ARHITECTURI COMPLEXE FOLOSITE ÎN PRELUCRAREA PARALELĂ A IMAGINILOR

dr. ing.	prof. dr. ing.	prof. dr. ing.	as. drd. ing.	ing.
Ştefan Mocanu	**Radu Dobrescu**	**Daniela Saru**	**Ramona Din**	**Andrei Grumăzescu**
smocanu@rdslink.ro	radud@isis.pub.ro	saru@aii.pub.ro	din.ramona@yahoo.com	andrei.grum@yahoo.com

Universitatea "Politehnica" Bucureşti

Rezumat: Simulatoare de antrenament, proiectare în diferite domenii de activitate, simulare şi vizualizare ştiinţifică, practică medicală sau „numai" jocuri 3D şi filme de animaţii, toate au în comun puterea de procesare de pe placa video. Susţinute de avansul tehnologic, dar şi datorită cererii pieţei pentru aplicaţii care să ruleze la rezoluţii mari şi cu performanţe HD (*High Definition*), procesoarele grafice (GPU) au evoluat foarte mult, transformându-se în arhitecturi complexe, mai puternice chiar decât CPU în anumite cazuri. Această lucrare îşi propune prezentarea succintă a unei arhitecturi performante de calcul paralel (CUDA) şi evidenţierea posibilităţilor pe care le oferă în diverse domenii, urmând ca un articol viitor să dezvolte studii de caz pentru cercetarea potenţialului de performanţă al acesteia în vederea implementării unor algoritmi optimizaţi pentru recunoaşterea, clasificarea şi procesarea imaginilor.

Cuvinte cheie: procesor grafic (GPU), CUDA, procesare paralelă, GPGPU, GUI, OpenCL.

Abstract: Training simulators, various design tools, science processes simulation and visualization, medical practice or „just" 3D games and animations, they all have in common the graphical card processing unit. Sustained by the technological progress and also due to the constant need for higher resolutions and speed for graphical applications which require HD (High Definition) performance, Graphical Processing Units (GPUs) are turning into increasingly complex architectures, sometimes even more powerful than the CPU. This paper presents Nvidia's general purpose parallel computing architecture, code name CUDA, while a future article will develop case studies focused on the research of its potential in highly optimized parallel algorithms for image recognizing, classifying and processing.

Key words: Graphical processing unit (GPU), CUDA, parallel computing, GPGPU, GUI, OpenCL.

1. Introducere

De-a lungul timpului, domeniul procesării grafice a reunit specialişti în matematică, informatică, proiectare şi chiar artişti, în încercarea de a crea o realitate virtuală bine conturată, care să îmbine caracteristicile mediului real cu informaţii generate de calculator. Fie că vorbim despre sisteme de realitate îmbogăţită – *Augmented Reality* (întâlnite adesea în domenii precum medicină, robotică sau aviaţia militară) în care un dispozitiv de tipul HMD (*Head Mounted Display*) suprapune date generate de calculator peste imaginea mediului real, despre sisteme de teleprezenţă cu ajutorul cărora operatorul poate observa şi controla acţiunile unui robot plasat într-un mediu inaccesibil sau periculos, despre sisteme proiective sau de simulare ori despre cea mai completă formă de realitate virtuală – *Immersive Virtual Reality*, în care contactul participantului cu lumea reală este întrerupt în totalitate, domeniile de aplicabilitate sunt din ce în ce mai vaste, iar suportul oferit de elementele grafice este de nepreţuit. Toate aceste sisteme au la bază procesarea grafică, adică ansamblul metodelor şi tehnicilor de conversie a datelor către şi de la un dispozitiv grafic prin intermediul calculatorului.

Astfel, procesarea imaginilor a reprezentat un punct de interes comun multor comunităţi ştiinţifice, urmărindu-se continuu reducerea timpului de procesare şi proiectarea unor algoritmi optimizaţi de prelucrare. În contextul evoluţiei din punct de vedere tehnologic, procesarea secvenţială a fost înlocuită cu procesarea paralelă. Aceasta a avut un impact deosebit asupra problemelor dintr-o gamă largă de domenii. Costurile mici ale tehnicilor de procesare paralelă, corelate cu cerinţele de performanţă din ce în ce mai mari ale aplicaţiilor, au fost unele dintre cele mai convingătoare argumente care au susţinut, de-a lungul timpului, acest model de programare aplicat deja cu succes în algoritmi de proiectare şi în aplicaţii din domeniul ingineriei (aviaţie, mecanică, motorizări, circuite integrate, sisteme microelectromecanice şi nanoelectromecanice), din domeniul ştiinţific (bioinformatică, fizică, chimie, astronomie, modelare matematică) sau din gama aplicaţiilor comerciale (servere web şi baze de date, extragerea informaţiilor, optimizarea deciziilor din domeniul afacerilor sau de pe diverse pieţe)

