Paralelizacija algoritama na heterogenim platformama kroz sustav OpenCL

Veljko Dragšić
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave
Fakultet elektrotehnike i računarstva
Unska 3, 10000 Zagreb, Hrvatska
Tel: 00 385 (0)95 8126 991, E-mail: veljko@kset.org

***Sažetak*—Sustav OpenCL olakšava paralelizaciju algoritama i pritom cilja na sklopovski heterogene platforme, konkretno centralne i grafičke mikroprocesore. Sustav je zanimljiv jer je prvi takve vrste, a pojavio se je 2009. godine. U ovom radu je kroz praktične primjere ispitana funkcionalnost i performanse samog sustava. Također su obrađene i trenutno dostupne platforme, točnije tipovi mikroprocesora, koje OpenCL podržava. Kroz dobivene rezultate je OpenCL uspoređen sa drugim sustavima, a posebna je pažnja usmjerena na usporedbu današnjih mikroprocesora i rezultata koje postižu ovisno o tipu problema.**

I. Uvod

Centralni mikroprocesor slijedno prolazi kroz strojni jezik programa i izvršava naredbe. Kako nikada neće postojati dovoljno brz mikroprocesor, logično je više procesora udružiti kako bi brže izveli željeni algoritam. Za tu priliku je potrebno slijedne algoritme paralelizirati, što ponekad nije nimalo jednostavan zadatak, te postoje različite tehnologije i principi kako bi se to ostvarilo.

Danas imamo na raspolaganju više tipova mikroprocesora specijaliziranih za različite probleme, poput centralnog ili grafičkog mikroprocesora. Time se pojavljuje problem prenosivosti programa između različitih platformi zbog njihove popriličnо različite arhitekture. Paralelizacija algoritama i njihova prenosivost zahtijeva drugačiji pristup dosadašnjem načinu oblikovanja algoritama.

OpenCL je sustav koji olakšava paralelizaciju algoritama, te je prvi koji omogućuje njihovu prenosivost između različitih platformi. Standard se pojavio 2009. godine, svaki proizvođač ga može implementirati jer je specifikacija otvorenog tipa, te se sastoji od sučelja za pristup sklopovlju (mikroprocesorima), biblioteka i *OpenCL C* programskog jezika.

II. OpenCL

Do ne tako davno se za procesorski zahtjevne algoritme oslanjalo isključivo na centralne mikroprocesore (engl. CPU), a od 2007. je počelo iskorištavanje i specijaliziranih grafičkih mikroprocesora (engl. GPU) u općenitije svrhe. Zadnjih nekoliko godina imamo na raspolaganju višejezgrene CPU-ove, a GPU-ovi su sami po sebi paralelizirane arhitekture, u oba slučaja se nameće potreba za paralelnim algoritmima. *Open Computing Language* cilja upravo na to područje, paralelizaciju algoritama i heterogene platforme.

Razvoj *OpenCL-a* je započela tvrtka *Apple*, a 2008. godine je upravljanje standardnom prepušteno konzorciju *Khronos* koji okuplja sve relevantnije kompanije u području mikroprocesora i multimedije, *Apple*, *AMD/ATI*, *IBM*, *Intel* i *Nvidia*. 2009. godine su se pojavile prve implementacije, podrška za CPU i GPU u *Mac OS X Snow Leopardu*, *Nvidia* je podržala svoje GPU-ove kroz upravljačke programe (engl. *drivers*), a isto tako i *AMD/ATI* svoje CPU-ove i GPU-ove.

Standard *OpenCL* cilja na područje od ugradbene (engl. *embedded*) i potrošačke (engl. *consumer*) programske opreme, pa sve do razine računarstva visokih performansi (engl. *High performance Computing, HPC*). Sustav je zadržan na niskoj razini koja je blizu samog sklopovlja, što omogućuje postizanje izrazito dobrih performansi.

III. Arhitektura OpenCL-a

Sustav se sastoji od hijerarhije 4 modela, platformskog, memorijskog, izvršnog i programskog.

A. Platformski model

Sastoji se od domaćina (osobno računalo) sa jednim ili više računalnih uređaja (engl. *compute device*), npr. CPU i/ili GPU. Svaki računalni uređaj može imati više računalnih jedinica (engl. *computing unit*), npr. jezgra CPU-a ili multiprocesor GPU-a, a svaka računalna jedinica može imati više procesirajućih elemenata (engl. *processing element*), poput procesora dretvi kod GPU-a.

