COMPUTATIONAL SCIENCE AND TECHNIQUES
Special Issue for Research Innovations Fundamentals 2011
Volume 1, Number 1, 2013, 36-44

Publisher: Klaipėda University
http://journals.ku.lt/index.php/CST
Online ISSN: 2029-9966

LYGIAGRETUSIS BENDROS PASKIRTIES PROGRAMAVIMAS NAUDOJANT NAUJOS KARTOS GRAFIKOS APDOROJIMO PROCESORIUS: CPU IR GPU LYGINAMOJI ANALIZĖ IR GALIMYBIŲ TYRIMAS

Donatas Krušna, Vitalijus Denisovas
Informatikos katedra, Klaipėdos universitetas, Herkaus Manto g. 84, LT-92294, Klaipėda
dasa666@gmail.com, vitalij@ik.ku.lt

Anotacija. Šiuolaikinių grafinių procesorių (GPU) skaičiavimų galia auga labai sparčiai, ir siekia 5.10 TFLOPs (5.10x10^12) 32 bitų ir 1.27 TFLOPs (1.27x10^12) 64 bitų slankaus kablelio skaičiaus operacijų per sekundę, kai tokio pat brangumo centriniai procesoriai (CPU) siekia tik 107 GFLOPs (107x10^9) 32 bitų slankaus kablelio skaičiaus operacijų. Toliau augant GPU skaičiavimų galiai mokslininkai ieško būdų kaip pažaboti šiuos resursus sprendžiant skaičiavimams imlius uždavinius. Tačiau kurti programas nevienalytėms lygiagrečių skaičiavimų platformoms, kurių tradicinės daugiabranduolinių procesorių CPU ir GPU metodikos yra labai skirtingos, lieka sudėtingu iššūkiu programuotojams, tai apsunkina galimybę lengvai išnaudoti nevienalyčių lygiagrečių CPU ir GPU platformų skaičiavimų galimybes. Šiame straipsnyje analizuojami naujausi lygiagretaus bendros paskirties programavimo modeliai heterogeninėms sistemoms ir aprašoma kalba, kuria rašant programinį kodą galima efektyviau išnaudoti šiuolaikinius programuojamus grafinius procesorius. Tokia CPU ir GPU simbiozė sukuria galimybes skaičiavimams imlias problemas (pavyzdžiui, daugiamačio kompiuterinio modeliavimo) spręsti tūkstančius kartų sparčiau nei iki šiol egzistavusiose vienalytėse sistemose.

Pagrindiniai žodžiai: programuojamas grafinis procesorius (GPU), lygiagretaus programavimo modeliai, bendrųjų skaičiavimų lygiagretinimas grafinio procesoriaus instrukcijų lygmenyje (GPGPU), atvira programavimo platforma OpenCL.

Įvadas

Šiuolaikinių procesorių architektūra lygiagretumu pasinaudojo našumui didinti. Susidūrus su techniniais iššūkiais, kai reikėjo padidinti procesoriaus našumą nekeliant procesoriaus taktinio dažnio ir įtampos reikalavimų, šiuolaikinių procesorių architektūra tiesiog padidino CPU branduolių skaičių, sujungdami juos į lygiagrečiai veikiančius modulius, taip dar labiau įkūnydami lygiagretinimo principus. Grafiniai procesoriai taip pat evoliucionavo iš riboto funkcionalumo atvaizdavimo prietaisų į programuojamus lygiagrečius procesorius. Turint omenyje, kad šiandienos sistemos yra komplektuojamos su lygiagrečiais CPU ir GPU, kurie gali vienu metu atlikti daug operacijų, dėl daugiabranduolinės jų architektūros, todėl svarbu, kad programuotojai visiškai išnaudotų šias heterogenines sistemas.