ori a sistemelor de calculatore (sisteme embedded, reţele, sisteme de detecţie şi prevenire a pătrunderii viruşilor)[2]. Pentru soluţionarea multor probleme de calcul paralel au fost dezvoltaţi algoritmi specifici ce au evoluat odată cu cerinţele aplicaţiilor. Un caz particular, în care modelul de programare paralelă a fost aplicat cu succes este procesarea imaginii – operaţii de pre-procesare, recunoaşterea formelor, identificarea obiectelor sau transformări discrete Fourier[5]. Dezvoltată la graniţa dintre procesarea grafică şi algoritmii paraleli, prelucrarea imaginilor a reprezentat un salt uriaş atât pentru cercetare, cât şi pentru inginerie, motiv pentru care tot mai multe eforturi au fost îndreptate către optimizarea acestor calcule şi reducerea timpului de execuţie a algoritmilor. Domenii precum medicina, astronomia, biologia sau criminalistica au la bază utilizarea şi interpretarea imaginilor specifice, astfel încât o prelucrare rafinată a acestora poate aduce avantaje considerabile.

Întorcându-ne în timp, conceptul de grafică digitală (*Computer Graphics*) a fost introdus de către William A. Fetter, grafician la Boeing, în anul 1960, pentru a descrie activitatea prin care încerca să proiecteze cât mai eficient spaţiul pentru cabina pilotului. Astfel, a fost realizată prima imagine digitală ortogonală a corpului uman (figura 1.1) şi primele animaţii computerizate 3D [1].

Figura 1.1. „Primul Om" – imagine realizată de W. Fetter, în 1960

În continuare, în pionieratul artei pe calculator, inginerii de la Bell Laboratoires şi-au îndreptat atenţia către captarea diverselor forme geometrice şi reproducerea acestora cu ajutorul tehnologiei existente. Drept urmare, au luat naştere lucrările lui Michael Noll (1962), ale matematicianului Frieder Nake (1965) şi cele ale lui Manfred Mohr.

În anii 1970, în cadrul centrului de cercetare de la Palo Alto (*Xerox PARC*), sunt puse bazele multor tehnologii actuale (reţele de calculatoare, transferuri de date, imprimare laser), însă una dintre cele mai importante inovaţii ale momentului a fost realizarea unei interfaţe grafice cu utilizatorul (*GUI – Graphical User Inteface*). O astfel de interfaţă putea aduce calculatorul în viaţa cotidiană, iar simpla afişare a unor linii de cod sau a unor comenzi devenea istorie, cedând locul reprezentărilor grafice şi imaginilor. Pe fundalul unui contur din ce în ce mai clar al proiectării calculatoarelor personale, Xerox creează prototipul Alto. În timp ce mare parte din cercetare se concentra pe rezolvarea sarcinilor numerice şi de control al datelor, inginerii de la Xerox şi-au îndreptat atenţia către procesarea grafică şi comunicaţii. Este adevărat că prima interfaţă grafică realizată de ei era greu de utilizat şi dependentă de puterea procesorului central pentru fiecare modificare a unui bit; însă primul pas era făcut. A urmat dezvoltarea unor algortimi din ce în cei mai performanţi pentru mişcarea ferestrelor pe ecran într-un timp cât mai scurt şi fără a supra-încărca procesorul. Au apărut circuite electronice specializate în controlul procesării grafice, cu rolul de a descongestiona unitatea centrală de prelucrare în momentul utilizării aplicaţiilor grafice. Spre sfârşitul anilor 1970, pe fondul unor factori externi precum rezistenţa guvernului, recesiunea economică şi chiar competiţia neloială, Xerox nu a mai putut ţine pasul. Decăderea uneia dintre cele mai importante companii în domeniu şi proiectarea, cu precădere, a circuitelor electronice destinate calculelor şi sarcinilor numerice, au făcut ca anii 1980 să nu aducă îmbunătăţiri semnificative în domeniul procesării

grafice. În iunie 1985, compania Hi – Toro din Santa Clara scoate, totuşi pe piaţă Amiga A1000 – calculatorul personal cu cele mai bune performanţe: 512K RAM (enorm pentru acel moment!), procesor Motorola 68000 la 14MHz şi o paletă, nemaiîntâlnită până atunci, de 4096 de culori. De asemenea dispunea, pentru prima dată, de un conector integrat video şi de câteva coprocesoare destinate prelucrărilor audio şi video. Chiar dacă intenţia producătorului nu a fost decât să se impună pe piaţa calculatoarelor personale, toate aceste caracteristici au făcut ca Amiga A1000 să fie considerat, totodată, şi primul calculator ce a oferit suport pentru aplicaţii multimedia [1].

