ISSN 1648-8776
JAUNŲJŲ MOKSLININKŲ DARBAI. Nr. 47 (1). 2017 Doi: 10.21277/jmd.v47i1.135

LYGIAGRETŪS SKAIČIAVIMAI SU CUDA

Julija Semenenko, Dmitrij Šešok
Vilniaus Gedimino technikos universitetas
El. p.: julija.semenenko@gmail.com, dmitrij.sesok@vgtu.lt

Įvadas

Skaičiavimų greitį galima padidinti keliais būdais: optimizuojant algoritmus, paskirstant uždavinio dalis per kelis kompiuterius (pvz., BOINC) arba, išskaidžius į paprastus veiksmus, atiduoti grafiniam procesoriui – GPU. Realizuoti GPU skaičiavimus leidžia lygiagrečių skaičiavimų architektūra CUDA (angl. *Compute Unified Device Architecture*) – panaudojant NVIDIA vaizdo plokštės pajėgumus spartesniems skaičiavimams atlikti.

Šio darbo tikslas yra ištirti grafinio procesoriaus našumo ypatumus sprendžiant skirtingus lygiagrečių skaičiavimų uždavinius.

Procesorius sudarytas iš kelių galingų branduolių, kurių tikslas – kuo greičiau atlikti jiems patikėtas užduotis. Kad našūs branduoliai negaištų laiko veltui, nemažą procesoriaus dalį užima spartinančioji atmintinė (angl. *cache*), į kurią įkraunami savo apdorojimo eilės laukiantys duomenys. Grafinis procesorius sudarytas iš daugelio smulkių multiprocesorių [3], kurie yra silpnesni galios atžvilgiu, palyginus su centrinio procesoriaus CPU branduoliais, tačiau jų yra žymiai daugiau. Jų tikslas – kuo greičiau įvykdyti kuo daugiau lygiagrečių užduočių, net jeigu kiekvienos atskirai paimtos užduoties atlikimo laikas bus ilgesnis negu CPU.

Taigi, jeigu užduotį galima išskaidyti į paprastesnius veiksmus, ypač jeigu jiems atlikti nereikalingi prieš tai buvusių skaičiavimų rezultatai, taikant GPU skaičiavimus galima tikėtis didelio vykdymo laiko pagreičio, lyginant su CPU. Ir atvirkščiai – nereikia tikėtis labai sutaupyti laiko bandant realizuoti sunkius, labai išsišakojusius sprendimus.

Lygiagretūs algoritmai turi nemažai skirtybių, galinčių daryti stiprią įtaką programos greitaveikai, todėl, norint efektyviai dirbti su CUDA, svarbu išmanyti ne tik taikytinus metodus, bet ir optimizacijos būdus, atminties tipų privalumus ir trūkumus, t. y. suprasti, kaip CUDA veikia.

Darbo metu realizuoti kelių uždavinių sprendimai remiasi įgytomis žiniomis, išbandytomis keliomis optimizacijomis ir palygintais skaičiavimo laiko rezultatais tarp GTX plokštės su CPU ir „Quadro“ plokštės. Pateiktos išvados, kada verta naudoti GPU, kada labiau apsimoka skaičiuoti su CPU.

Darbo tikslas – ištirti grafinio procesoriaus našumo skirtumus sprendžiant skirtingus lygiagrečių skaičiavimų uždavinius.

Darbo uždaviniai: atlikti literatūros analizę, išsiaiškinti lygiagretinimo su CUDA ypatumus, realizuoti taikomųjų uždavinių sprendimus ir juos optimizuoti, palyginti GPU ir CPU našumą sprendžiant skirtingus uždavinius.

Tyrimo metodai: literatūros analizė, skaitiniai eksperimentai.

Lygiagretūs skaičiavimai

Remiantis M. Flinn'o skaičiuojančiųjų mašinų architektūros taksonomija, GPU architektūra artimiausia SIMD (angl. *Single Instruction, Multiple Data*) architektūrai, tačiau galimi ir kiti variantai (išskyrus SISD). Pati NVIDIA vartoja SIMT sąvoką (angl. *Single Instruction Multiple Thread*).

Dažniausiai taikomas hibridinis skaičiavimų modelis – kai viena programa naudoja ir CPU, ir GPU. Kodas rašomas kaip įprasta, tik intensyvios skaičiavimo vietos perrašomos taip, kad būtų tinkamos vaizdo plokštės resursams panaudoti.

