# ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

# Μάθημα #7

# GPUs – CUDA (Εισαγωγή)

# Επιταχυντές / GPUs

Τι είναι μία GPU;

- Αντικατέστησε τον VGA Controller μετά το 2000+
- Είναι ένας συνεπεξεργαστής βελτιστοποιημένος για 2D/3D γραφικά, video, οπτικό υπολογισμό και απεικόνιση.
- Είναι ένας υψηλά/μαζικά παράλληλος (ως προς το υλικό) – πολυπύρηνος πολυεπεξεργαστής, βελτιστοποιημένος για τα παραπάνω (οπτικούς υπολογισμούς κλπ).
- Παρέχει οπτική αλληλεπίδραση πραγματικού χρόνου σε εφαρμογές υψηλών απαιτήσεων γραφικών και video.
- **Μπορεί να χρησιμοποιηθεί τόσο ως ένας σύγχρονος επεξεργαστής γραφικών όσο και ως μια πολυπύρηνη μονάδα παράλληλης επεξεργασίας.**
- **Ετερογενή συστήματα:** συνεργασία GPU με CPU (συνδυασμένη επεξεργασία, επιμερισμός και επικοινωνία)

# Επιταχυντές / GPUs

- Καλύτερη επίδοση ανά μονάδα ισχύος και κόστους από τους επεξεργαστές
- **Μαζικά πολυπύρηνες – πολύ αποδοτικές για εύκολα παραλληλοποιήσιμες εφαρμογές**
- Διαχείριση ιεραρχίας μνήμης από το λογισμικό
- **Χαμηλή επίδοση σε εφαρμογές με πολύπλοκες ροές ελέγχου**
- Απαιτούν εξειδικευμένα προγραμματιστικά μοντέλα και εργαλεία
- NVIDIA & CUDA

# Top 500 Supercomputers (2017)



| Rank | Site | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|---|
| 1 | National Supercomputing Center in Wuxi<br>China | **Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway<br>NRCPC | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 2 | National Super Computer Center in Guangzhou<br>China | **Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P<br>NUDT | 3,120,000 | 33,862.7 | 54,902.4 | 17,808 |
| 3 | Swiss National Super-computing Centre (CSCS)<br>Switzerland | **Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100<br>Cray Inc. | 361,760 | 19,590.0 | 25,326.3 | 2,272 |
| 4 | DOE/SC/Oak Ridge National Laboratory<br>United States | **Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x<br>Cray Inc. | 560,640 | 17,590.0 | 27,112.5 | 8,209 |
| 5 | DOE/NNSA/LLNL<br>United States | **Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom<br>IBM | 1,572,864 | 17,173.2 | 20,132.7 | 7,890 |

# Source: www.top500.org – Top List 2020

| Rank | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science - Japan | 7,299,072 | 415,530.0 | 513,854.7 | 28,335 |
| 2 | Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM, DOE/SC/Oak Ridge National Laboratory - United States | 2,414,592 | 148,600.0 | 200,794.9 | 10,096 |
| 3 | Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL - United States | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 4 | Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center - China | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 5 | Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT - National Super Computer Center in Guangzhou - China | 4,981,760 | 61,444.5 | 100,678.7 | 18,482 |
| 6 | HPC5 - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, Dell EMC, Eni S.p.A., Italy | 669,760 | 35,450.0 | 51,720.8 | 2,252 |

# ARIS Supercomputer (ΕΔΕΤ) (2016)



Στο αρχικό σύστημα των 426 επιμέρους κόμβων προστέθηκαν πρόσφατα **τρεις νέες υπολογιστικές νησίδες:**

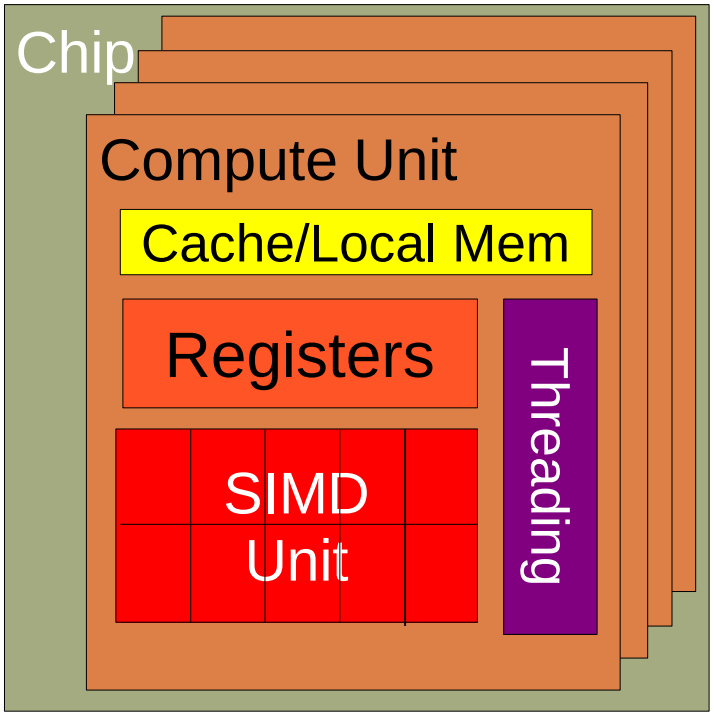
- νησίδα κόμβων μεγάλης μνήμης (fat nodes) που αποτελείται από 44 εξυπηρετητές Dell PowerEdge R820. Κάθε εξυπηρετητής προσφέρει 4 επεξεργαστές Intel Xeon E5-4650v2 και 512 GB κεντρικής μνήμης

- **νησίδα κόμβων GPU που αποτελείται από 44 εξυπηρετητές Dell PowerEdge R730. Κάθε εξυπηρετητής περιέχει 2 επεξεργαστές Intel Xeon E5-2660v3, 64 GB μνήμης και 2 κάρτες GPU NVidia K40, και**

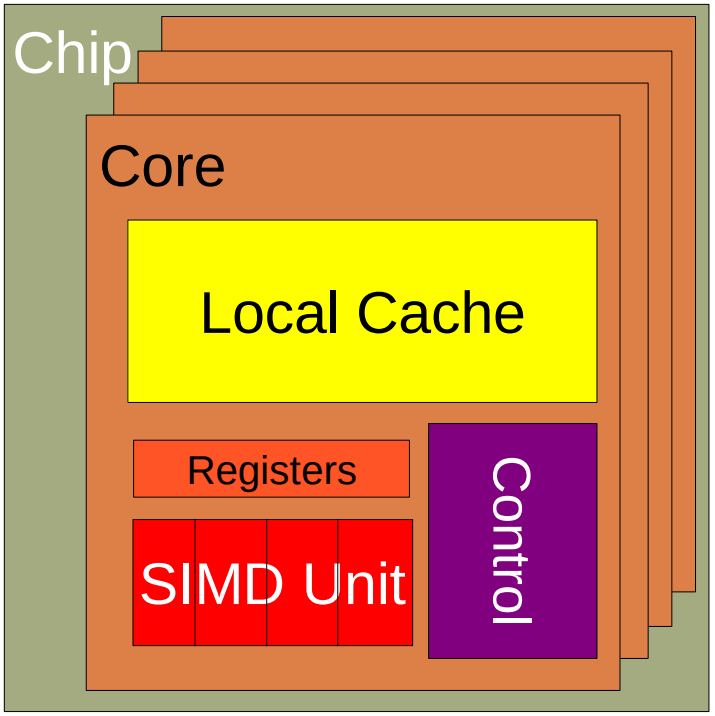
- νησίδα κόμβων που αποτελείται από 18 εξυπηρετητές Dell PowerEdge R730, καθένας εκ των οποίων περιέχει 2 επεξεργαστές Intel Xeon E5-2660v3, 64 GB μνήμης και 2 συνεπεξεργαστές Intel Xeon Phi 7120P.

