Programarea de kernel in OpenCL

Arhitectura NVIDIA CUDA

Implementarea nVidia pentru GPGPU se numeste CUDA (Compute Unified Device Architecture) si permite utilizarea limbajului C pentru programarea pe GPU-urile proprii. Framework-ul/arhitectura CUDA expune si API-ul de OpenCL prin intermediul caruia vom interactiona cu GPGPU-ul Nvidia Tesla disponibil pe ibm-dp.q.

Motivul discrepantei intre performanta in virgula mobila dintre CPU si GPU este faptul ca GPU sunt specializate pentru procesare masiv paralela si intensiva computational (descrierea perfecta a taskurilor de randare grafica) si construite in asa fel incat majoritatea tranzistorilor de pe chip se ocupa de procesarea datelor in loc de cachingul datelor si controlul fluxului executiei. GPU sunt potrivite pentru paralelismul de date (aceleasi instructiuni sunt executate in paralel pe mai multe unitati de procesare) intensiv computationale. Datorita faptului ca acelasi program este executat pentru fiecare element de date, sunt necesare mai putine elemente pentru controlul fluxului. Si deoarece calculele sunt intensive computational, latenta accesului la memorie poate fi ascunsa prin calcule in locul unor cache-uri mari pentru date.

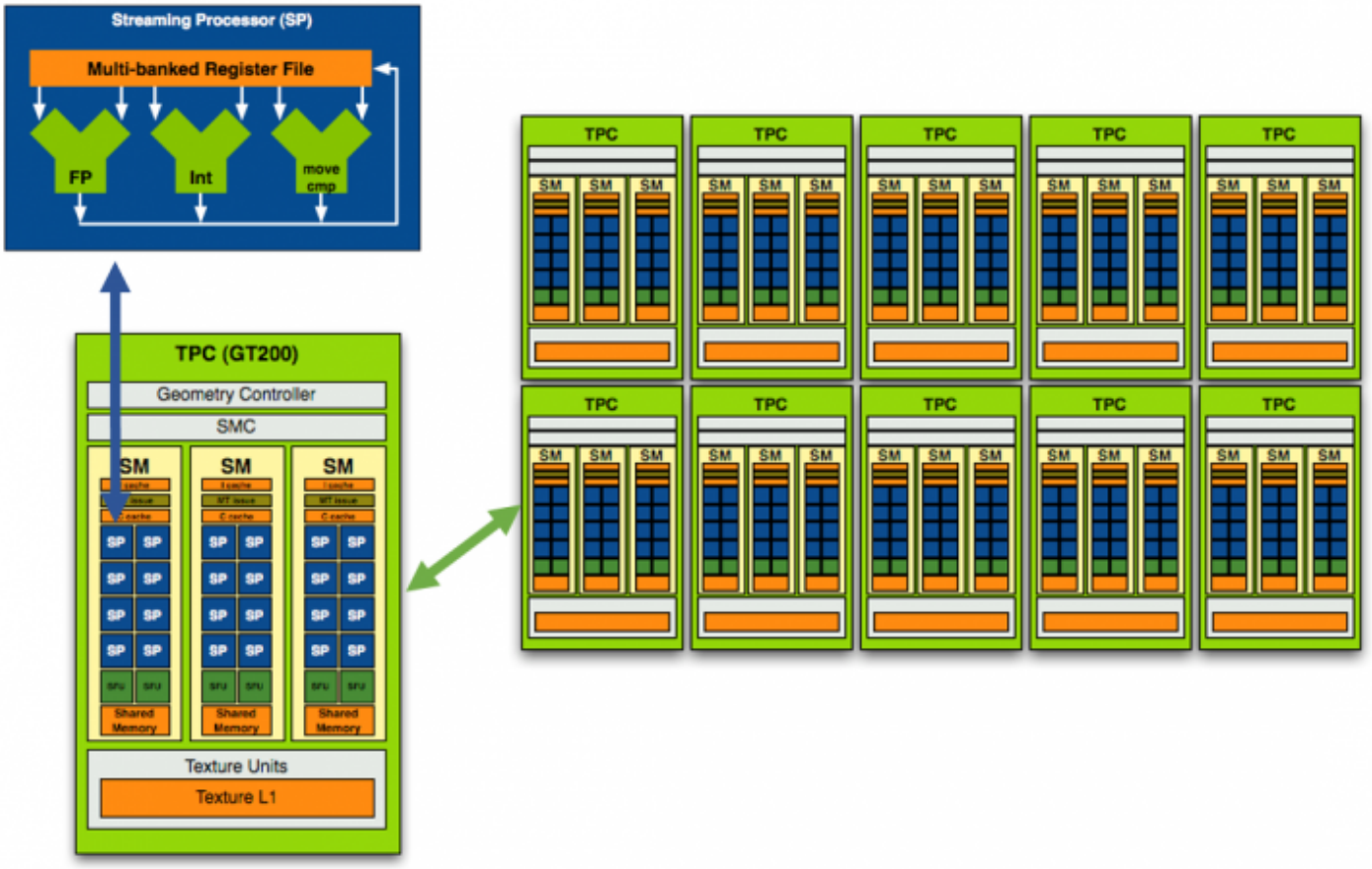
Studiu de caz, arhitectura CUDA