Skaičiuoti gali tekti su CPUx86, x86-64, *Itanium*, *SpursEngine* (*Cell*), NVIDIA GPU, AMD GPU, VIA (*S3 Graphics*) GPU [7], iš kurių kiekvienas procesorius turi savo SDK. Jiems priklauso ir CUDA. Apskritai galima išskirti dvi grupes: GPGPU (angl. *General-purpose computing for graphics processing units*) ir šešėliuokles (angl. *shader*).

CUDA. CUDA yra naši, nemokama, plačiai naudojama universitetinėse bendruomenėse (Harvardo, MIT, Stanfordo ir kt.), privačiose kompanijose („Adobe“, „MathWorks“) ir valstybinėse įmonėse (NASA) [1], tačiau turi rimtą trūkumą – kad ji veiktų, reikalinga NVIDIA vaizdo plokštė. Šiame darbe CUDA buvo pasirinkta dėl našumo [5], dėl iškelto tikslo įsigilinti į jos veikimo principus ir turint NVIDIA GTX 1060 plokštę, reikalingą tyrimams at-

likti. Reikėtų paminėti, kad CUDA galimybės plečiasi: pradedant nuo 5.5 CUDA versijos galima dirbti ir su ARM.

Yra sukurta ir nemažai oficialių, nemokamų CUDA bibliotekų lygiagretiems skaičiavimams atlikti, kurios pateikia vienus geriausių daugelio matematikos uždavinių sprendimus ir atlieka kitas funkcijas [4]. Tačiau ne visada ieškomas sprendimas jau egzistuoja. Svarbu suprasti, kaip veikia lygiagretūs skaičiavimai, į kokius aspektus būtina atsižvelgti. Tada galima bus kurti efektyvias programas, perprasti paruoštų bibliotekinių funkcijų turinius.

CUDA nėra tokia universali kaip „OpenCL“ modelis, veikiantis nepriklausomai nuo aplinkos (nors CUDA kodas veikia skirtingose NVIDIA plokštėse), tačiau yra pranašesnė lyginant skaičiavimus. Šešėliuokles pagal našumą CUDA, kaip GPGPU atstovė, irgi lenkia [5].

CUDA suderinama su „Windows“, MAC OS X ir „Linux“ operacinėmis sistemomis [2], galima naudotis C, C++, C#, FORTRAN programavimo kalbomis, „Python“ ir kitos kalbos turi savo papildinių programuojant su CUDA.

CUDA palaiko ir tokią GPU programavimo galimybę, kai skaičiavimai atliekami iš karto keliose vaizdo plokštėse. Tokiu atveju įrenginiai keičiasi duomenimis per PCIe trečios kartos magistralę. Tačiau šie GPU skaičiavimai supaprastėjo dirbant su CUDA 4.0 versija, kai buvo įvesta bendra adresų sritis visiems GPU ir šeimininkui. Kopijavimo operacijos sisteminės atminties nenaudoja, o duomenys kopijuojami tiesiogiai per GPUDirect 2.0 [9].

Svarbu suprasti, kad ne visada kodo lygiagretinimas reiškia atlikimo laiko sutrumpinimą. Labai išsišakojusius algoritmus, pavyzdžiui, archyvavimo, lygiagretinti neapsimoka, kaip ir mažų duomenų skaičiavimo algoritmą [6]. Prieš pradedant lygiagretinti reikia išskirti vietas, kurios bus perrašytos. Viso kodo lygiagretinti taip pat neapsimoka.

Eksperimentai

Buvo realizuoti du uždaviniai. Pirmasis, masyvų sudėtis, pasirinktas kaip paprastos realizacijos uždavinys, kuris puikiai atvaizduoja, kad ne visada verta taikyti lygiagrečius sprendimus. Antras uždavinys, matricų sandauga (ir optimizacijos), dažnai naudojamas kaip klasikinis lygiagretinimo pavyzdys, kadangi gijų skaičiavimai nėra tarpusavyje susieti.

Duotiesiems skaičiavimams su GPU buvo naudojama NVIDIA GTX 1060 plokštė (rezultatai pateikti skyriuose „Masyvų sudėtis“ ir „Matricų sandauga“) ir „Quadro“ K5000 (skyrius „Quadro“). CPU – Intel i7-7700, LGA1151.

