs: έμφαση στην ικανότητα διεκπεραίωσης (throughput)

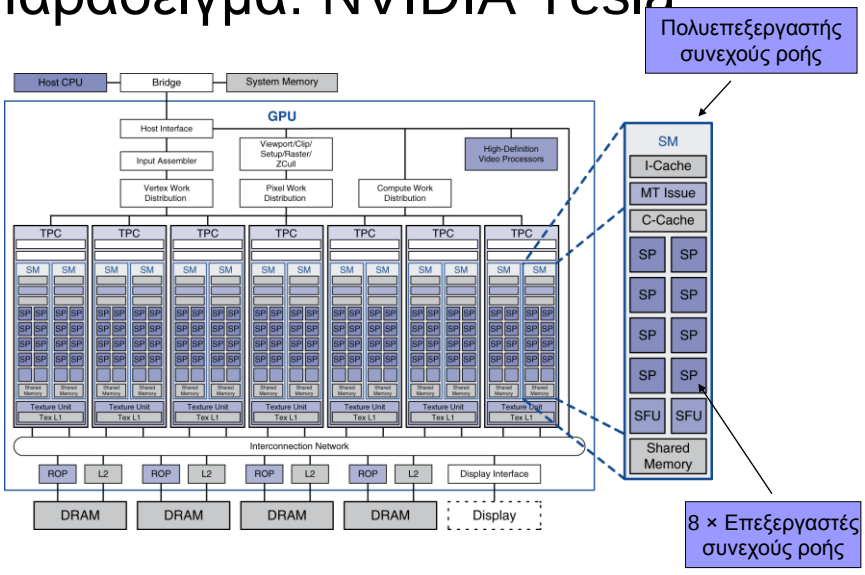
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies



Ετερογενής υπολογιστική: χρήση CPU και GPU

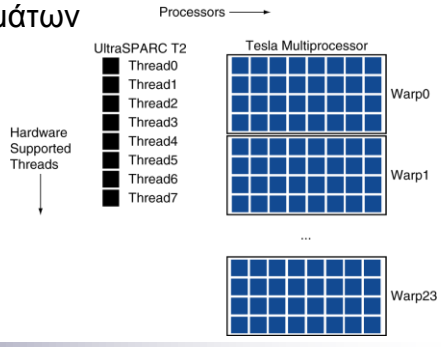
- CPUs for sequential parts where latency matters
  - CPUs can be 10+X faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
  - GPUs can be 10+X faster than CPUs for parallel code

Παράδειγμα: NVIDIA Tesla

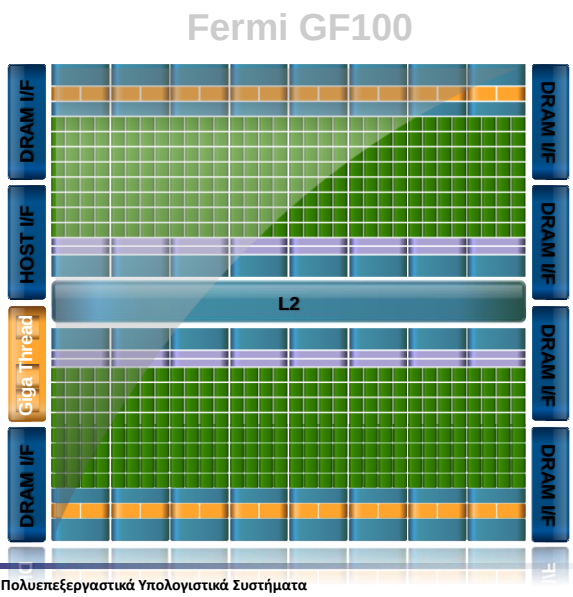


Παράδειγμα: NVIDIA Tesla

- Επεξεργαστές συνεχούς ροής (Streaming Processors – SP)
  - Μονάδες FP απλής ακρίβειας και ακέραιες μονάδες
  - Κάθε SP υποστηρίζει λεπτή πολυνημάτωση
- Στημόνι (warp): μπλοκ 32 νημάτων
  - Εκτελούνται παράλληλα
    - 8 SPs × 4 κύκλους ρολογιού
  - Υποστήριξη υλικού για 24 στημόνια
    - Καταχωρητές, PCs, …