Nelengvas uždavinys yra kurti programas heterogeninėms lygiagrečių skaičiavimų platformoms, kurių tradicinės CPU ir GPU programavimo metodikos yra labai skirtingos. CPU grindžiami lygiagretaus programavimo modeliai yra paremti standartais ir dažnai apima bendro vartojimo adresų erdvę ir neįtraukia vektorinių operacijų. Kai bendrosios

paskirties GPU programavimo modeliai apima sudėtingas hierarchines atminties struktūras, bei vektorines operacijas, tačiau yra priklausomi nuo platformos, gamintojo ar techninės įrangos komponentų. Šie apribojimai programuotojams, kurie iki šiol programavimo vieno branduolio architektūros procesorių pagrindu, apsunkina galimybę lengvai išnaudoti nevienalyčių lygiagrečių CPU ir GPU platformų skaičiavimų galimybes. Kuo toliau, tuo labiau atsiranda poreikis programų kūrėjams suteikti galimybę efektyviai išnaudoti nevienalytes skaičiavimų platformas pradedant nuo sparčių skaičiavimo serverių, stalinių kompiuterių iki nešiojamųjų kompiuterių ar net delninių įrenginių, turinčių plačią įvairovę lygiagrečių CPU, GPU ir kitokių procesorių, tokių kaip DSP ar Cell B.E. Todėl kaip sprendimas 2008 gruodžio 8 d. buvo išleista pirmoji bendrosios paskirties daugiabranduolinių procesorių bei heterogeninių sistemų programavimo kalba OpenCL (Open Computing Language), kuri nepriklausomai nuo techninės įrangos gamintojo leido grafiniams bei kitos įvairios paskirties procesoriams išnaudoti lygiagretaus programavimo modelių galimybes. Kadangi OpenCL yra įterptinė kalba ir jai reikalinga programa „šeimininkė“, tai dvi realiausios matematinio programavimo kalbos yra Matlab ir Phyton, kurios turi labai aiškią sintaksę ir stiprias matematinių modelių bei funkcijų bibliotekas.

1. Lygiagretaus programavimo sąvokos ir modeliai

Lygiagrečių programų paskirtis yra veikti ant daug, vienu metu veikiančių procesorių. Kiekvienas procesorius dirba su viena problemos dalimi ir jie visi veikia vienu metu. Geriausiu atveju „n“ procesorių sukoncentruotų spręsti vieną problemą, tai gali atlikti „n“ kartų greičiau nei bet koks pavienis procesorius. Nors ne visada pavyksta pasiekti šį idealų tiesinį pagreitį, tačiau lygiagretumas leidžia daugelio sudėtingų procesų veikimą galima paspartinti „n“ kartų.

Yra keli lygiagretaus programavimo bendro naudojimo modeliai:

- Bendrosios atminties (angl. *Shared Memory*),
- Užduočių (angl. *Threads*),
- Pranešimų perdavimo (angl. *Message Passing*),
- Duomenų paralelumo/lygiagretumo (angl. *Data Parallel*),
- Hibridinis (angl. *Hybrid*).

Šie lygiagretaus programavimo modeliai yra aukštesnio lygio techninės įrangos bei atminties architektūros abstrakcija. Todėl jie nėra priklausomi nuo konkretaus tipo mašinų ar architektūros. Tai leidžia pasiekti aukštą sąveikumo lygį be poreikio perrašyti programos kodą. Tačiau lyginant su aukšto lygio programavimo kalbomis šie modeliai vis dar yra vertinami kaip žemo lygio, nes daug lygiagretinimo užduočių išlieka programuotojo žinioje (Buck, 2006; Čiegis, 2001).

1.1 Bendrosios atminties modelis (Shared memory model)

Bendrosios atminties modelyje užduotys (angl. *tasks*) dalinasi bendra adresų erdve, iš kurios jos nuskaito ir rašo duomenis asinchroniniu principu. Įvairūs mechanizmai tokie kaip užraktai (angl. *locks*) gali būti naudojami valdyti priėjimą prie bendrosios atminties.

