ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Συστήματα Παράλληλης και Κατανεμημένης Επεξεργασίας

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No:19

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http://arch.icte.uowm.gr/mdasyg

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 του Πανεπιστημίου 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.

Περιεχόμενα

1. Σκοπός της άσκησης

- Εισαγωγή στην OpenCL με χρήση emulator σε περιβάλλον Microsoft Visual Studio.
- Δημιουργία αρχείου kernel για χρήση CPU και GPU_EMU.
- Εκτύπωση μηνύματος από τον κάθε slave.

2. Παραδοτέα

(A) 10 ερωτήσεις

(C) 6 ασκήσεις

(B) 1 Bonus

Η openCL *(Open Computing Language)* η οποία υλοποιήθηκε από την Khronos Group είναι μια γλώσσα παράλληλου υπολογισμού η οποία έχει ως στόχο την εκμετάλλευση της επεξεργαστικής ισχύος συσκευών. Συγκεκριμένα είναι συμβατή με διαφόρων ειδών συσκευές όπως επεξεργαστές κάρτες γραφικών κλπ. Δεν αναφέρεται σε μία εταιρία όπως η CUDA *(που είδαμε σε προηγούμενο εργαστήριο και απευθύνεται μόνο σε κάρτες γραφικών της Nvidia).* Η openCL λειτουργεί τόσο σε chipset AMD όσο και Nvidia κάτι που την κάνει πολύ διαδεδομένη για νέα τεχνολογία. Η openCL αν και είναι βασισμένη σε μια παλιά έκδοση της C έχει καλύτερες επιδόσεις σε αντικειμενοστραφή κώδικα γι αυτό και οι περισσότερες εφαρμογές που κυκλοφορούν είναι γραμμένες σε C++. Έτσι σε αυτό το εργαστήριο θα ξεφύγουμε λίγο από την ιδεολογία της C που ήμασταν μέχρι τώρα.

Σε τεχνικό επίπεδο παρατηρούμε ότι όπως και η CUDA είναι μια επέκταση της C++/C διότι χρησιμοποιείται το ίδιο συντακτικό και ίδιες εντολές με τις παραπάνω γλώσσες με επιπλέον πρόσβαση σε συναρτήσεις που επιτρέπουν εκμετάλλευση των συσκευών που αναφέραμε. Φυσικά οι συναρτήσεις αυτές βρίσκονται σε έτοιμες βιβλιοθήκες οι οποίες *(αφού εγκατασταθούν στο σύστημά μας)* εισάγονται στον κώδικα με include.

Βασικό πλεονέκτημά της είναι επίσης και η ετερογένεια ως προς τις συσκευές που τρέχει. Σε αντίθεση με την CUDA που τρέχει μόνο σε κάρτες γραφικών η openCL χρησιμοποιείται σε περισσότερες συσκευές όπως CPU, GPU, DSP κλπ. Έτσι σε αυτό το εργαστήριο θα χρησιμοποιήσουμε openCL και για την CPU και για την GPU.

Επειδή όπως αναφέραμε είναι νέα τεχνολογία πολλές συσκευές ακόμα δεν την υποστηρίζουν γι αυτό στα πλαίσια αυτού του εργαστηρίου θα χρησιμοποιηθεί προσομοιωτής *(emulator)* και εικονική GPU ώστε να γίνει μια εισαγωγή στην νοοτροπία αυτής της γλώσσας. Η συγκεκριμένη έκδοση του emulator δεν είναι έτοιμο εκτελέσιμο με γραφικό περιβάλλον αλλά χρησιμοποιείται σε ολοκληρωμένα περιβάλλοντα *ανάπτυξης (integrated development environment (IDE)).* Προτεινόμενο IDE για τον συγκεκριμένο emulator είναι το Visual studio της Microsoft *(καλύτερα η έκδοση του 2010 μιας και η έκδοση 2013 ζητάει εργαλεία του 2010).*

3. Εγκατάσταση

Απαραίτητα χαρακτηριστικά:

1. AMD Accelerated Parallel Processing (APP) SDK.
    a. http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-app- sdk/ downloads/.
2. AMD OpenCL Emulator http://ocl-emu.google code.com/svn/ trunk/.
3. AMD GPU με υποστήριξη OpenCL *(προτεινόμενο αλλά όχι απαραίτητο).*
4. Microsoft Visual Studio 2010.
5. Microsoft Windows XP or later.

4. Εγκατάσταση-παραμετροποίηση

1) Εγκατάσταση του Microsoft Visual Studio 2010.