Deşi procesoarele centrale au avut o evoluţie cu adevărat spectaculoasă încă din anii 1980, despre *chip*-urile de procesare grafică nu se poate spune acelaşi lucru. Abia anul 1991, când arhitectura 486 a procesorului central a permis pentru prima dată rularea rapidă a sistemului Windows (versiunea 3.0, 3.1 sau chiar 3.11), dă startul unei competiţii importante pe această piaţă. Totodată, este momentul în care coprocesorul matematic, disponibil pentru versiunile anterioare sub forma unui *chip* separat, este integrat la nivelul CPU, oferind, pentru prima dată, suport hardware pentru procesare paralelă în cadrul aceluiaşi microprocesor. Tot acum sunt lansate şi încearcă să se impună ca standarde, cele mai cunoscute API-uri din domeniu (OpenGL şi DirectX) şi apar primele tentative de procesoare 3D care, în realitate, nu erau decât versiuni îmbunătăţite ale procesoarelor 2D (motiv pentru care au eşuat). Primii care au reuşit să beneficieze de avantajele unui procesor 3D au fost cei de la 3dfx cu prototipul Voodoo. Cu toate că performanţele acestuia erau mult mai bune decât ale predecesorilor săi, o mare parte a calculelor matematice încă se făceau la nivelul coprocesorului matematic.

La sfârşitul anului 1999, Nvidia lansează produsul pe care, în campania de marketing, l-au denumit primul *procesor grafic* (*GPU – graphical processing unit*). Sub numele oficial GeForce 256 (NV10), acesta fusese construit să ofere suport pentru funcţii geometrice de tipul T&L (*Transform, clipping, and lighting*) şi să accelereze grafic bibliotecile Direct3D 7.0. A fost primul procesor grafic cu performanţe excepţionale care a surclasat cu uşurinţă toate plăcile video existente pe piaţă în momentul lansării. Au urmat generaţiile GeForce 2 care au adus o creştere a frecvenţei până la 200MHz, dar şi o unitate TMU (*Texture Map Unit*) şi GeForce 3 care oferă, pentru prima dată, suport pentru programarea anumitor unităţi funcţionale.

Ca şi în cazul procesoarelor centrale, concurenţa a obligat competitorii să-şi perfecţioneze produsele, să lanseze noi versiuni sau chiar să proiecteze alte prototipuri mai performante. Astfel, în paralel cu generaţiile GeForce de la Nvidia, cei de la ATI au lansat versiuni ale procesoarelor Radeon. Odată cu tehnologia *Unified Shader Architecture*, adoptată de Nvidia pentru cea de-a opta generaţie GeForce şi de către ATI pentru plăcile Radeon HD 2000, apare o flexibilitate a arhitecturii, iar unităţile funcţionale sunt folosite mult mai eficient. Acesta a fost momentul în care, printr-un nou proces de fabricaţie şi prin creşterea numărului de unităţi funcţionale (în contrast cu strategia de proiectare a CPU, unde lucrurile s-au concentrat pe îmbunătăţirea *pipeline*-ului, respectiv paralelizare la nivel de instrucţiune), *chip*-urile grafice au devenit mult mai performante decât unităţile centrale de prelucrare. Diferenţele de performanţă se datorează faptului că într-un GPU majoritatea tranzistorilor sunt dedicaţi procesării datelor, iar în cazul procesorului central sunt folosiţi în special pentru memorii tampon şi control. Aşadar, susţinute de avansul tehnologic, dar şi datorită cererii pieţei pentru aplicaţii 3D care să ruleze la rezoluţii mari şi cu performanţe HD, procesoarele grafice au evoluat foarte mult, transformându-se în arhitecturi complexe, *multithreaded*, multi-core, cu putere mare de procesare şi cu canale de comunicaţie cu lăţime mare de bandă. Mai mult, s-a constatat faptul că procesoarele grafice nu sunt exploatate suficient şi că ar putea fi redirecţionate către alte domenii decât grafica. Astfel, a apărut un nou concept numit GPGPU (*General Purpose computation on GPU*) [4].

Primele încercări de a utiliza GPU în alte domenii decât cel grafic nu sunt tocmai recente (2002), însă facilităţile oferite programatorilor de către producătorii plăcilor video au făcut acest domeniu mult mai accesibil cercetării şi chiar utilizării în aplicaţii de interes general doar în ultimii ani. Odată cu lansarea pe piaţă a modelului GeForce 8, Nvidia introduce şi tehnologia CUDA (*Compute Unified Device Architecture*) prin intermediul căreia placa video este privită

ca un coprocesor matematic al procesorului central. În 2007, pe lângă interfaţa de comunicare cu procesorul, a fost lansat şi un pachet software complet (compilator, biblioteci, documentaţie) care a uşurat munca utilizatorilor. De asemenea, platforma CUDA pune la dispoziţia dezvoltatorilor de aplicaţii un API specific, denumit CUDA API, care foloseşte limbaje de programare adaptate arhitecturii paralele.