# Διαφορετική φιλοσοφία σχεδιασμού μεταξύ CPU και GPU

## GPU
Throughput Oriented Cores

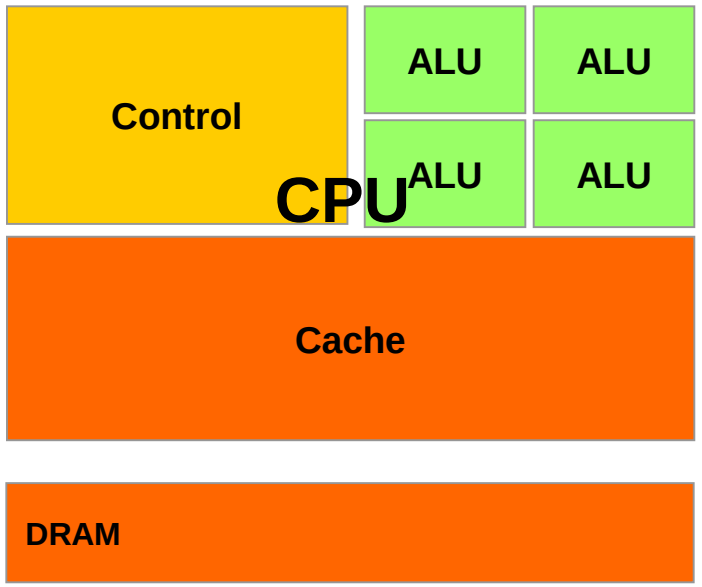



## CPU
Latency Oriented Cores



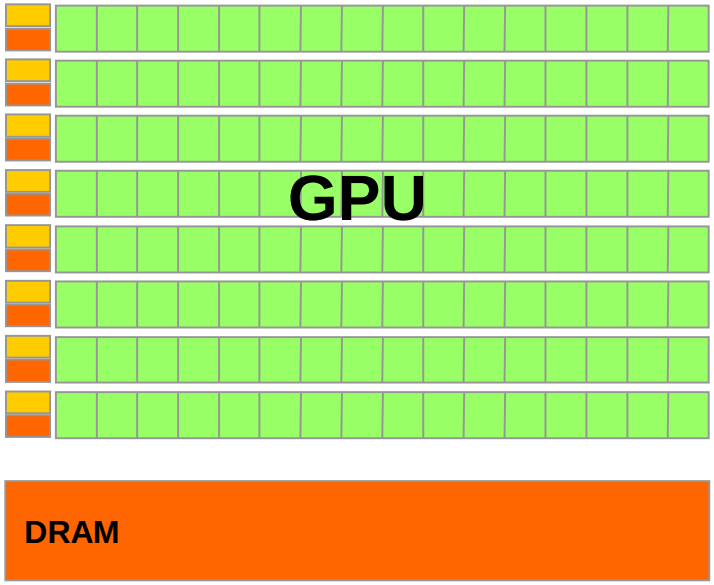
# CPU: Σχεδιασμός για μεγάλο χρόνο προσπέλασης στην μνήμη (latency)

- Μεγάλες κρυφές μνήμες
  - Μετατρέπουν αργές προσπελάσεις μνήμης σε γρηγορότερες προσπελάσεις στην κρυφή μνήμη
- Εξελιγμένος έλεγχος
  - Πρόβλεψη διακλαδώσεων για μείωση καθυστερήσεων
  - Προώθηση δεδομένων στον αγωγό για μείωση καθυστερήσεων
- Ισχυρή ALU
  - Μικρότερος χρόνος εκτέλεσης ανά εντολή



# GPU: Σχεδιασμός για διεκπεραιωτική ικανότητα (throughput)

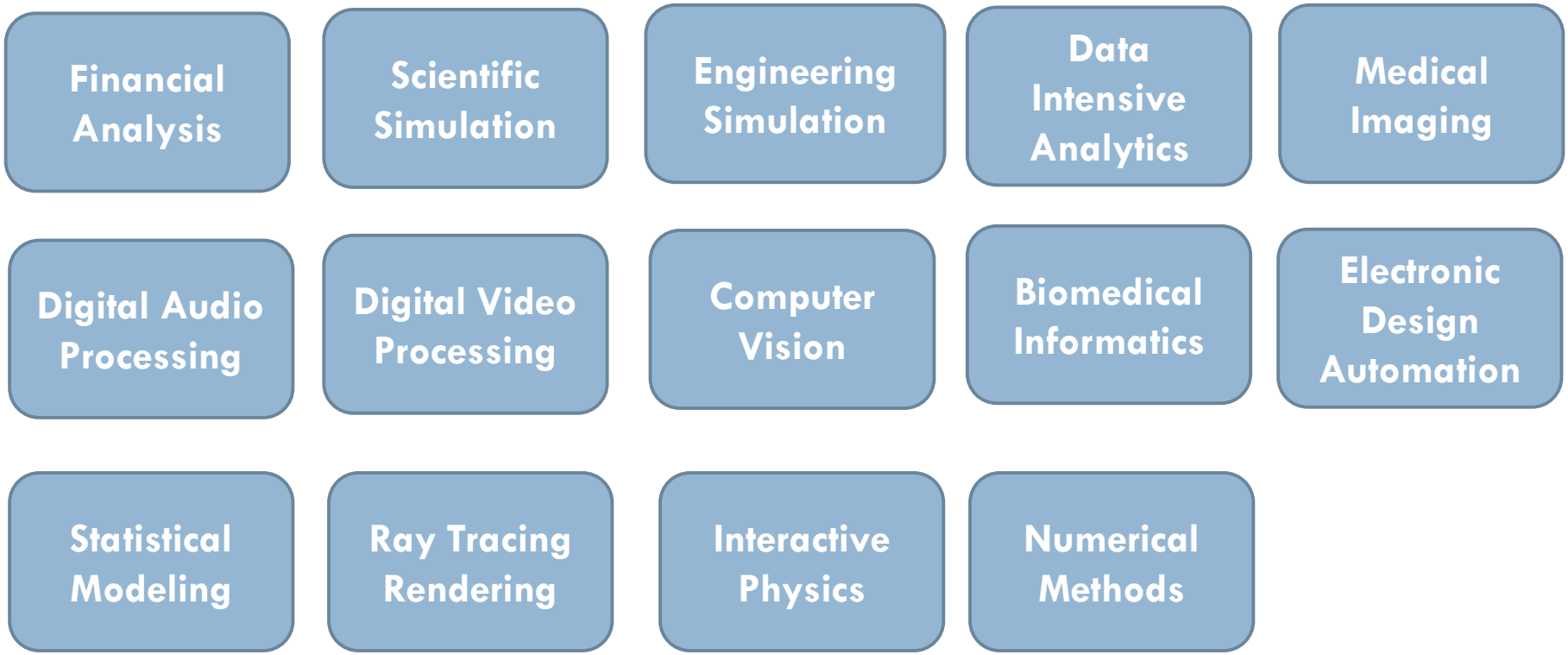
- Μικρές κρυφές μνήμες
  - Για την βελτίωση της ικανότητας διαβίβασης δεδομένων της μνήμης
- Απλός έλεγχος
  - Όχι πρόβλεψη διακλαδώσεων
  - Όχι προώθηση δεδομένων
- Ενεργειακά αποδοτικές ALUs
  - Πολλές, με μεγάλο χρόνο εκτέλεσης, αλλά με πολλά επίπεδα στον αγωγό
- Απαιτεί την ύπαρξη μεγάλου πλήθους νημάτων για την αντιμετώπιση των καθυστερήσεων από προσπελάσεις στην μνήμη



# Οι καλύτερες εφαρμογές χρησιμοποιούν και CPU και GPU

- CPUs για τα σειριακά τμήματα στα οποία έχει σημασία το latency
  - Οι CPUs μπορεί να είναι 10+X γρηγορότερες από τις GPUs για σειριακό κώδικα
- GPUs για τα παράλληλα τμήματα στα οποία έχει σημασία το throughput
  - Οι GPUs μπορεί να είναι 10+X γρηγορότερες από τις CPUs για παράλληλο κώδικα

