Lietuvos matematikos rinkinys
Lietuvos matematikų draugijos darbai, ser. B
53 t., 2012, 254–259

ISSN 0132-2818
www.mii.lt/LMR/

Greitosios Furjė transformacijos algoritmo CUDA architektūroje analizė

Beatričė Andziulienė, Evaldas Žulkas, Audrius Kuprinavičius

Klaipėdos universitetas, Informatikos inžinerijos katedra
H. Manto 84, LT-92294 Klaipėda
E. paštas: beata@ik.ku.lt, zulkas.e@gmail.com, kuprinavicius.a@gmail.com

Santrauka. Darbe aptariamas Greitosios Furjė transformacijos (GFT) algoritmas, realizuotas grafinių CUDA architektūros procesorių bendros paskirties skaičiavimų vykdymui. Analizuojama algoritmo struktūra ir algoritmo etapų vykdymo našumas. Panaudojant spartos analizės metodą, nustatytos algoritmo paskirstymo tarp centrinio ir grafinio procesoriaus galimybės, priklausomai nuo algoritmo etapų vykdymo trukmių ir algoritmo struktūros.
Raktiniai žodžiai: GFT, bendrosios paskirties grafininių procesorių algoritmai, CUDA.

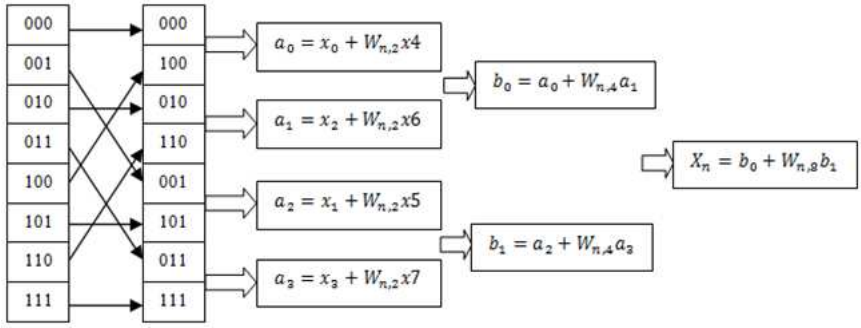
Įvadas

Informacinėse sistemose, norint išgauti reikalingą informaciją, reikia apdoroti didelius duomenų kiekius, todėl reikia naudoti greitus ir našius kompiuterius, kurie yra brangūs. Visą laiką augantis apdorojamų duomenų kiekis tokiose informacinėse sistemose sukuria poreikį kurti kuo didesnio produktyvumo kompiuterinius sprendimus, pavyzdžiui, panaudojant modernios kompiuterinės grafikos posistemės galimybes. Egzistuoja tokios sistemos, kuriose duomenys apdorojami centriniame procesoriuje, kartu išnaudojant didelio lygiagretinimo lygmens grafinės posistemės privalumus [3, 4, 11]. Greitosios Furjė transformacijos (GFT) taikymas ir įgyvendinimas grafiniame procesoriuje leidžia pasiekti maksimalų kompiuterinės sistemos skaičiavimo resursų panaudojimo efektyvumą, apjungiant centrinio ir grafinio procesoriaus galimybes, išgauti kuo didesnę skaičiavimų spartą [3, 11].

Kuriant duomenų apdorojimo ir skaičiavimų su duomenimis algoritmus, svarbu atkreipti dėmesį į bendrą proceso vykdymo greitį ir matematinių skaičiavimų efektyvumą. GFT algoritmas gali būti įvykdytas sparčiau naudojant GPU (angl. *graphics processing unit*), ir optimizuojant atomines matematines operacijas. Darbo tikslas – išanalizuoti GFT algoritmo vykdymą grafiniame CUDA architektūros (angl. *Compute Unified Device Architecture*) procesoriuje.