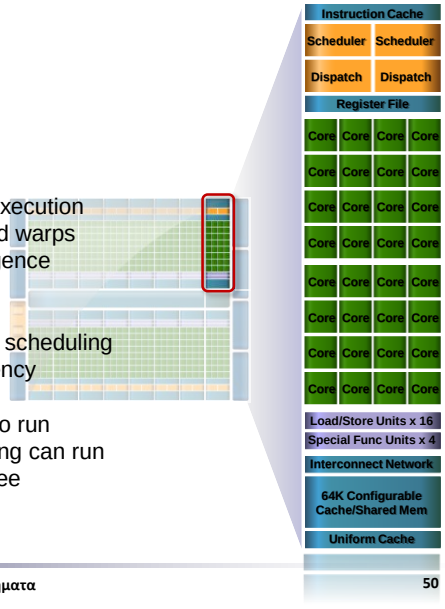


Παράδειγμα: NVIDIA Fermi



Παράδειγμα: NVIDIA Fermi

- 32 CUDA Cores per SM (512 total)
- 64KB of fast, on-chip RAM
  - Software or hardware-managed
  - Shared amongst CUDA cores
  - Enables thread communication
- SIMT (Single Instruction Multiple Thread) execution
  - threads run in groups of 32 called warps
  - HW automatically handles divergence
- Hardware multithreading
  - HW resource allocation & thread scheduling
  - HW relies on threads to hide latency
- Threads have all resources needed to run
  - any warp not waiting for something can run
  - context switching is (basically) free

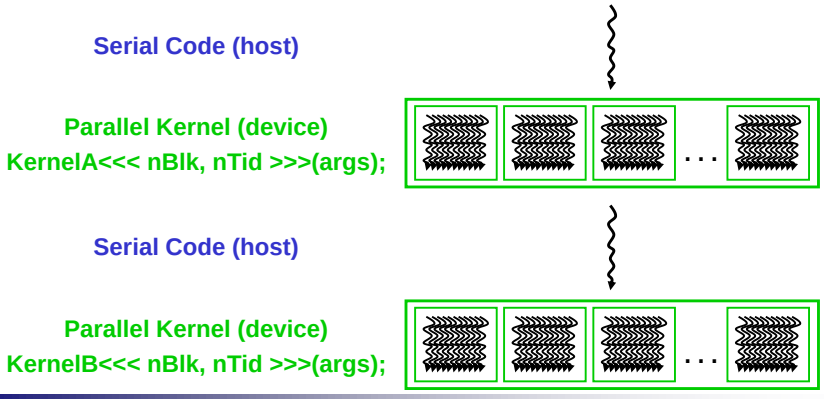


CUDA

- Σύντομη αναφορά στην CUDA
- Παράδειγμα: πρόσθεση διανυσμάτων

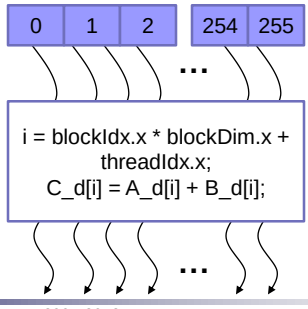
CUDA C – Μοντέλο εκτέλεσης

- Integrated host+device app C program
  - Serial or modestly parallel parts in **host** C code
  - Highly parallel parts in **device** SPMD kernel C code



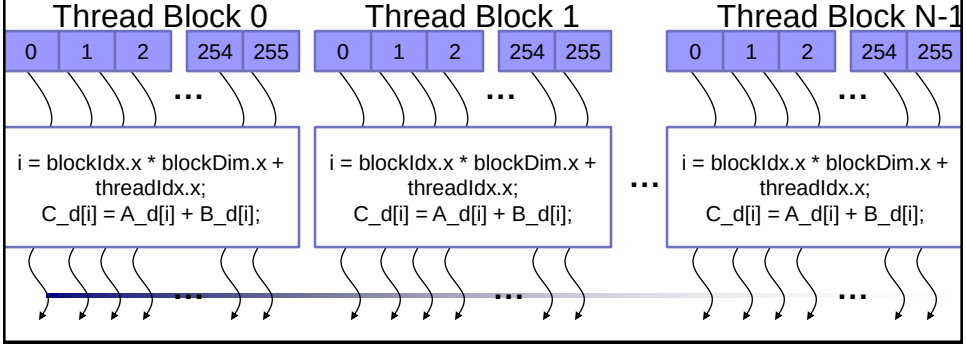
Πλέγμα παράλληλων νημάτων

- A CUDA kernel is executed by a grid (array) of threads
  - All threads in a grid run the same kernel code (SPMD)
  - Each thread has an index that it uses to compute memory addresses and make control decisions