B. Izvršni model

Izvođenje *OpenCL* aplikacije se odvija u dva dijela: programske jezgre (engl. *kernel*) se izvode na jednom ili više računalnih uređaja, a na domaćinu se izvodi program koji upravlja izvođenjem programskih jezgri, točnije brine se za stvaranje konteksta (engl. *contex*) za izvođenje, programskih slijedova (engl. *programming queues*) i stvaranje memorijskih međuspremnika (engl. *memory buffer*) za pisanje i čitanje podataka iz memorije uređaja.

Prilikom pokretanja *OpenCL* jezgri se mora definirati (globalna) veličina problema, za primjer možemo zamisliti broj redaka i stupaca matrice. Sustav zatim stvara indeksni prostor koji odgovara prije definiranim dimenzijama, te se za svaki njegov element pokreće po jedna radna jedinica (engl. *working-item*), tj. *OpenCL* jezgra koja se izvodi nad elementom matrice. Radne jedinice su grupirane u radne grupe (eng. *working-group*), a veličine radnih grupa

se također mogu definirati prilikom pokretanja *OpenCL* jezgri kroz lokalne veličine problema. Navedeni princip omogućava bolju granulaciju problema.

Svaka radna jedinica se može jednoznačno identificirati u indeksnom prostoru pomoću globalnih indeksa (engl. *global ID*). Svaka radna grupa se također može identificirati pomoću indeksa grupa (engl. *work-group ID*). Radne jedinice se također mogu indentificirati i pomoću lokalnih indeksa (engl. *local ID*) i grupa kojim pripadaju, odnosno svaka radna jedinica ima i svoj jednoznačni lokalni indeks u grupi u kojoj se nalazi (jedna radna jedinica se može nalaziti samo u jednog radnoj grupi).

C. Memorijski model

OpenCL poznaje 4 memorijska područja. Globalnoj memoriji mogu pristupati sve radne jedinice, a ona fizički odgovara radnoj memoriji CPU-a ili GPU-a. Konstantna memorija ostaje nepromijenjena za vrijeme izvođenja jezgri, te se za nju brine domaćin. Lokalna memorija je namijenjena radnim jedinicama unutar iste radne grupe, a njezina lokacija je ovisna o implementaciji, kod GPU-a je to priručna memorija mikroprocesora koja je znatno manja od radne memorije, ali joj je pristup daleko brži. Privatna memorija je namijenjena svakoj radnoj jedinici pojedinačno.

D. Programski model

Sustav OpenCL podržava dva programska modela, podatkovnu paralelizaciju (primarni) i paralelizaciju po poslovima. Do sada opisani model je podatkovna paralelizacija, a odvija se po principu SIMD, tj. ista operacija se izvršava nad različitim podacima. U modelu paralelizacije po poslovima programska jezgra se pokreće neovisno o indeksnom prostoru, a pretpostavljamo da jedna radna grupa sadrži samo jednu radnu jednicu. U ovom slučaju korisnik postiže paralelizam koristeći vektorske tipove podataka na OpenCL uređaju i stavljajući u red izvršavanja više poslova, te se brine oko sinkronizacije poslova.

IV. Komponente sustava

Sustav *OpenCL* omogućuje aplikacijama korištenje domaćina i dostupnih *OpenCL* uređaja kao jedno heterogeno paralelno računalo. Sustav se sastoji od sljedećih komponenata:

- *OpenCL* platformskog sloja: omogućuje programima na domaćinu otkrivanje *OpenCL* uređaja i njihovih karakteristika, te stvaranje *OpenCL* konteksta;
- *OpenCL* izvršnog okruženja (engl. *runtime*): omogućuje programu na domaćinu upravljanje kontekstom;
- *OpenCL* prevodioca: služi za stvaranje programa koji sadrže programske jezgre koje se izvršavaju na *OpenCL* uređajima. Tekst programa se piše u *OpenCL C* programskom jeziku.

A. OpenCL C

OpenCL C je proširenje *ISO C99* programskog jezika. Osim što je jezik proširen, tako su i neke njegove funkcionalnosti izbačene, tj. nisu dozvoljene, najčešće zbog načina izvođenja *OpenCL* programskih jezgri. To su pokazivači na funkcije, mogućnost rekurzije, varijabilne duljine nizova, te neke druge specifičnosti. S druge strane proširenja uključuju rad sa radnim jedinicama i grupama, vektorske tipove podataka, mogućnost sinkronizacije, te općenito neka proširenja za paralelizam, baratanje sa slikama, povezivanje sa *OpenGL* sustavom i tako dalje.