2) Εγκατάσταση AMD APP SDK.

a) Πρέπει αρχικά να απεγκαταστήσουμε *(αν έχουμε προηγούμενη έκδοση)*. Σε αυτήν την περίπτωση :

i) Πηγαίνουμε C:\AMD\Support\ streamsdk_v2_8_1_0_win64

(η στο αντίστοιχο για 32bit συστήματα)

ii) Τρέχουμε το Setup.exe και μετά την επιλογή costume επιλέγουμε τα SDK για απεγκατάσταση.

b) Έπειτα τρέχουμε το AMD-APP-SDK-v2.8.1.0-Windows-64.exe

3) Εγκατάσταση του EMU.

a) Κάνουμε extract τους φακέλους opencl_emu, opencl-emu_app

b) Στο C:\Program Files (x86) \AMD APP

c) ή C:\Program Files \AMD APP *(για 32bit)* κάνουμε έναν φάκελο με όνομα OpenCL Emulator και μέσα σε αυτόν κάνουμε copy-paste στους δύο προηγούμενους φακέλους *(opencl_emu, opencl-emu_app).*

4) Στήνουμε το περιβάλλον του EMU.

Χρησιμοποιούμε μεταβλητή περιβάλλοντος *(την CLEMU_ROOT)* και της τοποθετούμε το path το
C:\Program Files (x86) \AMD APP\OpenCL Emulator\opencl_emu

5. Ρυθμίζοντας το Project μας

1. Αρχικά ανοίγουμε το opencl-emu_appVC10 που βρίσκεται C:\Program Files (x86)\AMD APP\OpenCL Emulator\opencl-emu_app\src ή C:\Program Files \AMD APP\OpenCL Emulator\opencl-emu_app\src για *(32bit)*
2. Από την παλέτα solution explorer επιλέγουμε το HelloCL και πάμε Project Properties → C/C++ → General
3. Προσθέτουμε τα ακόλουθα στο additional include directories:

   $(ATISTREAMSDKROOT)\include

   $(CLEMU_ROOT)\clemu

   $(CLEMU_ROOT)\runCL

   $(CLEMU_ROOT)\SDKUtil

   $(CLEMU_ROOT)\include

4. Τώρα πάμε στο Preprocessor και προσθέτουμε: _GPU_EMU_IMPL
5. Έπειτα στο Configuration Properties → Linker → General προσθέτουμε τα παρακάτω στο additional library directories

   $(ATISTREAMSDKROOT)\lib\x86

   $(CLEMU_ROOT)\lib\debug\x86

   $(CLEMU_ROOT)\lib\x86

6. Μετά πάμε στο Linker → Input και προσθέτουμε στο Additional Dependencies

   OpenCL.lib

   runCl.lib

6. Εγγραφή κώδικα HelloCL

Αρχικά πρέπει να γνωρίζουμε ότι θα χρησιμοποιήσουμε ως γλώσσα προγραμματισμού την C++ μιας και η απόδοση της OpenCL στον αντικειμενοστραφή *(όπως γνωρίζουμε)* είναι μεγαλύτερη. Στο project μας που είναι ένα "απλό" hello World θα χρησιμοποιήσουμε δύο αρχεία σε C++. Το ένα θα αναφέρει τους kernels στους οποίους θα τρέξει το δεύτερο.

6.1 HelloCL_Kernels.cpp

Όσο αφορά το πρώτο αρχείο

HelloCL_Kernels.cpp

1. Κάνουμε define το KA με ifdef

   ```
   Ifdef KA
   ```

2. Χρειάζεται να κάνουμε include το path για το clemu_opencl.h και στην συνέχεια την ίδια την βιβλιοθήκη *(clemu_opencl.h)*
3. **(A1)** Δώστε το Path για το clemu_opencl.h
4. Ξεκινάμε μια συνάρτηση Kernel(hello).

   (A2) Ποιά η διαφορά μεταξύ _Kernel(hello)_ArgNULL και _kernel void hello()
5. Σε δύο μεταβλητές i,j θέτουμε τιμές από την συνάρτηση get_global_id().

   (A3) τί τύπου πρέπει να είναι οι μεταβλητές;
6. **(A4)** Αν χρησιμοποιήσω το όρισμα 0 στην get_global_id() τι θα μου επιστρέψει;
7. Μέσα στην συνθήκη

   ```
   if ( i== get_local_size(0) - 1 && j==get_local_size(1) -1 )
   ```

8. Κάνουμε τους ελέγχους (ifdef) για τον slave. Προσοχή οι πιθανές συσκευές slave είναι _GPU_EMU_IMPL, CPU_IMPL, GPU_IMPL
9. **(A5)** τί επιστρέφει η συνάρτηση `get_local_size(0);`
10. Τοποθετούμε το κείμενο που θέλουμε να τυπώσει η slave συσκευή μας
11. Στο τέλος τυπώστε την θέση που βρίσκεται ο συσκευή καθώς και την θέση στο group.