Šio modelio privalumai programuotojo požiūriu yra duomenų tarp užduočių bendrumas ir nepriklausomumas, tai leidžia supaprastinti programos kūrimo procesą, nes nereikia aprašyti duomenų ryšių tarp užduočių. Laikant duomenis lokaliai, procesoriui tenka rečiau kreiptis į atmintį, vykdyti sparčiosios atminties perrašymus. Taip pat sumažėja duomenų srautas, kuris atsiranda kai keli procesoriai naudoja tuos pačius duomenis. Tačiau spartos atžvilgiu tampa sunku suprasti ir valdyti duomenų lokalumą, kai naudojamų procesorių skaičiavimuose kiekis išauga, nes kiekvienas procesorius kreipdamasis į atmintį užrakiną magistralę ir kitiem procesoriams tenka laukti kol jie galės kreiptis į atmintį. Tokio tipo modelis turi kritinį tašką, kuomet didinant procesorių skaičių dėl labai apkrautos ir sudėtingos komunikacijos ir sinchronizacijos duomenų apdorojimo sparta nebeauga, o net pradeda mažėti (Buck, 2006; Čiegis, 2001).

1.2 Užduočių modelis (Threads model)

Lygiagretaus programavimo užduočių modelyje vienas procesas vienu metu gali turėti daug vykdymo kelių bei kartu dalintis programos resursais. Taip sutaupoma resursų, kurių reikėtų jei programos resursai būtų replikuojami kiekvienai užduočiai. Šį modelį galima apibūdinti kaip savarankišką programos dalių (paprogramių), kurios bendrauja viena su kita per globalią atmintį, darbą. Tai reikalauja sinchronizacijos priemonių, kurios užtikrintų, jog viena užduotis neatnaujins to pačio globalaus adreso bet kuriuo metu. Užduotys gali būti vykdomos, tačiau pagrindinė programa išlieka nepakitusi, tam kad teiktų reikalingus resursus kol aplikacija pasibaigs.

Užduočių modelis artimai siejamas su bendrosios atminties architektūra ir operacinėmis sistemomis. Iš programavimo perspektyvos užduočių realizacija gali būti paprogramių biblioteka, tačiau tokiu atveju programuotojas yra atsakingas už lygiagretumo įgyvendinimą, nes kompiliatorius nežino apie užduotis, jų kūrimą, sinchronizavimą, užrakinimą, bendrąją ir lokaliąją atmintį bei jos valdymą. Tačiau yra antras realizacijos būdas, kai užduočių modelis yra išreiškiamas per kompiliatoriaus direktyvų rinkinį. Tuomet programuotojui lieka tik nusakyti, kurias makro operacijas reikia lygiagretinti bei kaip valdoma atmintis (Buck, 2006; Čiegis, 2001).

1.3 Pranešimų perdavimo modelis (Message Passing Model)

Pranešimų perdavimo modelis – komunikacinis modelis naudojamas lygiagrečiuose skaičiavimuose, objektais paremtame programavime bei tarpprocesinėje komunikacijoje. Šitame modelyje procesai ar objektai gali siųsti ir gauti pranešimus (bitus, sudėtingas duomenų struktūras, ar net kodo segmentus) iš kitų procesų ar objektų.

Taip pat šitas modelis gali turėti ir pranešimų sinchronizaciją. Laukdami pranešimų procesai gali sinchronizuotis. Toks modelis yra komunikacijos tarp procesų ar objektų paradigma, kur pranešimas yra siunčiamas iš siuntėjo vienam ar keliems gavėjams. Modeliuojant pranešimų siuntimo sistemą yra galimi keli tokios sistemos variantai:

- kai pranešimai yra siunčiami patikimai,
- kai yra garantuojamas pranešimų pristatymo eiliškumas,

- kai pranešimai yra perduodami „one-to-one“, „one-to-many" („multicasting“ arba „broadcasting“), ar „many-to-one“ („client-server“) principais,
- kaip komunikacija vyksta sinchroniškai arba asinchroniškai.

Pranešimų siuntimo bibliotekos leidžia kurti efektyvias lygiagrečias programas paskirstytos atminties sistemoms. Taip pat šios bibliotekos turi funkcijas, kuriomis galima pranešimų sistemą inicializuoti bei konfigūruoti taip pat kaip ir sistemos duomenų siuntimą ar gavimą. Pasak Buck (2006) bei Čiegis (2001) dvi populiariausios pranešimų siuntimo sistemos, kurios naudojamos mokslinėms ir inžinerinėms aplikacijoms yra PVM (Parallel Virtual Machine) iš Oak Ridge National Laboratory ir MPI (Message Passing Interface) apibrėžta MPI forumu.

