Συστήματα Παράλληλης & Κατανεμημένης Επεξεργασίας

Ενότητα: Η γλώσσα OpenCL

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
http://arch.icte.uowm.gr/mdasyg
Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοπός της Ενότητας

- Η εισαγωγική παρουσίαση στην γλώσσα προγραμματισμού OpenCL για την εκμετάλλευση της παραλληλίας.

Τι είναι η OpenCL ;

- Open Computing Language.
- Γλώσσα προγραμματισμού για παράλληλη επεξεργασία.
- Είναι συμβατή με πολλών ειδών επεξεργαστές (Open) π.χ. GPU, CPU, DSP κ.α.

Συμβατότητα

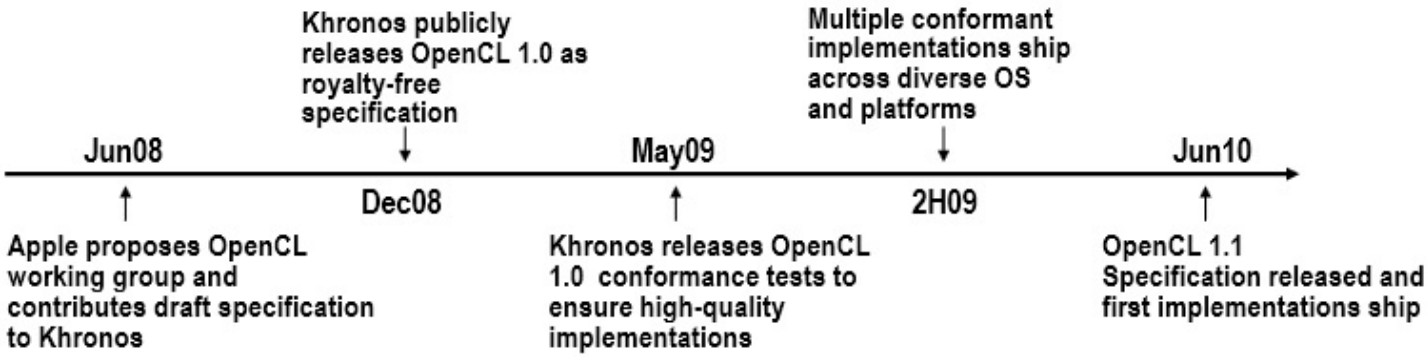
- Απαραίτητη προϋπόθεση είναι η συσκευή να υποστηρίζει OpenCL.
- Μπορεί να χρησιμοποιηθεί από πολλές συσκευές ανεξαρτήτως εταιρίας πχ. NVIDIA, AMD.
- Είναι βασισμένη σε μια παλιά έκδοση της C.

Ιστορικά στοιχεία (1/2)

- Αναπτύχθηκε από την Apple.
- Η Apple απευθύνθηκε στην Khronos Group για την δημιουργία της OpenCL.
- Στις 18 Νοεμβρίου 2008 ολοκληρώθηκε η υλοποίηση.
- Στις 8 Δεκεμβρίου 2008 κυκλοφόρησε η πρώτη έκδοση OpenCL 1.0.

Ιστορικά στοιχεία (2/2)

- Στις 9 Μαΐου η Khronos άρχισε τα τεστ της OpenCL 1.0.
- Στις 10 Ιουνίου κυκλοφόρησε η OpenCL 1.1.



Ετερογένεια

- Η OpenCL λειτουργεί σε πολλών τύπων hardware.
- Αυτό είναι το βασικό της πλεονέκτημα.
- Ένα πρόγραμμα σε OpenCL μπορεί να χρησιμοποιήσει όλο το υλικό της πλατφόρμας στην οποία τρέχει.

Βασική Ιδέα

- Η OpenCL χρησιμοποιεί έναν host ο οποίος εκτελεί το πρόγραμμα που γράφουμε σε γλώσσα C++, C# κ.α.
- Οι συσκευές (CPUs, GPUs, DSPs κ.α.) εκτελούν τον κώδικα OpenCL.
- Υπάρχει ξεχωριστός compiler για OpenCL.

Kernel

- Κώδικας OpenCL.
- Μία συνάρτηση που εκτελείται στις συσκευές.
- Ο host μπορεί να καλέσει μόνο kernel.

```
kernel void
dp_mul(global const float *a,
       global const float *b,
       global float *c)
{
  int id = get_global_id(0);

  c[id] = a[id] * b[id];

}
```

SIMT

- Single Instruction Multiple Thread.
- Δείχνει πως οι εντολές εκτελούνται στον host.
- Ο ίδιος κώδικας εκτελείται πολλές φορές σε διαφορετικό νήμα.
- Κάθε νήμα εκτελεί τον κώδικα με διαφορετικά δεδομένα.

Work Items

- Είναι παρόμοια με τα threads της CUDA.
- Οι μικρότερες μονάδες εκτέλεσης.
- Κάθε φορά που τρέχει ο kernel τρέχουν και πολλά work items.
- Το κάθε ένα τρέχει τον ίδιο κώδικα με διαφορετικά δεδομένα.
- Ο χρήστης μπορεί να επιλέξει τον αριθμό των work items.
- Το κάθε work item έχει το δικό του ID.