Kadangi kiekvienos operacijos laikas gali svyruoti, kiekvienas testas atliekamas 20 kartų ir imamas atlikimo laiko vidurkis. Atliekamų testų skaičius (20) pasirinktas kaip optimaliausias vidurkiams apskaičiuoti, atsižvelgiant į ilgai užtrunkančius CPU skaičiavimus. Eksperimentų vykdymo metu kompiuterio apkrova maksimaliai sumažinama.

Vykdymo laikas matuojamas pasitelkiant *<time.h>* biblioteką ir *cudaEvents*.

Kadangi gijos grupuojamos į lynus po 32 vienetus, duomenų skaičiai parinkti taip, kad dalytųsi iš 32 be liekanos – t. y. visi lynai būtų pilni. GTX plokštė yra 6 GB atminties, „Quardo“ – 4 GB. Pastebėta, kad, bandant dauginti 4096^2 elementų matricas, programos vykdymo laikas tampa lygus 0 s, kas reiškia, jog plokštė nesugeba apdoroti duomenų ir atlikti veiksmų, t. y. veikia nekorektiškai.

Masyvų sudėtis

Imami du A ir B masyvai (dydžiai kinta, žr. grafikus ir lenteles) ir užpildomi pagal formules. C masyvo elementai skaičiuojami sudėjus atitinkamus A ir B masyvų elementus:

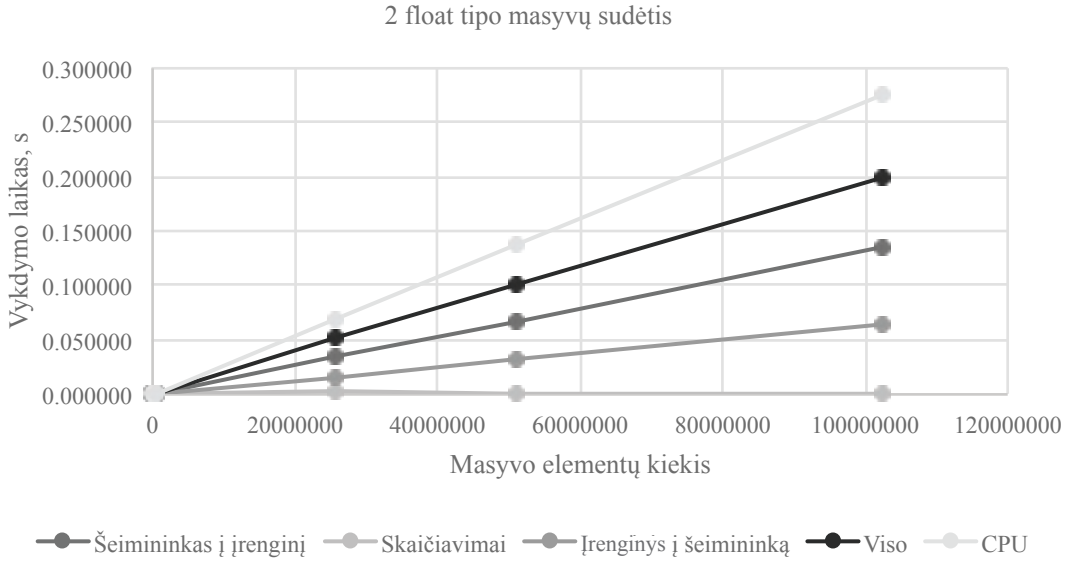
$$A_i = i \qquad B_i = j \qquad C_i = A_i + B_i \qquad (1)$$

Masyvams užpildyti atsitiktinių skaičių generatorius nenaudojamas, kad dėl psichologinio komforto visada būtų vienodos reikšmės. A ir B masyvų užpildymo laikas neįskaičiuojamas nei į GPU, nei į CPU vykdymo laiką. Algoritmo sudėtingumas – $O(N)$.

Duomenų skaičius šio skyriaus lentelėse ir grafikuose – tai elementų skaičius N kiekviename iš sudedamų A, B ir rezultato C masyvų. Atitinkamai per kiekvieną skaičiavimą du N duomenų masyvai yra kopijuojami į įrenginį iš šeimininko ir atgal grąžinamas N skaičius C masyvo elementų.