1 Furjė transformacijų taikymai

GFT naudojama fizinių mokslų srityje ir kompiuteriniuose algoritmuose, kuriuose nagrinėjamas signalas, turintis tam tikrą dažnį [3, 6, 7]. Ryšių teorijoje signalas įprastai suprantamas kaip įtampa, o Furjė transformacija skirta suprasti, kaip signalas elgiasi, kai jis apdorojamas filtrais, stiprintuvais ir dalinamas per ryšio kanalus. Netgi diskretinis skaitmeninis ryšys, kuriame naudojami 0 ir 1 duomenims persiųsti, yra tam tikro



1 pav. Duomenų masyvo iš 8 elementų dvejetainės apgrąžos metodas.

dažnio. Tokiu atveju Furjė transformacija taikoma tiek sudėtinguose tinklų valdymo įrenginiuose signalui analizuoti, tiek paprastoms kelių bitų perdavimo operacijoms atlikti. Furjė transformacija gali būti taikoma astronomijoje, kai iš signalo negalima išgauti reikiamos informacijos, apdorojant vaizdinę medžiagą, todėl reikia remtis radijo bangų spektro analize [8]. GFT taip pat taikoma signalų analizėje (koreliacijos skaičiavimuose), sistemų analizėje (sistemos išėjimo/įėjimo signalų skaičiavimuose), duomenų suspaudimo algoritmuose [1].

Dalykinės programos, paremtos grafinių procesorių skaičiavimais, pasiekia didesnį našumą lyginant su centrinio procesoriaus išteklius naudojančiomis programomis [9]. Grafinės plokštės atliekančios bendros paskirties skaičiavimų užduotis leidžia kurti sistemas, naudojančias mažiau elektros energijos (didesnis skaičiavimo našumas per galios vienetą). Mažesni užduotį atliekančios kompiuterinės sistemos kūrimo ir įrangos kaštai didina tokių, naudojančių GPU, sistemų panaudojimo galimybes.

Atliekant signalo spektro analizę, remiantis GFT [2, 5], algoritmas vienmačiam duomenų masyvui apdoroti gali būti atliekamas naudojant dvejetainę apgrąžą ir Danielson–Lanczos metodą [10, 12]. Pirmiausia originalus masyvas turi būti transformuotas, kad būtų galima atlikti Danielson–Lanzcos algoritmą. Pavyzdžiui, kompleksinių skaičių masyvas su indeksu turi transformuotis į masyvą su apgrąžos dvejetainiu indeksu. Jeigu dvejetainis indeksas yra 0b00001, tai dvejetainė apgrąža bus 0b10000. 1 pav. kairėje pusėje galima matyti, kas nutinka duomenų masyvui po transformacijos. Masyvas padalinamas į dvi lygias dalis pagal *indeksas* kintamąjį. Nagrinėjant duomenų apgrąžą pirmoje masyvo pusėje pastebimas veidrodinio atspindžio efektas kitoje dalyje. Šis efektas palengvina dvejetainės apgrąžos lygiagretinimą remiantis pirmos masyvo pusės indeksais ir naudojant juos antros masyvo dalies apgrąžai su *indeksas* + 1. Šį efektą galima taikyti, jeigu pakeitimai daromi tik pirmoje masyvo pusėje. Tai reiškia, kad reikšmė, kurios indeksas $k1$ yra sukeičiama vietomis su $k2$, kur $k2$ gaunamas apverčiant veidrodinio atspindžio principu binarinį skaičių $k1$, t. y. reikšminis bitas tampa mažiausiai reikšminiu. Turint naują indeksavimą, reikšmės apjungiamos pagal formulę $X_n = b_0 + W_{n,8} \cdot b_1$, kur $W_{n,8}$ yra periodinė seka su periodu n, b_0 ir b_1 – kintamieji su indeksu. 1 pav. vaizduoja visą GFT algoritmą vienmačiam masyvui, pradedant su įvesties duomenimis x_k ir baigiant su išvesties reikšme X_n.

Siekiant sumažinti apdorojimo laiką svarbu optimizuoti patį algoritmą. GFT optimizavimas paremtas duomenų išskaidymu į lyginių ir nelyginių indeksų duomenų masyvus, tokiu būdu algoritmo sudėtingumas mažėja iki $N \cdot \log_2 N$.

2 Algoritmo realizacija

Darbe naudojamas grafinis procesorius GeForce GT 540M. Grafinės kortos sistemos specifikacijos: 96 CUDA branduoliai, atminties pralaidumas – 28,8 GB/s, CPU/atminties taktinis dažnis – 1344/900 MHz. Algoritmas grafiniam procesoriui buvo pritaikytas ir realizuotas C#.NET kalba, remiantis centriniam procesoriui skirtu GFT algoritmu [12].