V. Arhitekture mikroprocesora

Za rad sa *OpenCL-om* je poželjno poznavati osnove rada mikroprocesora koje danas imamo na raspolaganju u računalima. Postojanje centralnog mikroprocesora je neizbježno za izvršavanje operacijskog sustava i korisničkih aplikacija, ali je i pojava specijaliziranog grafičkog mikroprocesora u računalima vrlo česta. *OpenCL* omogućava prenosivost istog teksta programa između CPU-a ili GPU-a, te je prvi takav sustav.

A. Centralni mikroprocesor - CPU

Osnovni princip funkcioniranja CPU je uglavnom ostao isti do danas, a temelji se na *Von Neumannovoj* arhitekturi. Mikroprocesor dohvaća redom naredbu po naredbu iz memorije, izvodi ju na aritmeto-logičkoj jedinici, a dobiveni rezultat zapiše nazad na memoriju. Radi se o slijednom izvođenju naredbi, a za jednu naredbu je najčešće potrebno po nekoliko ciklusa mikroprocesora. Današnji CPU-ovi su izuzetni kompleksni kako bi se navedeni princip ubrzao, neke od ugrađenih tehnologija su cjevovodi, superskalarnost, višedretvenost, *out-od-order-execution* i velike količine priručne memorije. Zadnjih godina su se pojavili višejezgreni CPU-ovi, a teko to možemo smatrati *"pravim"* paralelizmom na koji programer može imati utjecaj.

B. Grafički mikroprocesor - GPU

Grafički procesori su se pojavili dosta kasnije od CPU-a, tj. tek sredinom '90-ih godina. Kako su specijalizirani za prikaz 3D grafike i njihova arhitektura je u mnogočemu drugačija. Za razliku od CPU-a, GPU je već po arhitekturi paraleliziran, tj. sastoji se od puno (preko 100) procesora dretvi, te relativno male priručne memorije. Razlog je što se kod CPU puno tranzistora iskoristi za *"logiku"* procesora i priručnu memoriju, dok se kod GPU-a više tranzistora iskoristi za aritmetologičke jedinice. Iz tog razloga današnji GPU-ovi, sa jednakim brojem tranzistora kao i CPU-ovi, postižu daleko bolje performanse (za red veličine) u matematičkim kalkulacijama. U nazad zadnjih par godina se GPU počeo koristiti i u općenitije svrhe (engl. *GP/GPU*) od prikaza 3D grafike, te ga danas kroz sustav poput *OpenCL-a* možemo koristiti kao svojevrstan matematički koprocesor CPU-a.

C. Ostale i buduće arhitekture

Zadnjih godina su se pojavile, ali i počele najavljivati, arhitekture koje možemo nazvati *"hibridima"* između CPU-a i GPU-a. Za primjer možemo uzeti *IBM-ov Cell* procesor koji se sastoji od jezgre opće namjene na kojoj se izvršava operacijski sustav, te 8 specijaliziranih jezgri koje služe za matematičke kalkulacije. Jezgra opće namjene je u ulozi hipervizora nad preostalih 8 jezgri sa kojima je povezana

kroz izrazito brzu sabirnicu. Intel je također najavio novu arhitekturu zvanu Larrabbe koja bi se sastojala od većeg broja x86 jezgri.

Na temelju trenutne situacije dostupnih mikroprocesora možemo zaključiti kako u budućnosti možemo očekivati veće iskorištavanje platformi koje se sastoje od neizbježnog mikroprocesora opće namjene za izvršavanje operacijskog sustava, te vrste specijaliziranog mikroprocesora koji bi služio ako dodatni akcelerator. Ništa ne isključuje i korištenje *"hibridnog"* mikroprocesora koji objedinjuje obje arhitekture kroz više različitih jezgri.

VI. Ispitivanja

Za potrebe ispitivanja su napravljeni primjeri množenja matrica u više različitih tehnologija, *OpenCL*, *MPI* i *OpenMP*, te primjer simulacije međudjelovanja čestica u *OpenCL-u*. Testne platforme su bile *Nvidia FX570m* GPU i poslužitelj *Marvin* sa dva *Xeon E5504* CPU-a, svaki sa po 4 fizičke jezgre, bez *Hyperthreading* mogučnosti.