# Η χρήση υβριδικού παράλληλου προγραμματισμού διαδίδεται

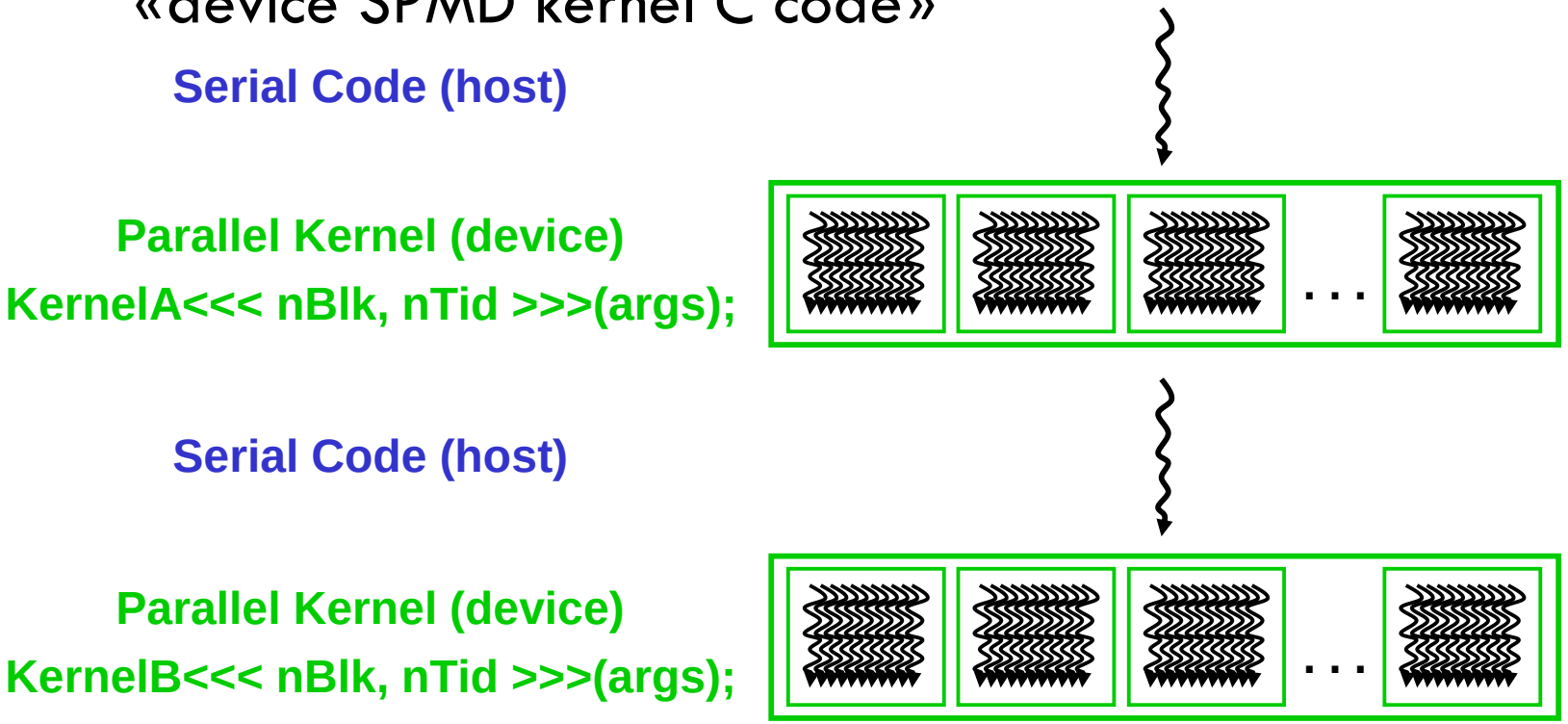


# GPU Gems

- Σειρά τριών βιβλίων
- Παραλληλοποίηση εφαρμογών για GPU
- 480 υποβολές στο GPU Computing Gems
  - 150 άρθρα έχουν συμπεριληφθεί στις τρεις εκδόσεις
- Ελεύθερα διαθέσιμα:
  - https://developer.nvidia.com/gpugems/GPUGems/gpugems_pref01.html

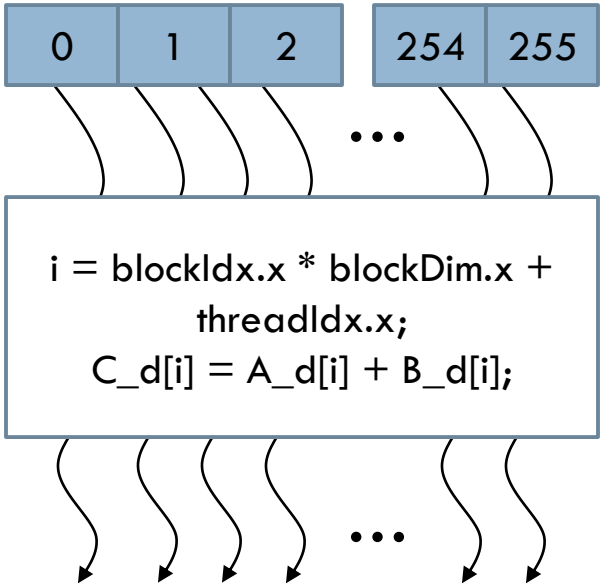
# CUDA /OpenCL – Μοντέλο εκτέλεσης

- Ενοποιημένο πρόγραμμα C για host+device
  - Σειριακός ή «ελαφρά» παράλληλος κώδικας C στον host
  - Παράλληλος κώδικας για την εξωτερική συσκευή σε «device SPMD kernel C code»



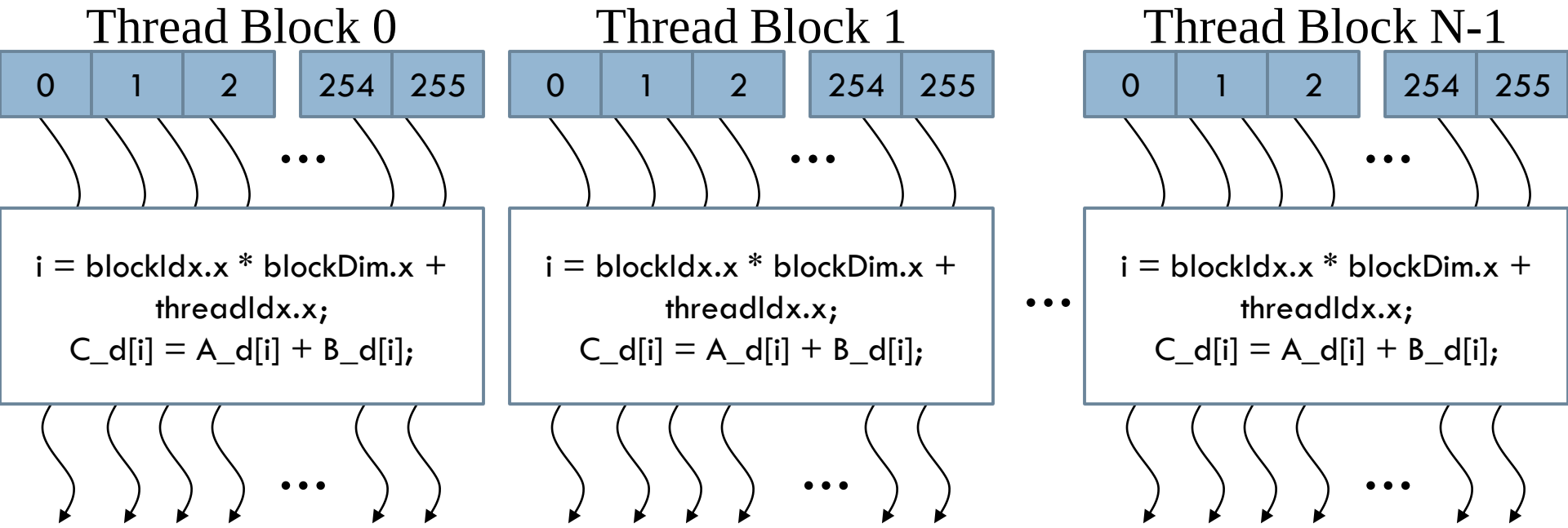
# Πίνακες Παράλληλων Νημάτων (Arrays of Parallel Threads)

- Μια συνάρτηση πυρήνα της CUDA (CUDA kernel) εκτελείται από ένα πλέγμα (ή πίνακα) νημάτων
  - Όλα τα νήματα του πλέγματος εκτελούν την ίδια συνάρτηση πυρήνα (SPMD)
  - Κάθε νήμα έχει ένα σύνολο (ακεραίων) δεικτών, για να υπολογίζει θέσεις μνήμης και να παίρνει αποφάσεις



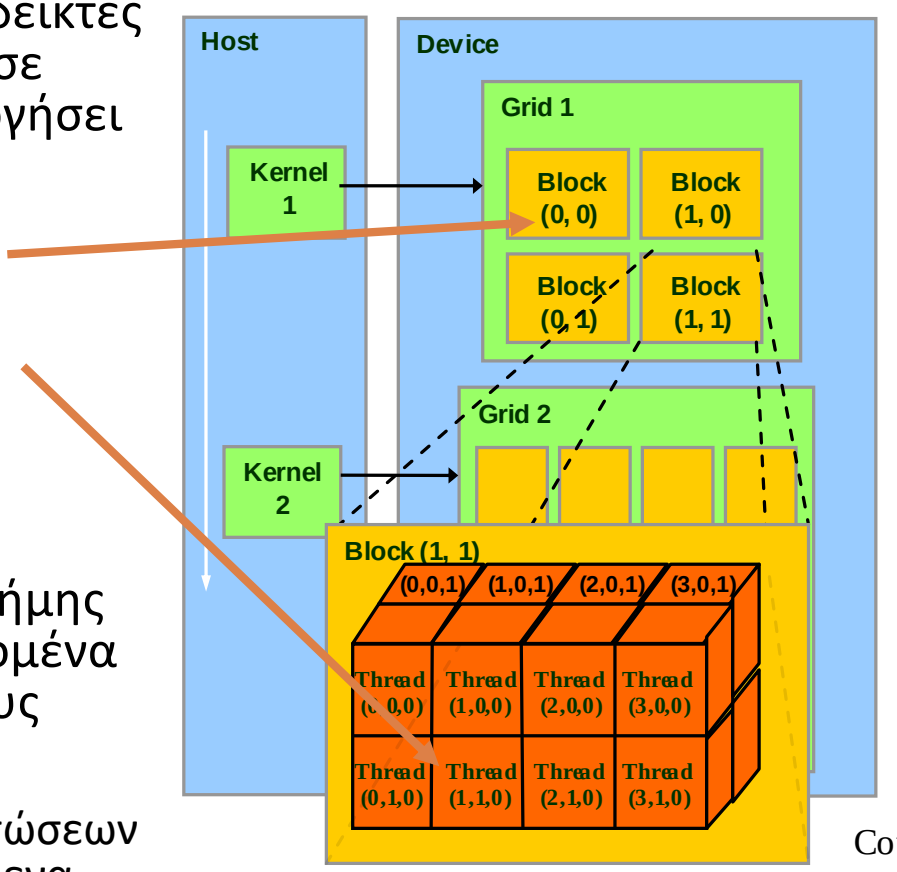
# Μπλοκ νημάτων (Thread Blocks): Κλιμακώσιμη συνεργασία