Prima arhitectura care a suportat modelul de programare CUDA/OpenCL a fost G80 (ex. Geforce 8800GTS, lansare in anul 2008). A fost proiectata modular astfel incat sa permita cresterea numarului de elemente computationale in generatii succesive.

Un Streaming Processor (SP) este un microprocesor cu executie secventiala, ce contine un pipeline, unitati aritmetico-logice (ALU) si de calcul in virgula mobila (FPU). Nu are un cache, fiind bun doar la executia multor operatii matematice. Un singur SP nu are performante remarcabile, insa prin cresterea numarului de unitati, se pot rula algoritmi ce se preteaza paralelizarii masive.

SP impreauna cu Special Function Units (SFU) sunt incapsulate intr-un Streaming Multiprocessor. Fiecare SFU contine unitati pentru inmultire in virgula mobila, utilizate pentru operatii transcendente (sin, cos) si interpolare. MT se ocupa cu trimiterea instructiunilor pentru executie la SP si SFU.

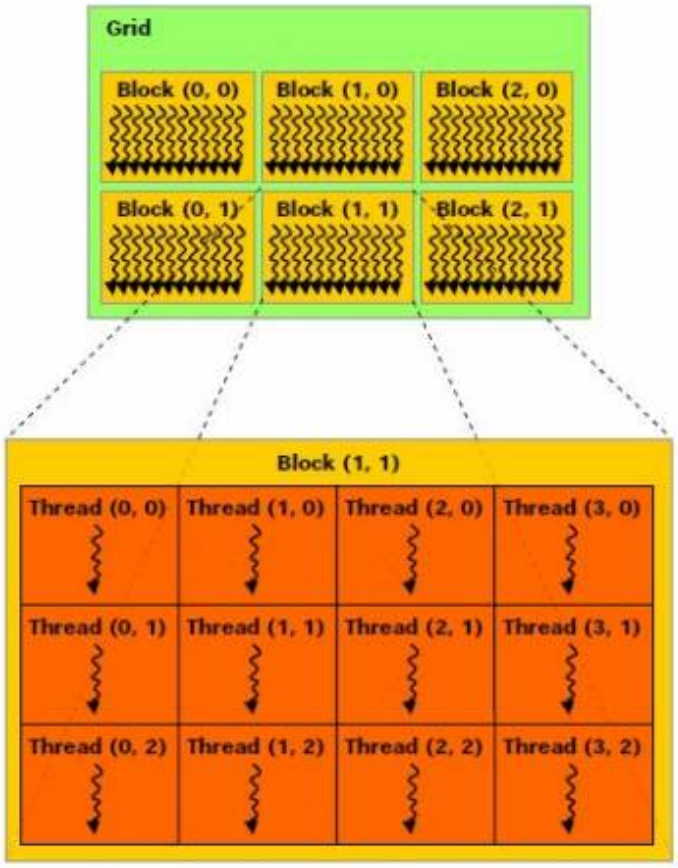
Pe langa acestea, exista si un cache (de dimensiuni reduse) pentru instructiuni, unul pentru date precum si memorie shared, partajata de SP-uri. Urmatorul nivel de incapsulare este Texture / Processor Cluster (TPC). Acesta contine SM-uri, logica de control si un bloc de handling pentru texturi. Acest bloc se ocupa de modul de adresare al texturilor, logica de filtrare a acestora precum si un cache pentru texturi.



