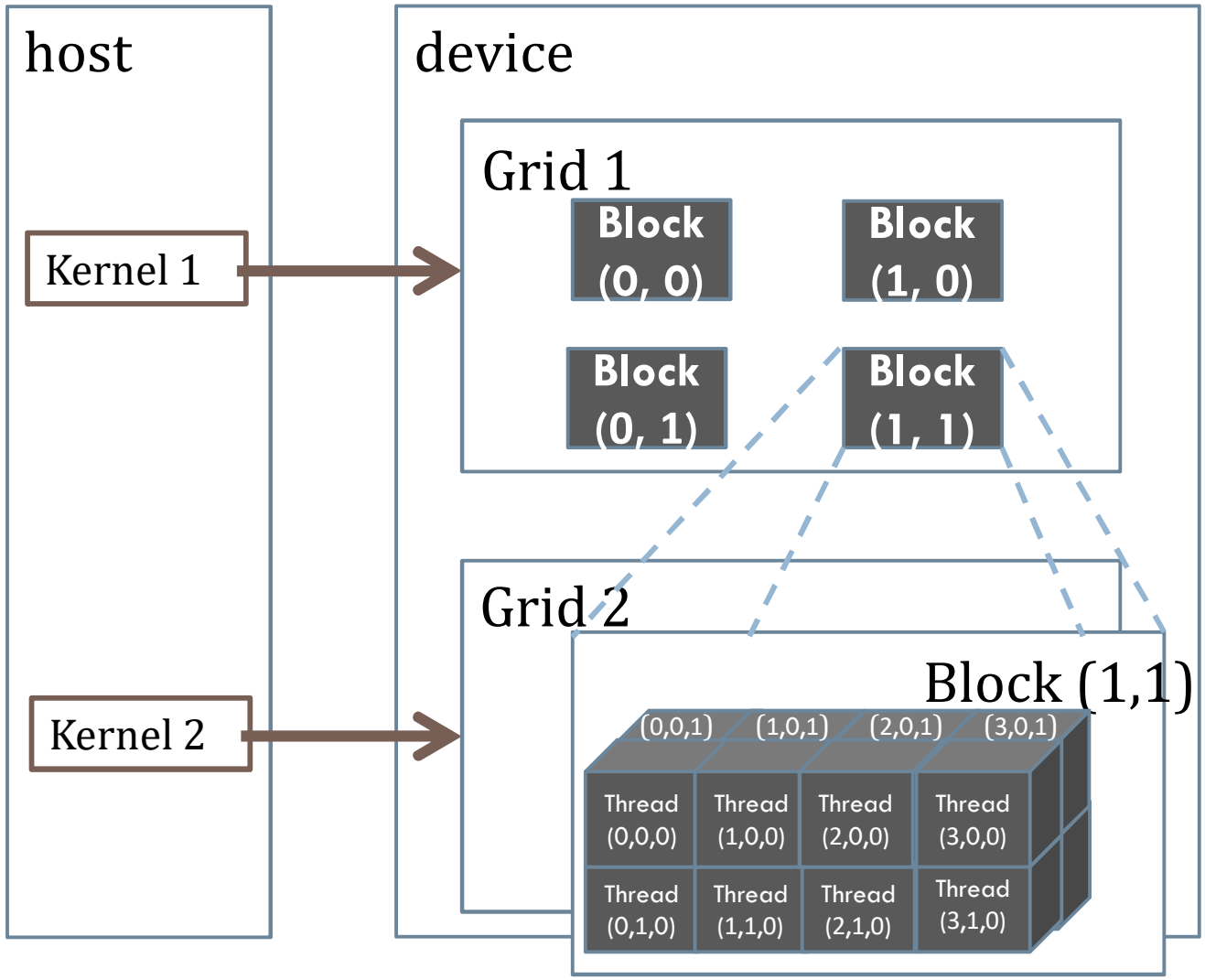
ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

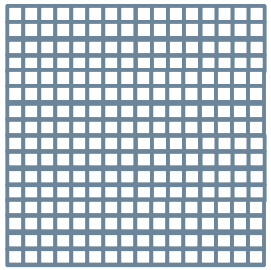
Μάθημα #8

CUDA (Μοντέλο Παραλληλισμού)