1.4 Duomenų lygiagretumo modelis (Data Parallel Model)

Duomenų lygiagretumas taip pat žinomas kaip ciklo lygio lygiagretumas kai skaičiavimai vyksta per kelis procesorius lygiagrečių skaičiavimų aplinkose. Duomenų lygiagretumas koncentruojasi į duomenų paskirstymą per skirtingus lygiagretaus skaičiavimo vienetus. Ši lygiagretumo forma glaudžiai susijusi su užduočių lygiagretumu, kai vietoj duomenų yra paskirstomi procesai.

Įsivaizduokime tipinį grafinį vamzdyną (angl. *pipeline*). Scena yra išskaidoma į poligonus, o jie dar padalinami į trikampius, kurių kiekvienas yra analizuojamas pritaikant jam apšvietimo modelį, geometriją, suteikiant tekstūrą, tam kad pabaigoje proceso įgauti atvaizdą. Operacijos, kurios yra taikomos šiems trikampiams yra identiškos viena kitai ir gali būti vykdomos nepriklausomai viena nuo kitos. Toks atvejis yra klasikinė duomenų lygiagretumo strategija, kuomet lygiagretumas leidžia spręsti atitinkamas problemas. Lygiagrečiojo duomenų modelio charakteristikos:

- Daugelis lygiagrečių skaičiavimų koncentruojasi į operacijų vykdymą duomenų rinkiniams, o duomenų rinkinių dažniausi pavidalai būna masyvai arba kubai.
- Užduočių rinkinys dirba ant tos pačios duomenų struktūros, tačiau kiekviena užduotis atskirai dirba su skirtinga tos struktūros dalimi.
- Užduotys atlieka vieną ir tą pačią operaciją ant savo duomenų dalies.

1.5 Mišrūs ir kiti modeliai (Hybrid and other models)

Egzistuoja ir kiti lygiagretaus programavimo modeliai. Mišrus (angl. *Hybrid*) modelis apjungią kelis ar daugiau lygiagretaus programavimo modelių. Vienas iš bendrinių mišriojo modelio pavyzdžių yra pranešimų siuntimo modelio (MPI) ir užduočių modelio (POSIX threads) arba bendrosios atminties modelio (OpenMP) kombinacija. Šis modelis labai gerai tinka sinchroninėms daugiabranduolinėms mašinoms.

Single Program Multiple Data (SPMD) yra aukšto lygio programavimo modelis, kuris gali būti pritaikytas bet kuriai iš prieš tai minėtų lygiagretaus programavimo modelių kombinacijų. Visos užduotys vykdo vieną programą vienu metu. Bet kuriuo programos veikimo metu užduotys gali vykdyti vienodas arba skirtingas instrukcijas. SPMD programos dažniausiai turi suprogramuotą logiką, tam kad skirtingos užduotys vykdytų tik tą dalį

programos, kuriai jos buvo suprogramuotos. Tai reiškia, kad užduotys neturi vykdyti visos programos, o tik jos dalį. Visos užduotys naudoja skirtingus duomenis.

Kaip ir SPMD, MPMD (Multiple Program Multiple Data) yra aukšto lygio programavimo modelis, kuris taip pat gali būti pritaikytas bet kuriai iš prieš tai minėtų lygiagretaus programavimo modelių kombinacijų. MPMD aplikacijos tipiškai turi kelis vykdomuosius objektus, failus, programas. Kol yra vykdoma aplikacija, užduotis gali vykdyti tą pačią ar kitą programą kaip ir kitos užduotys. Visos užduotys gali naudoti skirtingus duomenis (Buck, 2006, Čiegis, 2001).