- Διαίρεση πίνακα νημάτων σε περισσότερα μπλοκ
  - Νήματα στο ίδιο μπλοκ συνεργάζονται μέσω κοινής μνήμης, ατομικών εντολών και φραγμάτων (barriers)
  - Νήματα σε διαφορετικά μπλοκ δεν συνεργάζονται



# blockIdx και threadIdx

- Κάθε νήμα χρησιμοποιεί δείκτες για να αποφασίσει πάνω σε ποια δεδομένα θα επενεργήσει
  - blockIdx: 1D, 2D ή 3D
  - threadIdx: 1D, 2D ή 3D
  - gridDim, blockDim
- Απλοποιεί την διευθυνσιοδότηση της μνήμης όταν επεξεργάζονται δεδομένα σε πολυδιάστατους χώρους
  - Επεξεργασία εικόνας
  - Επίλυση διαφορικών εξισώσεων σε τρισδιάστατα αντικείμενα
  - …




Courtesy: NDVIA

# Γιατί όχι μόνο thread αλλά και block;

- Η φυσική υλοποίηση για το πλήθος thread έχει όρια

| Technical specifications | Compute capability (version) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1.0 | 1.1 | 1.2 | 1.3 | 2.x | 3.0 | 3.5 | 3.7 | 5.0 | 5.2 |
| Maximum dimensionality of grid of thread blocks | 2 | | | | 3 | | | | | |
| Maximum x-dimension of a grid of thread blocks | 65535 | | | | | $2^{31}$-1 | | | | |
| Maximum y-, or z-dimension of a grid of thread blocks | 65535 | | | | | | | | | |
| Maximum dimensionality of thread block | 3 | | | | | | | | | |
| Maximum x- or y-dimension of a block | 512 | | | | 1024 | | | | | |
| Maximum z-dimension of a block | 64 | | | | | | | | | |
| Maximum number of threads per block | 512 | | | | 1024 | | | | | |

CUDA C Programming Guide, Appendix G, Section G.1, Part of Table 13

- Για να έχουμε περισσότερα thread εισάγουμε την έννοια του πλέγματος (grid) από block

# CUDA Capability - Parameters

| Technical specifications | Compute capability (version) | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1.0 | 1.1 | 1.2 | 1.3 | 2.x | 3.0 | 3.2 | 3.5 | 3.7 | 5.0 | 5.2 | 5.3 | 6.0 | 6.1 | 6.2 | 7.0 | 7.2 | 7.5 | 8.0 | 8.6 |
| Maximum number of resident grids per device (concurrent kernel execution) | t.b.d. | | | | 16 | | 4 | | 32 | | | 16 | 128 | 32 | 16 | 128 | 16 | 128 | | |
| Maximum dimensionality of grid of thread blocks | 2 | | | | 3 | | | | | | | | | | | | | | | |
| Maximum x-dimension of a grid of thread blocks | 65535 | | | | $2^{31}-1$ | | | | | | | | | | | | | | | |
| Maximum y-, or z-dimension of a grid of thread blocks | 65535 | | | | | | | | | | | | | | | | | | | |
| Maximum dimensionality of thread block | 3 | | | | | | | | | | | | | | | | | | | |
| Maximum x- or y-dimension of a block | 512 | | | | 1024 | | | | | | | | | | | | | | | |
| Maximum z-dimension of a block | 64 | | | | | | | | | | | | | | | | | | | |
| Maximum number of threads per block | 512 | | | | 1024 | | | | | | | | | | | | | | | |
| Warp size | 32 | | | | | | | | | | | | | | | | | | | |
| Maximum number of resident blocks per multiprocessor | 8 | | | | | 16 | | | | 32 | | | | | | | | 16 | 32 | 16 |
| Maximum number of resident warps per multiprocessor | 24 | | 32 | | 48 | 64 | | | | | | | | | | | | 32 | 64 | 48 |
| Maximum number of resident threads per multiprocessor | 768 | | 1024 | | 1536 | 2048 | | | | | | | | | | | | 1024 | 2048 | 1536 |

# NVIDIA GPU Example

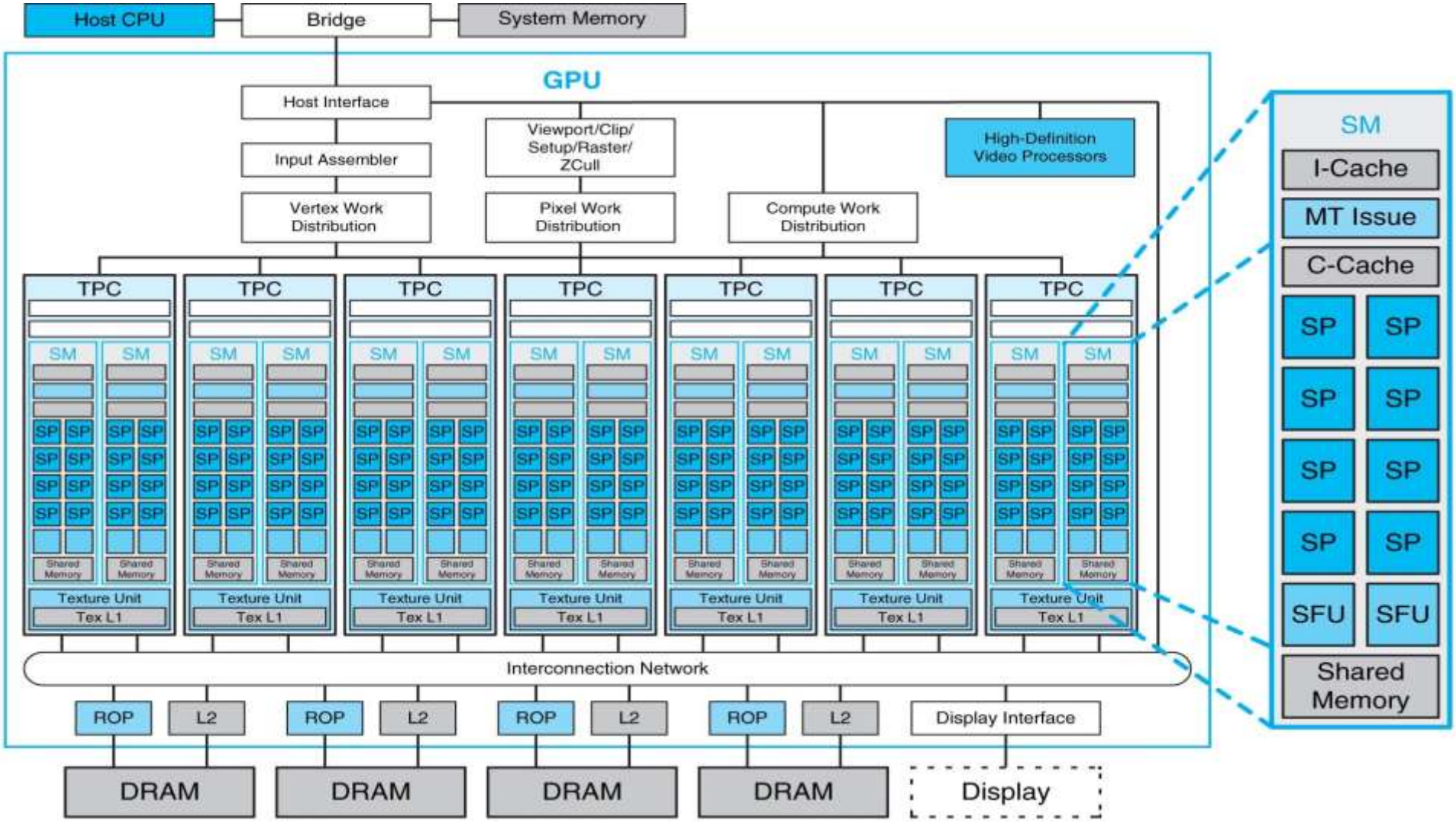
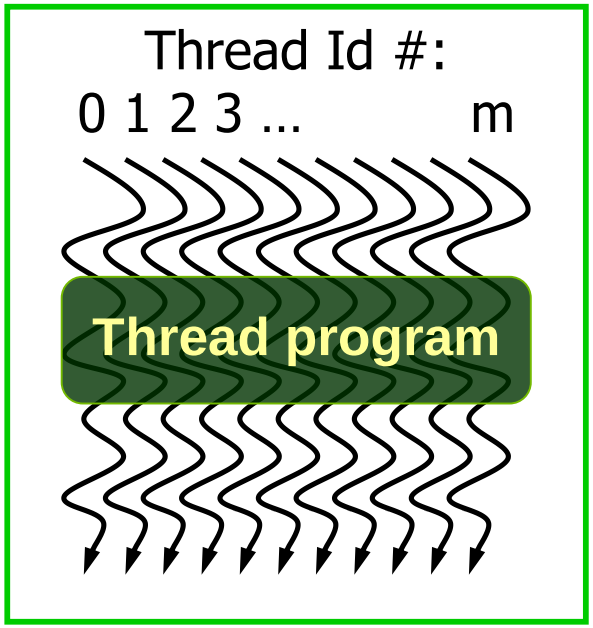