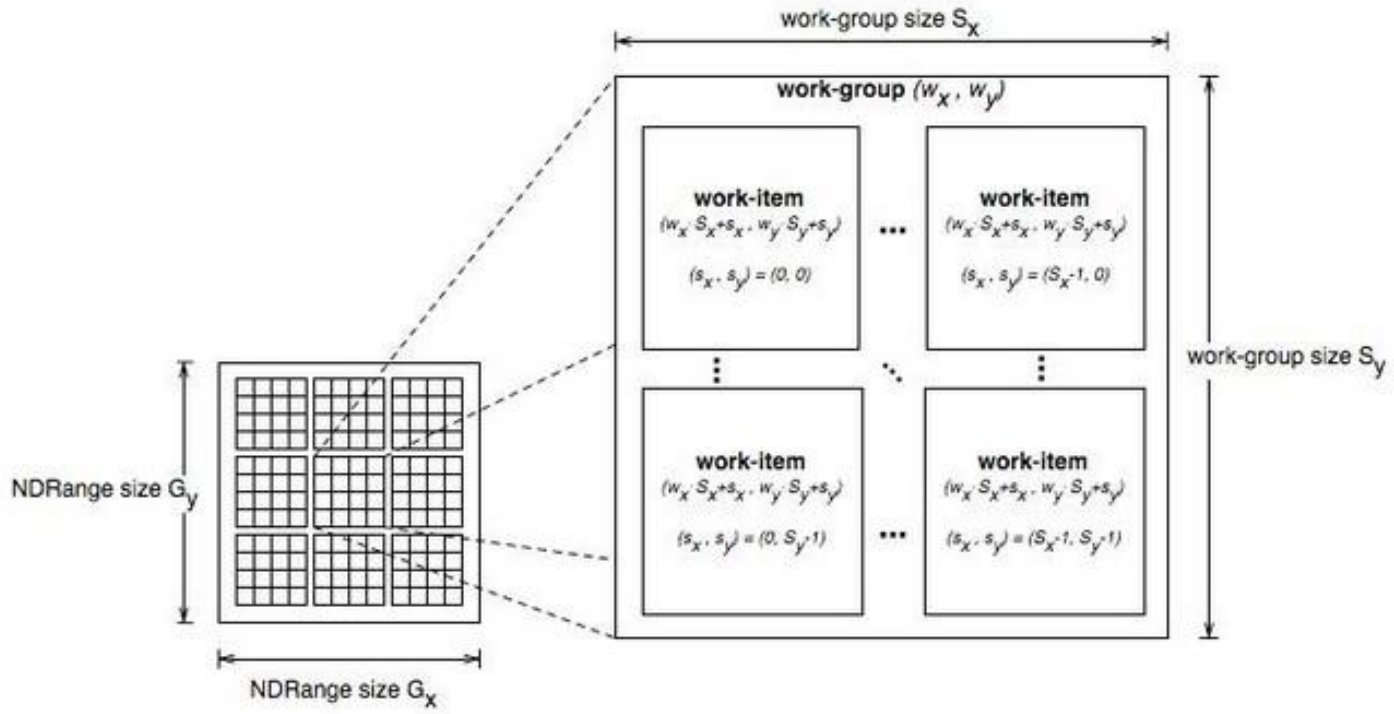
Work Groups

- Επιτρέπουν την επικοινωνία και την συνεργασία μεταξύ work items.
- Δείχνουν το πώς είναι οργανωμένα τα work items.
- Η οργάνωση είναι ένα N-διάστατο πλέγμα από work groups (N=1,2,3).
- Είναι παρόμοια με τα thread blocks της CUDA.
- Το κάθε work group έχει μοναδικό ID.

NDRange (1/2)

- Επίπεδο οργάνωσης.
- Δείχνει το πώς οργανώνονται τα work groups.
- N-διάστατο πλέγμα από work groups με N=1, 2, 3.

NDRange (2/2)

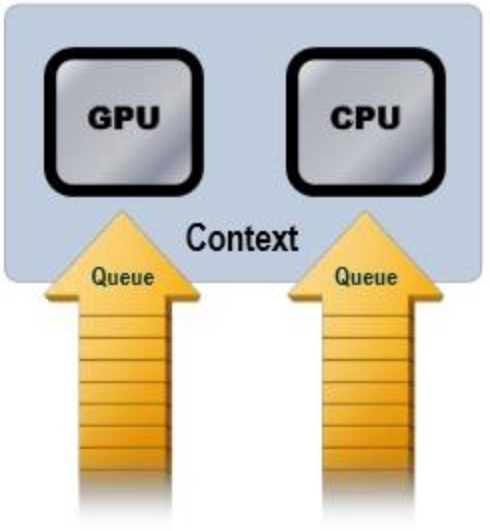

work-group size S_x
work-group (w_x, w_y)
work-item
$(w_x \cdot S_x+s_x, w_y \cdot S_y+s_y)$
$(s_x, s_y) = (0, 0)$
work-item
$(w_x \cdot S_x+s_x, w_y \cdot S_y+s_y)$
$(s_x, s_y) = (S_x-1, 0)$
work-item
$(w_x \cdot S_x+s_x, w_y \cdot S_y+s_y)$
$(s_x, s_y) = (0, S_y-1)$
work-item
$(w_x \cdot S_x+s_x, w_y \cdot S_y+s_y)$
$(s_x, s_y) = (S_x-1, S_y-1)$
work-group size S_y
NDRange size G_y
NDRange size G_x

Σύνταξη Kernel

- Η λέξη kernel είναι reserved.
- Οι συναρτήσεις kernel είναι σχεδόν πάντα void.
- Η λέξη global δείχνει πού βρίσκεται η μνήμη.
- Όλα τα αρχεία kernel έχουν κατάληξη .cl.
- Τα αρχεία .cl μπορούν να έχουν μόνο κώδικα OpenCL.

Χρήσιμα στοιχεία για τον Host

- Πλατφόρμα.
- Συσκευή.
- Γενικό Πλαίσιο (Context).
- Ουρά Εντολών.



Πλατφόρμα

- Είναι ο Host μαζί με μία συλλογή συσκευών.
- Χειρίζεται από το πλαίσιο OpenCL που επιτρέπει σε μία εφαρμογή να μοιράζεται πόρους και να τρέχει kernels σε συσκευές στην πλατφόρμα.
- Αντιπροσωπεύονται από το αντικείμενο cl_platform.

Συσκευή (1/2)

- Λόγω ετερογένειας μπορεί να είναι CPU, GPU, DSP, Accelerator, κ.α.
- Εκτελεί τον Kernel.
- Αντιπροσωπεύεται από το αντικείμενο cl_device.

Συσκευή (2/2)

- Κάθε συσκευή έχει τα δικά της χαρακτηριστικά.
- Δεν είναι όλα τα προγράμματα OpenCL συμβατά με όλες τις συσκευές.
- Όταν υπάρχει θέμα συμβατότητας μπορεί ο κώδικας να μην τρέχει βέλτιστα ή και καθόλου.
- Ο κώδικας του host μένει ίδιος για όλες τις συσκευές.

Γενικό Πλαίσιο (Context)

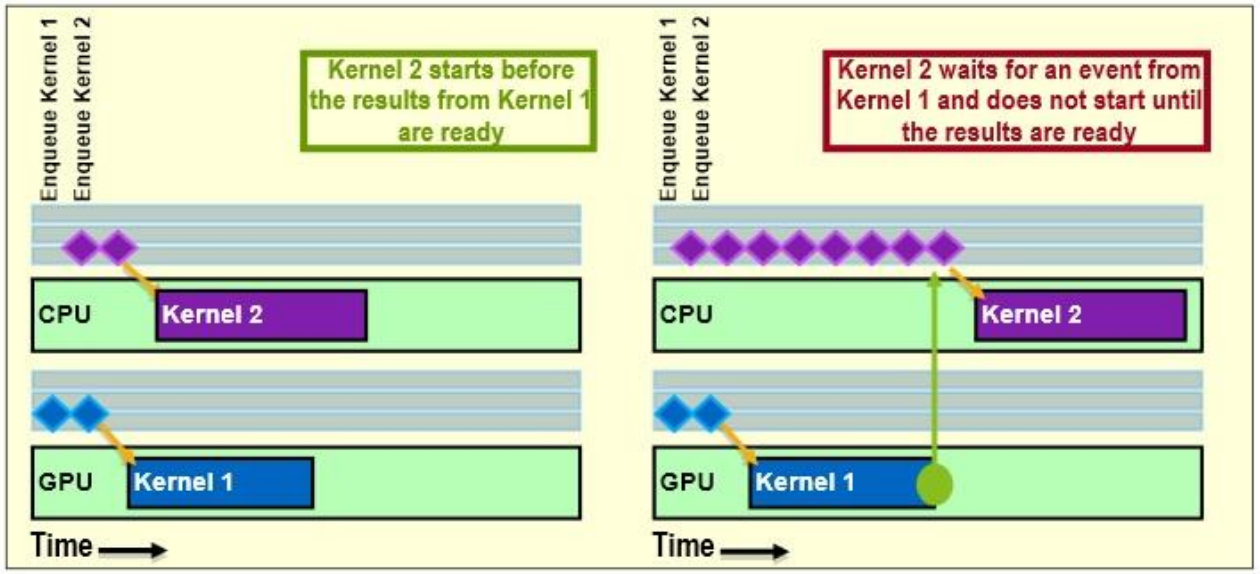
- Ορίζει το περιβάλλον της OpenCL.
- Περιβάλλον είναι οι kernels, οι συσκευές, η διαχείριση μνήμης, οι ουρές εντολών, κ.α.
- Αντιπροσωπεύεται από το αντικείμενο cl_context .