Nagrinėjamas signalas yra realiųjų skaičių masyvas. Egzistuoja specialių algoritmų realiems skaičiams apdoroti, tačiau darbe analizuojamas platesnį pritaikymą turintis kompleksinių skaičių algoritmas. Svarbus yra duomenų kiekis, kuris bus siunčiamas transformuoti ir duomenų pagrindinis (nešamasis) dažnis. Duomenų kiekis turi būti 2^n skaičius, tačiau visų duomenų persiųsti nebūtina, negauti duomenys užpildomi nuliais. Kai užpildoma realioji skaičių dalis masyve, menamoji dalis užpildoma nuliais. Norint sužinoti pagrindinį signalo dažnį, reikia rasti absoliutų masyvo maksimumą, ir dažnis bus to masyvo indekse. Jeigu maksimumas atsiranda kintamajame su indeksais, pvz., [92] [93] (realioji ir menamoji dalys), tada pagrindinis dažnis bus $92/2 = 46$ Hz. Antroji masyvo pusė, kuri yra pirmosios veidrodinis atspindys, turi būti ignoruojama. Vaizduojant Furjė signalą grafiškai, laiko ašis prieš ir po transformacijos nekinta.

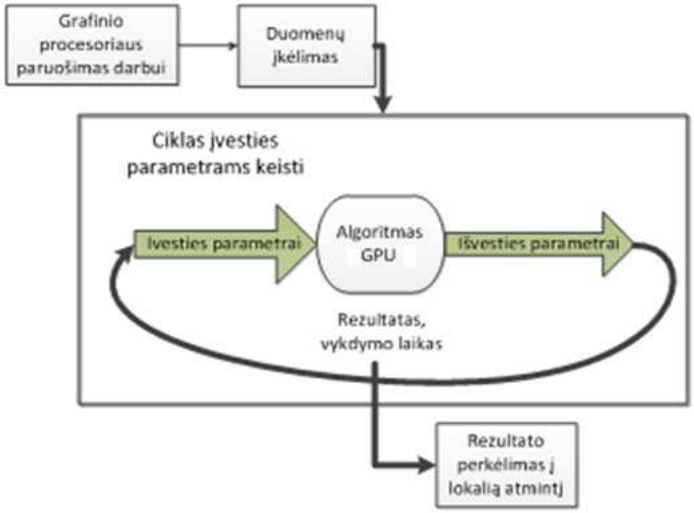
Išanalizavus GFT struktūrą ir panaudojus programinius kodo vykdymo spartos analizės įrankius, išsiaiškinta kiekvienos paprogramės dalies vykdymo trukmė: kintamųjų apibrėžimas trunka apie 0,1–0,5 proc., duomenų vektoriaus sudarymas – nuo 1–2 proc., dvejetainės apgrąžos panaudojimas – nuo 20–30 proc., o Danielson–Lanczos metodo taikymas užima 67,5–78,9 proc. viso vykdymo laiko. Prieš algoritmo naudojimą, visiems kintamiesiems išskiriama atminties lokalioje ir grafinėje atmintinėje. Pabrėžtina, kad dėl kelių procentų kodo vykdymo neverta pernešti dalies kodo centrinio procesoriaus apdorojimui. Tiek dvejetainės apgrąžos, tiek Danielson–Lanczos metodas yra lengvai lygiagretinami procesai, todėl visi duomenys perkeliami iš lokalios centrinio procesoriaus valdomos atminties į grafinės posistemės atmintį. Visos skaičiavimo procedūros atliekamos jau grafinio procesoriaus viduje, kol gaunamas galutinis rezultatas. Jeigu siekiama gauti nešamąjį signalo dažnį, kintamasis *pagrindinis dažnis* turi būti kopijuojamas atgal iš grafinės atminties į lokalią atmintį.