2. Lygiagrečiųjų grafinių procesorių panaudojimas bendros paskirties programavime

Grafiniai procesoriai yra skirti apdoroti vektorines operacijas, kurios dažnai atliekamos grafikoje. Dėl savo pobūdžio GPU yra veiksmingiausia priemonė spręsti problemas, kurių informaciją galima apdoroti srautiniu skaičiavimų modeliu. Srautiniai skaičiavimai (angl. *Stream computing*) nuo tradicinių skiriasi nuskaitymo principu. Duomenys yra nuskaitomi kaip srautas nuoseklių elementų, kur kiekvienas elementas yra įrašas naudojamas vienam elementariam skaičiavimui. Paprastai tokie duomenys yra nepriklausomi vieni nuo kitų. Sistema vykdo programą ar kernelį kiekvienam srauto įrašui ir galutinius duomenis patalpiną į rezultatų srautą. Tokiu būdu programuojamas grafinis procesorius vykdo viršūnių programas kiekvienai duomenų srauto viršūnei ir fragmento programas kiekvienam duomenų srauto fragmentui. Dėl šios priežasties skaičiavimai ant grafinių procesorių vyksta daug kartų sparčiau. Čia srautai tai duomenų rinkiniai, kurių duomenys gali būti apdorojami lygiagrečiai. Tačiau šios duomenų struktūros turi konkrečius apribojimus, kurie neleidžia bet kam ir bet kada kreiptis į jas, taip apsaugant duomenų vientisumą ir lygiagretų jų apdorojimą. Todėl tokius duomenis nuskaityti ir įrašyti gali tik specifinės operacijos.

Duomenys srautuose yra grupuojami pagal eilutes, kaip ir masyvai C kalboje. Toks išsidėstymas yra svarbus, kai yra naudojami skirtingi srauto operatoriai, visų pirma perduodant duomenis į arba iš srauto. Tai svarbu dėl duomenų paėmimo ir atidavimo specifikos.

Srautinės operacijos atliekamos taip:

R1 [a1, a2, ..., an]

+

R2 [b1, b2, ..., bn]

Kuo mažiau tokių operacijų yra sudaroma, tuo procesas vyksta greičiau. Teisingai (kaip pavaizduota aukščiau) sugrupavus duomenis registruose duomenims paimti gali pakakti dviejų operacijų, o patalpinti – vienos operacijos.

Kita GPU ir lygiagretaus programavimo specifika yra „kernel“ funkcijos, kurios vykdo operacijas su srauto elementais. Kernel funkcijos iškvietimas atlieka netiesioginį ciklą per srauto elementus, kur kernel funkcijos kodas yra iškviečiamas kiekvienam srauto elementui. Žemiau pateikiamas Kernel funkcijos pavyzdys, kuris vykdo vektorių sumą:

```
__kernel void vectorAdd(__global const float * a, __global const float * b, __global float * c)
{
// Vector element index
```

```
int nIndex = get_global_id(0);
c[nIndex] = a[nIndex] + b[nIndex]; }
```

Tam, kad sujungti GPU su CPU ar kitos paskirties procesoriumi ir vykdant matematines operacijas pasinaudoti GPU kaip koprocesoriumi yra bendros paskirties grafinių procesorių kalbos, tokios kaip: Brook, Cg, CUDA, OpenCL. Tačiau šiame straipsnyje pateiksime ir argumentuosime OpenCL kalbos pasirinkimą (General-Purpose Computation...).

3. OpenCL (Open Computing Language)

OpenCL kalba priešingai nei kitos GPGPU kalbos yra nepriklausoma nuo gamintojo, bei techninės įrangos. Tačiau bibliotekų bei klasių rinkinys skirtas programinei įrangai kurti (angl. *Framework*), kuris susideda iš kalbos, API (Application Interface), bibliotekų ir kompiliatoriaus su kita programine įranga, kuri tiesiogiai bendrauja su technine įranga ir įgalina OpenCL kalba parašytų programų veikimą.

OpenCL kalba nors ir skaitoma aukšto lygio technine įrangos abstrakcija, buvo skirta programuotojams ekspertams, kuriems reikia rašyti lankstų ir efektyvų platformų atžvilgiu kodą. T.y. bibliotekų rašytojams, techninės įrangos kūrėjams, kuriems reikia kurti su įvairiomis platformomis suderinamą žemo lygio programinę įrangą (angl. *Software drivers*), bei programuotojams orientuotiems į programų veikimo spartą.