**FIGURE A.2.5 Basic unified GPU architecture.** Example GPU with 112 streaming processor (SP) cores organized in 14 streaming multiprocessors (SMs); the cores are highly multithreaded. It has the basic Tesla architecture of an NVIDIA GeForce 8800. The processors connect with four 64-bit-wide DRAM partitions via an interconnection network. Each SM has eight SP cores, two special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory. Copyright © 2009 Elsevier, Inc. All rights reserved.

# 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!

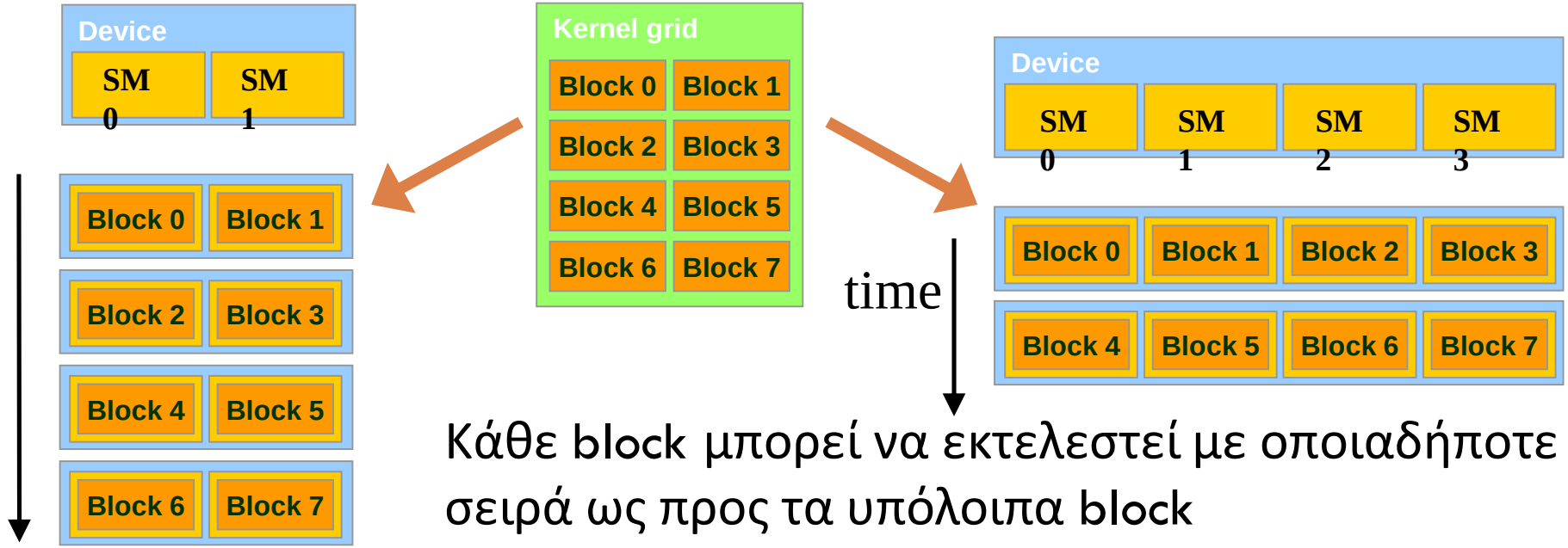
**CUDA Thread Block**




Courtesy: John Nickolls, NVIDIA

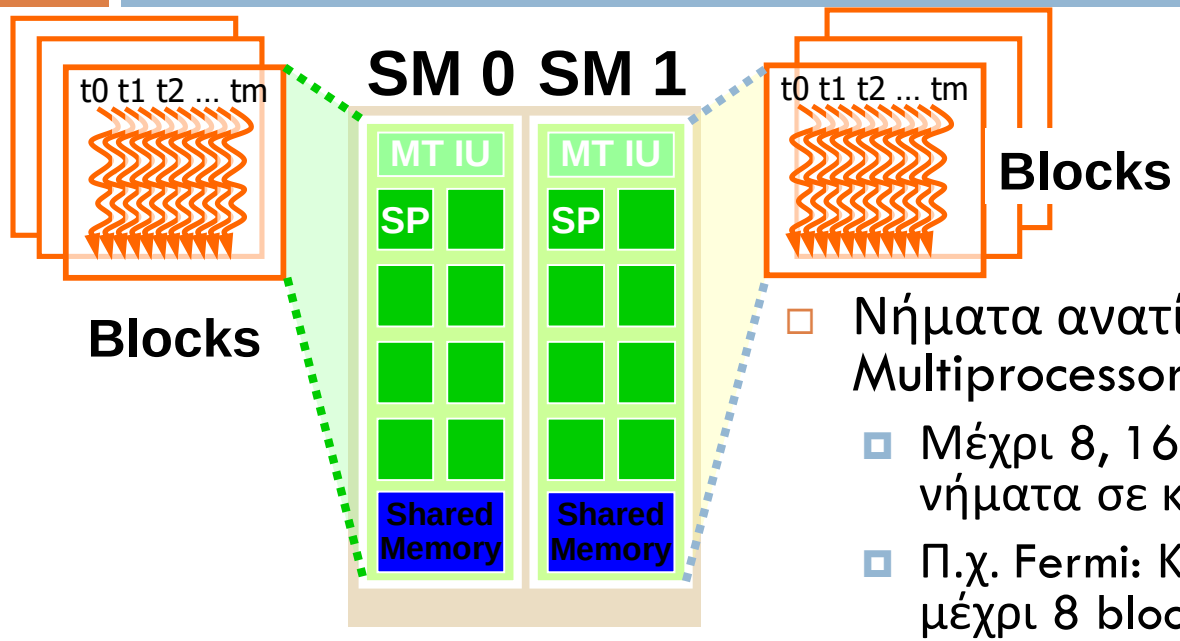
# Διάφανη κλιμακωσιμότητα

- 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

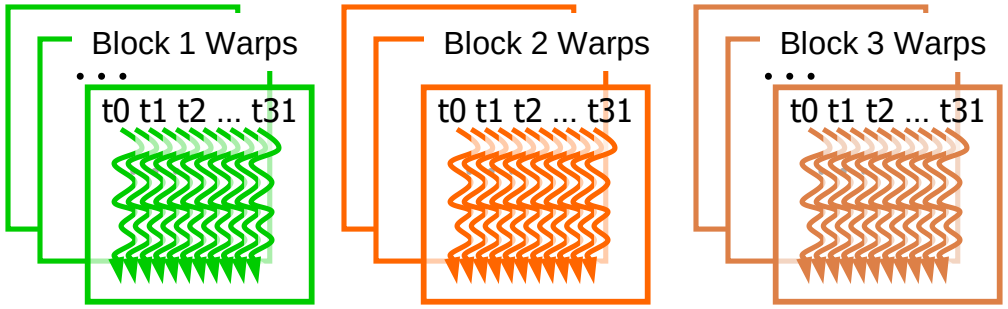
# Παράδειγμα: Εκτέλεση block νημάτων

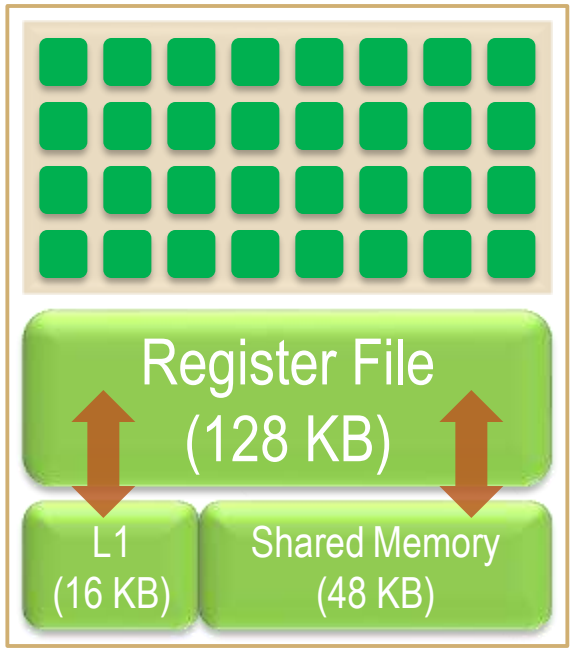



- Νήματα ανατίθενται σε Streaming Multiprocessors σε μεγέθη block
  - Μέχρι 8, 16, 32 blocks και 1024, 1536, 2048 νήματα σε κάθε SM, ανάλογα με την GPU
  - Π.χ. Fermi: Κάθε SM μπορεί να διαχειριστεί μέχρι 8 blocks και 1536 νήματα
    - Ίσως 256 (νήματα/block) * 6 blocks