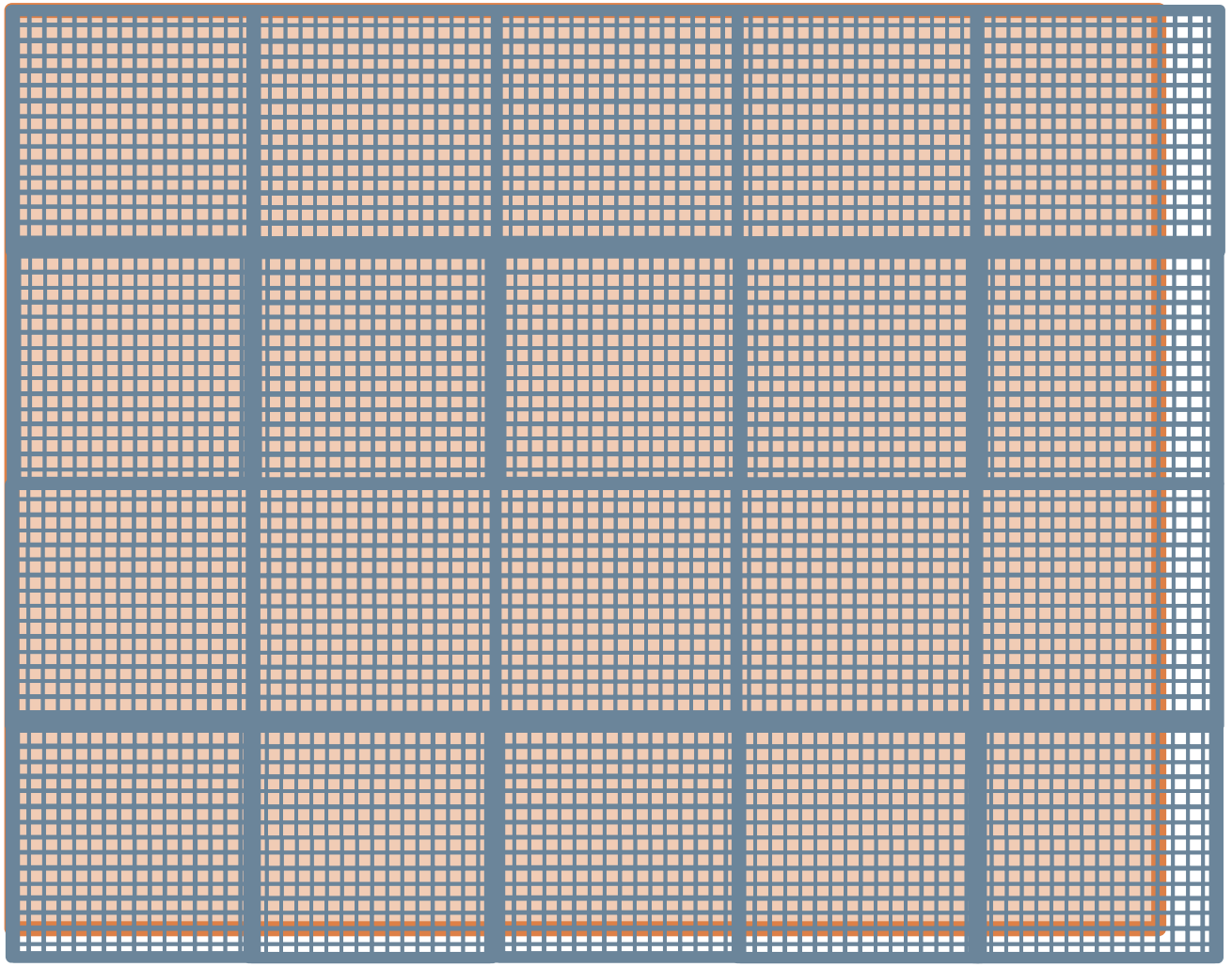
Παράδειγμα πολυδιάστατου πίνακα νημάτων



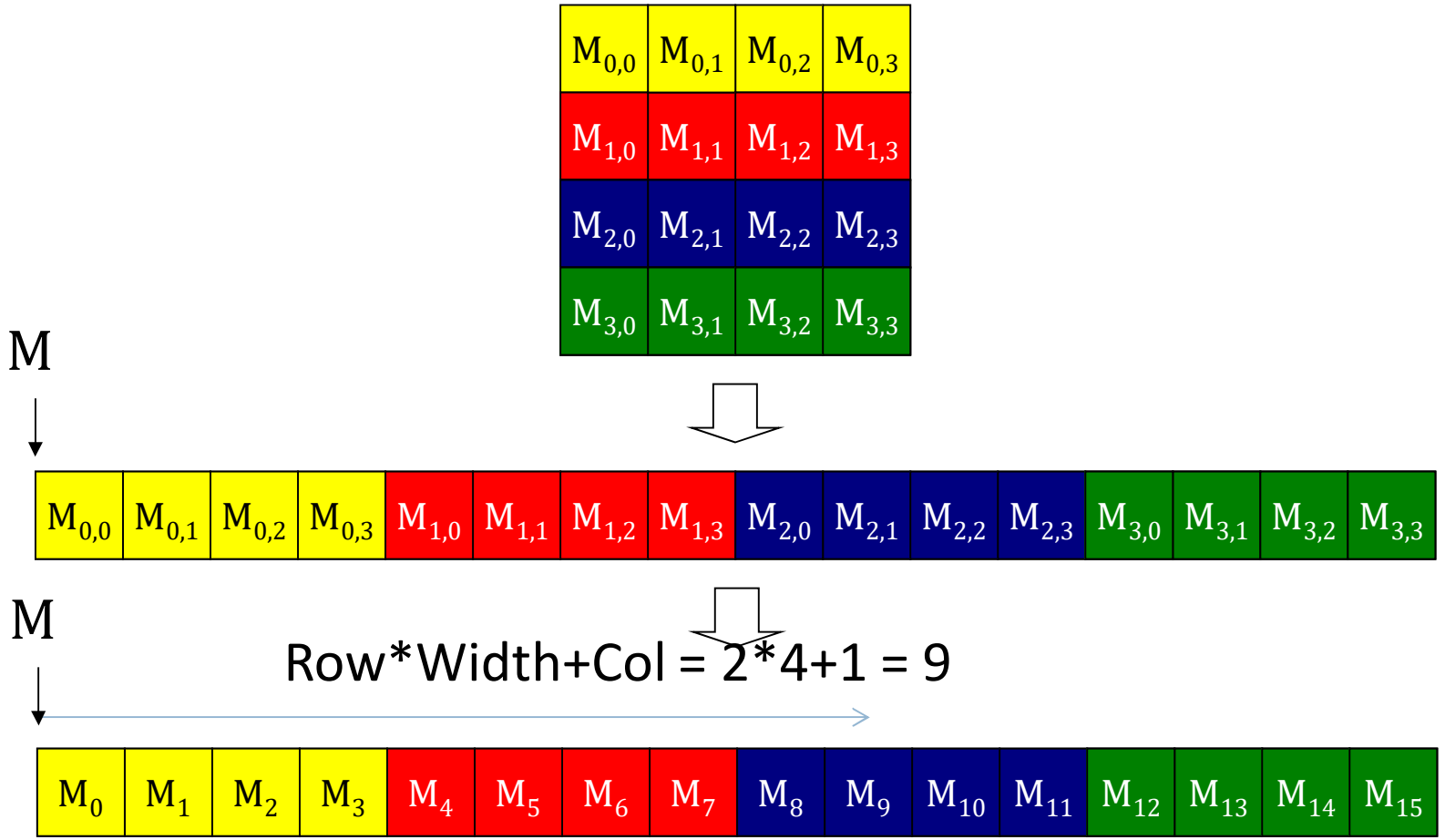
Επεξεργασία εικόνας με χρήση δισδιάστατου πίνακα νημάτων



16×16 blocks

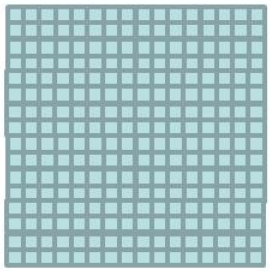


Απεικόνιση κατά γραμμές στην C/C++

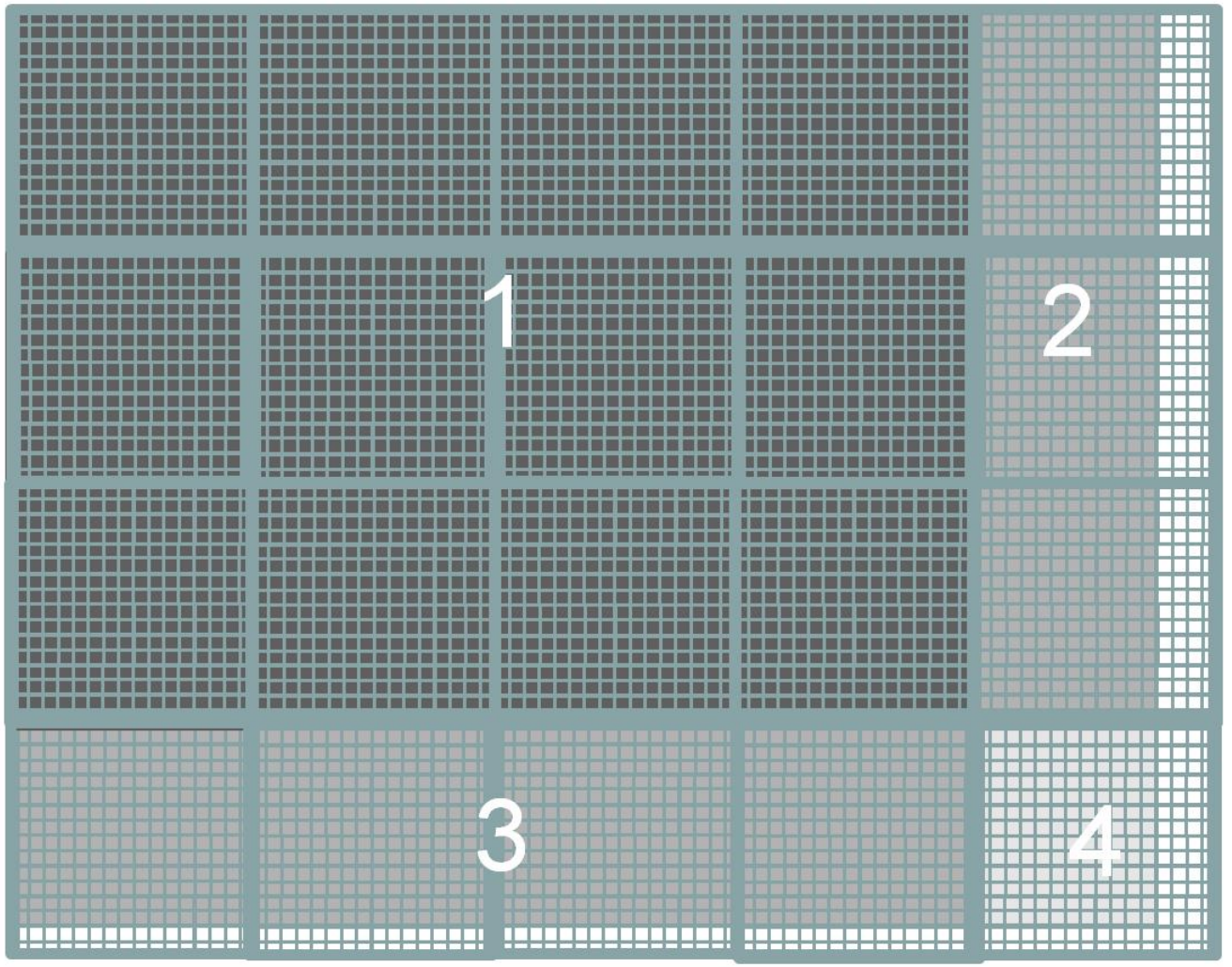


Κώδικας PictureKernel

```
__global__ void PictureKernel(float* d_Pin, float* d_Pout, int n,int m)
{
  // Calculate the row # of the d_Pin and d_Pout element to process
  int Row = blockIdx.y*blockDim.y + threadIdx.y;

  // Calculate the column # of the d_Pin and d_Pout element to process
  int Col = blockIdx.x*blockDim.x + threadIdx.x;

  // each thread computes one element of d_Pout if in range
  if ((Row < m) && (Col < n)) {
    d_Pout[Row*n+Col] = 2*d_Pin[Row*n+Col];
  }
}
```



16x16 block



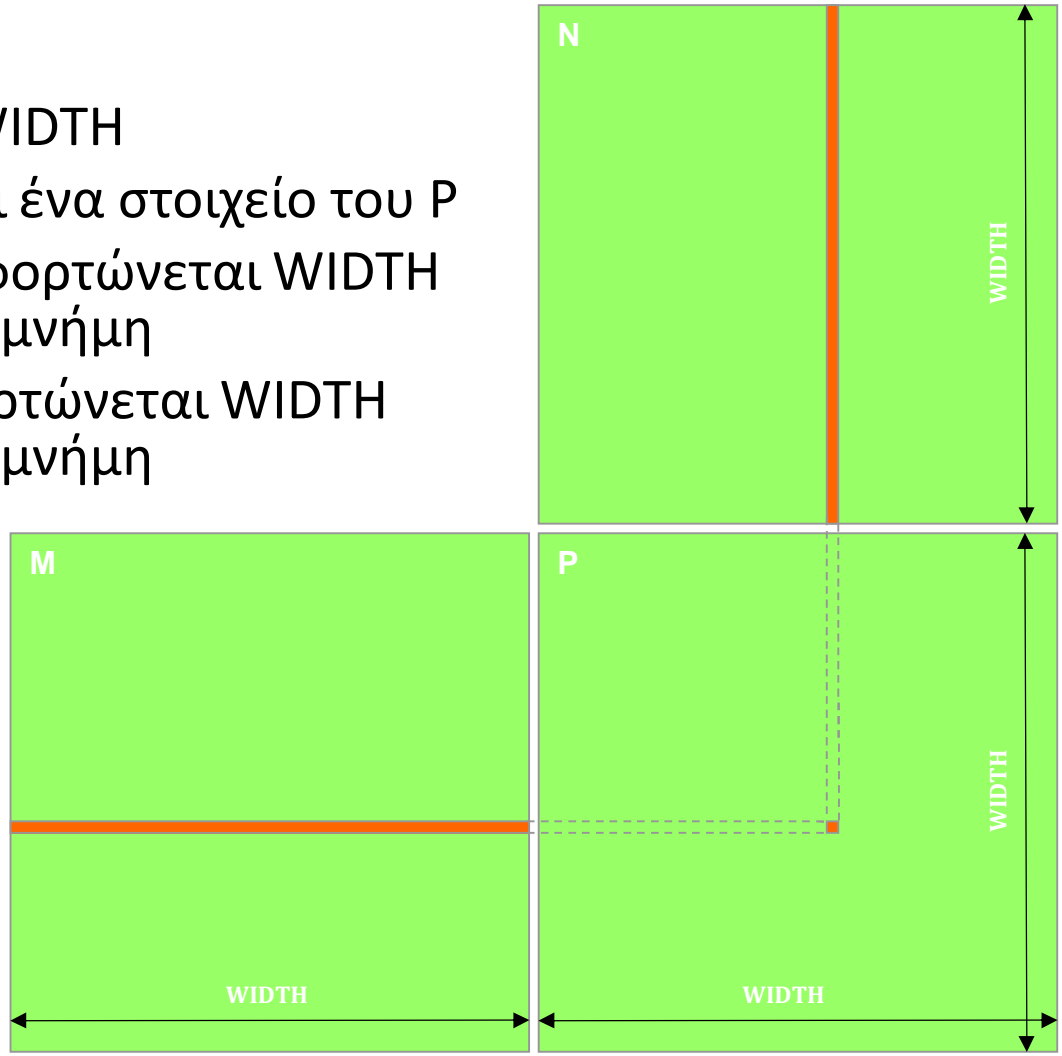
Κάλυψη εικόνας μεγέθους 76×62 με block μεγέθους 16×16