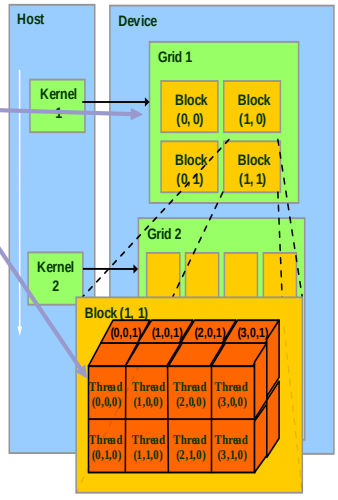
Μπλοκ νημάτων: Επεκτασιμότητα

- Divide thread array into multiple blocks
  - Threads within a block cooperate via **shared memory, atomic operations** and **barrier synchronization**
  - Threads in different blocks cannot cooperate

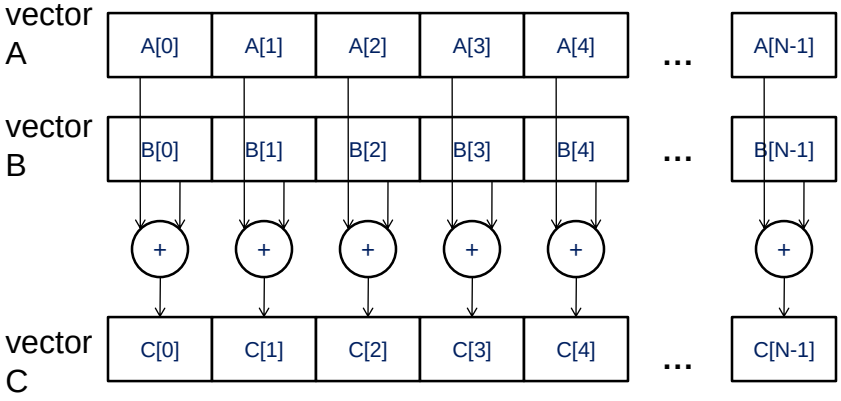


blockIdx και threadIdx

- Each thread uses indices to decide what data to work on
  - blockIdx: 1D, 2D, or 3D (CUDA 4.0)
  - threadIdx: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data



Πρόσθεση διανυσμάτων– εννοιολογική άποψη

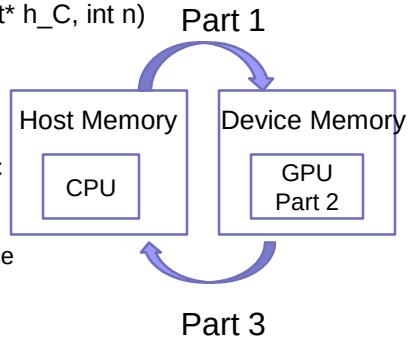


Πρόσθεση διανυσμάτων – Παραδοσιακός κώδικας C

```
// Compute vector sum C = A+B
void vecAdd(float* A, float* B, float* C, int n)
{
  for (i = 0; i < n; i++)
    C[i] = A[i] + B[i];
}

int main()
{
    // Memory allocation for A_h, B_h, and C_h
    // I/O to read A_h and B_h, N elements
    …
    vecAdd(A_h, B_h, C_h, N);
}
```

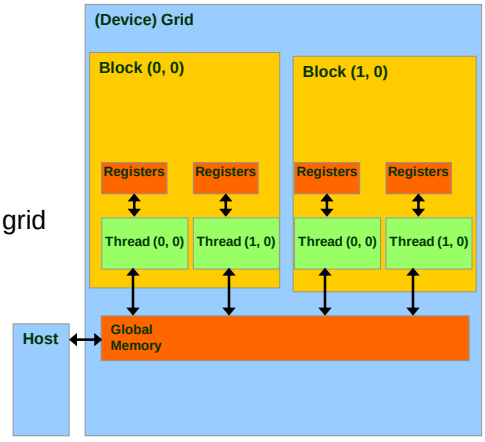

Ετερογενής υπολογιστική: vecAdd CUDA Host Code

```
void vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
  int size = n* sizeof(float);
  float* d_A, d_B, d_C;
  …
1. // Allocate device memory for A, B, and C
   // copy A and B to device memory

2. // Kernel launch code – to have the device
   // to perform the actual vector addition

3. // copy C from the device memory
   // Free device vectors
}
```



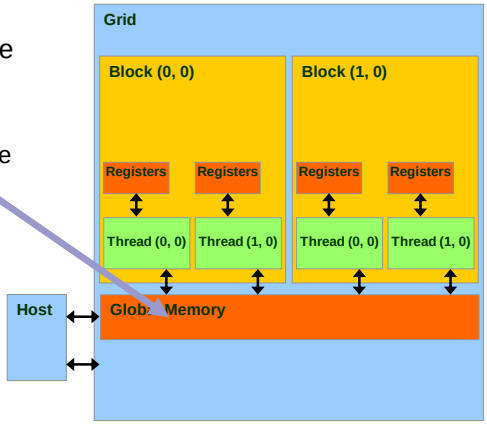
Μερική άποψη των μνημών στην CUDA

- Device code can:
  - R/W per-thread registers
  - R/W all-shared global memory
- Host code can
  - Transfer data to/from per grid global memory



