

# ARHITECTURI COMPLEXE FOLOSITE ÎN PRELUCRAREA PARALELĂ A IMAGINILOR

dr. ing.                      prof. dr. ing.              prof. dr. ing.              as. drd. ing.              ing.  
Ștefan 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" București

**Rezumat:** Simulatoare de antrenament, proiectare în diferite domenii de activitate, simulare și vizualizare științifică, practică medicală sau „numai” jocuri 3D și filme de animații, toate au în comun puterea de procesare de pe placa video. Susținute de avansul tehnologic, dar și datorită cererii pieței pentru aplicații care să ruleze la rezoluții mari și cu performanțe 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 își propune prezentarea succintă a unei arhitecturi performante de calcul paralel (CUDA) și evidențierea posibilităților pe care le oferă în diverse domenii, urmând ca un articol viitor să dezvolte studii de caz pentru cercetarea potențialului de performanță al acestora în vederea implementării unor algoritmi optimizați pentru recunoașterea, clasificarea și 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 specialiști în matematică, informatică, proiectare și chiar artiști, în încercarea de a crea o realitate virtuală bine conturată, care să îmbine caracteristicile mediului real cu informații generate de calculator. Fie că vorbim despre sisteme de realitate îmbogățită – *Augmented Reality* (întâlnite adesea în domenii precum medicină, robotică sau aviația militară) în care un dispozitiv de tipul HMD (*Head Mounted Display*) suprapune date generate de calculator peste imaginea mediului real, despre sisteme de teleprezență cu ajutorul cărora operatorul poate observa și controla acțiunile 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 neprețuit. Toate aceste sisteme au la bază procesarea grafică, adică ansamblul metodelor și tehnicilor de conversie a datelor către și de la un dispozitiv grafic prin intermediul calculatorului.

Astfel, procesarea imaginilor a reprezentat un punct de interes comun multor comunități științifice, urmărindu-se continuu reducerea timpului de procesare și proiectarea unor algoritmi optimizați de prelucrare. În contextul evoluției din punct de vedere tehnologic, procesarea secvențială 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 cerințele de performanță din ce în ce mai mari ale aplicațiilor, au fost unele dintre cele mai convingătoare argumente care au susținut, de-a lungul timpului, acest model de programare aplicat deja cu succes în algoritmi de proiectare și în aplicații din domeniul ingineriei (aviație, mecanică, motorizări, circuite integrate, sisteme microelectromecanice și nanoelectromecanice), din domeniul științific (bioinformatică, fizică, chimie, astronomie, modelare matematică) sau din gama aplicațiilor comerciale (servere web și baze de date, extragerea informațiilor, optimizarea deciziilor din domeniul afacerilor sau de pe diverse piețe)