6.2 HelloCL.cpp

Δεύτερο αρχείο

HelloCL.cpp

Κάνουμε include τις κατάλληλες βιβλιοθήκες όπως:
`runCL.h SDKUtil/SDKFile.hpp SDKUtil/SDKCommon.hpp CL/cl.hpp`.

Ορίζουμε `__NO_STD_VECTOR __NO_STD_STRING`

(A6) Για ποιόν λόγο κάνουμε το παραπάνω define εάν δεν το κάνουμε θα έχουμε κάποιο πρόβλημα;

Για να μπορέσουμε να εντοπίσουμε τον kernel που εκτελεί κάθε φορά τον κώδικα χρησιμοποιούμε την εντολή `#define DEFAULT_KERNEL_LOCATION "./"`

Μέσα στην main δίνουμε τις εντολές

```
int workgroup_sz = 64;
      cl_int status;
      ClKrnlArg Args[1024];

int localThreads[] = {8, workgroup_sz/8};
int globalThreads[] = {2*localThreads[0],2*localThreads[1]};
```

Στην συνέχεια κάνουμε κλήση της συνάρτησης callCL με slave την CPU

Η σύνταξη της callCL είναι:

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);

όπου στην θέση `const char * _program` θα βάλουμε το όνομα από το πρώτο αρχείο δηλαδή HelloCL_Kernels.cpp

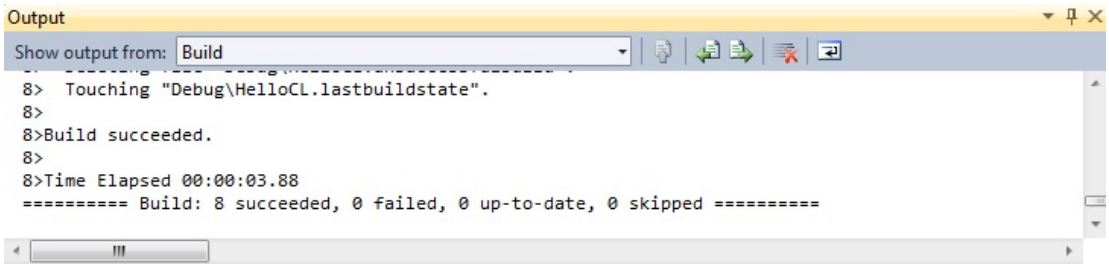
`_device_type`	Target device. Possible values: "cpu", "gpu" , "gpu_emu"
`_domainDim`	Work Dimension
`_domain[]`	Nd Range, Global worksize
`int _group[]`	Local workgroup size
`_program_location`	Kernel directory *(example: "c:/opencl_emu/test/")*
`_program`	Filename containing kernels *(example: "testKernel.cl")*
`_kernel_entry_name`	Specific kernel name
`_args`	Arguments passed to the kernel

Τέλος επιστρέφουμε με SDK_SUCCESS

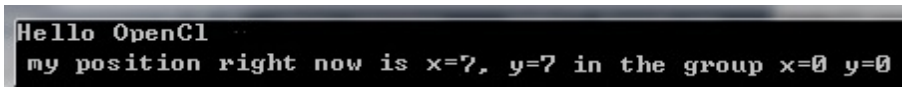
Σε αυτό το σημείο είμαστε έτοιμοι να τρέξουμε το πρώτο μας helloCL στον EMU χρησιμοποιώντας το περιβάλλον του Microsoft Visual Studio.

(C1) Τρέξτε το πρόγραμμα με ctrl+F5. Παραδοτέο το screenshot της εκτέλεσης.

Σε περίπτωση που δεν έγινε compile ελέγξτε προσεκτικά το κάτω παράθυρο *(output)* όπου εμφανίζει τα errors. Πρέπει να είναι της μορφής:



Αν πήγαν όλα καλά πρέπει να πάρετε κάτι τις μορφής



(A7) Προλάβατε να δείτε το αποτέλεσμα; Αν ναι κάνατε κάποια αλλαγή ώστε το παράθυρο να μην κλείνει τόσο γρήγορα;

Κάντε το ίδιο με slave μόνο την GPU_emu **(A8)** Σε ποιό από τα δύο αρχεία κάνατε αλλαγές; *(αιτιολογήστε)*

(C2)Τρέξτε το πρόγραμμα με ctrl+F5. Παραδοτέο το screenshot της εκτέλεσης.