Filosofia din spatele arhitecturii este permiterea rularii unui numar foarte mare de threaduri. Acest lucru este facut posibil prin paralelismul existent la nivel hardware.

Documentatia nVidia recomanda rularea unui numar cat mai mare threaduri pentru a executa un task. Arhitectura GT 200 suporta 30720 de threaduri active, iar Fermi 24576 (numarul a fost scazut pentru eficienta). Deci numarul este mult mai mare decat unitatile fizice existente pe chip. Acest lucru se datoreaza faptului ca un numar mare de threaduri poate masca latenta accesului la memorie.

Urmarind acelasi model modular ca si arhitectura, threadurile sunt incapsulate in blocuri (thread blocks), iar blocurile in grile (thread grid). Fiecare thread este identificat prin indexul threadului in bloc, indexul blocului in grila si indexul grilei. Indexurile threadurilor si ale blocurilor pot fi uni/bi/tri-dimensionale, iar indexul grilei poate fi uni sau bi-dimensional. Acest tip de impartire are rolul de a usura programare pentru probleme ce utilizeaza structuri de date cu mai multe dimensiuni. De exemplu pentru GT 200, numarul maxim de threaduri dintr-un bloc este de 512.



Threadurile dintr-un bloc pot coopera prin partajarea de date prin intermediul memoriei shared si prin sincronizarea executiei. Functia de bariera functioneaza doar pentru threadurile dintr-un bloc. Sincronizarea nu este posibila la alt nivel (intre blocuri/grila etc.).

Caracteristici relevante arhitectura dpdv API OpenCL, platforma NVIDIA, device Tesla M2070:

```
  Device Name                                    Tesla M2070
  Device OpenCL C Version                        OpenCL C 1.1
  Device Type                                    GPU
  Device Profile                                 FULL_PROFILE
  Device Topology (NV)                           PCI-E, 14:00.0
  Max compute units                              14
  Max clock frequency                            1147MHz
  Compute Capability (NV)                        2.0
  Max work item dimensions                       3
  Max work item sizes                            1024x1024x64
  Max work group size                            1024
  Preferred work group size multiple             32
  Warp size (NV)                                 32
```

Ierarhia de memorie

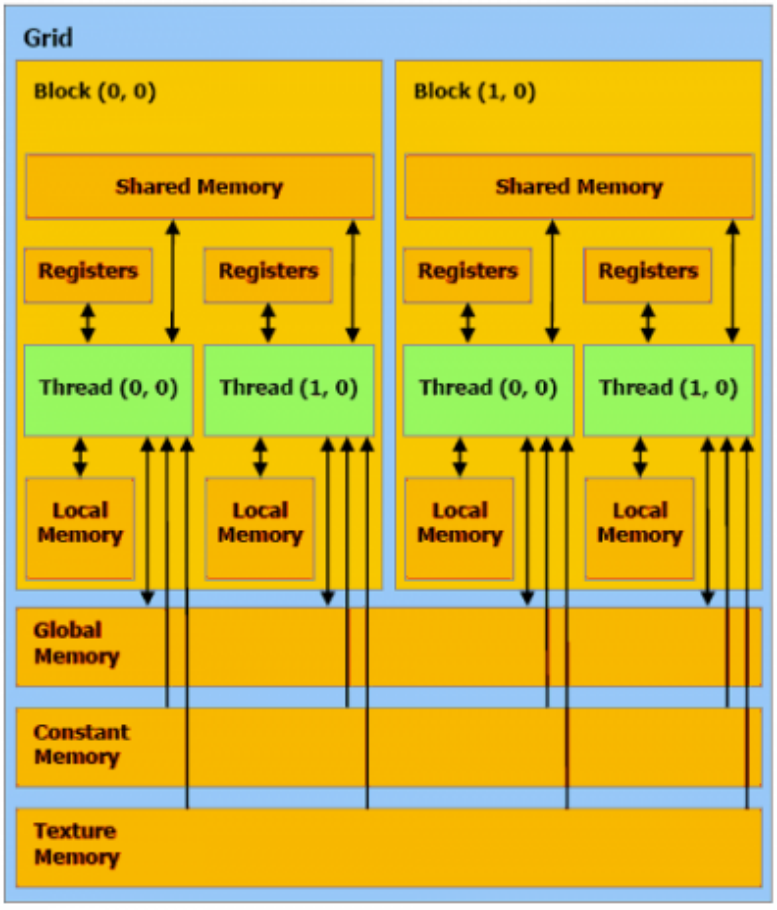
- Registri
  - cea mai rapida forma de memorie de pe multi-procesor
  - sunt accesibili doar de catre thread
  - durata de viata este aceeasi ca si a threadului
- Memoria partajata (shared memory)
  - poate fi la fel de rapida ca registrii atunci cand nu exista conflicte de acces intre threaduri
  - accesibila tuturor threadurilor dintr-un bloc
  - durata de viata este aceeasi ca si a blocului
- Memoria globala
  - poate fi de 150 de ori mai lenta decat registrii sau memoria partajata, mai ales pentru citiri sau scrieri divergente (uncoalesced)