Ένα απλό παράδειγμα Πολλαπλασιασμός μητρώων

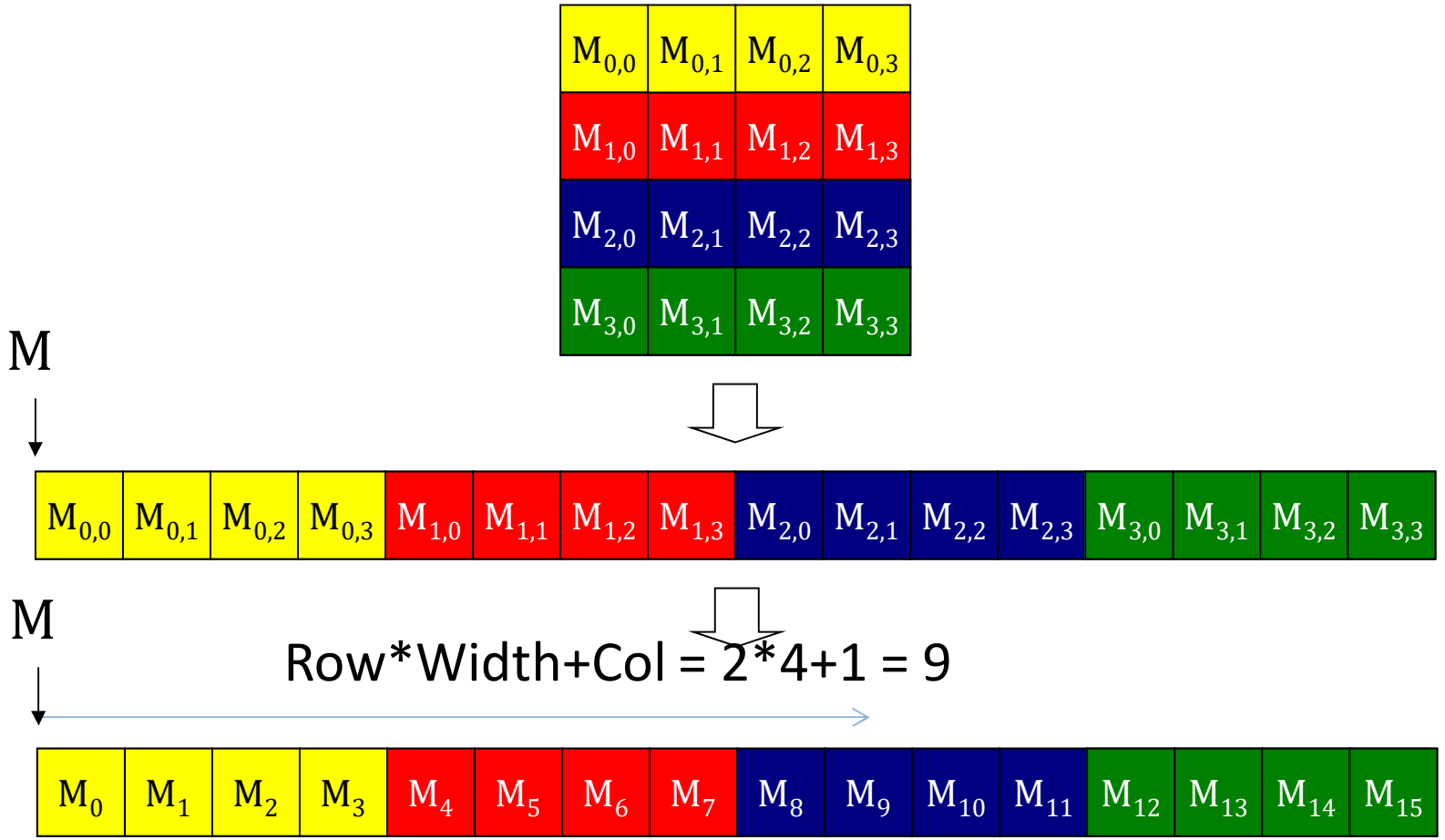
- Δείχνει τις βασικές δυνατότητες διαχείρισης μνήμης και νημάτων σε προγράμματα CUDA
  - Χρήση δεικτών (index) νημάτων
  - Διάταξη δεδομένων στην μνήμη
  - Χρήση καταχωρητών
- Για λόγους απλότητας
  - Υποθέτουμε τετραγωνικά μητρώα
  - Αφήνουμε την χρήση κοινής μνήμης για αργότερα

Πολλαπλασιασμός τετραγωνικών πινάκων

- P = M * N
  - Μεγέθους WIDTH x WIDTH
  - Κάθε νήμα υπολογίζει ένα στοιχείο του P
  - Κάθε γραμμή του M φορτώνεται WIDTH φορές από την global μνήμη
  - Κάθε στήλη του N φορτώνεται WIDTH φορές από την global μνήμη