Šeimininku (angl. *host*) vadinama viskas, kas priklauso x86, x64 architektūroms arba ARM procesoriams – tai gali būti kintamieji, atmintis, funkcijos ir t. t. Dažniausiai naudojama būtent atminčiai nurodyti. Įrenginiu (angl. *device*) vadinama viskas, kas priklauso GPU.

Nors GPU sugeba greitai atlikti skaičiavimus, yra viena problema – grafinė plokštė negali atlikti veiksmų su duomenimis, kurie nėra įdėti į vieną iš GPU atminčių tipą. Taigi, prieš pradedant GPU skaičiavimus, reikia nukopijuoti duomenis iš šeimininko į įrenginį (žr. 1.1 pav. „Šeimininkas į įrenginį“) ir, atlikus skaičiavimus, nukopijuoti rezultatą atgal į šeimininką (žr. 1.1 pav. „Įrenginys į šeimininką“).



1.1 pav. *Float* tipo masyvų sudėties laiko sąnaudos

Dėl šios priežasties, esant nedideliam duomenų skaičiui (užduoties atveju – kai masyvą sudaro iki 512 000 elementų), CPU yra pranašesnis, kadangi jam duomenų kopijuoti nereikia. Tačiau, peržengus nurodytą elementų skaičių, pranašesnis tampa GPU. Grafike galima detaliai matyti, kaip pasiskirsto GPU skaičiavimų laikas – patys skaičiavimai beveik neužtrunka, didžiąją dalį sudaro duomenų perkėlimas. „Iš viso“ grafike – tai duomenų kopijavimo pirmyn ir atgal bei pačių skaičiavimų laiko suma, t. y. visas GPU sugaištas laikas. Tačiau netgi su papildoma duomenų kėlimo pirmyn ir atgal užduotimi GPU įvykdo darbą greičiau, negu CPU atlieka skaičiavimus. Tai reiškia, kad, norint lygiagretinti užduotį, svarbu įvertinti duomenų kiekį: jeigu jis ganėtinai mažas, iš grafinio procesoriaus galima nesulaukti jokios laiko ekonomijos, ypač kopijuojant 64 bitų duomenų tipus (pvz., *double*). Jeigu su *int* ir *float* duomenų tipais dar pavyksta nežymiai sutaupyti dirbant su dideliais duomenų skaičiais, net ir su jais, dirbant su *double* duomenų tipu, nors skaičiavimų dalis su GPU ir įvykdoma greičiau negu su CPU, duomenų kopijavimo laiko sąnaudos yra didesnės už skaičiavimo laimėjimą (žr. 1.1 lentelę).

1.1 lentelė. **Double** ***tipo duomenų masyvų sudėties GTX plokštės laikas***

Procesas	**Elementų skaičius masyve**				
	51200	256000	512000	2560000	51200000
Šeimininkas į įrenginį, s	0.000200	0.000750	0.001350	0.066750	0.135450
Skaičiavimai, s	0.000050	0.000200	0.000350	0.001850	0.000100
Įrenginys į šeimininką, s	0.000050	0.000100	0.000750	0.032250	0.066100
Iš viso, s	0.000300	0.001050	0.002450	0.100850	0.201650
CPU, s	0.000100	0.001000	0.001500	0.069550	0.140450

Matricų sandauga

Matricų sandaugos uždavinys yra tapęs tradiciniu CUDA skaičiavimų pavyzdžiu, kadangi kiekvieno elemento skaičiavimai yra nepriklausomi ir gali būti vykdomi lygiagrečiai, kiekviena gija skaičiuoja savo elemento reikšmę. Yra sukurta ne viena biblioteka, leidžianti pasiekti geriausių rezultatų, tačiau čia pateikti pavyzdžiai parašyti „ranka“, kadangi jų tikslas – išaiškinti CUDA veikimo principus ir optimizavimo kelius, kuriuos jie atveria. C matricai-rezultatui gauti dauginamos dvi kvadratinės matricos A ir B, jos užpildomos iš anksto pagal formules:

$$A_{i,j} = i + j \qquad B_{i,j} = i - j \qquad C_{i,j} = \sum_{r=1}^{m} a_{i,r} b_{r,j} \ \ (i = 1, \dots m; j = 1, \dots m) \qquad (2)$$

Dėl skaitymo patogumo grafikuose ir lentelėse duomenų skaičius pateikiamas N^2 formatu, kur N^2 – tai kiekvienos A, B ir C matricų dydis.