OpenCL kalba skiriasi ir tuo, jog jos struktūra ir terminologija atsiriboja nuo grafinių procesorių ir jų vamzdyno terminologijos, kuri yra pakeičiama bendros paskirties logika ir terminais (Introduction to OpenCL Programming, 2010; ATI Stream Computing OpenCL, 2010; The OpenCL Specification). Šios kalbos hierarchinė struktūra susideda iš tokių modelių:

- Platform Model
- Memory Model
- Execution Model
- Programming Model

4. Lyginamoji CPU ir GPU analizė

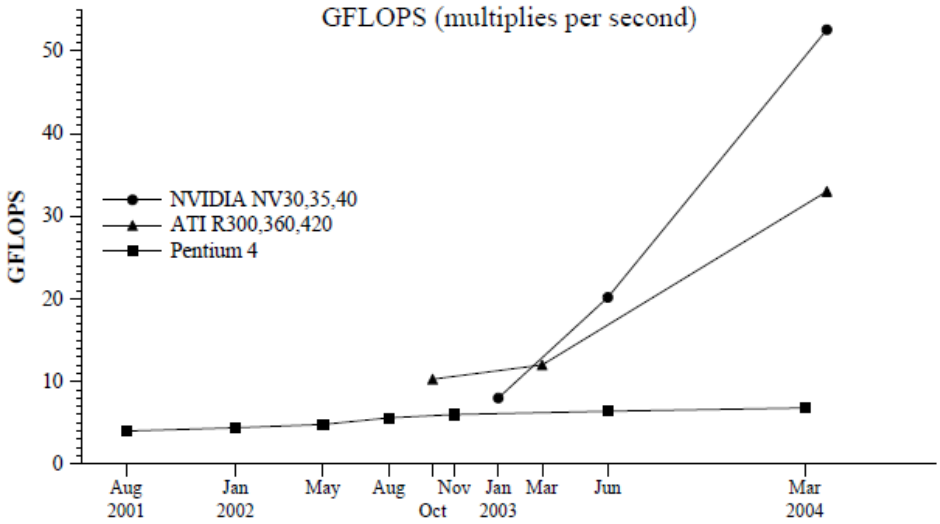
Daugelis straipsnių, forumų ir kitų šaltinių kalba apie CPU ir GPU santykinę spartą. Juose dažniausiai galima rasti diagramas rodančias nuo x10 iki x500 kartų didesnę spartą nei CPU. Tačiau tai priklauso nuo daugelio faktorių, kurie dažniausiai neminimi tuose šaltiniuose. Tie faktoriai tai operatyviosios atminties magistralės pralaidumas, procesoriaus sparčiosios atminties dydis. Bene svarbiausias faktorius tai pati užduotis, nuo kurios ir priklauso kaip stipriai GPU pralenks CPU skaičiavimuose.

Tam visų pirma problema turi būti laisvai daloma į daug dalių, kurios viena nuo kitos nepriklauso, nes GPU specifiškai suprojektuotas taip, jog turi daug vykdymo elementų, kurie vienu metu gali vykdyti daug vienodų skaičiavimų. Kitas kriterijus užduočiai yra slankaus kablelio skaičiai, nes GPU yra optimizuotas darbui su slankaus kablelio skaičių operacijomis.

Trečias reikalavimas yra tas, jog duomenys, su kuriais grafinis procesorius dirba, turi tilpti to procesoriaus vidinėje atmintyje, kurios sparta tarp procesoriaus ir jos yra žymiai didesnė nei PCI x16 ar platformos operatyviosios atminties magistralės. Todėl kuo mažiau vyksta įrašymų į globaliąją platformos atmintį, tuo didesnė yra sparta pačios aplikacijos bei skaičiavimų.

Paskutinė iš minimų šiame straipsnyje problemų, yra išsišakojimai, kurių turi būti vengiama. Nors šiuolaikiniai grafiniai procesoriai leidžia išsišakojimus, tačiau jų kaina spartos atžvilgiu yra didžiulė, nes jie padidina duomenų priklausomumą, kuris įtakoja didesnį skaičiavimų uždelstumą.