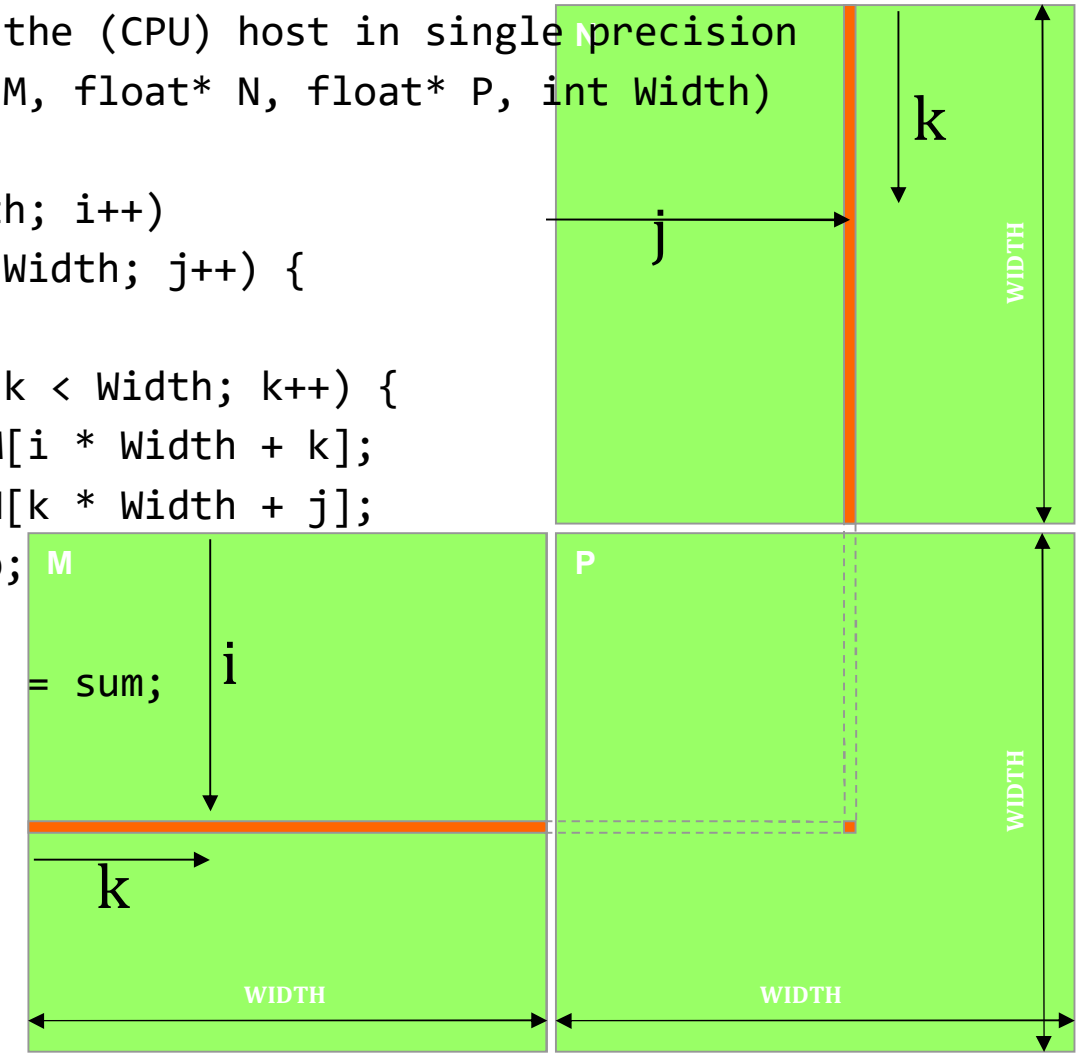


Απεικόνιση κατά γραμμές στην C/C++



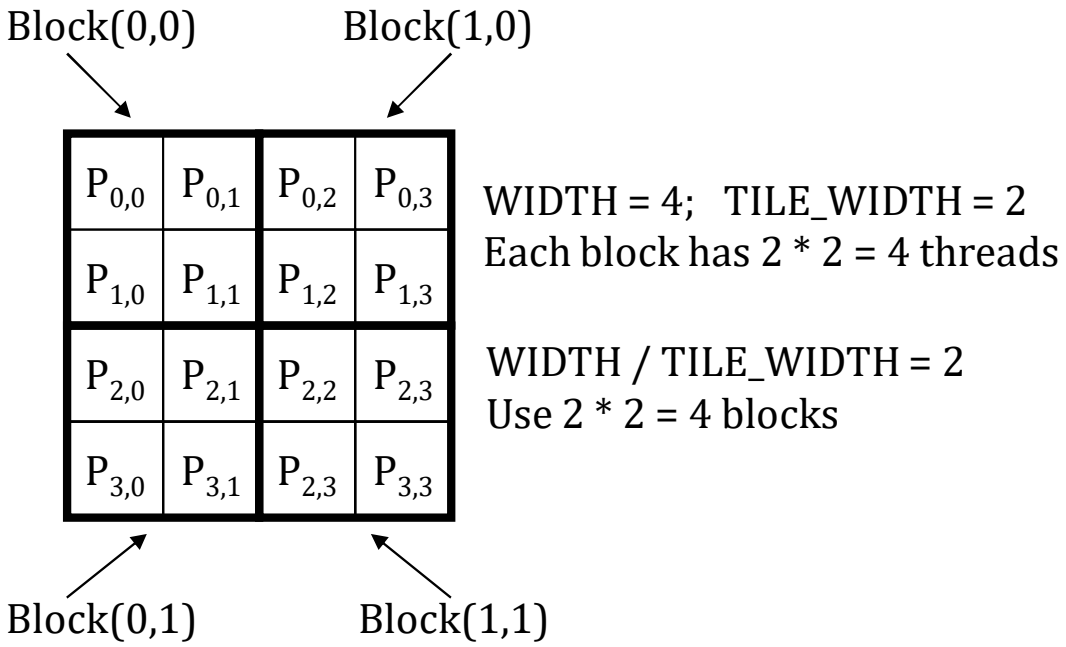
Πολλαπλασιασμός πινάκων Κώδικας για CPU

```
// Matrix multiplication on the (CPU) host in single precision
void MatrixMulOnHost(float* M, float* N, float* P, int Width)
{
    for (int i = 0; i < Width; i++)
        for (int j = 0; j < Width; j++) {
            double sum = 0;
            for (int k = 0; k < Width; k++) {
                double a = M[i * Width + k];
                double b = N[k * Width + j];
                sum += a * b;
            }
            P[i * Width + j] = sum;
        }
}
```



Συνάρτηση πυρήνα για μικρό παράδειγμα

- Κάθε block νημάτων θα αναλάβει τον υπολογισμό ενός υπομητρώου μεγέθους (TILE_WIDTH)^2 του τελικού μητρώου
  - Κάθε block έχει (TILE_WIDTH)^2 νήματα
- Δημιουργία δισδιάστατου πλέγματος block μεγέθους (WIDTH/TILE_WIDTH)^2



Ένα λίγο μεγαλύτερο παράδειγμα

$P_{0,0}$	$P_{0,1}$	$P_{0,2}$	$P_{0,3}$	$P_{0,4}$	$P_{0,5}$	$P_{0,6}$	$P_{0,7}$
$P_{1,0}$	$P_{1,1}$	$P_{1,2}$	$P_{1,3}$	$P_{1,4}$	$P_{1,5}$	$P_{1,6}$	$P_{1,7}$
$P_{2,0}$	$P_{2,1}$	$P_{2,2}$	$P_{2,3}$	$P_{2,4}$	$P_{2,5}$	$P_{2,6}$	$P_{2,7}$
$P_{3,0}$	$P_{3,1}$	$P_{3,2}$	$P_{3,3}$	$P_{3,4}$	$P_{3,5}$	$P_{3,6}$	$P_{3,7}$
$P_{4,0}$	$P_{4,1}$	$P_{4,2}$	$P_{4,3}$	$P_{4,4}$	$P_{4,5}$	$P_{4,6}$	$P_{4,7}$
$P_{5,0}$	$P_{5,1}$	$P_{5,2}$	$P_{5,3}$	$P_{5,4}$	$P_{5,5}$	$P_{5,6}$	$P_{5,7}$
$P_{6,0}$	$P_{6,1}$	$P_{6,2}$	$P_{6,3}$	$P_{6,4}$	$P_{6,5}$	$P_{6,6}$	$P_{6,7}$
$P_{7,0}$	$P_{7,1}$	$P_{7,2}$	$P_{7,3}$	$P_{7,4}$	$P_{7,5}$	$P_{7,6}$	$P_{7,7}$

WIDTH = 8; TILE_WIDTH = 2
Each block has 2 * 2 = 4 threads

WIDTH / TILE_WIDTH = 4
Use 4 * 4 = 16 blocks

Ένα λίγο μεγαλύτερο παράδειγμα

$P_{0,0}$	$P_{0,1}$	$P_{0,2}$	$P_{0,3}$	$P_{0,4}$	$P_{0,5}$	$P_{0,6}$	$P_{0,7}$
$P_{1,0}$	$P_{1,1}$	$P_{1,2}$	$P_{1,3}$	$P_{1,4}$	$P_{1,5}$	$P_{1,6}$	$P_{1,7}$
$P_{2,0}$	$P_{2,1}$	$P_{2,2}$	$P_{2,3}$	$P_{2,4}$	$P_{2,5}$	$P_{2,6}$	$P_{2,7}$
$P_{3,0}$	$P_{3,1}$	$P_{3,2}$	$P_{3,3}$	$P_{3,4}$	$P_{3,5}$	$P_{3,6}$	$P_{3,7}$
$P_{4,0}$	$P_{4,1}$	$P_{4,2}$	$P_{4,3}$	$P_{4,4}$	$P_{4,5}$	$P_{4,6}$	$P_{4,7}$
$P_{5,0}$	$P_{5,1}$	$P_{5,2}$	$P_{5,3}$	$P_{5,4}$	$P_{5,5}$	$P_{5,6}$	$P_{5,7}$
$P_{6,0}$	$P_{6,1}$	$P_{6,2}$	$P_{6,3}$	$P_{6,4}$	$P_{6,5}$	$P_{6,6}$	$P_{6,7}$
$P_{7,0}$	$P_{7,1}$	$P_{7,2}$	$P_{7,3}$	$P_{7,4}$	$P_{7,5}$	$P_{7,6}$	$P_{7,7}$

WIDTH = 8; TILE_WIDTH = 4
Each block has 4 * 4 =16 threads

WIDTH / TILE_WIDTH = 2
Use 2 * 2 = 4 blocks

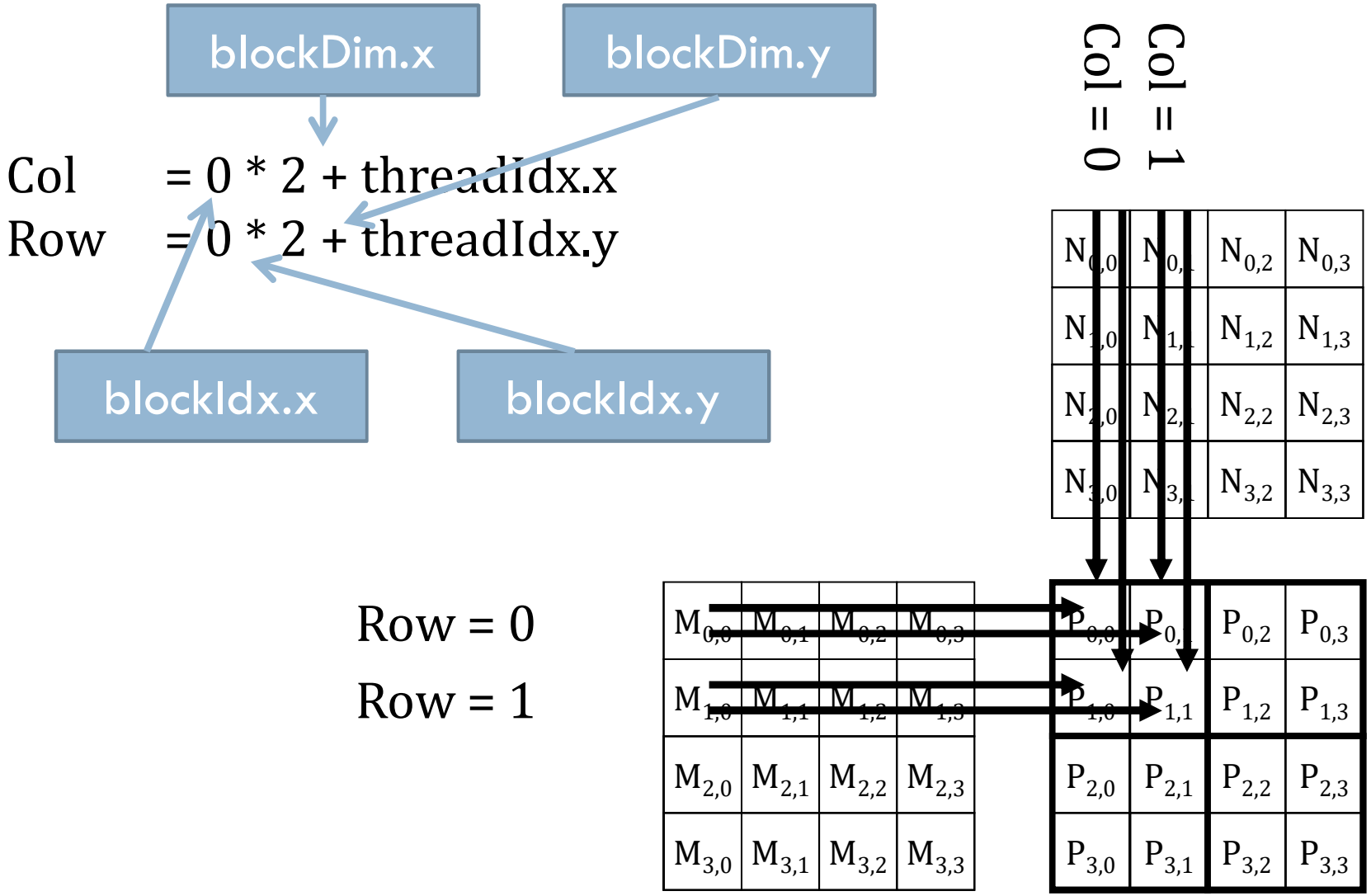
Κλήση συνάρτησης πυρήνα από τον host

```
// Setup the execution configuration
// TILE_WIDTH is a #define constant
dim3 dimGrid(Width / TILE_WIDTH, Width / TILE_WIDTH, 1);
dim3 dimBlock(TILE_WIDTH, TILE_WIDTH, 1);

// Launch the device computation threads!
MatrixMulKernel<<<dimGrid, dimBlock>>>(Md, Nd, Pd, Width);
```