    - accesibila atat de catre host cat si de device
    - are durata de viata a aplicatiei

- Memoria locala
    - o posibila capcana; se afla de fapt in memoria globala si are deci aceleasi penalitati ca si aceasta
    - in momentul in care compilatorul nu mai are loc in registri (si din alte motive…), va stoca datele in memoria locala in loc sa o faca in registri. Se examineaza codul .PTX pentru a determina acest lucru. Profilerul de asemenea va indica un numar mare de „local store” si „local load”.
    - este accesibila doar de catre thread
    - are durata de viata a threadului

- Memoria constanta
    - se afla in memoria globala
    - read-only din device context
    - cached

- Memoria dedicata texturilor
    - se afla in memoria globala
    - read-only
    - mod special de acces la date



Caracteristici relevante ierarhie memorie dpdv API OpenCL, platforma NVIDIA, device Tesla M2070:

```
  Global memory size                              5636554752 (5.249GiB)
  Address bits                                    64, Little-Endian
  Error Correction support                        Yes
  Max memory allocation                           1409138688 (1.312GiB)
  Alignment of base address                       4096 bits (512 bytes)
  Global Memory cache type                        Read/Write
  Global Memory cache size                        229376
  Global Memory cache line                        128 bytes
  Local memory type                               Local
  Local memory size                               49152 (48KiB)
  Registers per block (NV)                        32768
```

```
  Max constant buffer size                 65536 (64KiB)
  Max number of constant args              9
  Max size of kernel argument              4352 (4.25KiB)
```

Performanta diferitelor tipuri de memorie

Memoria locala si memoria globala nu sunt cached, deci orice acces la acestea genereaza un acces explicit la memorie. Este important deci de stiut care este costul pentru un astfel de acces.

Fiecare multiprocesor are nevoie de 4 cicluri de ceas pentru a transmite o instructiune de acces la memorie unui warp. Accesul la memoria locala sau globala dureaza inca 400-600 de cicluri de ceas.

Cu o diferenta de 100-150x in timpul de acces, este evident ca trebuie redus la minim accesul la memoria locala si reutilizarea datelor din memoria shared. De asemenea, latenta memoriei globale poate fi ascunsa destul de mult prin specificarea unui numar mare de blocuri pentru executie si folosirea variabilelor de tip registru, __shared__ si __constant__.

Cum accesul la memoria shared este mult mai rapid decat accesul la memoria globala, tehnica principala de optimizare este evitarea de bank conflicts (warp serialization). Insa, in ciuda vitezei mari a memoriei shared, imbunatatiri majore ale CUBLAS si CUFFT au fost aduse prin evitarea memoriei shared in favoarea registrilor – deci este recomandata folosirea acestora oricand este posibil.