Ουρά Εντολών (1/2)

- Οι εντολές OpenCL μπαίνουν σε μία ουρά.
- Η συσκευή εκτελεί τις εντολές που είναι στην ουρά εντολών.
- Δημιουργείται σε μία συγκεκριμένη συσκευή.
- Μπορούμε να έχουμε πάνω από μία ουρές για ανεξάρτητες εντολές.
- Αντιπροσωπεύεται από το αντικείμενο cl_command_queue.

Ουρά Εντολών (2/2)

- Μπορούν να χρησιμοποιηθούν events για να συγχρονιστούν οι εκτελέσεις των kernel μεταξύ ουρών.



Προγραμματισμός Host

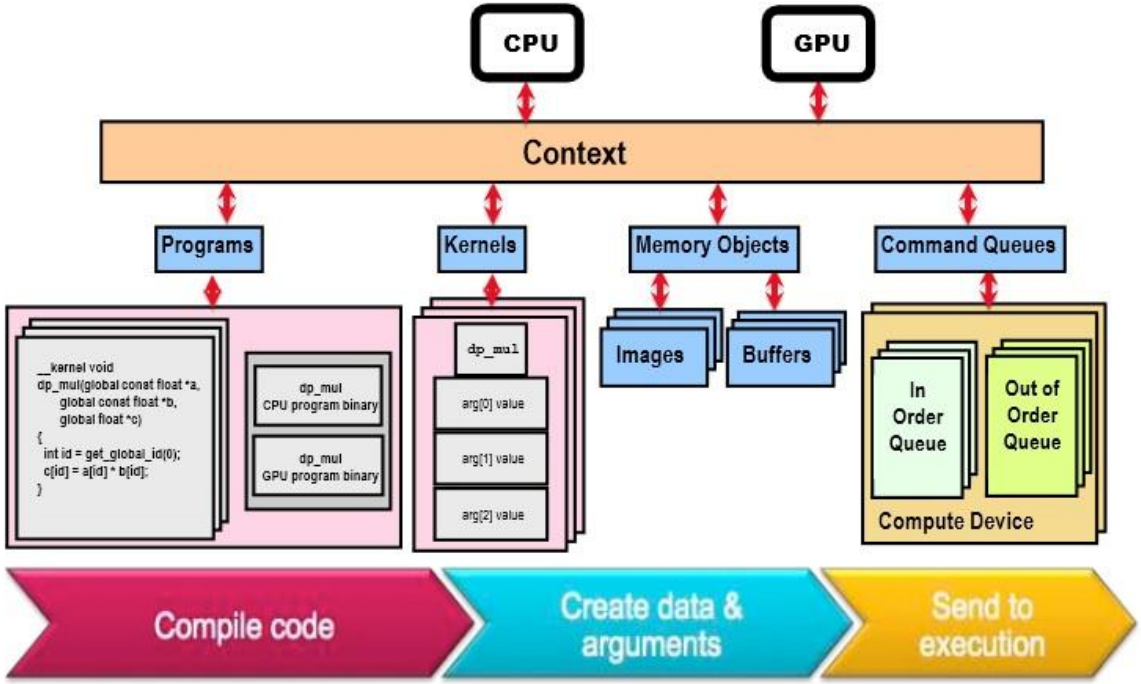
- Παράδειγμα:
  - Όπως βλέπουμε στον προγραμματισμό του host χρησιμοποιούνται τα αντικείμενα Πλατφόρμα, Συσκευή, Context, Ουρά Εντολών.

```
cl_int error = 0;   // Used to handle error codes
cl_platform_id platform;
cl_context context;
cl_command_queue queue;
cl_device_id device;

// Platform
error = oclGetPlatformID(&platform);
if (error != CL_SUCCESS) {
   cout << "Error getting platform id: " << errorMessage(error) << endl;
   exit(error);
}
// Device
error = clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
if (err != CL_SUCCESS) {
   cout << "Error getting device ids: " << errorMessage(error) << endl;
   exit(error);
}
// Context
context = clCreateContext(0, 1, &device, NULL, NULL, &error);
if (error != CL_SUCCESS) {
   cout << "Error creating context: " << errorMessage(error) << endl;
   exit(error);
}
// Command-queue
queue = clCreateCommandQueue(context, device, 0, &error);
if (error != CL_SUCCESS) {
   cout << "Error creating command queue: " << errorMessage(error) << endl;
   exit(error);
}
```

Σύνοψη

- Τα αντικείμενα της OpenCL και ο τρόπος που συνδέονται.

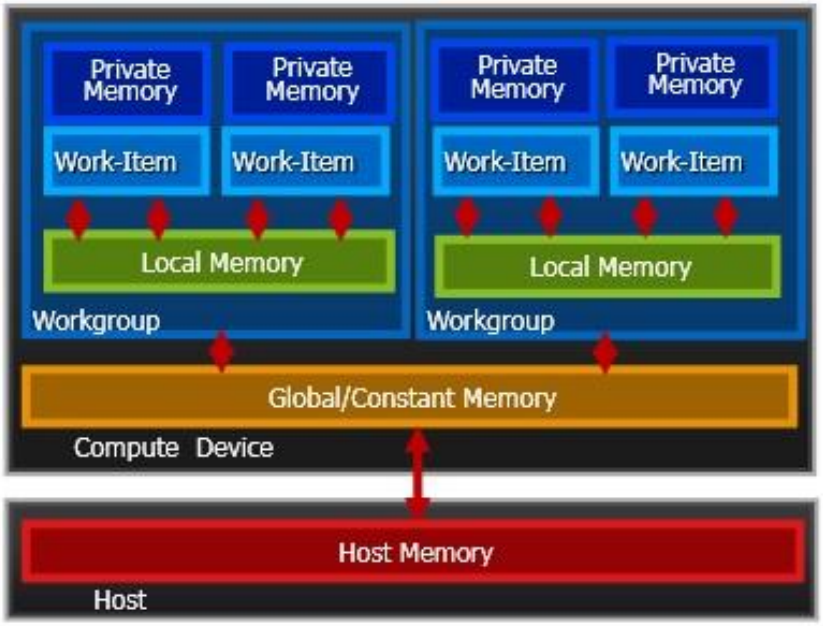


Μνήμη (1/2)

- Private Memory:
  - Ανά work item.
- Local Memory:
  - Μοιράζεται στο work group.
- Global Memory:
  - Ορατή σε όλα τα work groups.
- Constant Memory:
  - read only.
- Host Memory:
  - Στην CPU.

Μνήμη (2/2)

- Η ιεραρχία της μνήμης στην OpenCL φαίνεται στο διπλανό σχήμα.
- Όσο χαμηλότερα, τόσο μεγαλύτερη και αργότερη μνήμη.



Το Πρόγραμμα

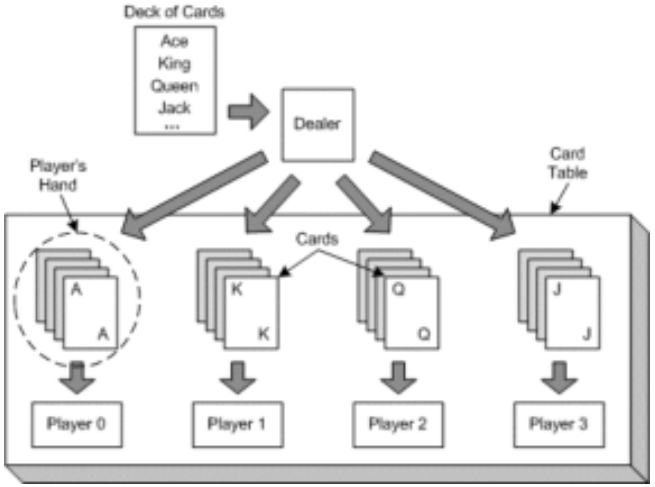
- Αποτελείται από ένα σύνολο από kernels , συναρτήσεις και δηλώσεις.
- Αντιπροσωπεύεται από το αντικείμενο cl_program.
- Στο πρόγραμμα πρέπει να αναφέρουμε ποια αρχεία αποτελούν το πρόγραμμα.