    - Ή 512 (νήματα/block) * 3 block, κλπ.
- Τα νήματα τρέχουν ταυτόχρονα
  - SM διατηρεί δείκτες νήματος/block
  - SM διαχειρίζεται/χρονοδρομολογεί την εκτέλεση των νημάτων

# Παράδειγμα: Χρονοπρογραμματισμός νημάτων

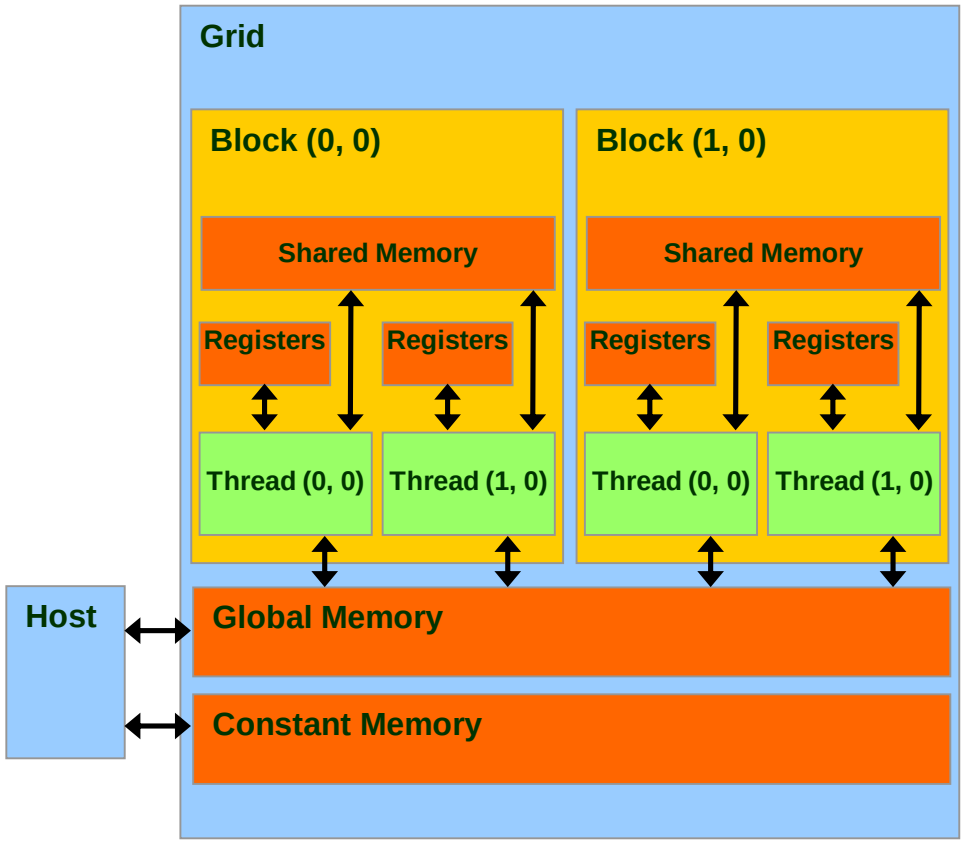
- Τα νήματα κάθε block εκτελούνται ανά 32 σε warp
  - Απόφαση υλοποίησης στο υλικό, όχι μέρος του προγραμματιστικού μοντέλου CUDA
  - Τα warp είναι οι μονάδες χρονοπρογραμματισμού σε κάθε SM
- Αν σε ένα SM έχουν ανατεθεί 3 block και κάθε block έχει 256 νήματα, πόσα warp υπάρχουν στο SM;
  - Κάθε block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



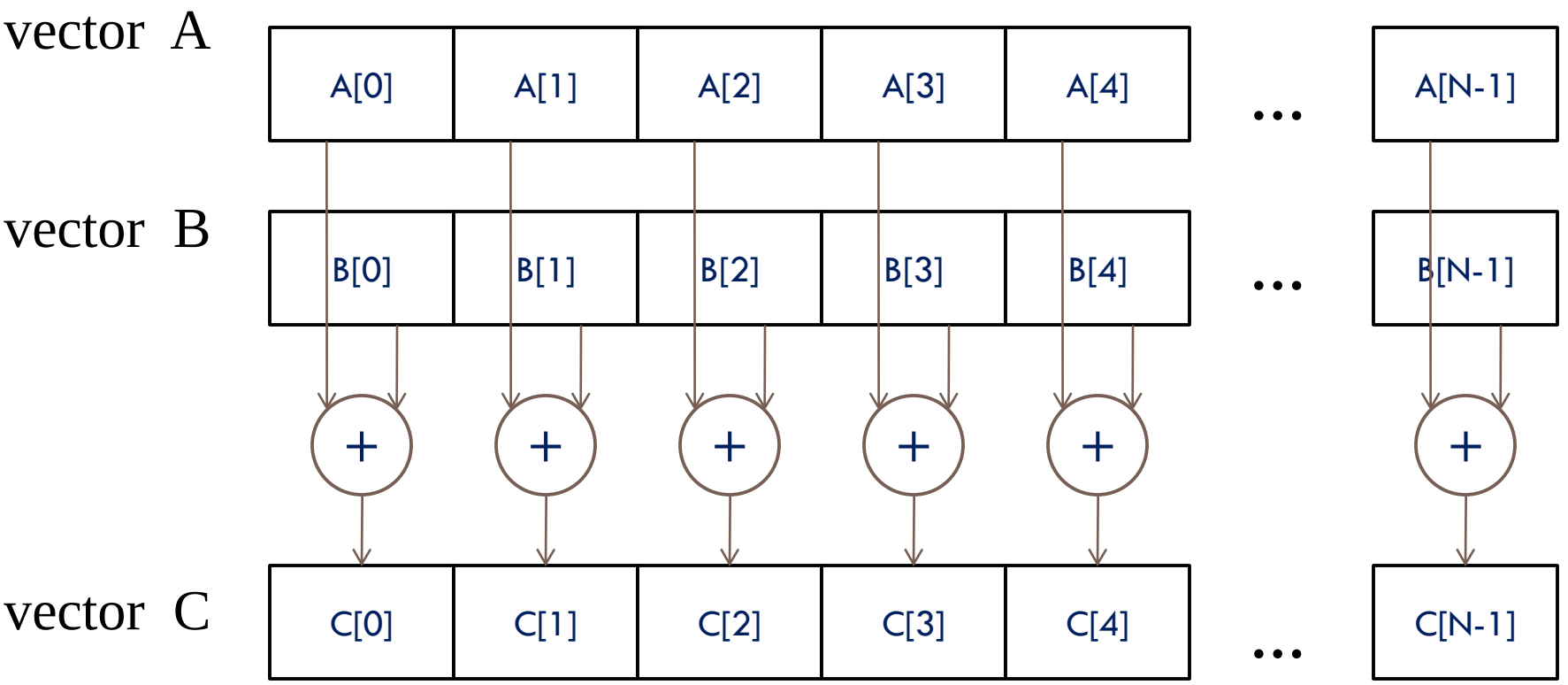



# Η ιεραρχία της μνήμης από την σκοπιά του προγραμματιστή

- Κάθε νήμα μπορεί να:
  - Διαβάσει/Γράψει σε **καταχωρητές** που ανήκουν στο νήμα (per thread registers) (~1 κύκλος)
  - Διαβάσει/Γράψει σε **κοινή μνήμη** που ανήκει στο block (per-block shared memory) (~5 κύκλοι)
  - Διαβάσει/Γράψει σε **καθολική μνήμη** που ανήκει στο πλέγμα (per-grid global memory) (~500 κύκλοι)
  - Διαβάσει από **constant μνήμη** που ανήκει στο πλέγμα (per-grid constant memory) (~5 κύκλοι αν υπάρχει στην κρυφή μνήμη)