Šiuo atveju bus patogiau naudoti dvimačius gijų ir blokų masyvus. Gijų masyvo matmenys = [32] [32], iš viso yra 1024 gijos; bloko masyvo matmenys priklausys nuo duomenų skaičiaus N^2 – [N/32][N/32]. Skaičius 32 pasirinktas pagal bankų kiekį bendrojoje atmintyje.

Paprastosios realizacijos be optimizacijų algoritmo sudėtingumas – $O(N^3)$.

Visiems trims duomenų tipams (*float*, *double* ir *int*) pateiktas šių realizacijų laikas kartu su duomenų kopijavimo sąnaudomis:

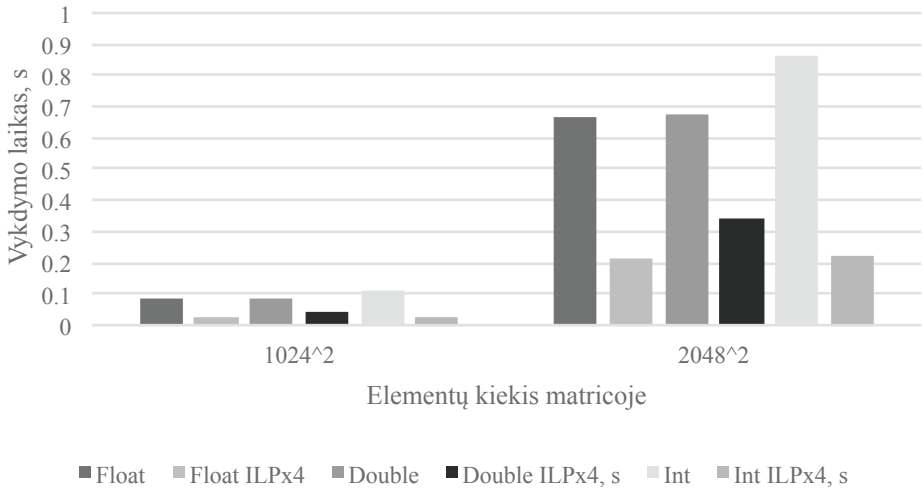
- paprastosios „eilutė*stulpelis“ realizacijos būdas (lentelėse vadinamas GPU);
- realizacija naudojant bendrąją atmintį vietoj globaliosios. Bendroji atmintis yra greitesnė, tačiau reikia deramai pasirūpinti duomenų išdėstymu. Jeigu vieno lyno N gijų kreipiasi į skirtingus 8 baitų žodžius, gulinčius viename banke, tai yra vertinama kaip N lygio konfliktas, ir laiko gali būti sugaištama net daugiau negu naudojant lėtesnę globaliąją atmintį („Bankų konfliktai“, žr. 1.2 lentelę);
- bendrosios atminties „bankų konfliktų“ sprendimas retkarčiais gali būti elementarus. Matricų sandaugos atveju užtenka pridėti papildomą stulpelį prie bendrosios atminties tinklo arba sukeisti *i* ir *j* indeksus, kreipiantis į elementą („Indeksų sukeitimas“, žr. 1.2 lentelę), ir jau matoma laiko ekonomija;
- įmanomas ne tik lygiagretinimas pagal gijas (angl. TLP, *Thread-Level Parallelism*), bet ir pagal instrukcijas (angl. ILP, *Instruction-Level Parallelism*), kai viena gija atlieka kelis skaičiavimus. Čia pateikiamas dviejų instrukcijų vienai gijai sprendimas („ILPx2“, žr. 1.2 lentelę);
- gilesnis ILP, kai viena gija atlieka 4 veiksmus („ILPx4“, žr. 1.2 lentelę).

Kadangi sandaugos algoritmas yra sudėtingesnis negu sudėties, gaunama akivaizdi ekonomija, palyginus su CPU veikimo laiku dirbant su visais duomenų tipais, netgi su „sunkiasvoriu“ *double* (žr. 1.2 lentelę).