ori a sistemelor de calculatoare (sisteme embedded, rețele, sisteme de detecție și prevenire a pătrunderii virusilor)[2]. Pentru soluționarea multor probleme de calcul paralel au fost dezvoltați algoritmi specifici ce au evoluat odată cu cerințele aplicațiilor. Un caz particular, în care modelul de programare paralelă a fost aplicat cu succes este procesarea imaginii – operații de pre-procesare, recunoașterea formelor, identificarea obiectelor sau transformări discrete Fourier[5]. Dezvoltată la granița dintre procesarea grafică și algoritmi paraleli, prelucrarea imaginilor a reprezentat un salt uriaș atât pentru cercetare, cât și pentru inginerie, motiv pentru care tot mai multe eforturi au fost îndreptate către optimizarea acestor calcule și reducerea timpului de execuție a algoritmilor. Domenii precum medicina, astronomia, biologia sau criminalistica au la bază utilizarea și 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 spațiul pentru cabina pilotului. Astfel, a fost realizată prima imagine digitală ortogonală a corpului uman (figura 1.1) și primele animații 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 Laboratories și-au îndreptat atenția către captarea diverselor forme geometrice și reproducerea acestora cu ajutorul tehnologiei existente. Drept urmare, au luat naștere lucrările lui Michael Noll (1962), ale matematicianului Frieder Nake (1965) și 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 (rețele de calculatoare, transferuri de date, imprimare laser), însă una dintre cele mai importante inovații ale momentului a fost realizarea unei interfațe grafice cu utilizatorul (*GUI – Graphical User Interface*). O astfel de interfață putea aduce calculatorul în viața cotidiană, iar simpla afișare a unor linii de cod sau a unor comenzi devenea istorie, cedând locul reprezentărilor grafice și 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 și de control al datelor, inginerii de la Xerox și-au îndreptat atenția către procesarea grafică și comunicații. Este adevărat că prima interfață grafică realizată de ei era greu de utilizat și dependentă de puterea procesorului central pentru fiecare modificare a unui bit; însă primul pas era făcut. A urmat dezvoltarea unor algoritmi din ce în ce mai performanți pentru mișcarea ferestrelor pe ecran într-un timp cât mai scurt și fără a supra-încărca procesorul. Au apărut circuite electronice specializate în controlul procesării grafice, cu rolul de a descongiona unitatea centrală de prelucrare în momentul utilizării aplicațiilor grafice. Spre sfârșitul anilor 1970, pe fondul unor factori externi precum rezistența guvernului, recesiunea economică și chiar competiția nelocală, Xerox nu a mai putut ține pasul. Decăderea uneia dintre cele mai importante companii în domeniu și proiectarea, cu precădere, a circuitelor electronice destinate calculelor și sarcinilor numerice, au făcut ca anii 1980 să nu aducă îmbunătățiri semnificative în domeniul procesării

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

Deși procesoarele centrale au avut o evoluție cu adevărat spectaculoasă încă din anii 1980, despre *chip*-urile de procesare grafică nu se poate spune același 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 competiții importante pe această piață. 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 aceluiași microprocesor. Tot acum sunt lansate și încearcă să se impună ca standarde, cele mai cunoscute API-uri din domeniu (OpenGL și DirectX) și apar primele tentative de procesoare 3D care, în realitate, nu erau decât versiuni îmbunătățite ale procesoarelor 2D (motiv pentru care au eșuat). Primii care au reușit să beneficieze de avantajele unui procesor 3D au fost cei de la 3dfx cu prototipul Voodoo. Cu toate că performanțele acestuia erau mult mai bune decât ale predecesorilor săi, o mare parte a calculelor matematice încă se făceau la nivelul coprocesorului matematic.

La sfârșitul 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 funcții geometrice de tipul T&L (*Transform, clipping, and lighting*) și să accelereze grafic bibliotecile Direct3D 7.0. A fost primul procesor grafic cu performanțe excepționale care a surclasat cu ușurință toate plăcile video existente pe piață în momentul lansării. Au urmat generațiile GeForce 2 care au adus o creștere a frecvenței până la 200MHz, dar și o unitate TMU (*Texture Map Unit*) și GeForce 3 care oferă, pentru prima dată, suport pentru programarea anumitor unități funcționale.

Ca și în cazul procesoarelor centrale, concurența a obligat competitorii să-și perfecționeze produsele, să lanseze noi versiuni sau chiar să proiecteze alte prototipuri mai performante. Astfel, în paralel cu generațiile 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 generație GeForce și de către ATI pentru plăcile Radeon HD 2000, apare o flexibilitate a arhitecturii, iar unitățile funcționale sunt folosite mult mai eficient. Acesta a fost momentul în care, printr-un nou proces de fabricație și prin creșterea numărului de unități funcționale (în contrast cu strategia de proiectare a CPU, unde lucrurile s-au concentrat pe îmbunătățirea *pipeline*-ului, respectiv paralelizare la nivel de instrucțiune), *chip*-urile grafice au devenit mult mai performante decât unitățile centrale de prelucrare. Diferențele de performanță se datorează faptului că într-un GPU majoritatea tranzistorilor sunt dedicați procesării datelor, iar în cazul procesorului central sunt folosiți în special pentru memorii tampon și control. Așadar, susținute de avansul tehnologic, dar și datorită cererii pieței pentru aplicații 3D care să ruleze la rezoluții mari și cu performanțe HD, procesoarele grafice au evoluat foarte mult, transformându-se în arhitecturi complexe, *multithreaded*, multi-core, cu putere mare de procesare și cu canale de comunicație cu lățime mare de bandă. Mai mult, s-a constatat faptul că procesoarele grafice nu sunt exploatate suficient și că ar putea fi redirecționate 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 facilitățile oferite programatorilor de către producătorii plăcilor video au făcut acest domeniu mult mai accesibil cercetării și chiar utilizării în aplicații de interes general doar în ultimii ani. Odată cu lansarea pe piață a modelului GeForce 8, Nvidia introduce și tehnologia CUDA (*Compute Unified Device Architecture*) prin intermediul căreia placa video este privită