Παιχνίδι με τράπουλα (1/2)

- Ο dealer μοιράζει τα χαρτιά στους παίκτες.
- Ο κάθε παίκτης βλέπει τα χαρτιά του και αποφασίζει με ποια στρατηγική είναι καλύτερο να παίξει.
- Οι παίκτες δεν μπορούν να ανταλλάξουν χαρτιά ούτε να δουν τι χαρτιά έχουν οι άλλοι παίκτες.
- Οι παίκτες μπορούν να ζητήσουν επιπλέον χαρτιά από τον dealer.
- Ο dealer πραγματοποιεί την επιθυμία τους.

Παιχνίδι με τράπουλα (2/2)

- Η εικόνα δεξιά μας δείχνει το παιχνίδι με τράπουλα.
- Οι παίκτες που κάθονται στο τραπέζι του παιχνιδιού δεν είναι υποχρεωμένοι να παίξουν.
- Μπορούν να συμμετέχουν στο παιχνίδι μόνο όσοι κάθονται στο τραπέζι.



OpenCL και τράπουλα (1/4)

- Ο host είναι ο dealer.
- Η συσκευή είναι ο παίκτης.
- Οι kernels είναι τα χαρτιά.
- Το πρόγραμμα είναι η στοίβα με τα χαρτιά.
- Η ουρά εντολών είναι τα χαρτιά που έχει ο κάθε παίκτης.
- Το context είναι το τραπέζι.

OpenCL και τράπουλα (2/4)

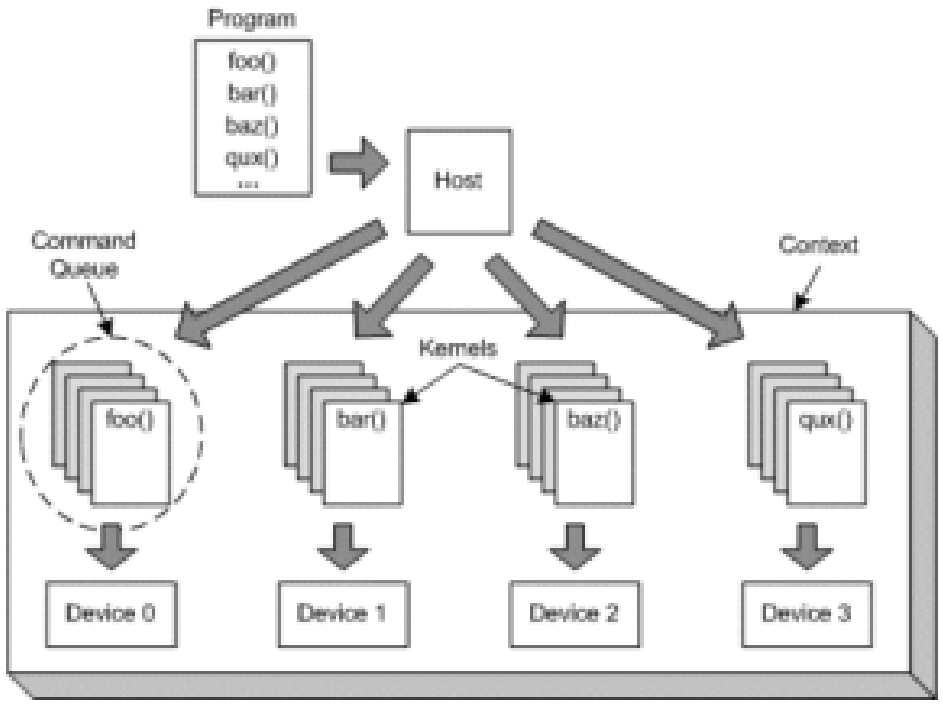
- Συσκευή - Παίκτης:
  - Ο παίκτης παίρνει κάρτες από τον dealer.
  - Η συσκευή παίρνει kernels από τον host.
- Kernels - Κάρτες:
  - Ο dealer μοιράζει κάρτες στους παίκτες.
  - Ο host στέλνει kernels στις συσκευές.
- Πρόγραμμα - Στοίβα χαρτιών:
  - Ο dealer παίρνει χαρτιά από την στοίβα.
  - Ο host παίρνει kernels από το πρόγραμμα.

OpenCL και τράπουλα (3/4)

- Ουρά εντολών - χαρτιά του παίκτη.
- Ο κάθε παίκτης συγκεντρώνει το κάθε χαρτί που λαμβάνει στα υπόλοιπα.
- Η κάθε συσκευή συγκεντρώνει τους kernels στην ουρά εντολών.
- Context – Τραπέζι.
- Το τραπέζι επιτρέπει στους παίκτες να διακινούν τα χαρτιά.
- Το context επιτρέπει στις συσκευές να λαμβάνουν kernels και να διακινούν δεδομένα.

OpenCL και τράπουλα (4/4)

- Στο σχήμα δεξιά βλέπουμε πώς λειτουργούν οι πέντε παραπάνω δομές.
- Το πρόγραμμα περιέχει πολλές συναρτήσεις.



Αντιθέσεις στην αναλογία (1/2)

- Στο παράδειγμά μας δεν υπάρχει η έννοια της πλατφόρμας.
- Ο dealer δεν επιλέγει ποιοι παίκτες θα καθίσουν στο τραπέζι. Ο host επιλέγει ποιες συσκευές θα τοποθετηθούν στο context.
- Ο dealer δεν μπορεί να δώσει την ίδια κάρτα σε δύο και πάνω παίκτες. Ο dealer μπορεί να δώσει τον ίδιο kernel στις ουρές εντολών από πάνω από μία συσκευές.

Αντιθέσεις στην αναλογία (2/2)

- Δεν αναφέρεται πώς οι συσκευές εκτελούν τους kernels. Οι συσκευές περιέχουν στοιχεία επεξεργασίας τα οποία επεξεργάζονται κάποια από τα δεδομένα.
- Ο dealer μοιράζει τα χαρτιά κυκλικά. Ο host δεν έχει περιορισμό στον τρόπο που μοιράζει kernels.

Σχολείο και ασκήσεις (1/7)

- Το σχολείο περιέχει αίθουσες.
- Οι αίθουσες φιλοξενούν τάξεις.
- Οι τάξεις έχουν μαθητές.
- Οι μαθητές έχουν να λύσουν προβλήματα μαθηματικών.
- Στις αίθουσες υπάρχουν πίνακες.
- Το σχολείο έχει έναν κεντρικό πίνακα.

Σχολείο και ασκήσεις (2/7)

- Ας υποθέσουμε πως:
  - Οι μαθητές έχουν ένα πρόβλημα μαθηματικών.
  - Ο κάθε ένας οφείλει να λύσει το πρόβλημα με διαφορετικούς αριθμούς.
  - Αφού το λύσει πρέπει να το παρουσιάσει στον πίνακα.
  - Οι τελικές απαντήσεις αναρτώνται στον κεντρικό πίνακα.
  - Όταν αναρτηθούν, οι μαθητές σχολούν.