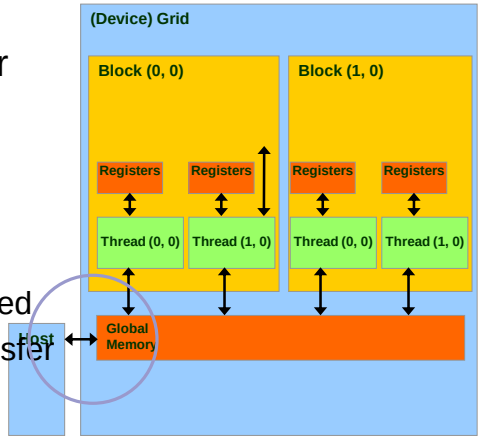
Συναρτήσεις API για την διαχείριση της μνήμης της συσκευής

- cudaMalloc()
  - Allocates object in the device global memory
  - Two parameters
    - **Address of a pointer** to the allocated object
    - **Size of** allocated object in terms of bytes
- cudaFree()
  - Frees object from device global memory
    - **Pointer** to freed object



Συναρτήσεις API για μεταφορά δεδομένων μεταξύ Host & Device

- cudaMemcpy()
  - memory data transfer
  - Requires four parameters
    - Pointer to destination
    - Pointer to source
    - Number of bytes copied
    - Type/Direction of transfer




```
void vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
  int size = n * sizeof(float);
   float* d_A, d_B, d_C;

1. // Transfer A and B to device memory
   cudaMalloc((void **) &d_A, size);
   cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
   cudaMalloc((void **) &B_d, size);
   cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Allocate device memory for
    cudaMalloc((void **) &d_C, size);

2.  // Kernel invocation code – to be shown later
   ...
3.   // Transfer C from device to host
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
     // Free device memory for A, B, C
    cudaFree(d_A); cudaFree(d_B); cudaFree (d_C);
}
```

Παράδειγμα: Vector Addition Kernel

Device Code

```
// Compute vector sum C = A+B
// Each thread performs one pair-wise addition
__global__
void vecAddKernel(float* A, float* B, float* C, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if(i<n) C[i] = A[i] + B[i];
}

int vectAdd(float* h_A, float* h_B, float* h_C, int n)
{
        // d_A, d_B, d_C allocations and copies omitted
    // Run ceil(n/256.0) blocks of 256 threads each
    vecAddKernel<<<ceil(n/256.0), 256>>>(d_A, d_B, d_C, n);
}
```


Παράδειγμα: Vector Addition Kernel

```
// Compute vector sum C = A+B
// Each thread performs one pair-wise addition
__global__
void vecAddkernel(float* A_d, float* B_d, float* C_d, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if(i<n) C_d[i] = A_d[i] + B_d[i];
}
```

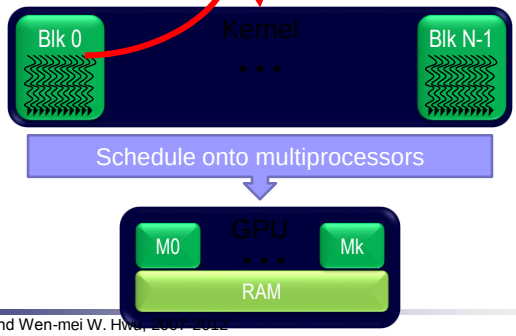
Host Code

```
int vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
 // d_A, d_B, d_C allocations and copies omitted
 // Run ceil(n/256.0) blocks of 256 threads each
  vecAddKernnel<<<ceil(n/256.0),256>>>(d_A, d_B, d_C, n);
}
```

Η εκτέλεση του Kernel

```
__host__
Void vecAdd()
{
  dim3 DimGrid = (ceil(n/256.0),1,1);
  dim3 DimBlock = (256,1,1);

vecAddKernel<<<DimGrid,DimBlock>>>(A_
d,B_d,C_d,n);
}
```

```
__global__
void vecAddKernel(float *A_d,
  float *B_d, float *C_d, int n)
{
  int i = blockIdx.x * blockDim.x
    + threadIdx.x;

  if( i<n ) C_d[i] = A_d[i]+B_d[i];
}
```



Schedule onto multiprocessors