ca un coprocesor matematic al procesorului central. În 2007, pe lângă interfața de comunicare cu procesorul, a fost lansat și un pachet software complet (compilator, biblioteci, documentație) care a ușurat munca utilizatorilor. De asemenea, platforma CUDA pune la dispoziția dezvoltatorilor de aplicații un API specific, denumit CUDA API, care folosește limbaje de programare adaptate arhitecturii paralele.

## 2. Platforma CUDA

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

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

Firele de execuție pot fi identificate și 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 execuția mai multor blocuri este posibilă în orice ordine, secvențial sau paralel. Un grid reprezintă, de fapt, un șir de blocuri care execută același *kernel*, citesc date din memoria globală și 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, regiștrii, o locație proprie în memoria comună, date de intrare și date de ieșire. Fiecare bloc are alocat un spațiu de memorie comună, vizibil tuturor firelor de execuție din bloc. Pe lângă accesul la memoria comună, toate firele de execuție au drept de citire a memoriei de constante și a celei de texturi.

### 2.1 Interfețe de programare

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

Interfața C pentru CUDA expune modelul de programare sub forma unui set minimal de extensii ale limbajului clasic C, în care programatorul poate defini funcții (*kernels*) care, atunci când sunt apelate, sunt executate de  $N$  ori în paralel de către  $N$  fire de execuție CUDA pe procesorul grafic, spre deosebire de modul clasic, în care erau executate o dată, pe CPU. Un *kernel* se definește folosind declarația `_global_` înaintea numelui funcției și specificând numărul firelor de execuție 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];
}