Συνάρτηση πυρήνα

```
// Matrix multiplication kernel – per thread code
__global__ void MatrixMulKernel(float* d_M, float* d_N,
                                float* d_P, int Width)
{

  // Pvalue is used to store the element of the matrix
  // that is computed by the thread
  float Pvalue = 0;
```

Επεξεργασία block (0, 0) για TILE_WIDTH = 2

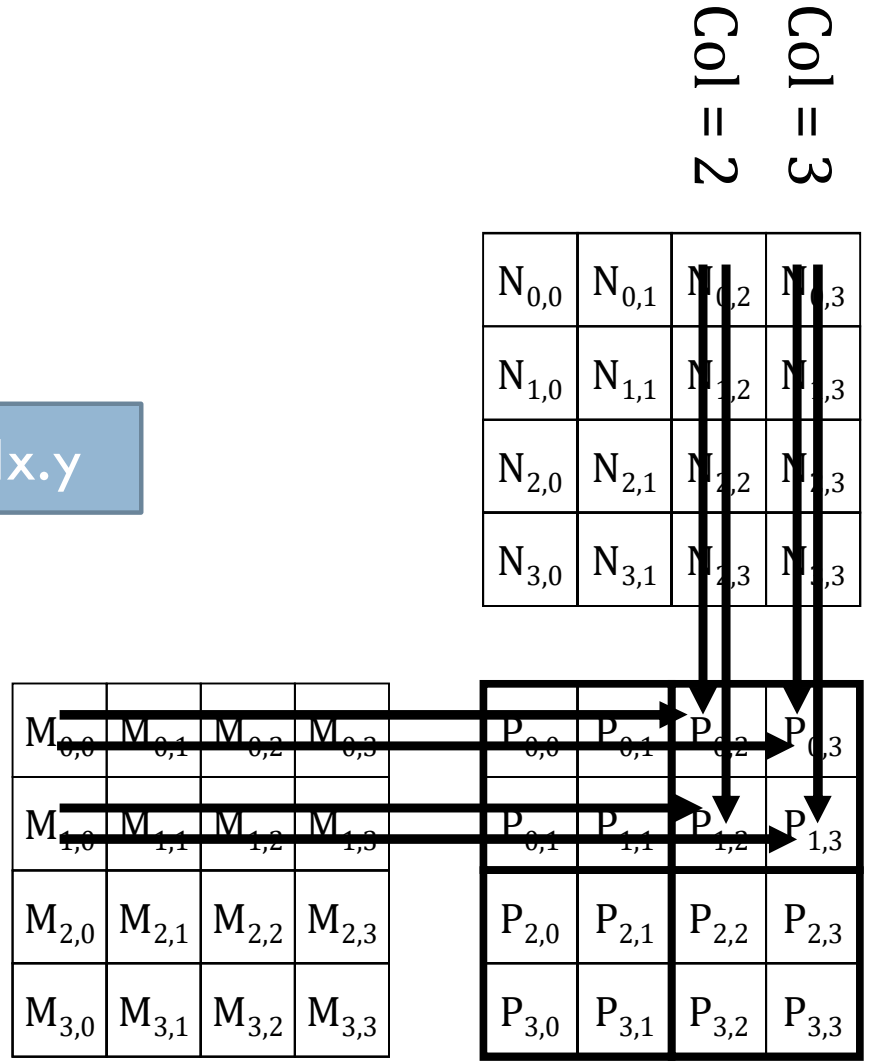


Επεξεργασία block (0, 1)

Col = 1 * 2 + threadIdx.x
Row = 0 * 2 + threadIdx.y

blockIdx.x

blockIdx.y



Row = 0

Row = 1

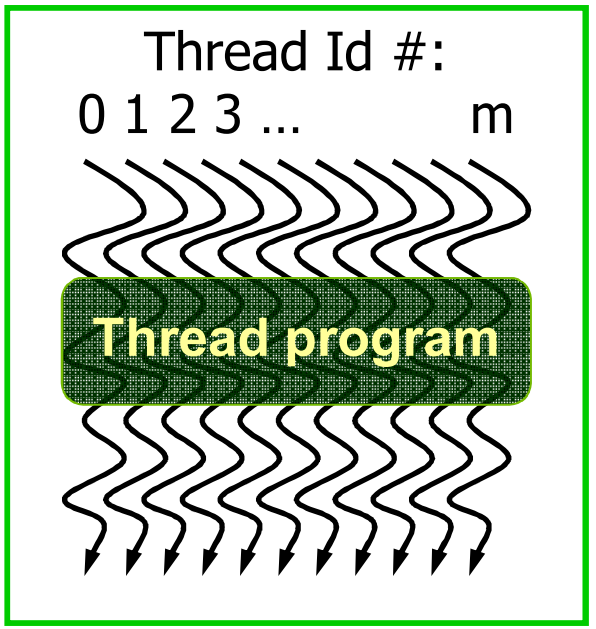
Μια πρώτη, απλή συνάρτηση πυρήνα για πολλαπλασιασμό πινάκων

```
__global__ void MatrixMulKernel(float* d_M, float* d_N,
                                float* d_P, int Width)
{
  // Calculate the row index of the d_P element and d_M
  int Row = blockIdx.y * blockDim.y + threadIdx.y;
  // Calculate the column idenx of d_P and d_N
  int Col = blockIdx.x * blockDim.x + threadIdx.x;

  if ((Row < Width) && (Col < Width)) {
    float Pvalue = 0;
    // each thread computes one element of the block sub-matrix
    for (int k = 0; k < Width; k++)
      Pvalue += d_M[Row * Width + k] * d_N[k * Width + Col];

    d_P[Row*Width+Col] = Pvalue;
  }
}
```

Block νημάτων στην CUDA

- Όλα τα νήματα ενός block εκτελούν την ίδια συνάρτηση πυρήνα (SPMD)
- Ο προγραμματιστής ορίζει τα χαρακτηριστικά του block:
  - Μέγεθος block από 1 έως **1024** νήματα
  - Διαστάσεις πλέγματος block (1D, 2D ή 3D)
  - Διαστάσεις πλέγματος νημάτων
- Τα νήματα έχουν δείκτες (indices) εντός του block
  - Ο κώδικας πυρήνα χρησιμοποιεί τους δείκτες νημάτων και block για να επιλέξει τμήμα υπολογισμών και να διευθυνσιοδοτήσει μνήμη
- Τα νήματα του ίδιου block μπορούν να μοιραστούν δεδομένα και να συγχρονιστούν μεταξύ τους όσο εκτελούν το τμήμα των υπολογισμών τους
- Τα νήματα διαφορετικών block δεν μπορούν να συνεργαστούν
  - Κάθε block μπορεί να εκτελεστεί με οποιαδήποτε σειρά σε σχέση με τα άλλα block!

CUDA Thread Block



Courtesy: John Nickolls, NVIDIA

Ιστορία παράλληλης επεξεργασίας

- 1η γενιά – Οι εντολές εκτελούνται σειριακά με βάση την σειρά εμφάνισης τους στο πρόγραμμα, μια κάθε φορά

- Παράδειγμα:

Cycle	1	2	3	4	5	6
Instruction1	Fetch	Decode	Execute	Memory		
Instruction2					Fetch	Decode

Ιστορία (Συνέχεια)

- 2η γενιά – Οι εντολές εκτελούνται σειριακά με βάση την σειρά εμφάνισης τους στο πρόγραμμα, αλλά όπως σε μια «γραμμή παραγωγής» (Pipeline)

- Example:

Cycle	1	2	3	4	5	6
Instruction1	Fetch	Decode	Execute	Memory		
Instruction2		Fetch	Decode	Execute	Memory	
Instruction3			Fetch	Decode	Execute	Memory

Ιστορία – Παραλληλισμός στο επίπεδο εντολών (Instruction Level Parallelism)

- 3η γενιά – Οι εντολές εκτελούνται ταυτόχρονα
- Παράδειγμα 1:

```
c = b + a;
d = c + e;
```

Μη παραλληλοποιήσιμο

- Παράδειγμα 2:

```
a = b + c;
d = e + f;
```

Παραλληλοποιήσιμο

Παραλληλισμός στο επίπεδο εντολών (Συνέχεια)

- Δύο μορφές:
  - Superscalar: Προσκόμιση, αποκωδικοποίηση και εκτέλεση πολλαπλών εντολών ταυτόχρονα. Η εκτέλεση μπορεί να είναι «εκτός σειράς» (Out of order)

Cycle	1	2	3	4	5
Instruction1	Fetch	Decode	Execute	Memory	
Instruction2	Fetch	Decode	Execute	Memory	
Instruction3		Fetch	Decode	Execute	Memory
Instruction4		Fetch	Decode	Execute	Memory

  - VLIW: Κατά τον χρόνο μεταγλώττισης γίνεται ομαδοποίηση πολλαπλών ανεξάρτητων εντολών σε μια μεγάλη εντολή και η επεξεργασία γίνεται θεωρώντας αυτές ως ανεξάρτητες μονάδες.