2. Platforma CUDA

Platformă software dedicată soluţionării problemelor ce pot fi exprimate prin paralelism la nivel de date, CUDA reprezintă o extensie a limbajului C, la baza căreia se regăsesc noţiuni de abstractizare destinate exclusiv programării paralele (ierarhizarea firelor de execuţie pe grupe, memoria comună sau bariere de sincronizare), principii pentru decongestionarea procesorului central şi câteva completări ale sintaxei.

Considerat la nivel arhitectural un coprocesor matematic al CPU, procesorul grafic reprezintă de fapt o colecţie de nuclee capabile să execute în paralel un număr foarte mare de fire de execuţie (*threads*). De exemplu, în cazul unui GPU din primele serii GeForce 8 pot fi executate până la 12 288 astfel de sarcini[3]. Deşi arhitectura CUDA impune un anumit stil de programare paralelă, nu obligă programatorul să se ocupe de gestiunea explicită a firelor de execuţie. Acesta are, în general, sarcina de a partiţiona problema în sub-probleme ce urmează a fi rezolvate prin cooperare între firele de execuţie, mecanism gestionat cu succes de către cei de la Nvidia. Metoda de descompunere a problemelor menţine expresivitatea limbajului, oferind în acelaşi timp o scalabilitate transparentă, deoarece fiecare sub-problemă poate fi programată să ruleze pe oricare nucleu de procesare. Astfel, o funcţie compilată sub CUDA, denumită generic *kernel*, poate să ruleze pe oricâte nuclee îi sunt puse la dispoziţie, întrucât resursele pot fi alocate fie de către pogramator, fie de către sistem. De asemenea, se asigură şi compatibilitatea programelor scrise pe arhitecturi mai vechi, gen 8800 GTS, cu cele noi, de tipul GTX 200, dar şi o creştere a performanţelor, în raport exponenţial cu numărul de unităţi funcţionale.

Firele de execuţie pot fi identificate şi organizate în blocuri (*threads blocks*) care, la rândul lor, formează grid-uri (*grids of thread blocks*). Fiecare bloc poate fi identificat, în interiorul grid-ului, cu ajutorul unui index, iar execuţia mai multor blocuri este posibilă în orice ordine, secvenţial sau paralel. Un grid reprezintă, de fapt, un şir de blocuri care execută acelaşi *kernel*, citesc date din memoria globală şi asigură mecanismele de sincronizare în cazul apelurilor dependente. Unui *thread* i se asociază automat un *threadID*, accesibil în interiorul *kernel-ului*, un contor de program, regiştrii, o locaţie proprie în memoria comună, date de intrare şi date de ieşire. Fiecare bloc are alocat un spaţiu de memorie comună, vizibil tuturor firelor de execuţie din bloc. Pe lângă accesul la memoria comună, toate firele de execuţie au drept de citire a memoriei de constante şi a celei de texturi.

2.1 Interfeţe de programare

La momentul actual, există două interfeţe de programare destinate arhitecturii CUDA: *C pentru CUDA* – interfaţă destinată exclusiv programelor scrise în extensia limabjului C şi *CUDA driver API*. Un program trebuie să folosească numai una dintre aceste intefeţe, ele excluzându-se reciproc.

Interfaţa C pentru CUDA expune modelul de programare sub forma unui set minimal de extensii ale limbajului clasic C, în care programatorul poate defini funcţii *(kernels)* care, atunci când sunt apelate, sunt executate de *N* ori în paralel de către *N* fire de execuţie CUDA pe procesorul grafic, spre deosebire de modul clasic, în care erau executate o dată, pe CPU. Un *kernel* se defineşte folosind declaraţia `_global_` înaintea numelui funcţiei şi specificând numărul firelor de execuţie pentru fiecare apel, cu următoarea sintaxă[7]:

```
_global_ void vecAdd (float* A, float* B, float* C)
 { int i = threadIdx. X;
   C [i] = A [i] + B [i];
 }
int main ()
 { //Kernel invocation
   VecAdd<<<1, N>>> (A, B, C);
 }
```

Orice fişier sursă care conţine extensii, trebuie compilat cu *nvcc*, compilator la nivel de driver, pus la dispoziţie gratuit de Nvidia. Acelaşi program CUDA poate conţine cod mixt, adică atât cod destinat procesorului grafic, cât şi cod pentru CPU, sarcina de a separa cele două tipuri revenindu-i compilatorului. În urma procesului de separare poate rezulta fie cod C (codul sursă destinat execuţiei pe CPU) care, în urma unei compilări cu instrumente sotfware specifice CPU, va rula ca un proces normal, fie cod PTX (*Parallel Thread eXecution*) care va fi compilat cu *nvcc* şi va rula pe GPU. Compilatorul *nvcc* suportă şi cod C++, destinat funcţiilor ce rulează pe CPU, însă concepte de programare orientată pe obiecte precum noţiunea de clasă sau de moştenire încă nu au fost implementate (conform Nvidia CUDA Programming Guide v2.3.1). Totuşi, Nvidia promite să ofere suport complet pentru C++, dar şi optimizări pentru OpenCL, Fortran sau DirectCompute pentru versiunile ulterioare.