Σχολείο και ασκήσεις (3/7)

- Οι μαθητές μίας τάξης χρησιμοποιούν τον ίδιο πίνακα αλλά έχουν ξεχωριστά τετράδια.
- Οι μαθητές μίας τάξης δεν γνωρίζουν πότε σχολούν οι μαθητές μίας άλλης τάξης.
- Όταν τελειώσει μία τάξη, μία άλλη μπορεί να χρησιμοποιήσει την αίθουσα.

Σχολείο και ασκήσεις (4/7)

- Οι μαθητές του σχολείου έχουν να λύσουν το ίδιο πρόβλημα με διαφορετικές τιμές.
- Οι τιμές για τον κάθε μαθητή είναι ανακοινωμένες στον κεντρικό πίνακα του σχολείου.
- Στην αρχή της σχολικής μέρας οι μαθητές πηγαίνουν να βρουν τους αριθμούς που τους αντιστοιχούν.

Σχολείο και ασκήσεις (5/7)

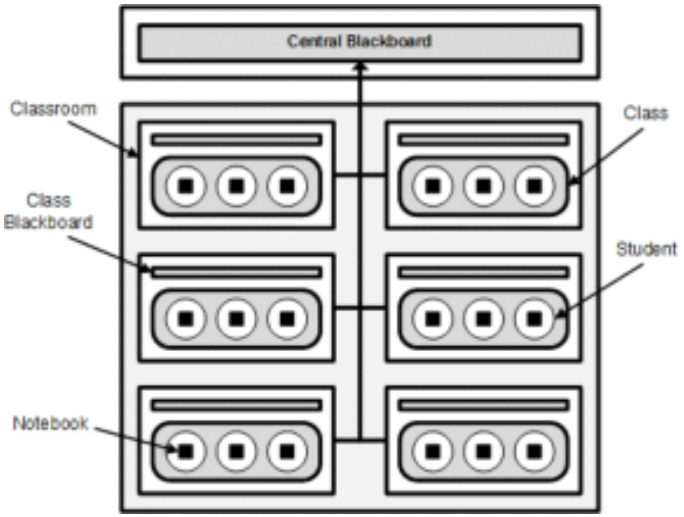
- Η διαδικασία εύρεσης των αριθμών από τον κεντρικό πίνακα είναι εξαιρετικά χρονοβόρα λόγω του μεγάλου διαδρόμου.
- Οι πίνακες στις αίθουσες είναι αρχικά κενοί και πολύ μικρότεροι από τον κεντρικό.
- Μόλις ο κάθε μαθητής λύσει το πρόβλημά του στο τετράδιό του, το παρουσιάζει στον πίνακα της τάξης.
- Στην συνέχεια γράφει τα αποτελέσματα στον κεντρικό πίνακα.

Σχολείο και ασκήσεις (6/7)

- Ο κάθε μαθητής πηγαίνει στον κεντρικό πίνακα δύο φορές:
  - Για να πάρει τα δεδομένα.
  - Για να γράψει τις απαντήσεις.
- Η κάθε τάξη έχει έναν μοναδικό αριθμό.
- Ο κάθε μαθητής έχει δύο αριθμούς:
  - Έναν μοναδικό ανάμεσα στους άλλους μαθητές της ίδιας τάξης.
  - Έναν που προσδιορίζει την τάξη.

Σχολείο και ασκήσεις (7/7)

- Η εικόνα δεξιά δείχνει την σχέση των:
  - Κύριος πίνακας.
  - Αίθουσα.
  - Τάξη.
  - Πίνακας τάξης.
  - Μαθητής.
  - Τετράδιο.

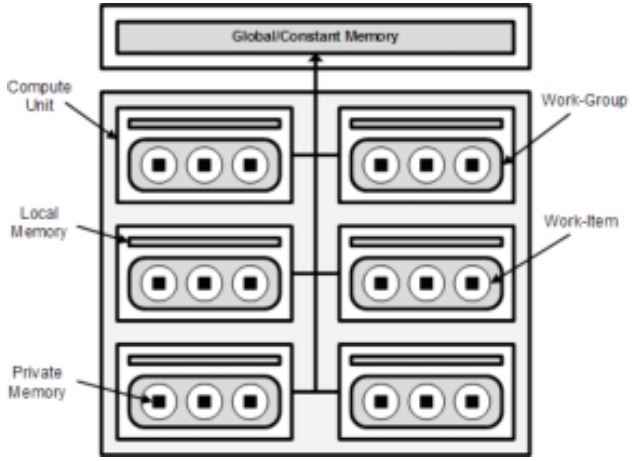


Σχολείο και Συσκευές (1/5)

- Το σχολείο είναι η συσκευή OpenCL.
- Το πρόβλημα μαθηματικών είναι ο kernel.
- Ο μαθητής είναι ένα work-item.
- Η τάξη είναι ένα work-group.
- Η αίθουσα είναι ένα αντικείμενο επεξεργασίας (πυρήνας επεξεργαστή).

Σχολείο και Συσκευές (2/5)

- Στο σχήμα δεξιά βλέπουμε την δομή της συσκευής.
- Προφανώς όπως μία αίθουσα μπορεί να φιλοξενήσει μία μόνο τάξη, ένα αντικείμενο επεξεργασίας μπορεί να φιλοξενήσει ένα μόνο work-group.



Σχολείο και Συσκευές (3/5)

- Ο κεντρικός πίνακας είναι η **Global Memory.**
- Είναι διαθέσιμη και στον host και στην συσκευή.
- Όταν ο host μεταφέρει δεδομένα στην συσκευή τα δεδομένα αποθηκεύονται στην Global Memory.
- Είναι η μεγαλύτερη μνήμη στην συσκευή.
- Όμως έχει τον μεγαλύτερο χρόνο προσπέλασης από τα work-items.

Σχολείο και Συσκευές (4/5)

- Ο πίνακας της τάξης είναι η **Local Memory.**
- Τα work-items μπορούν να την προσπελάσουν πιο γρήγορα από την Global (περίπου 100 φορές).
- Είναι επίσης μικρότερη από την Global.
- Επειδή είναι πολύ ταχύτερη όμως είναι ιδανική για την αποθήκευση των ενδιάμεσων δεδομένων των work-items.
- Τα work-items στο ίδιο work-group μπορούν να προσπελάσουν το ίδιο block της Local Memory.

Σχολείο και Συσκευές (5/5)

- Το τετράδιο του κάθε μαθητή είναι η **Private Memory.**
- Το κάθε work-item έχει την δική του Private Memory.
- Μπορεί να την προσπελάσει γρηγορότερα από τις Local, Global, Constant.
- Δεν υπάρχει όμως αρκετός χώρος στην Private Memory.
- Πρέπει να μην χρησιμοποιείται μεγάλη ποσότητα.

Compile

- Για το compile χρειαζόμαστε πάνω από ένα αρχεία.
- Συνήθως είναι δύο:
  - Το αρχείο Kernel.
  - Το αρχείο host.
  - Τα αρχεία αυτά αποτελούν το πρόγραμμα.
  - Οι kernels κάνουν compile όταν τρέχουν.

Το πρόγραμμα του host

- Ελέγχει την εκτέλεση των kernels.
- Μαζεύει πληροφορίες για τις διαθέσιμες συσκευές.
- Μεταφέρει δεδομένα στις συσκευές.
- Δημιουργεί αντικείμενα του kernel.
- Τρέχει τον kernel.
- Διαβάζει τα αντικείμενα της μνήμης.