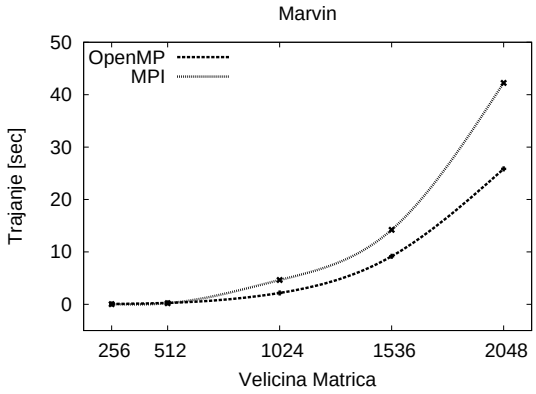
A. Množenje matrica

Ispitivanja su provedena za različite veličine kvadratnih matrica i različite tipove podataka. Za primjer u *OpenCL-u* su napravljene tri programske jezgre, prva jezgra koristi globalni indeksni prostor, druga grupe i lokalni indeksni prostor, a treća je ista kao i druga, ali još dodatno koristi lokalnu *OpenCL* memoriju. *MPI* i *OpenMP* primjeri su izvedeni na *Marvinu*, dok je *OpenCL* primjer izveden na *Nvidia* GPU-u.

Slika 1. prikazuje postignute rezultate primjera u *MPI-u* i *OpenMP-u*. Navedena veličina matrica je broj elemenata po jednoj dimenziji, dakle ukupan broj elemenata matrice odgovara kvadratu navedene vrijednosti. Korišten je tip podataka sa pomičnim zarezom (engl. *float*). U oba primjera su iskorištene sve dostupne jezgre, tj. njih 8. *MPI* je očekivano postigao nešto lošije rezultate zbog same složenosti sustava u odnosu na *OpenMP*, konkretno je riječ o vremenskom trošku raspodjele podataka i sinkronizacije. Rezultati sa cjelobrojnim tipom podataka (engl. *integer*) su postigli jednake rezultate, dok su oni dvostruke preciznosti (engl. *double*) potrajali nešto duže.

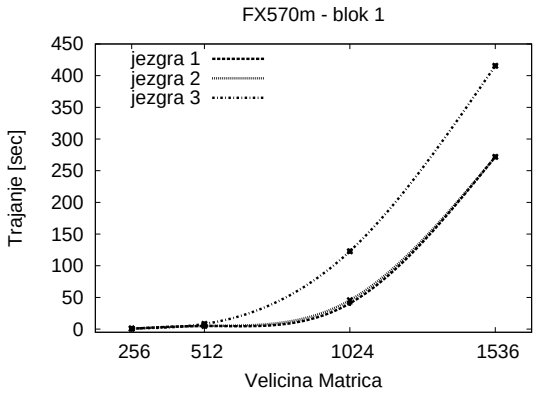
Kod izrade *OpenCL* jezgri posebno treba obratiti pažnju na veličinu lokalnog indeksnog prostora. Njegova veličina može odigrati značajnu ulogu u postizanju performansa zbog načina na koji GPU izvodi dretve odnosno programske jezgre. U slučaju matrica lokalni indeksni prostor možemo zamisliti kao podjelu matrice na podmatrice, npr. matrica dimenzija 16x16 se sastoji od 16 podmatrica veličina 4x4. Ovisno o toj podjeli i GPU grupira dretve, što je važno zbog *"skrivanja"* latencije pristupa memoriji, točnije dok procesor dretvi izvodi jednu dretvu, ostalih nekoliko dohvaća podatke i sprema se na izvođenje, tako se cijeli proces izvođenja značajno ubrzava jer nema *"praznog"* hoda.

Za potrebe ispitivanja su napravljene tri programske jezgre, prva pristupa elementima matrice preko globalnih indeksa, druga koristi indekse grupa i lokalne indekse elemenata unutar tih grupa. Jezgra 3 pristupa elementima na isti načina kao i druga, ali dodatno koristi i lokalnu *OpenCL* memoriju što poboljšava rezultate čak za red veličine.