1.2 lentelė. **Double *tipo duomenų matricų sandaugos laikai***

Realizacija	Elementų skaičius matricoje		
	512^2	1024^2	2048^2
GTX, s	0,006800	0,091100	0,684022
„Bankų konfliktai“, s	0,007050	0,098150	0,760320
Indeksų sukeitimas, s	0,006100	0,094700	0,734625
ILPx2, s	0,003350	0,028650	0,195650
ILPx4, s	0,003600	0,025550	0,185100
CPU, s	0,422400	3,393300	27,428799

1.2 pav. pavaizduoti visų trijų duomenų tipų paprastų realizacijų laikai ir geriausios optimizacijos, keturių instrukcijų ILP vykdymo laikai. Čia pateikti tik atlikti skaičiavimai GTX plokštėje be duomenų kopijavimo. Gerai matyti, kad *int* leidosi labiausiai optimizuojamas (iki 4 kartų), *double* sumažino laiką tik per pusę. Dirbant su CPU sugaištama žymiai daugiau laiko visais skaičiavimo atvejais – nuo 40 iki 130 kartų lėčiau, palyginus su GPU, įskaitant duomenų persiuntimą ILP realizacijai.



1.2 pav. *Float*, *double* ir *int* duomenų tipų masyvų sandaugos laikai ir ILPx4 optimizacija

„Quadro“

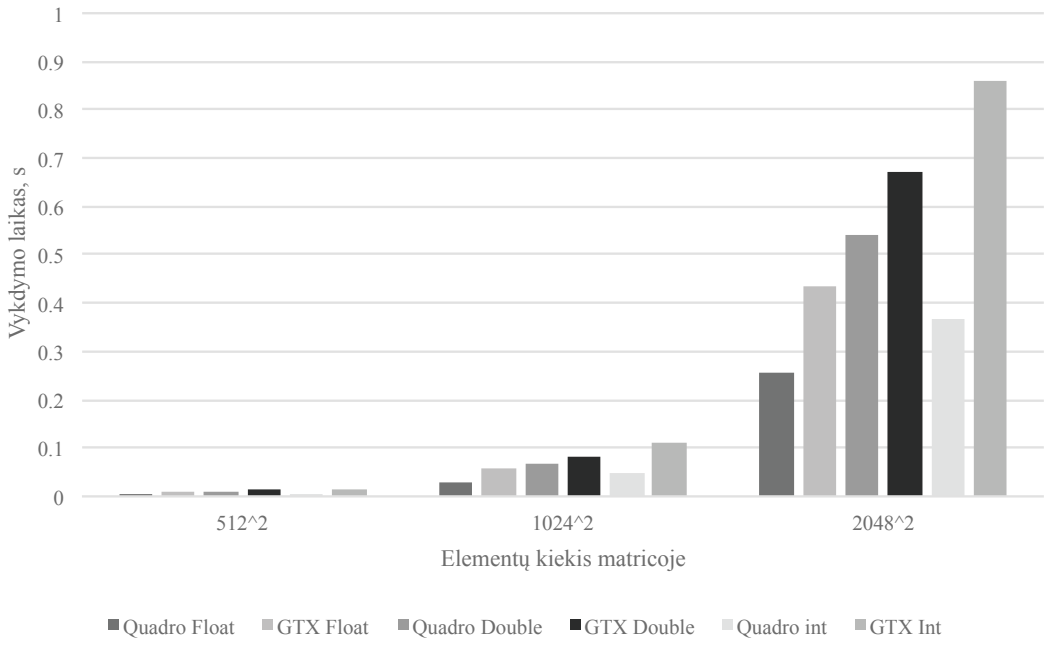
Buvo atlikti skaičiavimai su VGTU „Vilko“ klasteryje esančia NVIDIA „Quadro K5000“ grafine plokšte, kuri irgi naudoja CUDA 8.0 (žr. 1.3 lentelę). „Quadro“ nuo GTX skiriasi tuo, kad pirmasis tipas naudojamas būtent lygiagretiems skaičiavimams atlikti, o antrasis – vaizdui pateikti ir žaidimams. Yra ir trečiasis tipas – „Tesla“, skirtas serveriams. Jis šiame darbe nenagrinėjamas.

Masyvų sudėties atveju gaunamas nedidelis (iki 0,009 s sudedant 51 200 000 *double* elementų (žr. 2.14 lentelę, kuri atvaizduoja $\Delta T = T_{GTX} - T_{Quadro}$)), palyginus su GTX vykdymo laiko ekonomija, laiko skirtumas arba net ilgesnis vykdymas su mažais duomenų skaičiais (neigiamos reikšmės – 1.3 lentelėje). Taigi net su specializuota „Quadro“ lygiagretinti nesudėtingus algoritmus neapsimoka.