Memoria shared este impartita in module de memorie identice, denumire bancuri de memorie (memory banks). Fiecare banc contine o valoare succesiva de 32 biti (de exemplu, un int sau un float), astfel incat accesele consecutive intr-un array provenite de la threaduri consecutive sa fie foarte rapid. Bank conflicts au loc atunci cand se fac cereri multiple asupra datelor aflate in acelasi banc de memorie. Cand acest lucru are loc, hardware-ul serializeaza operatiile cu memoria (warp serialization), fortand toate threadurile sa astepte pana cand toate cererile de memorie sunt satisfacute. Insa, daca toate threadurile citesc aceeasi adresa de memorie shared, este invocat automat un mecanism de broadcast iar serializarea este evitata. Mecanismul de broadcast este foarte eficient si se recomanda folosirea sa de oricate ori este posibil.

OpenCL date scalare si vectoriale

Pentru a transpune si utiliza dimensiunea mare a registrilor OpenCL dispune atat de date tip scalar cat si de date tip vectorial.

Datele de tip vector sunt utilizate de către operațiile SIMD, și au următoarele proprietăți:

- din date de tip scalar se poate face cast la un tip de date vector
- dintr-un tip de date vector se poate face cast la un alt tip de date vector

Exemple date tip scalar:

OpenCL Type (device)	API Type (host)	Descriere
char	cl_char	8 bit
short	cl_short	16 bit
int	cl_int	32 bit
uint	cl_uint	32 bit signed
long	cl_long	64 bit
ulong	cl_ulong	64 bit signed
float	cl_float	32 bit
double	cl_double	64 bit (optional)

Exemple date tip vectorial:

Tipul de date	Descriere	Valori
charn	cl_charn	8 bit
shortn	cl_shortn	16 bit
intn	cl_intn	32 bit
uintn	cl_uintn	32 bit signed
longn	cl_longn	64 bit

ulongn	cl_ulongn	64 bit signed
floatn	cl_floatn	32 bit
doublen	cl_doublen	64 bit (optional)

Calculul in dubla precizie (double IEEE754) in OpenCL este optional, dar suportat in cazul NVIDIA Tesla M2070. Pentru activare trebuie definit explicit in fisierul de kernel OpenCL.

```
#pragma OPENCL EXTENSION cl_khr_fp64 : enable
```

Caracteristici relevante tipuri date dpdv API OpenCL, platforma NVIDIA, device Tesla M2070:

```
  Preferred / native vector sizes
    char                                                 1 / 1
    short                                                1 / 1
    int                                                  1 / 1
    long                                                 1 / 1
    half                                                 0 / 0        (n/a)
    float                                                1 / 1
    double                                               1 / 1        (cl_khr_fp64)
  Half-precision Floating-point support           (n/a)
  Single-precision Floating-point support         (core)
    Denormals                                     Yes
    Infinity and NANs                             Yes
    Round to nearest                              Yes
    Round to zero                                 Yes
    Round to infinity                             Yes
    IEEE754-2008 fused multiply-add               Yes
    Support is emulated in software               No
    Correctly-rounded divide and sqrt operations  Yes
  Double-precision Floating-point support         (cl_khr_fp64)
    Denormals                                     Yes
    Infinity and NANs                             Yes
    Round to nearest                              Yes
    Round to zero                                 Yes
    Round to infinity                             Yes
    IEEE754-2008 fused multiply-add               Yes
    Support is emulated in software               No
    Correctly-rounded divide and sqrt operations  No
```

OpenCL functii matematice

Functiile matematice primesc ca argumente ori date de tip scalar ori de tip vectorial.
Exemple functii matematice generale :

Functie	
T acos(T)	Arc cosine
T asin(T)	Arc sine
T log(T)	Natural logarithm
T log2(T)	Base 2 logarithm
T log10(T)	Base 10 logarithm
T exp(T)	Exponential base e
T exp2(T)	Exponential base 2
T exp10(T)	Exponential base 10
T mad(Ta, Tb, Tc)	Approximates a * b + c

Functiile matematice pot avea suport integral hardware sau doar in sofware. Prin compilarea de kernel cu anumite flaguri precum “-cl-fast-relaxed-math” putem imbunatatii performanta cu pretul preciziei. La baza, unitatile GPU au avut implementate functii matematice cu precizie variabila spre a avea o performanta cat mai mare. Seria de GPGPU-uri Tesla este orientata strict spre calcul tip HPC si astfel are ajustata arhitectura spre a integra unitati de inalta precizie atat pentru float cat si double (IEEE 754).