Taigi, kuo problema ar užduotis geriau atitinka šiuos reikalavimus, tuo ji sparčiau veiks GPU. Visgi gali būti atvejų kai CPU veikia greičiau. Kad taip nutiktų reikėtų nepaisyti visų minėtus punktų. Be to, jei duomenų kiekis būtų neekvivalenčiai mažas GPU skaičiavimo galiai, tada tokios programos inicializacija ant GPU užimtų daug daugiau resursų ir laiko nei ant CPU, kas įtakotų ir bendrą programos spartą. Tačiau ir CPU turi savo optimizacijos sąlyčio taškus, kurie gali stipriai pagerinti tam tikrų skaičiavimų spartą. Visgi GPU išlieka spartesnė intensyviuose skaičiavimuose, nei CPU, dėl savo specifinės architektūros (Buck, 2006; Intel® Software Network; Lee et al., 2010).



1 pav. Slankiojo kablelio operacijų per sekundę kiekio kitimas metų eigoje.

5. Naujausios programinės priemonės ir rekomendacijos jas parenkant

Pagrindiniai grafinių procesorių gamintojai yra ATI ir NVIDIA. Tiek NVIDIA, tiek ATI turi išleidusios programinę įrangą, kuri yra tarsi sluoksnis tarp techninės ir programinės įrangos (OpenCL drivers). Todėl pradedant domėtis grafinio programavimo įrankiais siūloma pradėti nuo šių gamintojų programinės įrangos kūrėjų portalų.

Kalbant apie matematinių operacijų intensyvumą ir skaičiavimų imlumą, realaus gyvenimo problemų matematiniuose modeliuose, yra keli pasirinkimai. AMD Core Math Library for Graphic Processors (ACML-GPU), tai ACML bibliotekos plėtinys, kuris priklausomai nuo matematinių operacijų sugeba dinamiškai pasirinkti kur tą kodą taikyti ar: ant CPU ar ant GPU. Ši biblioteka suteikia rinkinį išsamiai optimizuotų, lygiagrečių

matematinių metodų, skirtų mokslinėm, inžinerinėm, intensyvaus skaičiavimo reikalaujančiom aplikacijom. Pavyzdžiui ACML puikiai tinka orų modeliavimui, skysčių dinamikos skaičiavimams, finansinei analizei, naftos bei kuro aplikacijoms ir dar daug kitų aplikacijų. Analogą turi ir NVIDIA – CUDA Math. Šias aplikacijas siūloma rinktis siekiant programuoti vienos arba kitos kompanijos įrankiais ant tos pačios kompanijos grafinių procesorių.

MATLAB įrankio OpenCL biblioteka skirta tam tikrus skaičiavimus paleisti ant GPU ir taip išnaudoti MATLAB lankstumą ir intuityvumą, bei GPU skaičiavimo galią. Ši priemonė tinkama mokslininkams, neturintiems gilių žinių kaip išnaudoti GPU privalumus. Pradiniai žingsniai paleidžiant skaičiavimus ant GPU yra labai panašūs kaip ir programuojant bet kuria kita kalba: C\C++, Python ar kitomis.

Dar vienas vertas dėmesio įrankis yra OpenCL Studio. Tai integruota kūrimo sistema (IDE), kuri apima visus reikalingus įrankius OpenCL aplikacijos kūrimui, klaidų taisymui, monitoringui, bei testavimui. Ši sistema naudoja Lua skriptų kalbą OpenCL kalbos programų inicializavimui ir paleidimui. Tai puikus įrankis kurti tiek grafinius modelius, nes savyje ši sistema turi ir OpenGL objektus, bei kurti bendros paskirties aplikacijas.