Επιπλέον έχετε την δυνατότητα να εκτελέσετε τον κώδικα ταυτόχρονα σε 2 slaves. Αυτό θα το επιτύχετε κάνοντας τα εξής:

1. **Στο πρώτο αρχείο (HelloCL_Kernels.cpp) πρέπει να υπάρχει δήλωση και των δύο συσκευών slave στην προκειμένη περίπτωση και CPU και GPU_EMU**
2. **Στο δεύτερο αρχείο (HelloCL.cpp) πρέπει να υπάρχει κλήση της callCL και για τις δύο συσκευές slaves στην προκειμένη περίπτωση και CPU και GPU_EMU**

Τρέξτε τον κώδικα με slaves την CPU και την EMU_GPU

Πολλές φορές χρειάζονται συνθήκες που να ελέγχουν κάποιες περιπτώσεις σφάλματος ώστε να αποφεύγονται κρασαρίσματα άγνωστης αιτίας και γενικά να κάνουν το πρόγραμμά μας πιο σταθερό.

(A9) Τί επιστρέφει η callCL σε περίπτωση επιτυχίας;

Τροποποιήστε τον κώδικα ώστε να γίνεται έλεγχος της callCL για εκτέλεση μόνο για slave CPU.

(C3) Τρέξτε το πρόγραμμα με ctrl+F5. Παραδοτέο το screenshot της εκτέλεσης.

Τροποποιήστε τον κώδικα ώστε να γίνεται έλεγχος της callCL για εκτέλεση για slave CPU και GPU.

(C4) Τρέξτε το πρόγραμμα με ctrl+F5. Παραδοτέο το screenshot της εκτέλεσης.

7. Μέτρηση χρόνου εκτέλεσης προγράμματος

Μετρήστε με την χρήση κατάλληλης βιβλιοθήκης τον χρόνο εκτέλεσης του προγράμματος χρησιμοποιώντας ως slave την CPU *(μετρήστε μόνο τον χρόνο που εκτελείται η openCL ώστε να παρατηρήσουμε τυχόν διαφορά όσο αφορά την ταχύτητα εκτέλεσης.)*

(C5) Τρέξτε το πρόγραμμα με ctrl+F5. Παραδοτέο το screenshot της εκτέλεσης.

Κάντε το ίδιο και για την εκτέλεση με slave την GPU_EMU. Προσοχή θέλουμε να μετρήσουμε τον χρόνο εκτέλεσης μόνο του κώδικα που τρέχει στον slave. Να είστε προσεκτικοί όσο αφορά το ποιό αρχείο επεξεργάζεστε όσο και σε ποιό σημείο τοποθετείτε τις μεταβλητές t1,t1.

(C6) Τρέξτε το πρόγραμμα με ctrl+F5. Παραδοτέο το screenshot της εκτέλεσης.

(A10) Παρατηρείτε διαφορά ως προς τον χρόνο εκτέλεσης; Είναι αναμενόμενο το αποτέλεσμα αν ναι γιατί;

(B) Εάν έχετε κάποια κάρτα γραφικών η οποία να υποστηρίζει OpenCL δοκιμάστε να επαναλάβετε τα παραπάνω βήματα χρησιμοποιώντας ως slave την κάρτα γραφικών. Παρατηρείτε μεγάλη διαφορά μεταξύ του χρόνου εκτέλεσης του προγράμματος σε EMU_GPU και σε GPU; Η διαφορά είναι αναμενόμενη; Πού πιστεύετε ότι οφείλεται η διαφορά στον χρόνο εκτέλεσης;

8. Περιορισμοί της συγκεκριμένης έκδοσης emu

1. Μοντέλα: δεν υπάρχει δυνατότητα δημιουργίας 3D domain αλλά μόνο 1D και 2D
2. Οι συναρτήσεις πυρήνα δεν μπορούν να κληθούν μέσα από άλλο πυρήνα
3. Τύπος εικόνας: δεν υποστηρίζεται filtering
4. Εκφράσεις του τύπου s6789i δεν υποστηρίζονται αλλά μόνο .s0, .s1, .s3, .even, .odd
5. Τα διανύσματα αντί για uint2 val2 = uint2(v0, v1); να γίνονται:

```
uint2 val2;
          val2.x = v0;
          val2.y = v1;
```

Και αντί για:

```
uint2 val00 = uint2(val0, val0);
```

να γίνεται:

```
uint2 val00 = (uint2)val0;
```

9. Γνωστά προβλήματα του EMU

1. Δεν μπορεί να χρησιμοποιήσει τα ίδια system memory buffer pointers ως διαφορετικά callCL επιχειρήματα *(arguments)*.
2. Τόσο οι στατικές όσο και οι δυναμικές τοπικές δηλώσεις μνήμης δεν μπορούν να χρησιμοποιηθούν από την ίδια κλήση callCL.