CUDA driver API este o interfaţă care oferă funcţii utile pentru configurarea directă a GPU, încărcarea programelor *kernel* ca module binare CUDA sau sub formă de cod de asamblare, pentru inspectarea parametrilor, lansarea modulelor în execuţie şi transmiterea rezultatelor execuţiei către CPU.

Procesul de execuţie al unui program tipic CUDA începe prin rularea codului destinat CPU. Când se apelează o funcţie *kernel*, execuţia se transferă unui procesor grafic unde se generează un număr mare de fire de execuţie (un grid) pentru a putea fi lansate în paralel mai multe task-uri. La încheierea execuţiei *grid-ului*, rularea aplicaţiei revine pe CPU până în momentul în care este apelată o altă funcţie *kernel* (figura 2.1.). Atât procesorul central, cât şi procesorul grafic dispun de spaţii proprii de memorie, astfel încât, pentru a permite execuţia unui *kernel* programatorul trebuie să aibă în vedere alocarea spaţiului de memorie necesar, transferul datelor de la CPU către GPU, respectiv transferul de la GPU către CPU şi eliberarea memoriei la încheierea execuţiei [4].

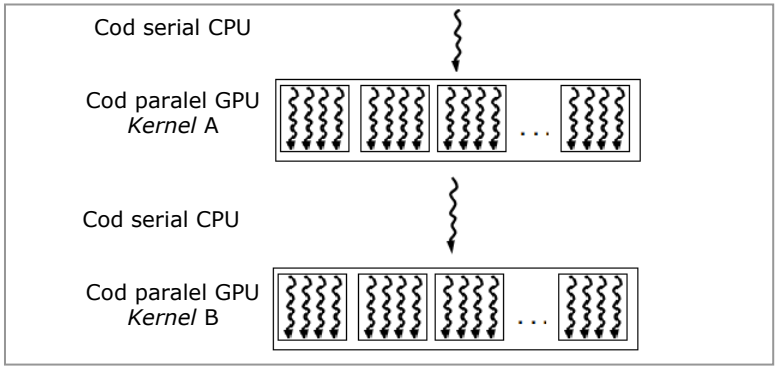

Figura 2.1. Execuţia unui program CUDA

Interfaţa C pentru CUDA include un *runtime API* care, împreună cu driver API, furnizează suport pentru alocarea şi eliberarea memoriei, transferul datelor între GPU şi CPU, gestiunea sistemelor cu mai multe procesoare grafice şi alte mecanisme utile pentru atingerea unor performanţe din ce în ce mai bune. Spre deosebire de interfaţa C, CUDA driver API necesită mai mult cod şi este mai greu de programat, însă oferă un control mai riguros al codului şi al

resurselor. În afară de CUDA driver API, interfaţa generică de programare a unităţilor de tip *device* permite scrierea funcţiilor folosind DirectX Compute sau OpenCL.

Încă de la lansarea pe piaţă a primelor produse dedicate procesării paralele pe GPU, Nvidia a reuşit să se impună cu un avantaj de invidiat pe atunci: „a ascuns" programatorului arhitectura hardware în spatele unei interfeţe de programare, oferindu-i funcţii predefinite pentru cazurile în care nu vrea să lucreze direct la nivel hardware. Acest lucru a permis dezvoltarea aplicaţiilor dedicate GPU fără implicarea programatorilor în gestiunea resurselor hardware, dar şi o compatibilitate fără costuri foarte mari a aplicaţiilor vechi pe platforme noi. Odată cu creşterea interesului faţă de noile tehnologii şi cu diversificarea domeniilor de aplicabilitate, necesitatea gestionării de către programator a resurselor în vederea obţinerii unor performanţe net superioare a condus la lansarea unui set complet de instrumente software numit Nvidia OpenCL. Pentru asigurarea corespondenţei dintre principiile de programare OpenCL şi suportul hardware, fiecare multi-procesor al GPU a fost proiectat ca un corespondent al unităţii de calcul OpenCL. Un multi-procesor execută un *thread* CUDA pentru fiecare instrucţiune OpenCL, respectiv un bloc de *thread-uri* pentru un grup de instrucţiuni. Pentru obţinerea unor rezultate cât mai bune, se recomandă un grad cât mai mare de paralelizare a codului secvenţial, reducerea transferurilor de date dintre CPU şi GPU chiar şi atunci când acest lucru presupune execuţia unor funcţii *kernel* pe GPU fără obţinerea unor performanţe semnificativ mai mari în comparaţie cu rularea pe CPU, reducerea accesului la memoria globală şi asigurarea unui acces omogen ori de câte ori apare necesitatea accesării acesteia, evitarea utilizării mai multor căi de execuţie în cadrul aceluiaşi *warp* şi optimizarea instrucţiunilor.