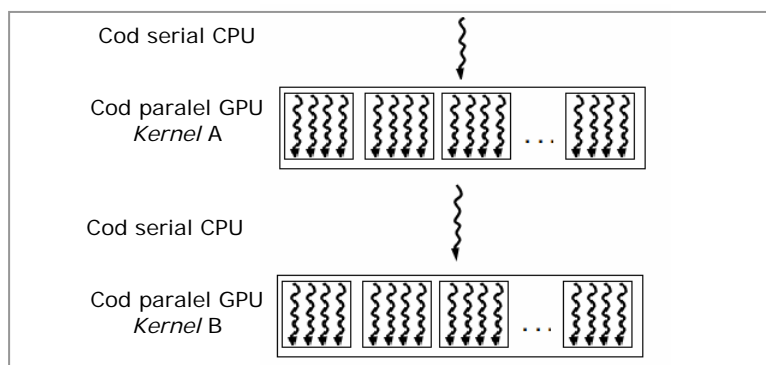
int main ()
{
    //Kernel invocation
    VecAdd<<<1, N>>> (A, B, C);
}

```

Orice fișier sursă care conține extensii, trebuie compilat cu *nvcc*, compilator la nivel de driver, pus la dispoziție gratuit de Nvidia. Același program CUDA poate conține cod mixt, adică atât cod destinat procesorului grafic, cât și 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 execuției pe CPU) care, în urma unei compilări cu instrumente software specifice CPU, va rula ca un proces normal, fie cod PTX (*Parallel Thread eXecution*) care va fi compilat cu *nvcc* și va rula pe GPU. Compilatorul *nvcc* suportă și cod C++, destinat funcțiilor ce rulează pe CPU, însă concepte de programare orientată pe obiecte precum noțiunea de clasă sau de moștenire încă nu au fost implementate (conform Nvidia CUDA Programming Guide v2.3.1). Totuși, Nvidia promite să ofere suport complet pentru C++, dar și optimizări pentru OpenCL, Fortran sau DirectCompute pentru versiunile ulterioare.

CUDA driver API este o interfață care oferă funcții 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 execuție și transmiterea rezultatelor execuției către CPU.

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



**Figura 2.1. Execuția unui program CUDA**

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

resurselor. În afară de CUDA driver API, interfața generică de programare a unităților de tip *device* permite scrierea funcțiilor folosind DirectX Compute sau OpenCL.

Încă de la lansarea pe piață a primelor produse dedicate procesării paralele pe GPU, Nvidia a reușit să se impună cu un avantaj de invidiat pe atunci: „a ascuns” programatorului arhitectura hardware în spatele unei interfețe de programare, oferindu-i funcții predefinite pentru cazurile în care nu vrea să lucreze direct la nivel hardware. Acest lucru a permis dezvoltarea aplicațiilor dedicate GPU fără implicarea programatorilor în gestiunea resurselor hardware, dar și o compatibilitate fără costuri foarte mari a aplicațiilor vechi pe platforme noi. Odată cu creșterea interesului față de noile tehnologii și cu diversificarea domeniilor de aplicabilitate, necesitatea gestionării de către programator a resurselor în vederea obținerii unor performanțe net superioare a condus la lansarea unui set complet de instrumente software numit Nvidia OpenCL. Pentru asigurarea corespondenței dintre principiile de programare OpenCL și suportul hardware, fiecare multi-procesor al GPU a fost proiectat ca un corespondent al unității de calcul OpenCL. Un multi-procesor execută un *thread* CUDA pentru fiecare instrucțiune OpenCL, respectiv un bloc de *thread-uri* pentru un grup de instrucțiuni. Pentru obținerea unor rezultate cât mai bune, se recomandă un grad cât mai mare de paralelizare a codului secvențial, reducerea transferurilor de date dintre CPU și GPU chiar și atunci când acest lucru presupune execuția unor funcții *kernel* pe GPU fără obținerea unor performanțe semnificativ mai mari în comparație cu rularea pe CPU, reducerea accesului la memoria globală și asigurarea unui acces omogen ori de câte ori apare necesitatea accesării acesteia, evitarea utilizării mai multor căi de execuție în cadrul aceluiași *warp* și optimizarea instrucțiunilor.

Pentru dezvoltarea aplicațiilor care să exploateze noua arhitectură, programatorii au la dispoziție un set complet de instrumente software, exemple și documentație, oferite gratuit: *runtime C* care oferă suport pentru execuția funcțiilor C standard pe GPU și permite trecerea către limbaje de nivel înalt precum Java, Fortran sau Python; Nvidia Nexus[7] – primul mediu de dezvoltare proiectat exclusiv pentru aplicații CUDA C; OpenCL și DirectX Compute; biblioteci care includ funcții matematice complexe optimizate pentru arhitectura CUDA; compilator Nvidia C (*nvcc*), mediu de depanare CUDA (*cuda-gdb*); ghid de programare (*CUDA Programming Guide*); specificații API și exemple de funcții *kernel* care ilustrează tehnici de programare pentru o gamă largă de algoritmi și aplicații.