1.3 lentelė. ***Masyvų sudėties vykdymo laiko skirtumas tarp GTX ir „Quadro“***

Duomenų tipas	**Elementų skaičius masyve**					
	512	51200	256000	512000	25600000	51200000
float	0,000093	0,000134	–0,000056	0,000054	0,001348	0,003872
double	0,000091	–0,00005	–0,000238	0,000129	0,000983	0,009139
int	–0,000012	0,000060	–0,000069	0,000000	0,001370	0,003422

Geresnė situacija yra su matricų sandauga. Čia matoma iki 0,5 s ekonomija atliekant veiksmą su 2048^2 *int* tipo elementais arba iki 2,4 karto greitesni skaičiavimai su *int* tipo elementais, palyginus su GTX.



1.3 pav. Matricų sandaugos vykdymo laikas su „Quadro“ ir GTX plokštėmis

Kitaip negu su sudėtimi, su matricų sandauga matoma laiko ekonomija, kuri didėja priklausomai nuo elementų skaičiaus. Matomas aiškus *int* tipo paspartinimas – tai, ko trūko GTX, skirtam iš esmės apdoroti skaičius su slankiuoju kableliu (žr. 1.3 pav.). Grafike pateikti paprasčiausio sprendimo be optimizacijų vykdymo laikai.

Kitas skirtumas tarp GTX ir „Quadro“ skaičiavimų – „bankų konfliktų“ sprendimas. Jeigu GTX optimaliausias sprendimas yra indeksų sukeitimas (x ⇔ y), tai „Quadro“ rodo geresnius rezultatus pridėjus papildomą stulpelį bendrosios atminties tinkle, nors toks metodas net pailgina skaičiavimus GTX atveju (žr. 1.4 lentelę). Lentelėje pateikti *float* tipo skaičiavimo duomenys.

1.4 lentelė. ***„Bankų konflikto“ sprendimų laikai su GTX ir „Quadro“ plokštėmis***

Realizacija	Elementų skaičius matricoje		
	512^2	1024^2	2048^2
Bendroji atmintis GTX, s	0,009006	0,064550	0,512150
Papildomas stulpelis GTX, s	0,015519	0,118800	0,947400
Indeksų sukeitimas GTX, s	0,007991	0,061600	0,488100
Bendroji atmintis „Quadro“, s	0,004000	0,037000	0,298000
Papildomas stulpelis „Quadro“, s	0,001000	0,014000	0,111000
Indeksų sukeitimas „Quadro“, s	0,003050	0,031000	0,253000

Nors „Quadro“ CC ir mažesnis (3.0, palyginti su GTX 6.1), netgi esant tokiam architektūrų skirtumui skaičiavimams skirta plokštė pasiekia žymiai geresnių rezultatų. Duomenų kopijavimas užtrunka iki 0,002 s trumpiau (su 2048^2 *double* duomenų tipo elementais).

Išvados

Skaičiuojant su CUDA, ganėtinai ilgai užtrunka duomenų kopijavimas iš šeimininko į įrenginį ir atgal, kas netgi gali pasiglemžti visą atliekant skaičiavimus sutaupytą laiką. Dėl šios priežasties apsimoka lygiagretinti sprendimus, atliekančius daug veiksmų, pavyzdžiui, matricų daugybą (sudėtingumas – $O(n^3)$), ir neapsimoka lygiagretinti paprastos masyvų sudėties (sudėtingumas – $O(n)$), nes duomenų kopijavimas panaikins skaičiavimo laiko sąnaudų ekonomiją.

Priklausomai nuo duomenų apimties, svarbu parinkti tinkamą blokų ir tinklo dydį, neišskirti daugiau atminties, negu reikia, kad SM galėtų apdoroti po kelis blokus. Duomenys turi būti įrašyti efektyviai, kad galima būtų naudoti užklausų transakcijų jungimą. Naudojantis bendrąja atmintimi reikia pasirūpinti, kad gijų kreipimasis į atmintį nesukeltų „bankų konflikto“ (specifiniais atvejais padeda tokie elementarūs veiksmai kaip fikcinis matricos transponavimas, t. y. *i* ir *j* indeksų sukeitimas per kreipimąsi). ILP naudojimas gali papildomai kelis kartus paspartinti skaičiavimus.