OpenCL functii sincronizare

Modelul de executie NDRANGE denota o grupare a problemei pe arhitectura. Prin dimensiunea locala putem asigura o grupare/comunicare a thread-urile (work-items) dintr-un “local workgroup”. Deasemenea in cazul “local workgroup” avem

suplimentar definite mai multe functii de sincronizare. Memoria shared este disponibila doar la nivel de local workgroup. Cum impartim problema dpdv global-local este foarte important si denota singura modalitate de scheduling la nivel de API. Cum se face de fapt scheduling este o combinatie intre diferitele nivele hardware de scheduling cat si driverul de GPU - de multe ori nedeterminist la nivel de API.

Exemple de functii de sincronizare (local workgroup):

Functie	
`void barrier (cl_mem_fence_flags flags))`	Work-items in a work-group must execute this before any can continue
`void mem_fence (cl_mem_fence_flags flags)`	Orders loads and stores of a workitem executing a kernel
`void read_mem_fence (cl_mem_fence_flags flags)`	Orders memory loads
`void write_mem_fence (cl_mem_fence_flags flags)`	Orders memory stores

OpenCL functii profiling

Pentru a evalua timpul unei operatii trimise in coada de executie avem nevoie de suport special atat hardware cat si software. In acest moment NVIDIA nu ofera unelte software dedicate pentru OpenCL ci doar pentru CUDA. In API-ul de OpenCL sunt definite insa metode a putea face profiling pe care stack-ul celor de la NVIDIA le implementeaza.

Interogarea proprietatilor OpenCL referitor la coada de executie, platforma NVIDIA, device Tesla M2070:

```
  Queue properties
    Out-of-order execution                     Yes
    Profiling                                  Yes
  Profiling timer resolution                   1000ns
  Execution capabilities
    Run OpenCL kernels                         Yes
  Concurrent copy and kernel execution (NV)    Yes
```

Exercitii

Urmariti indicatiile todo si documentatia oficiala OpenCL pentru a rezolva exercitiile.
OpenCL refcard 1.2 [https://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf] prezinta o listare a tuturor functiilor OpenCL 1.2 - atat host cat si device.

Intrati pe frontend-ul `fep.grid.pub.ro` folosind contul de pe cs.curs.pub.ro. Executați comanda `qlogin -q ibm-dp48.q` pentru a accesa una din stațiile cu GPU-uri. Modificarile se vor face in host.cpp (ex 1,2) cat si in device.cl (ex 2).

1. (3p) Completati fisierul “host.cpp” astfel incat sa faceti profiling la executia de kernel (urmariti indicatiile todo). Afisati timpul de executie al functiei kernel folosind profiling cu events.
2. (3p) Modificati fisierul “host.cpp” si efectuati profiling la urmatoarele dimensiuni de buffer: 4M, 16M, 32M si 64M. Cum variaza executia de kernel fata de executia intregului program ?
3. (2p) Adaugati optiunea de profiling si la transferurile de memorie. Ce observati ?
4. (2p) Compilati kernelul cu optiunea “-cl-fast-relaxed-math”. Cum impacteaza optiunea timpul de executie ? Dar rezultatele ?

Resurse

Schelet Laborator 11

Solutie Laborator 11

Enunt Laborator 11

- Responsabili laborator: Grigore Lupescu

Referinte

- Documentatie oficiala OpenCL:
  - https://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/

[https://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/]

- https://www.khronos.org/files/opencl-1-1-quick-reference-card.pdf [https://www.khronos.org/files/opencl-1-1-quick-reference-card.pdf]
- https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf [https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf]

- Alte:
  - https://www.khronos.org/registry/ [https://www.khronos.org/registry/]
  - https://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml/clBuildProgram.html [https://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml/clBuildProgram.html]