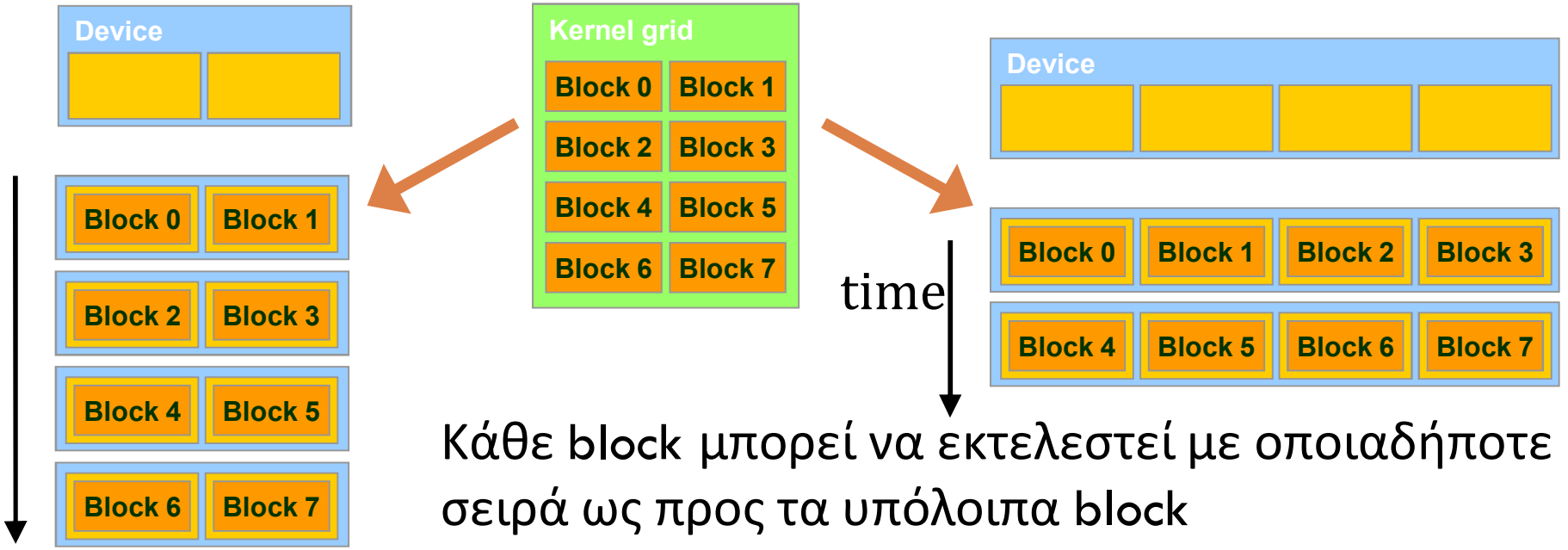
Ιστορία (Συνέχεια)

- 4η γενιά – Πολυνημάτωση (Multi-threading): Πολλαπλά νήματα εκτελούνται μέσω εναλλαγής ή πραγματικά ταυτόχρονα στον ίδιο επεξεργαστή ή πυρήνα (θα το επανεξετάσουμε όμως…)

- 5η γενιά - Multi-Core: Πολλαπλά νήματα εκτελούνται ταυτόχρονα σε πολλαπλούς επεξεργαστές ή/και πυρήνες

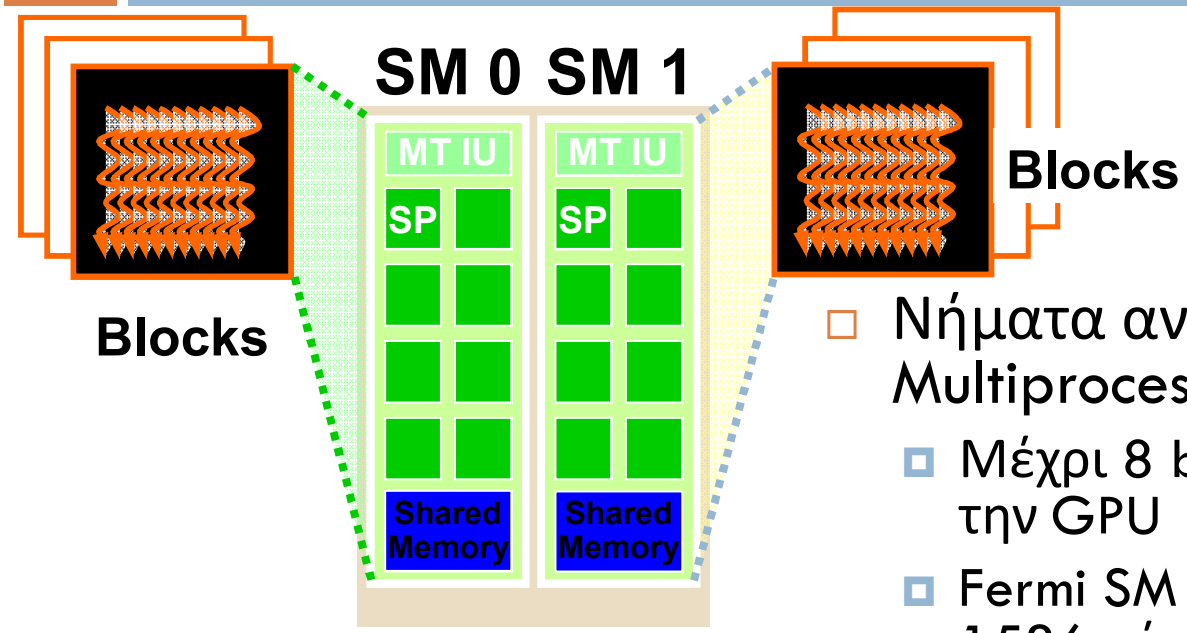
Διάφανη κλιμακωσιμότητα

- Το υλικό είναι ελεύθερο να αντιστοιχίσει οποιοδήποτε block σε οποιονδήποτε επεξεργαστή, οποιαδήποτε χρονική στιγμή
  - Μια συνάρτηση πυρήνα παρουσιάζει κλιμακωσιμότητα ως προς οποιοδήποτε πλήθος επεξεργαστών



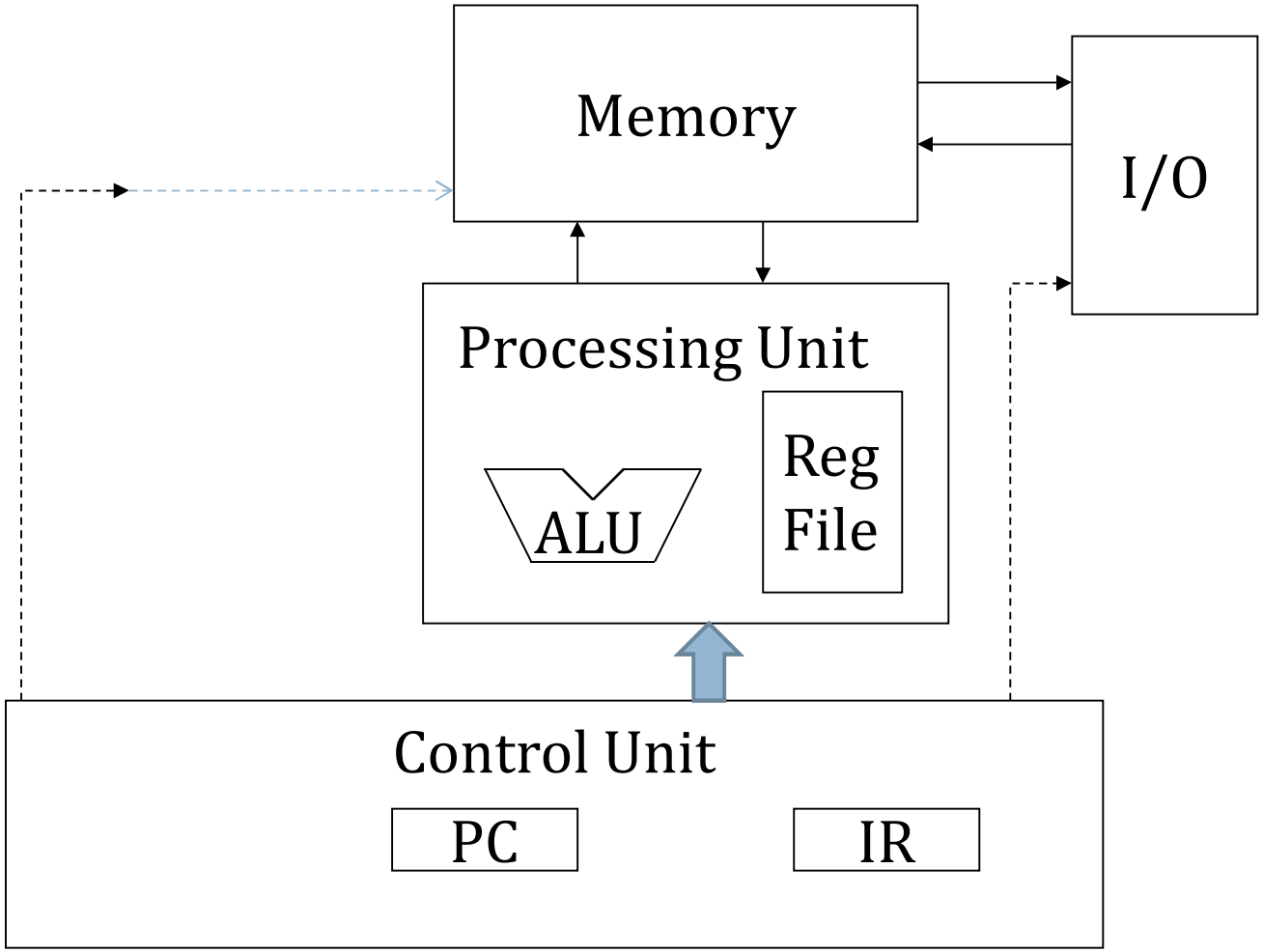
Κάθε block μπορεί να εκτελεστεί με οποιαδήποτε σειρά ως προς τα υπόλοιπα block