Išvados

- Šiuolaikinė lygiagrečių heterogeninių sistemų programavimo kalba OpenCL leidžia užduotį išdalinti atitinkamai paskirstant darbus tarp centrinio ir grafinio procesoriaus, kas įtakoja stipriai išaugančią tokio tipo programų spartą.
- Kadangi grafiniai procesoriai aritmetiniuose ir vektoriniuose skaičiavimuose naudoja slankaus kablelio operacijas, jie smarkiai lenkia centrinius bendrosios paskirties procesorius.
- Naujoji lygiagrečių heterogeninių sistemų programavimo kalba OpenCL palengvino programų kūrimą tokioms sistemoms, nes ši kalba nepriklauso nuo platformos ar gamintojo, bei yra įterptinė. Tai suteikia galimybę parašius biblioteką ją taikyti norimoje bendrosios paskirties, ar kitokioje specifinėje programavimo kalboje.
- Vis daugiau atsiranda įrankių pritaikytų tiek žemo lygio programuotojams, tiek mokslininkams ar inžinieriams, kuriantiems programas ar bibliotekas šiuolaikiniams grafiniams procesoriams bei nevienalytėms sistemoms.
- Tyrimas parodė, jog daugėjant branduolių skaičiui lygiagrečiose nevienalytėse ar vienalytėse sistemose lygiagrečių programų kūrimas darosi sudėtingesnis, nes komunikacija ir sinchronizacija tampa vis keblesnė. Tai reiškia, jog yra taškas, kurį pasiekus didinant branduolių skaičių programos sparta nebedidės ir net turi tendenciją mažėti.
- Lygiagretus programavimas bei grafiniai procesoriai su srautiniais skaičiavimų metodais stipriai sumažino bendrosios atminties programavimo modeliu keliamą problemą, nes tik galutiniai duomenys yra įrašomi į RAM. Tačiau tai vis dar išlieka silpniausia grandimi kuriant bet kokio tipo programas.

Literatūra

Buck, I. (2006). Stream computing on graphics hardware. A Dissertation.

Čiegis, R. (2001). Lygiagretieji algoritmai, Vilnius: Technika.

Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal R., Dubey P. (2010). Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. ISCA '10 Proceedings of the 37th Annual International Symposium on Computer Architecture: 451–460.

AMD Developer Central. http://developer.amd.com/pages/default.aspx

ATI Stream Computing OpenCL. (2010). Programming Guide by AMD.

Intel® Software Network. CPU parallel computing vs GPU parallel computing. http://software.intel.com/en-us/forums/showthread.php?t=66235

Introduction to OpenCL Programming. (2010). Publication #: 137-41768-10, by AMD.

General-Purpose Computation Using Graphics Hardware. http://gpgpu.org/

MATLAB Adds GPGPU Support. 2010-09-20. http://www.hpcwire.com/features/MATLAB-Adds-GPGPU-Support103307084.html

NVIDIA Developer Zone. http://developer.nvidia.com/

The OpenCL Specification. Version: 1.0. (2011). Khronos OpenCL Working Group. http://developer.amd.com/wordpress/media/2012/10/opencl-1.2.pdf

GENERAL PURPOSE PARALLEL PROGRAMING USING NEW GENERATION GRAPHIC PROCESSORS: CPU VS GPU COMPARATIVE ANALYSIS AND OPPORTUNITIES RESEARCH

Donatas Krušna, Vitalij Denisov

Summary

OpenCL, a modern parallel heterogeneous system programming language, enables problems to be partitioned and executed on modern CPU and GPU hardware; this increases performance of such applications considerably. Since GPU's are optimized for floating point and vector operations and specialize in them, they outperform general purpose CPU's in this field greatly. This language greatly simplifies the creation of applications for such heterogeneous system since it is cross-platform, vendor independent and is embeddable, hence letting it be used in any other general purpose programming language via libraries. There is more and more tools being developed that are aimed at low level programmers and scientists or engineers alike, that are developing applications or libraries for CPU's and GPU's of today as well as other heterogeneous platforms. The tendency today is to increase the number of cores or CPU's in hopes of increasing performance, however the increasing difficulty of parallelizing applications for such systems and the even increasing overhead of communication and synchronization are limiting the potential performance. This means that there is a point at which increasing cores or CPU's will no longer increase applications performance, and even can diminish performance. Even though parallel programming and GPU's with stream computing capabilities have decreased the need for communication and synchronization (since only the final result needs to be committed to memory), however this still is a weak link in developing such applications.

Key words: GPU, parallel programming models, open programming platform OpenCL, GPGPU general purpose graphic processing units.