Net ir senesnės „Kepler“ architektūros su CC (angl. *Compute Capability*) 3.0 specializuota skaičiavimams skirta „Quadro“ K5000 plokštė pasiekia iki 2,5 karto geresnių rezultatų (*int* tipo matricų sandauga) negu bendrosios paskirties „Pascal“ CC 6.1 GTX 1060.

Geriausia optimizacija matricų sandaugai atlikti – 4 instrukcijų ILP. Didžiausia laiko ekonomija (iki 160 kartų) pasiekiama dauginant 2048^2 *double* elementų matricas, mažiausia (iki 40 kartų) – 512^2 *int* tipo matricas, lyginant GTX grafinės plokštės ir CPU vykdymo laikus.

Literatūra

1. Cole C., 2014, *CUDA In Action Spotlights*. Prieiga per internetą: http://www.nvidia.com/object/cuda-in-action.html [žiūrėta 2017-04-29].
2. *CUDA Toolkit Documentation v8.0.61*. Prieiga per internetą: http://docs.nvidia.com/cuda/index.html#axzz4fWu7lTPu [žiūrėta 2017-04-28].
3. Glaskowsky P. N., 2009, *NVIDIA's Fermi: The First Complete GPU Computing Architecture*. Prieiga per internetą: http://www.nvidia.com/content/pdf/fermi_white_papers/p.glaskowsky_nvidia's_fermi-the_first_complete_gpu_architecture.pdf [žiūrėta 2017-06-02].
4. *GPU-Accelerated Libraries*. Prieiga per internetą: https://developer.nvidia.com/gpu-accelerated-libraries [žiūrėta 2017-04-28].
5. Karimi K., Dickson N. G., Hamze F., *A Performance Comparison of CUDA and OpenCL*. Prieiga per internetą: https://arxiv.org/ftp/arxiv/papers/1005/1005.2581.pdf [žiūrėta 2017-04-29].
6. Langer B., 2015, *Arbitrary-Precision Arithmetics on the GPU*. Prieiga per internetą: http://old.cescg.org/CESCG-2015/papers/Langer-Arbitrary-Precision_Arithmetics_on_the_GPU.pdf [žiūrėta 2017-04-20].
7. *OpenCL. Что это такое и зачем он нужен? (если есть CUDA)*. Prieiga per internetą: [žiūrėta 2017-04-29].
8. Давлеткалиев Р., 2011, Введение в параллельные вычисления. Prieiga per internetą: https://habrahabr.ru/post/126930/ [žiūrėta 2017-04-29].
9. Перепелкин Е., 2012, *NVIDIA CUDA и OPENACC*. Prieiga per internetą: https://www.youtube.com/playlist?list=PLhlTilzRdxykC7dmxh0Xp2kJNy0LSTUfO [žiūrėta 2017-04-29].

Summary

PARALLEL COMPUTING WITH CUDA

J. Semenenko, D. Šešok

The principles of computing with NVIDIA's parallel computing platform CUDA are analysed in the paper. Two numerical experiments, array addition and matrix multiplication and optimizing matrix multiplication (shared memory, bank conflict solutions, instruction-level parallelism) with Geforce and Quadro graphics cards and the CPU were run. Time values for computing int, float, double data types are provided in the paper.

Keywords: CUDA, graphics processing unit, CPU, parallel computing, numerical experiment.

Santrauka

LYGIAGRETŪS SKAIČIAVIMAI SU CUDA

J. Semenenko, D. Šešok

Straipsnyje pateikiami NVIDIA CUDA skaičiavimų technologijos veikimo principai, darbo su CUDA ypatumai. Su „GeForce“ ir „Quadro“ grafinėmis plokštėmis bei CPU atlikti du skaitiniai eksperimentai – masyvų sudėtis ir matricų sandauga, matricų sandaugos optimizacijos (bendroji atmintis, „bankų konfliktų“ sprendimai, lygiagretinimas pagal instrukcijas), analizuojami vykdymo laiko sąnaudų rezultatai dirbant su *int*, *float* ir *double* duomenų tipais ir skirtingais duomenų skaičiais.

Prasminiai žodžiai: CUDA, grafinis procesorius, procesorius, lygiagretūs skaičiavimai, skaitiniai eksperimentai.

Įteikta 2017-06-03
Priimta 2017-06-23