__kernel ή kernel

- Τύπος συνάρτησης.
- Μπορεί να εκτελεστεί μόνο από την συσκευή.
- Δηλώνει ότι η συγκεκριμένη συνάρτηση είναι kernel.
- Μπορεί να κληθεί από τον host ή από κάποια άλλη kernel συνάρτηση.

__attribute__

- Είναι η βάση για τον υπολογισμό της χρησιμοποίησης bandwidth της CPU.
- vec_type_hint (<type*n*>):
  - Τύπος δεδομένων.
- ((work_group_size_hint(*X*, *Y*, *Z*))):
  - Μέγεθος του work group που μπορεί να χρησιμοποιηθεί.
- ((reqd_work_group_size(*X*, *Y*, *Z*))):
  - Μέγεθος του work group που μπορεί να χρησιμοποιηθεί ως local work size.

Παράδειγμα kernel

- Παράδειγμα με διαφορετικές επιλογές attribute σε κάθε kernel.

```
// autovectorize assuming float4 as the
// basic computation width
__kernel __attribute__((vec_type_hint(float4)))
void foo( __global float4 *p ) { ....

// autovectorize assuming double as the
// basic computation width
__kernel __attribute__((vec_type_hint(double)))
void foo( __global float4 *p ){ ....

// autovectorize assuming int (default)
// as the basic computation width
__kernel
void foo( __global float4 *p ){ ....
```

Δήλωση Kernel

- __Kernel NAME
  - Απλή δήλωση του kernel.
- __KernelGrpSz1 NAME (GROUP_SZ1)
  - Με στατικό μονοδιάστατο μέγεθος GROUP_SZ1.
- __KernelGrpSz2NAME (GROUP_SZ1, GROUP_SZ2)
  - Με στατικό δισδιάστατο μέγεθος.

Παράδειγμα

- __kernel void square(__global float* input, __global float* output, const unsigned int count).

- Το όνομα του kernel είναι square.
- Τα input, output και count είναι οι μεταβλητές που χρειάζεται η συνάρτηση.

Στοιχεία του Kernel (1/3)

- Ο kernel έχει κάποια στοιχεία/γνωρίσματα.
- Ορίζονται στο αρχείο kernel.
- Τα Arguments συντάσσονται μέσα στην δήλωση συνάρτησης kernel.
- Πρέπει να ταιριάζουν με τα στοιχεία που δίνει ο host στον kernel που θα δούμε παρακάτω.

Ορισμός Στοιχείων του Kernel (1/5)

- __Arg(TYPE, NAME)
  - Argument NAME τύπου TYPE.
- __ArgFirst(TYPE, NAME)
  - Το πρώτο argument σε μία λίστα με arguments.
- __ArgLast(TYPE, NAME)
  - Το τελευταίο argument σε μία λίστα με arguments.

Ορισμός Στοιχείων του Kernel (2/5)

- __ArgNULL
- Ο kernel δεν έχει arguments.
- __Local(TYPE, NAME, SIZE)
- Στατικός local πίνακας.
- __Return
- Η τελευταία δήλωση του kernel, με αυτήν την εντολή τερματίζει το αρχείο του kernel.

Ορισμός Στοιχείων του Kernel (3/5)

- Παράδειγμα _Local
- __Local(float4, localPos, 256);
- Στατικός πίνακας με γνωρίσματα:
  - Όνομα: localPos.
  - Τύπος: float4.
  - Μέγεθος: 256.

Ορισμός Στοιχείων του Kernel (4/5)

Παράδειγμα:

Ένας kernel χωρίς arguments.

```
__Kernel(hello)
__ArgNULL
{
...
__Return
}
```

Ορισμός Στοιχείων του Kernel (5/5)

- Παράδειγμα kernel με arguments:

__Kernel(nbody_sim) __ArgFirst(__global float4*, pos) __Arg(__global float4*, vel) __Arg(uint, numBodies) __Arg(float, deltaTime) __ArgLast(float, epsSqr) { …

Στοιχεία για τον Kernel (2/3)

- Εκτός από τα στοιχεία (Arguments) του kernel, πρέπει να ορίσουμε arguments για τον kernel.
- Τα ορίζουμε στο αρχείο του κώδικα του host.
- Ο ορισμός είναι ίδιος με αυτόν στο αρχείο του kernel.
- Τα στοιχεία που δίνουμε στον kernel πρέπει να ταιριάζουν με τα arguments στο αρχείο του kernel.

Στοιχεία για τον Kernel (3/3)

- Παράδειγμα πρώτου argument στο αρχείο host (δημιουργεί αντικείμενα μνήμης):

```
_ArgFirst(Args, CL_ARG_INPUTOUTPUT_PTR | CL_ARG_POPULATE_PTR,
        cl_float*,
        pos,
        numBodies*sizeof (cl_float4) );
```

Κλήση Kernel

- Στο αρχείο του host χρειάζεται να καλέσουμε τους kernels.
- Για να καλέσουμε έναν kernel χρειαζόμαστε μία ειδική συνάρτηση.
- Μία συνάρτηση είναι η CallCL.
- Είναι λιγότερο περίπλοκη από άλλες συναρτήσεις κλήσης.
- Είναι συμβατή για CPU, GPU και EMU_GPU κλήσεις.

Συνάρτηση CallCL (1/3)

```
int callCL(const char *_device_type,
    int _domainDim,
    int _domain[],
    int _group[],
    const char *_program_location,
const  char *_program,
    const char*_kernel_entry_name,
ClKrnlArg*_args = 0);
```

Συνάρτηση CallCL (2/3)

_device_type

- Η συσκευή (π.χ. “cpu”, “gpu”, “gpu_emu”).

_domainDim

- Η διάσταση του project μας.

_domain[]

- Το ολικό μέγεθος.

int _group[]

- Το local μέγεθος.

Συνάρτηση CallCL (3/3)

_program_location

- Η τοποθεσία του kernel στον υπολογιστή.

_program

- Το όνομα του αρχείου των kernels.

_kernel_entry_name

- Το ειδικό όνομα kernel.

_args

- Τα arguments που θα περάσουν στον kernel.

Barrier

- barrier (cl_mem _ fence_flags *flags*):
  - Όλα τα work-items σε ένα work-group που εκτελούν τον kernel πρέπει να εκτελέσουν αυτήν την συνάρτηση πριν κάποιο από αυτά συνεχίσει την εκτέλεση μετά το barrier.
  - Είναι χρήσιμο όταν τα work-items γράφουν στον buffer και θέλουν στην συνέχεια να διαβάσουν αυτό που έγραψαν.

Barrier Flags

- cl_mem _ fence_flags *flags:*
  - CLK_LOCAL_MEM_FENCE
    - Η συνάρτηση barrier θα κάνει flush όλες στις μεταβλητές στην local memory ή θα βάλει φράχτη μνήμης στην ουρά για να υπάρχει σωστή σειρά εκτελέσεων στην local memory.
  - CLK_GLOBAL_MEM_FENCE
    - Η συνάρτηση barrier θα βάλει φράχτη μνήμης στην ουρά για να υπάρχει σωστή σειρά εκτελέσεων στην global memory.

Βασικές Εντολές (1/9)

- get_global_id (uint *dimindx*)
  - Επιστρέφει το ID του work-item για την διάσταση dimindx.
  - Π.χ. get_global_id(0).
- get_local_size (uint *dimindx*)
  - Επιστρέφει τον αριθμό των local work- items.
  - Π.χ. get_local_size (0).