Pentru dezvoltarea aplicaţiilor care să exploateze noua arhitectură, programatorii au la dispoziţie un set complet de instrumente software, exemple şi documentaţie, oferite gratuit: *runtime C* care oferă suport pentru execuţia funcţiilor C standard pe GPU şi permite trecerea către limbaje de nivel înalt precum Java, Fortran sau Python; Nvidia Nexus[7] – primul mediu de dezvoltare proiectat exclusiv pentru aplicaţii CUDA C; OpenCL şi DirectX Compute; biblioteci care includ funcţii matematice complexe optimizate pentru arhitectura CUDA; compilator Nvidia C (*nvcc*), mediu de depanare CUDA (cudagdb); ghid de programare (*CUDA Programming Guide*); specificaţii API şi exemple de funcţii *kernel* care ilustrează tehnici de programare pentru o gamă largă de algoritmi şi aplicaţii.

2.2 Implementare hardware

Din punct de vedere hardware, platforma de dezvoltare a aplicaţiilor GPGPU presupune lucrul cu unul sau mai multe procesoare grafice (*devices*) şi procesorul central (*host*) al sistemului. Astfel, un program poate fi considerat un ansamblu de sub-programe care pot rula fie pe GPU, fie pe CPU. În general, se recomandă implementarea codului pentru CPU atunci când acesta nu poate fi paralelizat sau gradul de paralelizare este foarte mic, altfel performanţele obţinute în urma execuţiei pe GPU fiind mult mai bune.

Arhitectura CUDA este construită pe o matrice de multi-procesoare (SM – *Streaming Multiprocessors*) în care ierarhia firelor de execuţie corespunde chiar ierarhiei procesoarelor din cadrul GPU. Când un program care rulează pe GPU invocă un *kernel*, blocurile sunt numărate şi distribuite către multi-procesoarele libere. Firele de execuţie ale unui bloc rulează concurent pe nucleele multi-procesoarelor, în grupuri a câte 32, purtând numele de *warps* (acest termen provine din asemănarea arhitecturii cu un război de ţesut, unde mai multe fire formează o urzeală). Pentru a putea lucra cu mii de fire de execuţie (pentru a obţine cele mai bune performanţe, se recomandă programatorilor să lucreze cu cel puţin 5 000 de fire de execuţie concurente!), multi-procesorul implementează o arhitectură denumită SIMT (*Single Instruction, Multiple Thread*). Fiecare fir de execuţie rulează pe un anumit nucleu, în mod independent, cu propria adresă de instrucţiune şi propriul registru de stare, iar atunci când trebuie să execute mai multe blocuri, le împarte în *warps*, grupuri ce vor fi programate pentru execuţie de către unitatea *warp scheduler*. Modul în care un bloc este organizat în grupuri este întotdeauna acelaşi, astfel încât fiecare *warp* va conţine fire de execuţie consecutive, începând cu *id* – ul primului fir. Unitatea SIMT este cea care selectează un *warp* pentru execuţie şi emite următoarea

instrucţiune către firele de execuţie active ale acestuia. Deşi programatorul poate ignora modul intern de funcţionare a unităţii SIMT, printr-o proiectare a codului cu evidenţierea firelor de execuţie în cadrul unui *warp* se pot aduce îmbunătăţiri considerabile performanţelor aplicaţiei.

Nvidia a anunţat lansarea pe piaţă a unei noi versiuni a arhitecturii CUDA. Sub numele de cod Fermi, aceasta va oferi suport pentru un număr de până la 512 unităţi funcţionale CUDA, organizate în 16 SM a câte 32 de nuclee fiecare (de patru ori mai multe decât în arhitecturile actuale). Fiecare nucleu va dispune de o unitate ALU cu o precizie pe 32 de biţi pentru toate instrucţiunile, o unitate de calcul în virgulă mobilă – FPU, 16 unităţi de stocare, 4 unităţi funcţionale speciale destinate funcţiilor de tipul sinus, cosinus, rădăcină pătrată, funcţie inversă, o unitate *Dual Warp Scheduler* ce va permite planificarea şi execuţia concurentă a două grupuri *warps* şi 64 KB memorie locală comună – un mijloc excelent de a reduce semnificativ traficul extern şi de a asigura reutilizarea datelor interne.Totodată, Nvidia promite că va fi primul procesor grafic ce va oferi suport pentru detecţia şi corectarea erorilor – *ECC* (*Error Correcting Code*) *Memory Support*[6]. Odată cu introducerea tehnologiei Fermi, Nvidia se angajează să asigure suport şi pentru operaţii în virgulă mobilă cu dublă precizie şi o cadenţă a instrucţiunilor de tipul adunare sau înmulţire de 16 operaţii pe ciclu de ceas. Una dintre secvenţele de instrucţiuni des întâlnite în grafică, algebră liniară sau aplicaţii ştiinţifice dedicate este înmulţirea a două numere şi adunarea produsului cu al treilea număr (D = A× B + C). Arhitecturile anterioare permit accelerarea operaţiilor printr-o unitate MAD (M*ultiply – Add*) şi execuţia în cadrul unui singur ciclu de ceas, însă prin trunchierea rezultatului înmulţirii. Conform producătorului, Fermi va implementa o nouă unitate FMA (*Fused Multiply – Add*) prin intermediul căreia sunt evitate pierderile intermediare de informaţie.