# Πρόσθεση διανυσμάτων

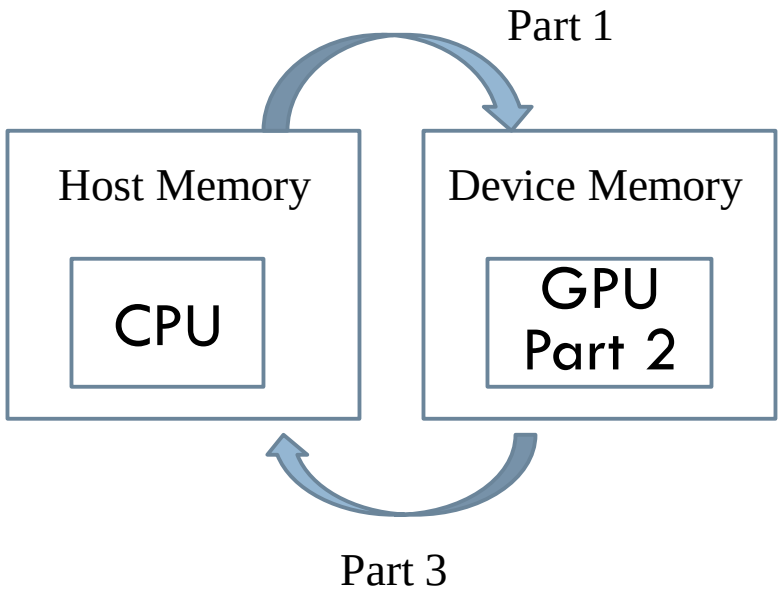


# Πρόσθεση διανυσμάτων – Κώδικας C

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

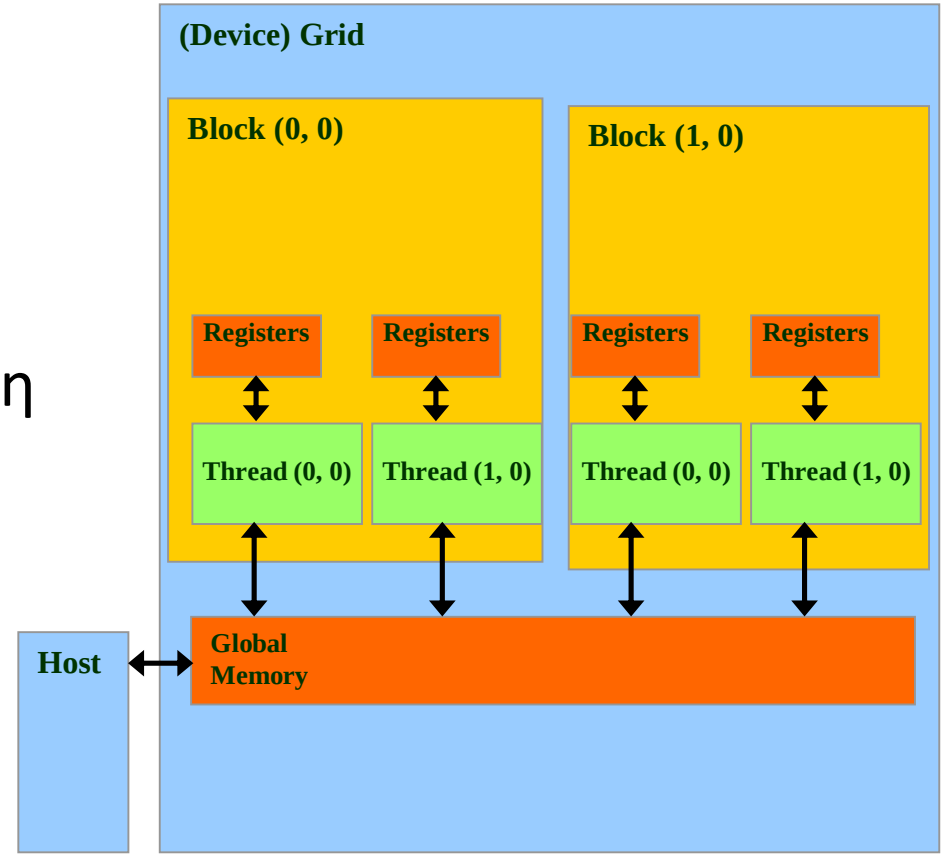
# Πρόσθεση διανυσμάτων σε ετερογενές σύστημα – Κώδικας στον host

```
#include <cuda.h>
void vecAdd(float* A, float* B, float* C, int n)
{
   int size = n* sizeof(float);
   float* A_d, B_d, C_d;
   …
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 επισκόπηση της μνήμης στην CUDA

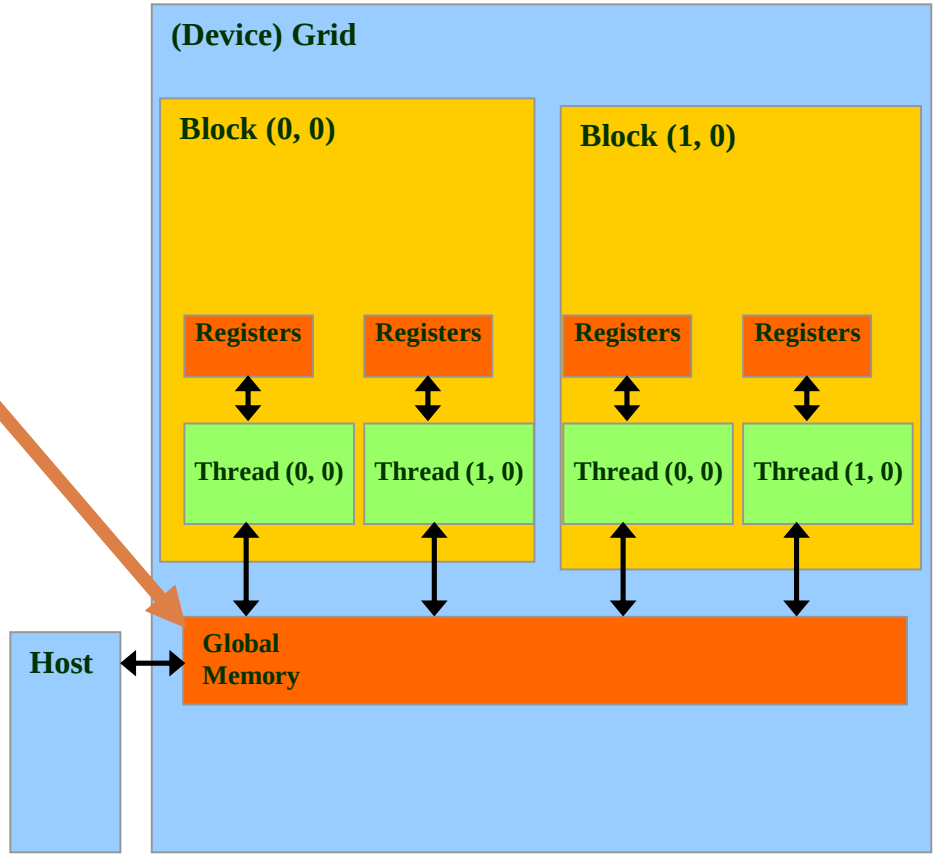
- Ο κώδικας στο device:
  - Διαβάζει/Γράφει καταχωρητές ανά νήμα
  - Διαβάζει/Γράφει καθολική (global) μνήμη ανά πλέγμα
- Ο κώδικας στον host:
  - Μεταφέρει δεδομένα από/προς την καθολική μνήμη ανά πλέγμα




Θα δούμε περισσότερα

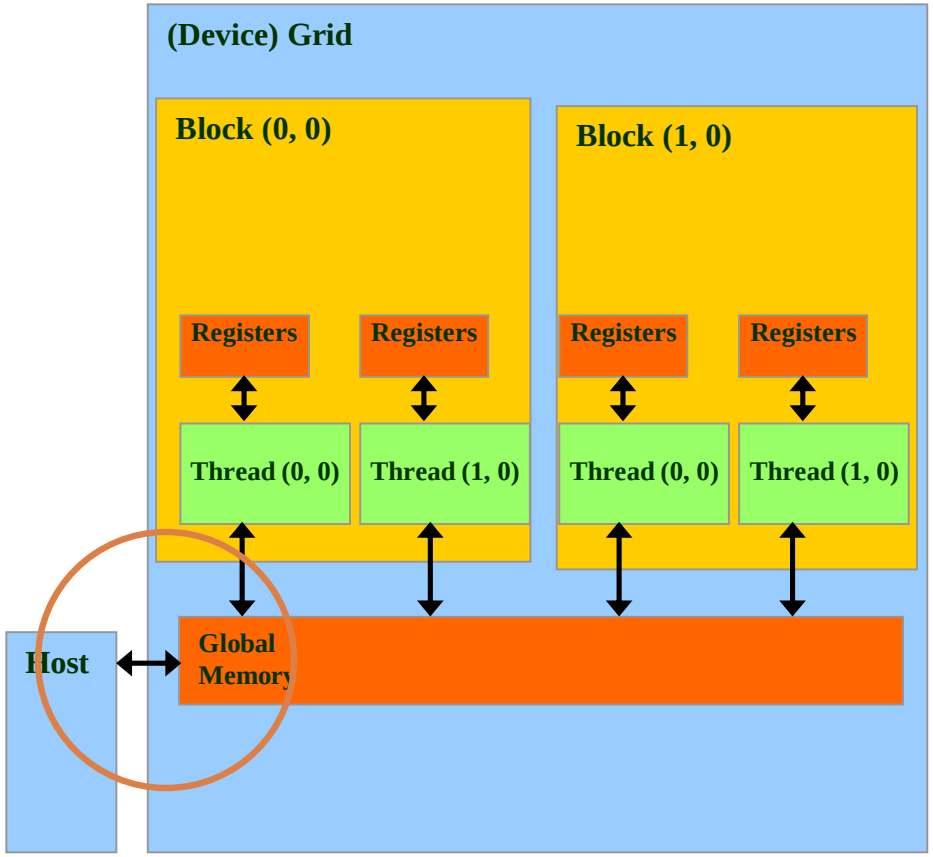
# Διεπαφή της CUDA για διαχείριση μνήμης του device

- cudaMalloc()
  - Δεσμεύει αντικείμενο στην καθολική μνήμη του device
  - Δύο παράμετροι
    - Δείκτης προς το δεσμευμένο αντικείμενο
    - Μέγεθος αντικειμένου σε bytes
- cudaFree()
  - Ελευθερώνει αντικείμενο από την καθολική μνήμη του device
  - Μία παράμετρος
    - Δείκτης προς αντικείμενο



# Διεπαφή για μεταφορά δεδομένων

- cudaMemcpy()
  - Μεταφορά δεδομένων
  - 4 παράμετροι
    - Δείκτης στο αντικείμενο-προορισμός
    - Δείκτης στο αντικείμενο-πηγή
    - Πλήθος bytes
    - Κατεύθυνση μεταφοράς
  - Η μεταφορά προς το device είναι ασύγχρονη
    - Το πρόγραμμα στον host συνεχίζει άμεσα την εκτέλεση του