## 2.2 Implementare hardware

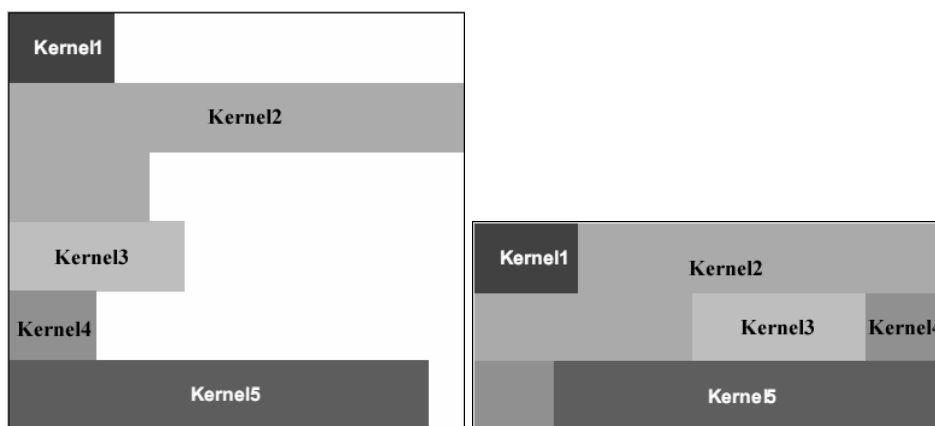
Din punct de vedere hardware, platforma de dezvoltare a aplicațiilor GPGPU presupune lucrul cu unul sau mai multe procesoare grafice (*devices*) și procesorul central (*host*) al sistemului. Astfel, un program poate fi considerat un ansamblu de sub-programe care pot rula fie pe GPU, fie pe CPU. În general,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

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

instrucțiune către firele de execuție active ale acestuia. Deși programatorul poate ignora modul intern de funcționare a unității SIMT, printr-o proiectare a codului cu evidențierea firelor de execuție în cadrul unui *warp* se pot aduce îmbunătățiri considerabile performanțelor aplicației.

Nvidia a anunțat lansarea pe piață a unei noi versiuni a arhitecturii CUDA. Sub numele de cod Fermi, aceasta va oferi suport pentru un număr de până la 512 unități funcționale 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 biți pentru toate instrucțiunile, o unitate de calcul în virgulă mobilă – FPU, 16 unități de stocare, 4 unități funcționale speciale destinate funcțiilor de tipul sinus, cosinus, rădăcină pătrată, funcție inversă, o unitate *Dual Warp Scheduler* ce va permite planificarea și execuția concurrentă a două grupuri *warps* și 64 KB memorie locală comună – un mijloc excelent de a reduce semnificativ traficul extern și de a asigura reutilizarea datelor interne. Totodată, Nvidia promite că va fi primul procesor grafic ce va oferi suport pentru detecția și corectarea erorilor – *ECC (Error Correcting Code) Memory Support*[6]. Odată cu introducerea tehnologiei Fermi, Nvidia se angajează să asigure suport și pentru operații în virgulă mobilă cu dublă precizie și o cadență a instrucțiunilor de tipul adunare sau înmulțire de 16 operații pe ciclu de ceas. Una dintre secvențele de instrucțiuni des întâlnite în grafică, algebră liniară sau aplicații științifice dedicate este înmulțirea a două numere și adunarea produsului cu al treilea număr ( $D = A \times B + C$ ). Arhitecturile anterioare permit accelerarea operațiilor printr-o unitate MAD (*Multiply – Add*) și execuția în cadrul unui singur ciclu de ceas, însă prin trunchierea rezultatului înmulțirii. Conform producătorului, Fermi va implementa o nouă unitate FMA (*Fused Multiply – Add*) prin intermediul căreia sunt evitate pierderile intermediare de informație.

Dezvoltată ca o continuare a unei platforme stabile, Fermi promite să deschidă noi perspective pentru procesarea la nivelul GPU. Prin valorificarea resurselor hardware și software pentru prelucrarea paralelă, resurse proiectate inițial pentru grafică 3D, programatorii pot oferi soluții celor mai variate probleme din diverse domenii. O parte a aplicațiilor, 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 complecși, într-un timp foarte scurt și cu o precizie uimitoare întrucât, pe lângă suportul hardware prezentat anterior, Fermi promite să ofere posibilitatea execuției concurente a *kernel-urilor* (spre deosebire de CUDA, unde execuția concurrentă este limitată numai la nivelul *thread-urilor* din cadrul aceluiși *kernel*) – caracteristică denumită de Nvidia procesare *PhysX* (figura 2.2.). Această tehnologie va permite și programelor care execută un număr redus de *kernel-uri* să exploateze întreaga putere a procesorului grafic.



**Figura 2.2. Execuție secvențială vs. execuție concurentă funcții *kernel***

Pornind de la specificațiile comunicate de producător, Fermi pare că va surclasa cu ușurință arhitecturile scoase pe piață de către cei mai importanți competitori din domeniul procesoarelor

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