Βασικές Εντολές (2/9)

- clGetPlatformIDs
  - Επιστρέφει την λίστα των διαθέσιμων πλατφορμών.
- clGetDeviceIDs
  - Επιστρέφει την λίστα των διαθέσιμων συσκευών:
- clCreateContext
  - Δημιουργεί ένα γενικό πλαίσιο (context).

Βασικές Εντολές (3/9)

- clCreateCommandQueue
  - Δημιουργεί ουρά εντολών σε μία συγκεκριμένη συσκευή.
- clGetCommandQueueInfo
  - Προβάλλει πληροφορίες για την ουρά εντολών.
- clCreateBuffer
  - Δημιουργεί ένα αντικείμενο buffer.

Βασικές Εντολές (4/9)

- clEnqueueWriteBuffer
  - Βάζει στην ουρά εντολές για να γράψει σε έναν buffer από την μνήμη του host.
- clEnqueueCopyBuffer
  - Βάζει στην ουρά εντολές για να αντιγράψει έναν buffer από έναν άλλο buffer.
- clEnqueueReadBuffer
  - Βάζει στην ουρά εντολές για να διαβάσει δεδομένα από έναν buffer.

Βασικές Εντολές (5/9)

- clCreateProgramWithSource
  - Δημιουργεί ένα αντικείμενο προγράμματος για ένα context και φορτώνει τον πηγαίο κώδικα στο αρχείο προγράμματος.
- clBuildProgram
  - Δημιουργεί εκτελέσιμο ενός προγράμματος.

Βασικές Εντολές (6/9)

- clCreateKernel
  - Δημιουργεί ένα αντικείμενο kernel.
- clSetKernelArg
  - Δίνει την τιμή του argument για ένα συγκεκριμένο argument του kernel.
- clEnqueueNDRangeKernel
  - Βάζει σε σειρά εντολές για να εκτελεστεί ένας kernel σε μία συσκευή.

Βασικές Εντολές (7/9)

- clFlush
  - Εκδίδει τις εντολές σε ουρά σε μία ουρά εντολών στην κατάλληλη συσκευή.
- clFinish
  - Μπλοκάρει την εκτέλεση μέχρι να μπουν όλες οι εντολές στην ουρά που πρέπει και να ολοκληρωθεί η εκτέλεσή τους.

Βασικές Εντολές (8/9)

- clReleaseKernel
  - Μειώνει τον αριθμό των πυρήνων.
- clReleaseProgram
  - Μειώνει τον αριθμό των προγραμμάτων.
- clReleaseMemObject
  - Μειώνει τον αριθμό των αντικειμένων μνήμης.

Βασικές Εντολές (9/9)

- clReleaseCommandQueue
  - Μειώνει τον αριθμό των αντικειμένων ουράς εντολών.
- clReleaseContext
  - Μειώνει τον αριθμό των αντικειμένων γενικών πλαισίων.

Abstract Τύποι Δεδομένων (1/2)

- Στην OpenCL εισάγονται νέοι τύποι δεδομένων όπως οι παρακάτω:
  - _cl_platform_id
    - Το id της πλατφόρμας.
  - _cl_device_id
    - Το id της συσκευής.
  - _cl_context
    - Ένα context.

Abstract Τύποι Δεδομένων (2/2)

_cl_command_queue
- Μία ουρά εντολών.

_cl_mem
- Ένα αντικείμενο μνήμης.

_cl_program
- Ένα πρόγραμμα.

_cl_kernel
- Ένας kernel.

Πολυπύρηνοι Επεξεργαστές

- Χρησιμοποιούνται οι SIMD επεκτάσεις συνόλων εντολών.
  - (πχ x86 SSE και Power/VMX).
  - SIMD - Singe Instruction Multiple Data.
- Οι επεξεργαστές x86 κάνουν καλύτερη χρήση του SSE όταν οι kernels είναι τύπου float4.
- Οι υλοποιήσεις CPU συχνά χαρτογραφούν τους χώρους μνήμης μίας cache ώστε ένας kernel να χρησιμοποιεί Constant και Local Memory ταυτόχρονα.

Επεξεργαστές Cell (1/2)

- Η IBM κυκλοφόρησε μία εργαλειοθήκη OpenCL που υποστηρίζει Cell και Power επεξεργαστές σε πλατφόρμα Linux.
- Υποστηρίζει ενσωματωμένα προφίλ.
- Χρησιμοποιεί τεχνικές λογισμικού για να ομαλύνει κάποιες από τις διαφορές ανάμεσα σε Cell και συμβατικούς επεξεργαστές.

Επεξεργαστές Cell (2/2)

- Οι προσπελάσεις Global μνήμης αποδίδουν καλύτερα όταν οι τελεστές είναι πολλαπλάσιοι των 16 bytes.
  - Πχ float4.
- Η χρήση μεγαλύτερων τύπων διανυσμάτων αυξάνει περισσότερο την απόδοση βοηθώντας τον compiler με τους βρόχους.
  - Πχ float16.

Επεξεργαστές Γραφικών (1/4)

- Η OpenCL παρέχει APIs για την μεταφορά δεδομένων βασισμένη στην CPU μεταξύ ανεξαρτήτων host και συστημάτων μνήμης GPU.
- Οι νέες GPU παρέχουν πρόσβαση στην μνήμη του host μέσω των PCI-e.
- Κάποιες GPU επιτρέπουν την χαρτογράφηση στον χώρο διευθύνσεων του host παρέχοντας το κατάλληλο hardware για την υποστήριξη πρόσβασης σε αρχεία που διαβάζονται ή γράφονται μία φορά κατά την διάρκεια εκτέλεσης του kernel, χωρίς αντίγραφα.

Επεξεργαστές Γραφικών (2/4)

- Η NVIDIA και η AMD έχουν κυκλοφορήσει εφαρμογές OpenCL για τις GPU τους.
- Οι συσκευές αυτές απαιτούν μεγάλο αριθμό από work-items και work-groups για να καλύψουν το hardware και να κρύψουν την καθυστέρηση.

Επεξεργαστές Γραφικών (3/4)

- Η NVIDIA χρησιμοποιεί αρχιτεκτονική κλιμακωτού (scalar) επεξεργαστή.
- Δίνει την δυνατότητα να εργαστεί με υψηλή απόδοση στους περισσότερους τύπους δεδομένων OpenCL.
- Η AMD χρησιμοποιεί αρχιτεκτονική διανυσματικού (vector) επεξεργαστή.
- Έχουν υψηλή απόδοση όταν τα work-items λειτουργούν σε τύπους διανυσμάτων τεσσάρων στοιχείων (πχ. float4).

Επεξεργαστές Γραφικών (4/4)

- Ένας διανυσματικός kernel μπορεί να αποδώσει καλά σε x86 CPU, NVIDIA και AMD GPUs.
- Όμως ο kernel θα είναι λιγότερο ευανάγνωστος από τον αντίστοιχο scalar.
- Οι διαφορές στις αρχιτεκτονικές χαμηλού επιπέδου GPU περιλαμβάνουν διακυμάνσεις στην cache μνήμη και το ποια μοντέλα προσπέλασης μνήμης δημιουργούν συγκρούσεις που επηρεάζουν τον kernel.

OpenCL vs. CUDA (1/7)

- Μοναδική εταιρία που υποστηρίζει CUDA:
  - Nvidia.
- Εταιρίες που υποστηρίζουν OpenCL:
  - Nvidia.
  - AMD.
  - Apple.
  - Intel.
  - IBM.
  - Portable OpenCL.