```
void vecAdd(float* A, float* B, float* C, int n)
{
    int size = n * sizeof(float);
    float* A_d, B_d, C_d;

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

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

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

# Παράδειγμα: Πρόσθεση διανυσμάτων

Device Code

```
// 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];
}

int vectAdd(float* A, float* B, float* C, int n)
{
    // A_d, B_d, C_d allocations and copies omitted
    // Run ceil(n/256) blocks of 256 threads each
    vecAddKernel<<<ceil(n/256), 256>>>(A_d, B_d, C_d, n);
}
```

# Παράδειγμα: Πρόσθεση διανυσμάτων

```
// 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 vectAdd(float* A, float* B, float* C, int n)
{
    // A_d, B_d, C_d allocations and copies omitted
    // Run ceil(n/256) blocks of 256 threads each
    vecAddKernel<<<ceil(n/256), 256>>>(A_d, B_d, C_d, n);
}
```

# More on Kernel Launch

Host Code

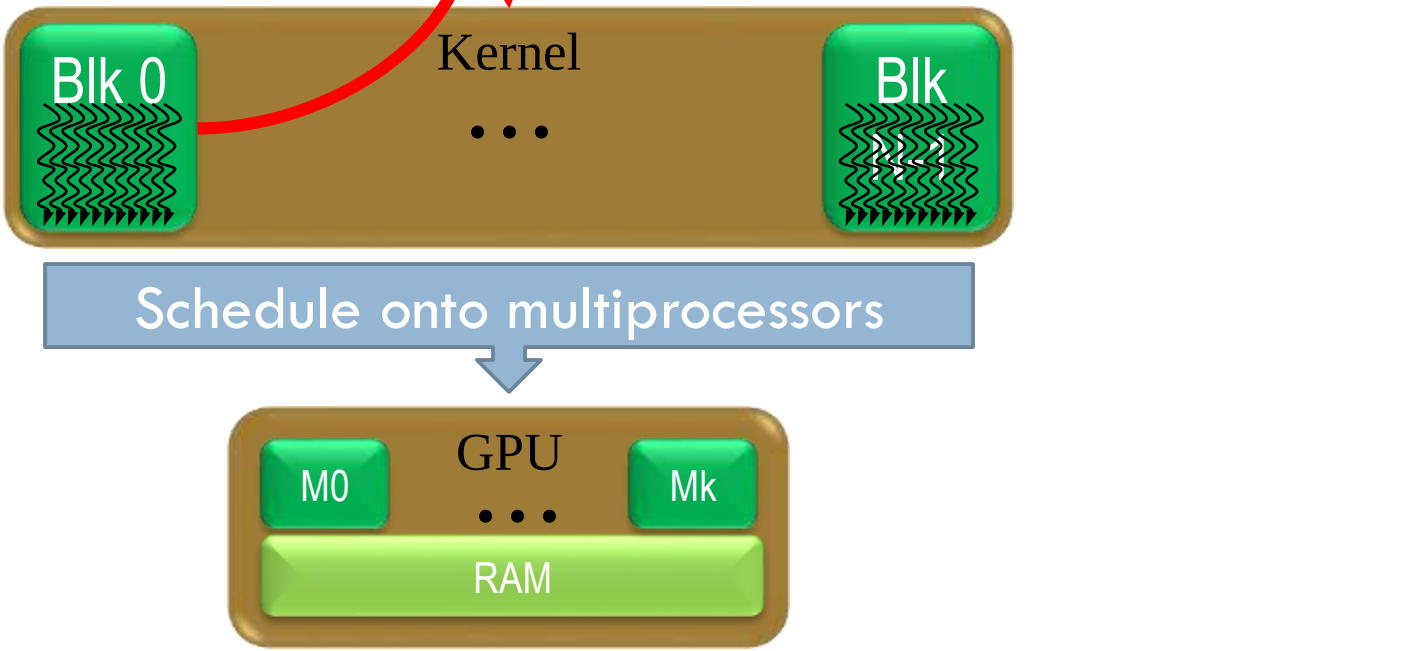
```
int vectAdd(float* A, float* B, float* C, int n)
{
    // A_d, B_d, C_d allocations and copies omitted
    // Run ceil(n/256) blocks of 256 threads each
    dim3 DimGrid(n/256, 1, 1);
    if (n % 256) DimGrid.x++;
    dim3 DimBlock(256, 1, 1);

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

- Η κλήση σε μια συνάρτηση πυρήνα είναι ασύγχρονη από την CUDA 1.0 και μετά
  - Απαιτείται συγχρονισμός αν χρειάζεται σύγχρονη εκτέλεση

# Επισκόπηση εκτέλεσης συνάρτησης πυρήνα

```
__host__
void vecAdd()
{
  dim3 DimGrid = (ceil(n/256,1,1);
  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];
}
```



# Δήλωση συναρτήσεων στην CUDA

- __global__ ορίζει μια συνάρτηση πυρήνα
  - Κάθε "__" αποτελείται από δύο underscore
  - Μια συνάρτηση πυρήνα πρέπει να επιστρέφει void
- __device__ και __host__ μπορούν να χρησιμοποιηθούν μαζί

| | Executed on the: | Only callable from the: |
|---|---|---|
| `__device__ float DeviceFunc()` | device | device |
| `__global__ void KernelFunc()` | device | host |
| `__host__ float HostFunc()` | host | host |

# Μετάφραση προγραμμάτων CUDA