Dezvoltată ca o continuare a unei platforme stabile, Fermi promite să deschidă noi perspective pentru procesarea la nivelul GPU. Prin valorificarea resurselor hardware şi software pentru prelucrarea paralelă, resurse proiectate iniţial pentru grafică 3D, programatorii pot oferi soluţii celor mai variate probleme din diverse domenii. O parte a aplicaţiilor, precum prelucrarea imaginii sau codarea / recodarea video, implică în continuare procesare grafică, însă, dacă pentru CPU timpul necesar procesării este de ordinul orelor, în cazul prelucrării la nivelul unui GPU cu suport CUDA sau Fermi durata poate fi redusă la câteva minute. Mai mult chiar, odată cu lansarea acestei tehnologii, vor putea rula în paralel algoritmi foarte complecşi, într-un timp foarte scurt şi cu o precizie uimitoare întrucât, pe lângă suportul hardware prezentat anterior, Fermi promite să ofere posibilitatea execuţiei concurente a *kernel-urilor* (spre deosebire de CUDA, unde execuţia concurentă este limitată numai la nivelul *thread-urilor* din cadrul aceluiaşi *kernel*) – caracteristică denumită de Nividia procesare *PhysX* (figura 2.2.). Această tehnologie va permite şi programelor care execută un număr redus de *kernel-uri* să exploateze întreaga putere a procesorului grafic.

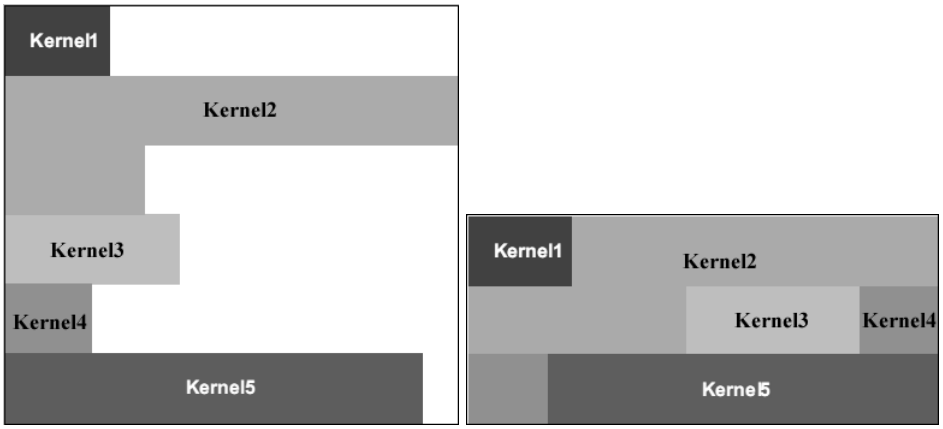

Figura 2.2. Execuţie secvenţială vs. execuţie concurentă funcţii *kernel*

Pornind de la specificaţiile comunicate de producător, Fermi pare că va surclasa cu uşurinţă arhitecturile scoase pe piaţă de către cei mai importanţi competitori din domeniul procesoarelor

grafice. Totuşi, răspunsul la întrebările dacă această arhitectură face mult aşteptatul pas către estomparea diferenţelor dintre GPU şi CPU şi dacă oferă posibilitatea înlocuirii unuia cu celălalt, este în continuare negativ. Chiar dacă, aparent, se mai vorbeşte numai despre câteva mari provocări rămase nerezolvate în tehnologia GPU (printre care dimensiunea relativ mică a memoriei GPU şi operaţii I/O indirecte către memoria GPU), viitorul apropiat pare să rezerve un loc bine stabilit atât pentru GPU, cât şi pentru CPU în domeniul procesării.

Aşa cum era de aşteptat, avantajele aduse de arhitectura CUDA nu au trecut neobservate în mediul profesional. În martie 2009, MotionDSP Inc. lansează pe piaţă *vReveal* [7] – tehnologie revoluţionară de recodare video ce permite utilizatorilor îmbunătăţirea semnalului video al *clip*-urilor filmate cu telefoane mobile, camere video sau digitale; în aprilie 2009, Nero anunţă lansarea unei versiuni actualizate a aplicaţiei „Nero Move it" care permite reducerea timpului de codare video de până la cinci ori; în august 2009, Nvidia anunţă un parteneriat cu Lowry Digital pentru a restaura înregistrarea aselenizării navetei NASA Apollo 11, iar exemplele nu se opresc aici [7]. Toate acestea au în comun suportul oferit de tehnologia CUDA, ce permite dezvoltarea şi utilizarea unor algortimi de procesare de o complexitate mult mai mare decât ar fi fost posibil altfel şi un timp de execuţie de zeci, chiar sute de ori mai mic.