Slika 1. *OpenMP - MPI*, Marvin

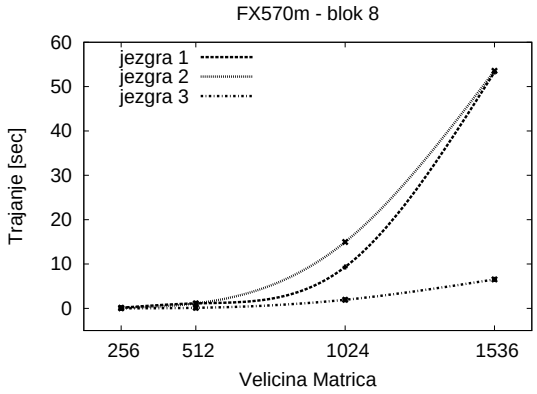
Slika 2. prikazuje postignute rezultate za veličinu lokalnog indeksnog prostora od jedan. Jezgra 3 koja koristi lokalnu memoriju je postigla najlošije rezultate jer dodatno kopira podatke iz globalne u lokalnu memoriju, što nema smisla ako je veličina lokalnog prostora jedan element. Jezgre 1 i 2 su očekivano postigle iste rezultate, jedina razlika je što jezgra 2 koristi grupe i lokalne indekse što nema utjecaja na brzinu.



Slika 2. *OpenCL*, veličina bloka 1

Na slici 3. se vide znatno bolje performanse svih primjera prilikom korištenja većeg lokalnog prostora, konkretno se koriste podmatrice veličine 8x8 elemenata. Jezgra 3 je postigla daleko bolje rezultate jer svaku podmatricu prekopira u lokalnu memoriju, tako da se podacima u globalnoj memoriji pristupa samo za potrebe kopiranja, a zatim se sve obavlja u lokalnoj memoriji kojoj je daleko brži pristup. Također treba primjetiti kako i jezgre 1 i 2 također postižu bolje rezultate nego u prethodnom primjeru zbog boljeg grupiranja dretvi.

Korisnik ne treba eksplicitno definirati veličinu lokalnog indeksnog prostora, već se sustav sam može pobrinuti oko toga, no najčešće se preporuča da korisnik sam ispita sa kojom veličinom se postižu najbolji rezultati. U ovom slučaju je veličina bloka od 8x8 postizala bolje rezultate



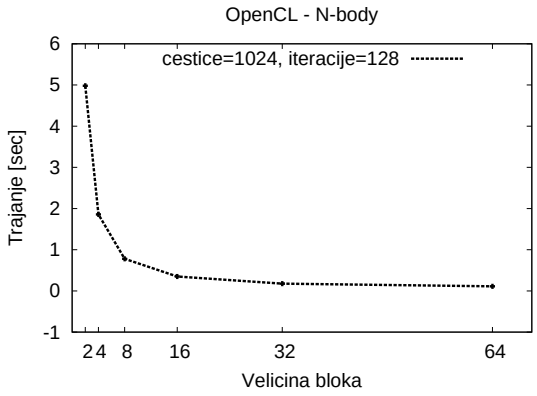
Slika 3. *OpenCL*, veličina bloka 8

od 4x4, ali je već veličina 16x16 postizala lošije rezultate. Vjerovatan razlog je izrazito mala lokalna OpenCL memorija, svega 16kB po procesoru dretvi, a kako jedan procesor izvodi po nekoliko dretvi vjerovatno dolazi do problema u sinkronizaciji.

B. Simulacija međudjelovanja čestica

Primjerom simulacije međudjelovanja čestica (engl. *n-body*) se za veliki broj čestica u 3D prostoru kroz niz iteracija promatra njihovo međudjelovanje. Svaka čestica ima svoj položaj u prostoru, masu i trenutnu brzinu, a svakom iteracijom se računaju privlačne sile između svih čestica i određuje njihov novi položaj. Navedeni primjer je računski, odnosno procesorski, izuzetno zahtjevan zbog velikog broja čestica i iteracija.

Slika 4. prikazuje utjecaj veličine lokalnog prostora na postignute rezultate, broj čestica je 1024, a iteracija 128. Očekivano su postignuti bolji rezultati za veći blok čestica. Pritom treba imati na umu da je lokalna memorija izuzetno mala u odnosu na globalnu, pa treba biti pažljiv oko njenog korištenja. Povećanjem broja čestica i iteracija naravno se i duljina izvođenja povećava eksponencijalno.