Παράδειγμα: Εκτέλεση block νημάτων

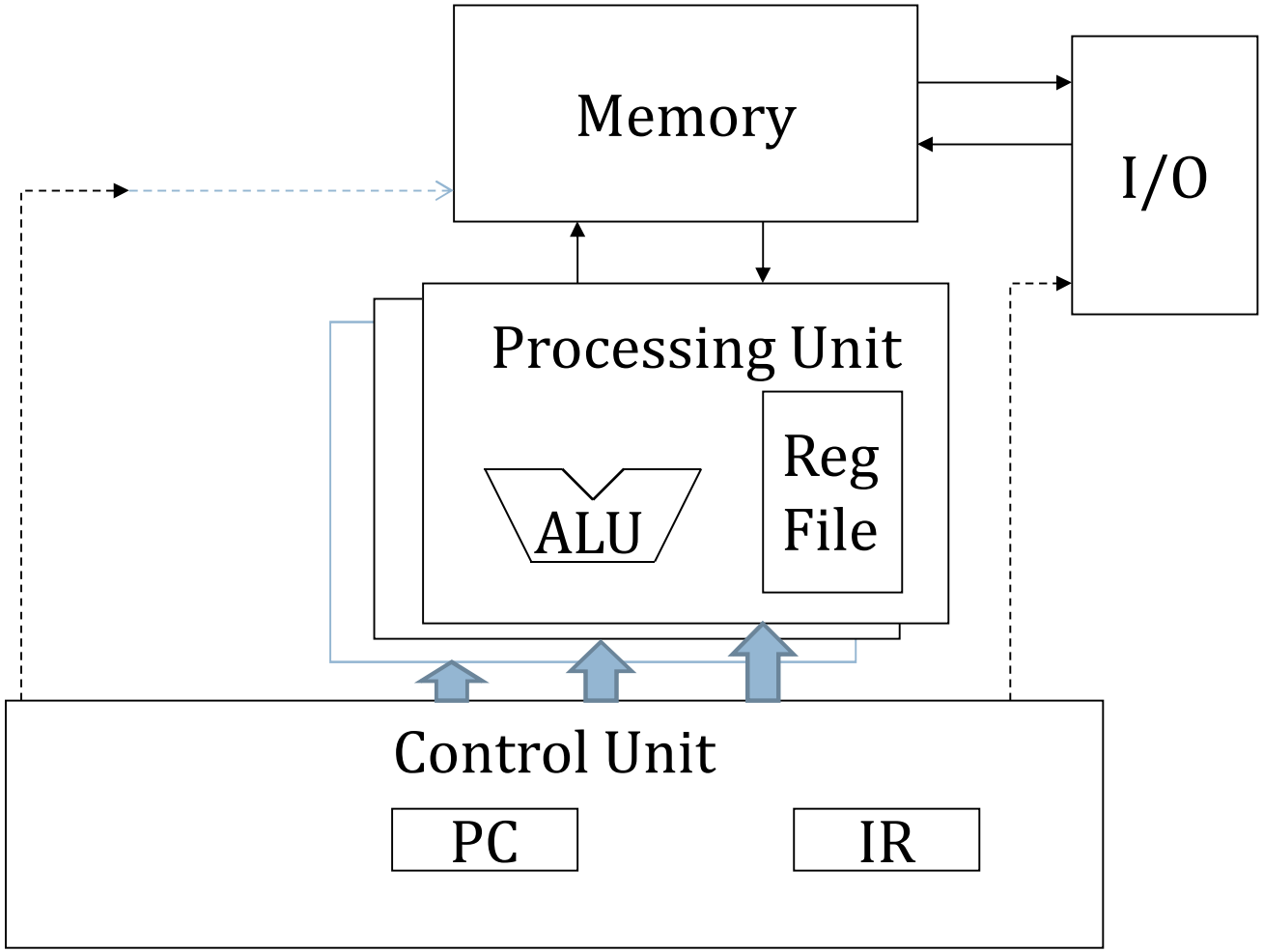


- Νήματα ανατίθενται σε Streaming Multiprocessors σε μεγέθη block
  - Μέχρι 8 block σε κάθε SM, ανάλογα με την GPU
  - Fermi SM μπορεί να διαχειριστεί μέχρι 1536 νήματα
    - Ίσως 256 (νήματα/block) * 6 blocks
    - Ή 512 (νήματα/block) * 3 block, κλπ.
- Τα νήματα τρέχουν ταυτόχρονα
  - SM διατηρεί δείκτες νήματος/block
  - SM διαχειρίζεται/χρονοδρομολογεί την εκτέλεση των νημάτων

Το μοντέλο Von-Neumann

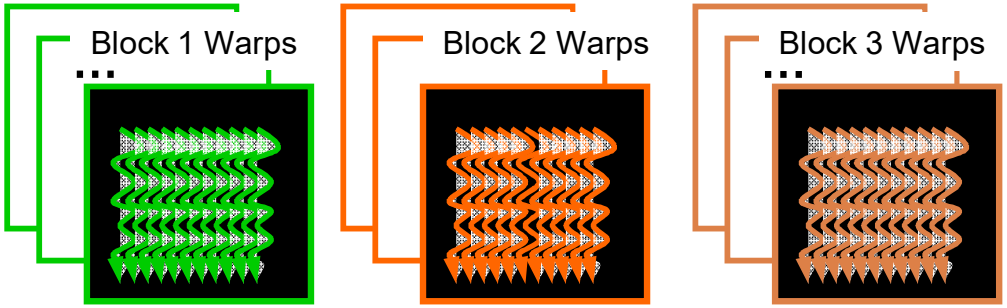


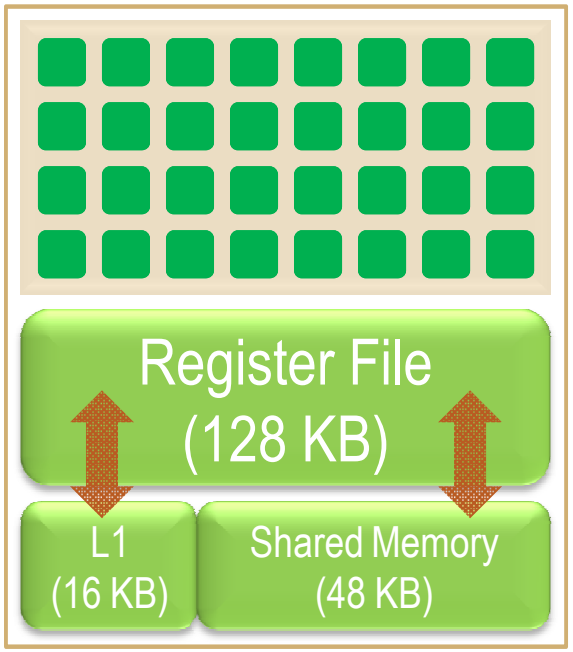
Το μοντέλο Von-Neumann με μονάδες SIMD



Παράδειγμα: Χρονοπρογραμματισμός νημάτων

- Τα νήματα κάθε block εκτελούνται ανά 32 σε warp
  - Απόφαση υλοποίησης στο υλικό, όχι μέρος του προγραμματιστικού μοντέλου CUDA
  - Τα warp είναι οι μονάδες χρονοπρογραμματισμού σε κάθε SM
- Αν σε ένα SM έχουν ανατεθεί 3 block και κάθε block έχει 256 νήματα, πόσα warp υπάρχουν στο SM;
  - Κάθε block αποτελείται από 256/32 = 8 warp
  - Συνολικά 8 * 3 = 24 warp





Επιστροφή στην εκτέλεση προγράμματος

- Κάθε εντολή πρέπει να προσκομιστεί από την μνήμη, να αποκωδικοποιηθεί και μετά να εκτελεστεί
- Υπάρχουν τριών ειδών εντολές: Εκτέλεση πράξης, Μεταφορά δεδομένων και Ελέγχου ροής του προγράμματος
- Ένα παράδειγμα εκτέλεσης των σταδίων μιας εντολής είναι το παρακάτω:

Fetch | Decode | Execute | Memory

Εντολές εκτέλεσης πράξης

- Παράδειγμα εντολής:

  ADD R1, R2, R3

- Στάδια εκτέλεσης εντολής:

  Fetch | Decode | Execute | Memory

Εντολές μεταφοράς δεδομένων

- Παραδείγματα εντολών:

```
LDR   R1, R2, #2
STR   R1, R2, #2
```

- Στάδια εκτέλεσης εντολής:

Fetch | Decode | Execute | Memory

Εντολές ελέγχου ροής

- Παράδειγμα εντολής:

  BRp #-4

- Αν η συνθήκη είναι θετική, τότε μεταφέρσου πίσω τέσσερις εντολές

- Στάδια εκτέλεσης εντολής:

  Fetch | Decode | Execute | Memory

Πως διαμοιράζονται τα block νημάτων

- Τα block νημάτων διαμοιράζονται σε warp
  - Οι δείκτες (indices) των νημάτων σε ένα warp είναι συνεχόμενα και αυξάνουν
  - Το warp 0 ξεκινάει με το νήμα 0
- Ο διαμοιρασμός είναι πάντα ο ίδιος
  - Μπρούμε να το εκμεταλλευτούμε στον έλεγχο ροής του προγράμματος
  - Ωστόσο το ακριβές μέγεθος των warp μπορεί να αλλάξει από γενιά σε γενιά στο υλικό
  - Καλύπτουμε το θέμα στην συνέχεια
- ΔΕΝ ΠΡΕΠΕΙ να βασιζόμαστε στην σειρά εκτέλεσης των warp
  - Αν υπάρχουν εξαρτήσεις μεταξύ νημάτων τότε πρέπει να χρησιμοποιηθεί η συνάρτηση __syncthreads() για να πάρουμε σωστά αποτελέσματα (περισσότερα στην συνέχεια)

Εντολές ελέγχου ροής

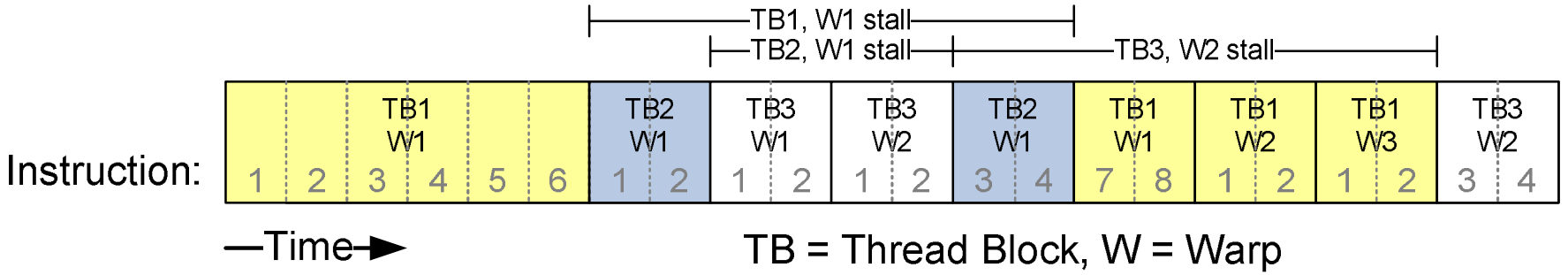
- Κύριο πρόβλημα με εντολές ελέγχου ροής
  - Απόκλιση στην ροή εκτέλεσης εντολών (divergence) των νημάτων με αρνητική επίδραση στην απόδοση
  - Νήματα στο ίδιο warp ακολουθούν διαφορετικά μονοπάτια εκτέλεσης
  - Διαφορετικά μονοπάτια εκτέλεσης σειριοποιούνται στις υπάρχουσες GPU
    - Τα μονοπάτια που ακολουθούν τα νήματα σε ένα warp ακολουθούνται ένα προς ένα μέχρι να μην υπάρχουν άλλα μονοπάτια

Εντολές ελέγχου ροής (Συνέχεια)

- Συνήθης περίπτωση: αποφυγή divergence όταν ο έλεγχος είναι συνάρτηση του δείκτη (index) νήματος
  - Παράδειγμα με divergence:
    - `if (threadIdx.x > 2) { }`
    - Δημιουργεί δύο διαφορετικά μονοπάτια εκτέλεσης εντολών για τα νήματα ενός block
    - Συμβαίνει γιατί ο έλεγχος υποχρεώνει λιγότερα νήματα από το μέγεθος ενός warp να ακολουθήσουν κάθε μονοπάτι
    - Τα νήματα 0, 1 και 2 ακολουθούν διαφορετικό μονοπάτι εκτέλεσης από τα υπόλοιπα νήματα του πρώτου warp
  - Παράδειγμα χωρίς divergence:
    - `if (threadIdx.x / WARP_SIZE > 2) { }`
    - Επίσης δημιουργεί δύο διαφορετικά μονοπάτια εκτέλεσης για νήματα στο ίδιο block
    - Όμως το πλήθος των νημάτων που ακολουθούν κάθε μονοπάτι είναι πολλαπλάσιο του μεγέθους του warp
    - Όλα τα νήματα ενός warp ακολουθούν το ίδιο μονοπάτι

Παράδειγμα: Χρονοπρογραμματισμός νημάτων (Συνέχεια)

- Τα SM υλοποιούν χρονοπρογραμματισμό των warp με μηδενική επιβάρυνση
  - Κάθε χρονική στιγμή 1 ή 2 warp εκτελούνται από ένα SM
  - Τα warp των οποίων η επόμενη προς εκτέλεση εντολή έχει τα έντελα της έτοιμα προς χρήση μπορεί να επιλεγεί προς εκτέλεση
  - Τα έτοιμα προς εκτέλεση warp επιλέγονται για εκτέλεση με μια πολιτική χρονοδρομολόγησης βασισμένη σε προτεραιότητες
  - Όλα τα νήματα ενός warp που επιλέχθηκε προς εκτέλεση, εκτελούν τις ίδιες εντολές



Θέματα επιλογής μεγέθους block

- Για έναν πολλαπλασιασμό πινάκων που χρησιμοποιεί block, ποια είναι η καλύτερη επιλογή;
  - Block μεγέθους 8x8, 16x16 ή 32x32;
- Για 8x8, θα έχουμε 64 νήματα ανά block
  - Κάθε SM μπορεί να διαχειριστεί μέχρι 1536 νήματα, οπότε χρειαζόμαστε 24 block για να εκμεταλλευτούμε πλήρως το SM
  - Όμως κάθε SM μπορεί να διαχειριστεί μέχρι 8 blocks, άρα μόλις 512 νήματα θα πάνε σε κάθε SM!
- Για 16x16, θα έχουμε 256 νήματα ανά block
  - Κάθε SM μπορεί να διαχειριστεί μέχρι 1536 νήματα, οπότε χρειαζόμαστε 6 block για να εκμεταλλευτούμε πλήρως το SM
  - Εκμεταλλευόμαστε πλήρως τις δυνατότητες του SM (εκτός αν υπάρχουν άλλα θέματα στην κατανομή των πόρων, όπως πλήθος καταχωρητών, κλπ)
- Για 32x32, θα έχουμε 1024 νήματα ανά block
  - Κάθε SM μπορεί να διαχειριστεί μόνο ένα block
  - Εκμεταλλευόμαστε μόλις τα 2/3 της χωρητικότητας του SM σε νήματα
  - Λειτουργεί από την έκδοση 3.0 και μετά της CUDA
    - Πιθανόν πολύ μεγάλο για προηγούμενες εκδόσεις

Περισσότερες ερωτήσεις; Διαβάστε το Κεφάλαιο 4!

ΜΕΡΙΚΕΣ ΕΠΙΠΛΕΟΝ ΔΥΝΑΤΟΤΗΤΕΣ ΤΗΣ ΔΙΕΠΑΦΗΣ ΠΟΥ ΠΡΟΣΦΕΡΕΙ Η CUDA

Διεπαφή (Application Programming Interface ή API)

- Η διεπαφή είναι μια επέκταση της γλώσσας προγραμματισμού C
- Αποτελείται από:
  - Επεκτάσεις της γλώσσας προγραμματισμού
    - Για την εκτέλεση τμημάτων κώδικα στο device
  - Μια βιβλιοθήκη χρόνου εκτέλεσης (runtime library) που αποτελείται από:
    - Ένα κοινό μέρος για το host και το device το οποίο προσφέρει ενσωματωμένους τύπους διανυσμάτων (vector types) και ένα υποσύνολο της βιβλιοθήκης χρόνου εκτέλεσης της C
    - Ένα τμήμα για το host για την διαχείριση ενός ή περισσότερων device από τον host
    - Ένα τμήμα για το device που προσφέρει συναρτήσεις μόνο για αυτό

Κοινό μέρος βιβλιοθήκης χρόνου εκτέλεσης: Μαθηματικές συναρτήσεις

- pow, sqrt, cbrt, hypot
- exp, exp2, expm1
- log, log2, log10, log1p
- sin, cos, tan, asin, acos, atan, atan2
- sinh, cosh, tanh, asinh, acosh, atanh
- ceil, floor, trunc, round
- Κλπ.
  - Όταν εκτελείται στον host, η συνάρτηση χρησιμοποιεί την αντίστοιχη υλοποίηση της βιβλιοθήκης χρόνου εκτέλεσης της C (αν υπάρχει)
  - Οι συναρτήσεις υποστηρίζονται μόνο για μεταβλητές και όχι για διανυσματικούς τύπους δεδομένων (vector types)

Τμήμα μόνο για το device: Μαθηματικές συναρτήσεις

- Μερικές μαθηματικές συναρτήσεις (π.χ. sin(x)) έχουν ειδικές εκδόσεις για το device που έχουν μικρότερη ακρίβεια αλλά είναι πιο γρήγορες (π.χ. __sin(x))
  - __pow
  - __log, __log2, __log10
  - __exp
  - __sin, __cos, __tan