Așa cum era de așteptat, avantajele aduse de arhitectura CUDA nu au trecut neobservate în mediul profesional. În martie 2009, MotionDSP Inc. lansează pe piață *vReveal* [7] – tehnologie revoluționară de recodare video ce permite utilizatorilor îmbunătățirea semnalului video al clipurilor filmate cu telefoane mobile, camere video sau digitale; în aprilie 2009, Nero anunță lansarea unei versiuni actualizate a aplicației „Nero Move it” care permite reducerea timpului de codare video de până la cinci ori; în august 2009, Nvidia anunță un parteneriat cu Lowry Digital pentru a restaura înregistrarea aselenizării navei NASA Apollo 11, iar exemplele nu se opresc aici [7]. Toate acestea au în comun suportul oferit de tehnologia CUDA, ce permite dezvoltarea și utilizarea unor algoritmi de procesare de o complexitate mult mai mare decât ar fi fost posibil altfel și un timp de execuție de zeci, chiar sute de ori mai mic.

### 3. Concluzii

Deși GPU este deja un veteran pe piața procesoarelor, exploatarea puterii sale de prelucrare a devenit, din ce în ce mai mult, o provocare, atât în spectrul cercetării, cât și în rândurile arhitecților IT sau ale celor pasionați de arta programării. Unul dintre motivele principale pentru care procesoarele grafice au reușit să pătrundă pe o piață deja dezvoltată este de natură economică. Deși procesul de fabricație nu este nici mai ușor, nici mai puțin costisitor decât în cazul CPU, procesoarele grafice au avut avantajul acoperirii unei nișe pe cât de specializate, pe atât de atractive pentru un număr de utilizatori în continuă creștere: piața jocurilor. La momentul actual, din ce în ce mai multe domenii (medicină, astronomie, chimie, fizică, modelare matematică, aviație civilă sau militară, biologie, criminalistică, criptologie, mecanică etc.) beneficiază de optimizarea timpului de procesare prin exploatarea puterii procesorului grafic. Atragerea utilizatorilor a continuat și dincolo de granițele celor fascinați de efectele vizuale, prin lansarea pe piață a arhitecturilor ce oferă suport pentru rezolvarea problemelor de calcul complex și de modelare sau simulare a sistemelor din domenii de interes general. Plăcile video, existente acum pe piață, depășesc cu mult performanțele oricărei unități 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 și nu pe control, așa cum s-a întâmplat, de-a lungul timpului, în cazul procesorului central.

Lansată pe piață sub numele *Compute Unified Device Architecture* sau CUDA, platforma de la Nvidia oferă, poate pentru prima dată, un mediu aproape complet de dezvoltare și optimizare a aplicațiilor complexe pe GPU. Pe de o parte, prin CUDA C, modelul de programare este accesibil chiar și celor nefamiliarizați cu metode avansate de prelucrare paralelă, iar pe de altă parte, OpenCL oferă suport pentru programare *low – level*, permițând astfel obținerea unor performanțe semnificative în comparație cu rezultatele obținute pe CPU. Mai mult decât atât, arhitectura hardware este disponibilă pe ultimele plăci video proiectate și lansate pe piață (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 și de ultimele generații de arhitecturi lansate de Nvidia. Deoarece activitatea de cercetare a colectivului de autori are drept obiectiv dezvoltarea și optimizarea unor algoritmi de calcul paralel distribuit pentru procesarea imaginii, într-un articol viitor vom prezenta un studiu de caz asupra potențialului de performanță al arhitecturii CUDA pentru calculul paralel al histogramei unei imagini pe 256 niveluri de gri și vom evidenția diferențele dintre rezultatele obținute 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, documentație 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, documentație Internet, [http://www.nvidia.com/docs/IO/55972/220401\\_Reprint.pdf](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, documentație Internet, [http://www.nvidia.com/content/PDF/fermi\\_white\\_papers/NVIDIA\\_Fermi\\_Compute\\_Architecture\\_Whitepaper.pdf](http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf)
7. \*\*\*: Nvidia Documentation, 2009, documentație Internet, <http://www.nvidia.com>