OpenCL vs. CUDA (2/7)

Ορολογίες:

CUDA	**OpenCL**
• GPU	Device
• Multiprocessor	Compute Unit
• Scalar Core	Processing Element
• Global Memory	Global Memory
• Shared Memory	Local Memory
• Local Memory	Private Memory
• Kernel	Program
• Block	Work-Group
• Thread	Work-Item

OpenCL vs. CUDA (3/7)

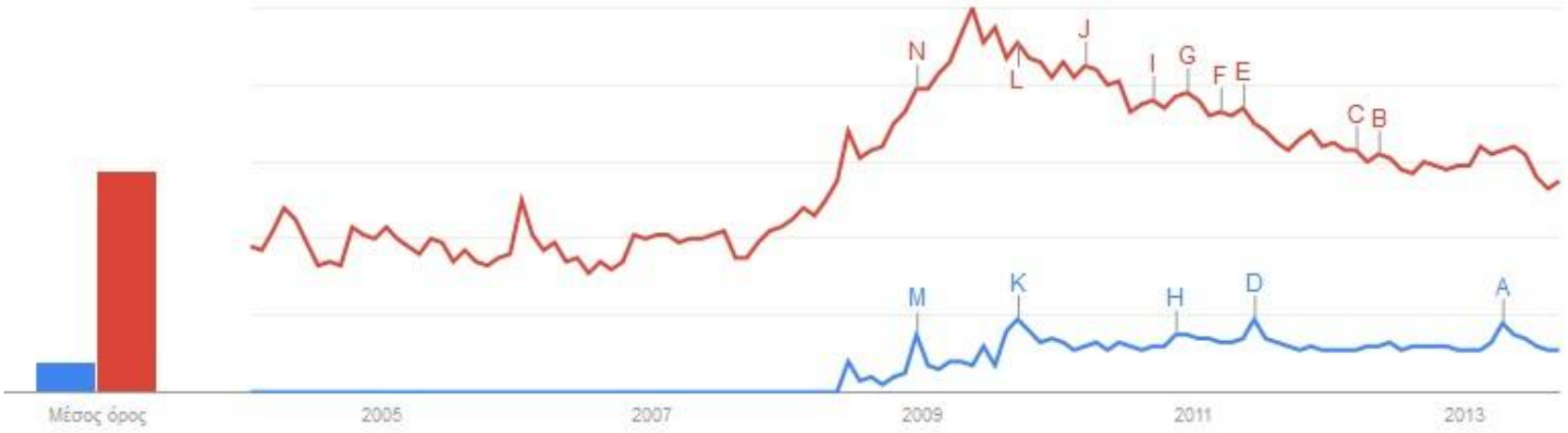
- Στην CUDA υπάρχουν καλύτερα εργαλεία, πχ. CUBLAS, CUFFT, runtime API
- Για την γλώσσα C η CUDA έχει το προβάδισμα, ενώ στην C++ προτιμάται η OpenCL.
- Η OpenCL μπορεί να βάλει στην ουρά εντολών δείκτες για συναρτήσεις CPU ενώ η CUDA όχι.

OpenCL vs. CUDA (4/7)

- Για το ίδιο hardware οι δύο γλώσσες θα έπρεπε θεωρητικά να προσφέρουν ίδια ταχύτητα.
- Κάποιες δοκιμές δείχνουν πως η OpenCL σε Nvidia είναι 10% αργότερη λόγω της CUDA.
- Δεν είναι εύκολο όμως να τις συγκρίνουμε αντικειμενικά σε ταχύτητα.

OpenCL vs. CUDA (5/7)

- Το παρακάτω διάγραμμα μας δείχνει τις αναζητήσεις στο Google για CUDA (κόκκινο) και OpenCL (μπλε).

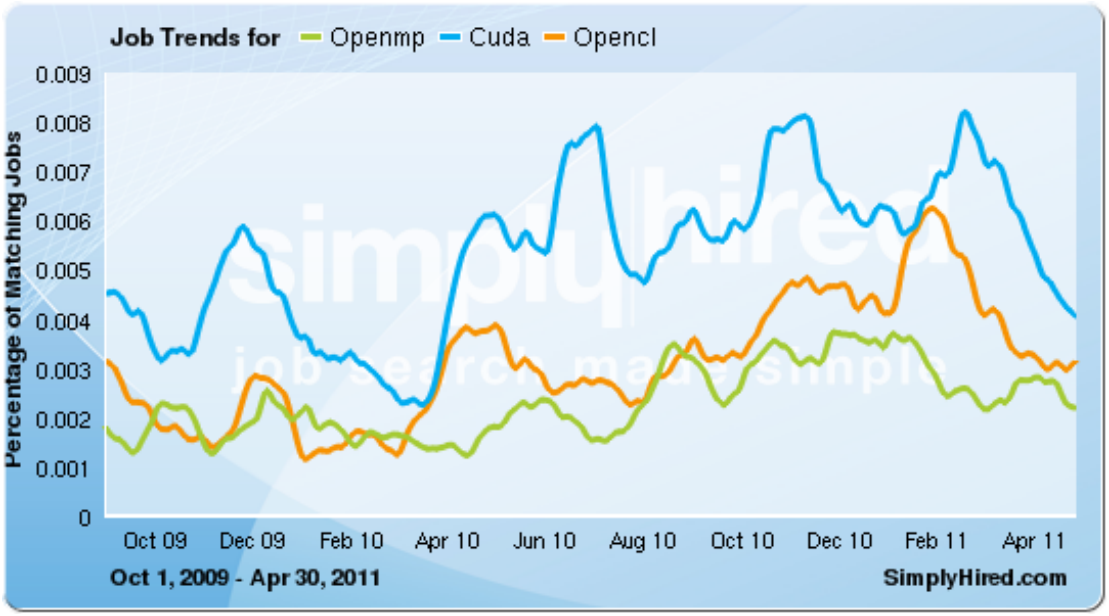


OpenCL vs. CUDA (6/7)

- Όπως βλέπουμε η CUDA είναι πιο δημοφιλής από την OpenCL.
- Βλέπουμε όμως επίσης πως η CUDA «πέφτει» σε δημοτικότητα τα τελευταία χρόνια.
- Δεν αποκλείεται σε λίγο καιρό να δούμε την OpenCL να φτάνει, ακόμα και ξεπερνάει την CUDA.

OpenCL vs. CUDA (7/7)

- Εδώ βλέπουμε πως η CUDA πάλι έχει το προβάδισμα, όμως σε ορισμένα σημεία η OpenCL την ανταγωνίζεται επάξια.



Nvidia vs. AMD

- Οι αρχιτεκτονική της Nvidia και της AMD είναι πολύ διαφορετικές μεταξύ τους. Αυτός είναι ο λόγος που ένας kernel που έχει γραφεί για AMD δεν θα τρέχει εξίσου γρήγορα σε Nvidia.
- Άρα αν θέλουμε να συγκρίνουμε τις δύο αρχιτεκτονικές με benchmark το αποτέλεσμα θα εξαρτηθεί από τον kernel που θα χρησιμοποιήσουμε.

Πηγές

- http://www.khronos.org
- http://www.thebigblob.com
- http://www.drdobbs.com
- http://www.ncbi.nlm.nih.gov
- http://streamcomputing.eu/
- http://wiki.tiker.net/
- http://developer.amd.com
- http://www.cmsoft.com.br/
- http://opencl.codeplex.com
- http://gfxspeak.com

Τέλος Ενότητας