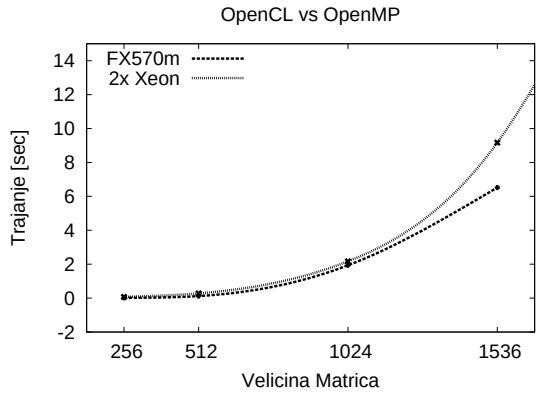


Slika 4. *OpenCL*, *N-body*, veličine lokalnih blokova

C. Usporedba rezultata OpenCL-a (GPU) i OpenMP-a (CPU)

Posebno je zanimljivo usporediti postignute rezultate *OpenCL-a* na GPU-u i *OpenMP-a* na CPU-u kako bi se okvirno usporedio utjecaj same arhitekture mikroprocesora na rezultate. Korišten je primjer množenja matrica, za *OpenCL* je korištena programska jezgra 3 sa lokalnom veličinom indeksnog prostora od 8x8 elemenata, ona je ujedno postigla i najbolje rezultate u prethodnom ispitivanju.

Slika 5. prikazuje performanse *Marvina* i Nvidia GPU-a za koje bi mogli tvrditi kako su neočekivani. Razlog je što korišteni GPU radi na taktu od 475MHz dok *Marvinovi* CPU-ovi rade na 2GHz, također treba spomenuti kako je GPU generacijski iz 2007. godine, dok je CPU arhitekture koja se pojavila tek 2009. godine, također ne treba ni zanemariti da je jedan *Xeon E5504* nekoliko puta skuplji od *Nvidie FX570m*. Sve navedene tvrdnje idu u prilog CPU-u, ali upravo tu dolazi do izražaja paralelizirana arhitektura GPU-a koja je specijalizirana za matematičke kalkulacije, te iz tog razloga postiže bolje performanse.



Slika 5. *OpenCL FX570m* : *OpenMP* Marvin

Bitno je napomenuti kako su danas dostupni GPU-ovi za red veličine brži od ovdje korištenog, te da je ugradnja više GPU-ova u jedno računalo daleko jednostavnija i jeftinija nego gomilanje CPU-ova u skupim poslužiteljima, grozdovima itd. Danas imamo priliku iskorištavati snagu GPU-ova i za općenite svrhe, tako da svakako treba voditi računa o odabiru platforme prilikom rješavanja problema.

VII. ZAKLJUČAK

OpenCL je prvi sustav koji omogućuje izvođenje istog programskog koda na različitim platformama, olakšava postupak paralelizacije, a njegova primjena može varirati od korisničkih aplikacija pa sve do računarstva visokih performansi. *OpenCL* implementacije su već dostupne iako još nisu dovoljno zastupljene da bi se nametnuo kao standard, ali se to može očekivati jer je otvorene specifikacije u čijoj su izradi sudjelovale sve relevantnije kompanije.

S druge strane danas imamo na raspolaganju mikroprocesore različitih arhitektura i svrha, te se nameće pitanje kako ih što bolje i jednostavnije iskoristiti, a *OpenCL* popunjava

upravo tu prazninu. U budućnosti svakako možemo očekivati korištenje GPU-ova u računalima u općenitije svrhe, naravno uz postojanje neophodnog CPU-a. Također se može očekivati i češća pojava *"hibridnih"* mikroprocesora koji sadrže veći broj jezgri, bile one opće namjene ili specijalizirane (npr. za matematičke kalkulacije).

LITERATURA

[1] Khronos OpenCL Working Group, 2009: The OpenCL Specification, version 1.0 rev 48

[2] Nvidia, 2009: OpenCL Programming for the CUDA Architecture, version 2.3

[3] Nvidia, 2009: OpenCL Programming Guide for the CUDA Architecture, version 2.3

[4] Nvidia, 2009: OpenCL Best Practices Guide, version 1.0

[5] Andreas Klöckner, prosinac 2009: http://mathema.tician.de/software/pyopencl/

[6] Wikipedia, 12.02.2010: http://en.wikipedia.org/wiki/N-body_problem