3. Concluzii

Deşi GPU este deja un veteran pe piaţa procesoarelor, exploatarea puterii sale de prelucrare a devenit, din ce în ce mai mult, o provocare, atât în spectrul cercetării, cât şi în rândurile arhitecţilor IT sau ale celor pasionaţi de arta programării. Unul dintre motivele principale pentru care procesoarele grafice au reuşit să pătrundă pe o piaţă deja dezvoltată este de natură economică. Deşi procesul de fabricaţie nu este nici mai uşor, nici mai puţin costisitor decât în cazul CPU, procesoarele grafice au avut avantajul acoperirii unei nişe pe cât de specializate, pe atât de atractive pentru un număr de utilizatori în continuă creştere: piaţa jocurilor. La momentul actual, din ce în ce mai multe domenii (medicină, astronomie, chimie, fizică, modelare matematică, aviaţie civilă sau militară, biologie, criminalistică, criptologie, mecanică etc.) beneficiază de optimizarea timpului de procesare prin exploatarea puterii procesorului grafic. Atragerea utilizatorilor a continuat şi dincolo de graniţele celor fascinaţi de efectele vizuale, prin lansarea pe piaţă a arhitecturilor ce oferă suport pentru rezolvarea problemelor de calcul complex şi de modelare sau simulare a sistemelor din domenii de interes general. Plăcile video, existente acum pe piaţă, depăşesc cu mult performanţele oricărei unităţi centrale de procesare, când vine vorba despre calcule în virgulă mobilă – lucru de altfel normal, dacă avem în vedere modul de proiectare specific *chip*-urilor grafice, în care accentul a fost pus pe procesarea efectivă a datelor şi nu pe control, aşa cum s-a întâmplat, de-a lungul timpului, în cazul procesorului central.

Lansată pe piaţă sub numele *Compute Unified Device Architecture* sau CUDA, platforma de la Nvidia oferă, poate pentru prima dată, un mediu aproape complet de dezvoltare şi optimizare a aplicaţiilor complexe pe GPU. Pe de o parte, prin CUDA C, modelul de programare este accesibil chiar şi celor nefamiliarizaţi cu metode avansate de prelucrare paralelă, iar pe de altă parte, OpenCL oferă suport pentru programare *low – level*, permiţând astfel obţinerea unor performanţe semnificative în comparaţie cu rezultatele obţinute pe CPU. Mai mult decât atât, arhitectura hardware este disponibilă pe ultimele plăci video proiectate şi lansate pe piaţă (începând cu seria GeForce 8), iar pachetul software este disponibil gratuit, pe site–ul producătorului.

În cadrul acestei lucrări am urmărit prezentarea celor mai importante aspecte legate de procesarea grafică şi de ultimele generaţii de arhitecturi lansate de Nvidia. Deoarece activitatea de cercetare a colectivului de autori are drept obiectiv dezvoltarea şi optimizarea unor algoritmi de calcul paralel distribuit pentru procesarea imaginii, într-un articol viitor vom prezenta un studiu de caz asupra potenţialului de performanţă al arhitecturii CUDA pentru calculul paralel al histogramei unei imagini pe 256 niveluri de gri şi vom evidenţia diferenţele dintre rezultatele obţinute în urma prelucrării algoritmului pe unitatea centrală de prelucrare, respectiv pe procesorul grafic.

BIBLIOGRAFIE

1. **CARLSON, W.**: A Critical History of Computer Graphics and Animation, Universitatea de Stat din Ohio, documentaţie Internet, http://design.osu.edu/carlson/history/lessons.html
2. **GRAMA, A., A. GUPTA, G. KARYPIS, V. KUMAR**: Introduction to Parallel Computing, Second Edition, Pearson Education, 2003
3. **HALFHILL, T.**: Parallel Processing with CUDA – Nvidia's High – Performance Computing Platform Uses Massive Multithreading, 2008, documentaţie Internet, http://www.nvidia.com/docs/IO/55972/220401_Reprint.pdf
4. **KIRK, D., W. HWU**: Programming Massively Parallel Processors, Curs Universitatea din Illinois, 2007
5. **WILKINSON, B., M. ALLEN**: Parallel Programming – Techniques and Applications Using Networked Workstations and Parallel Computers, Second Edition , Pearson Education, 2005
6. ***: Nvidia's Next Generation CUDA Compute Architecture: Fermi Version 1.1, Nvidia, 2009, documentaţie Internet, http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf
7. ***: Nvidia Documentation, 2009, documentaţie Internet, http://www.nvidia.com