3 Algoritmo tikrinimas ir testo rezultatai

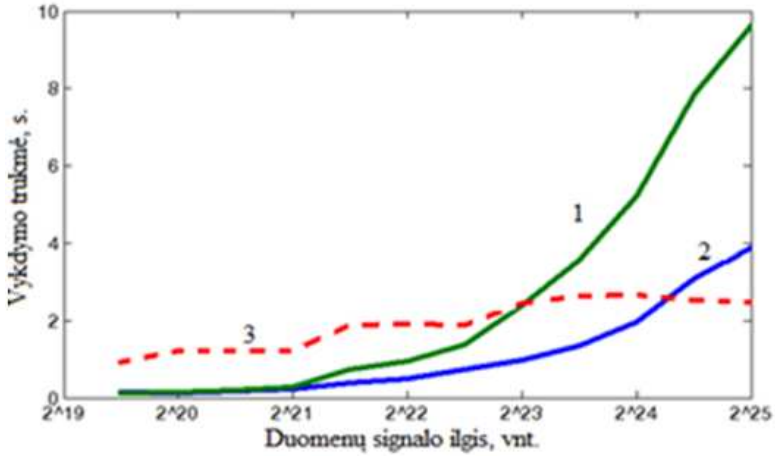
Algoritmas buvo tikrinamas su netikrais, kompiuteriu generuotais, duomenimis: $\sin(100\pi) + \sin(220\pi) +$ triukšmas. Triukšmas formuojamas pridedant prie kiekvienos diskrečios signalo reikšmės po atsitiktinį skaičių. Įvedus triukšmą signalas pasikeičia – signalo periodiškumas ir simetriškumas sumažėja.

Centrinio ir grafinio procesoriaus spartos palyginimas. Norint rasti santykį tarp centrinio ir grafinio procesorių skaičiavimų atliekant GFT algoritmą spartos, reikia išsiaiškinti, kokie turi būti programinio kodo metodo ciklo įvesties parametrai, geriausiai atspindintys ryšį tarp vykdymo spartos. Reikia atkreipti dėmesį, kad grafinio procesoriaus inicializavimas trunka kelias sekundes, todėl pirmųjų bandymų duomenys nebūna kokybiški ir tikslūs.

Įvertinant vykdymo spartą grafiniame CUDA architektūros procesoriuje, reikia papildomai nagrinėti grafinio procesoriaus paruošimą darbui, duomenų įkėlimą į GPU



2 pav. Algoritmo spartos testavimas.



3 pav. Grafinio ir centrinio procesoriaus vykdymo spartos santykis: 1-CPU; 2-GPU; 3-CPU/GPU.

atmintį ir rezultato perkėlimą atgal į lokalią darbinę atmintį (2 pav.). Geriausiai spartos analizės būdu algoritmą tikrinti su skirtingais ciklo įvesties parametrais, kurie turi nurodyti duomenų ilgį. Šiuo atveju grąžinamas rezultatas turi būti sudarytas iš paties rezultato duomenų ir kiekvieno etapo vykdymo laiko.

Dėl grafinės posistemės operatyviosios atminties apribojimų, buvo nustatyti didžiausi duomenų kiekiai ($n = 19$–25), kuriuos Furjė transformacijos algoritmas sėkmingai apdoroja. Centrinio ir grafinio procesorių skaičiavimo spartos santykio kitimas priklausomai nuo duomenų kiekio parodytas 3 pav. (3 kreivė). Prie tam tikro duomenų signalo ilgio, grafinio ir centrinio procesoriaus užduoties įvykdymo sparta keičiasi

priklausomai nuo vykdomų instrukcijų, naudojamų duomenų kiekio ir procesorių architektūros. Matyti, kad vykdant GFT grafiniame procesoriuje skaičiavimų sparta išauga iki dviejų kartų.

Centrinių procesorių architektūrai skirtą GFT kodą pritaikant grafinių procesorių architektūrai, kodas papildomas lygiagretaus duomenų apdorojimo ir darbo paskirstymo srauto procesoriams algoritmo dalimis. Grafiniame procesoriuje, dėl didelio srauto procesorių skaičiaus, architektūriškai lengviau įgyvendinti masinį lygiagretinimą, tačiau centrinio procesoriaus nuoseklaus vykdymo sparta išlieka svarbi.

Dėl kelių (< 10 proc.) procentų veiksmų, atliekamų centriniame procesoriuje dažniausiai neverta aukoti programinio kodo kūrimo ir testavimo laiko, nes išgaunamas nedidelis našumas, o perkeliant duomenis iš lokalios į grafinę atmintį sugaištama daug laiko. Kuo smulkesnė užduotis, tuo centrinis procesorius ją vykdo greičiau, nes prieš pirmąją operaciją CUDA architektūroje grafinė posistemė inicializuojama (iki kelių sekundžių) bendros paskirties užduotims spręsti.

Išvados

1. Įvertinus Greitosios Furjė transformacijos pritaikomumą grafinių procesorių architektūroms iš centrinių procesorių architektūrai skirto kodo, realizacija netampa sudėtinga, apdorojant duomenis lygiagrečiai ir paskirstant užduotį srauto procesoriams.
2. Panaudojant spartos analizės metodą, nustatytos algoritmo paskirstymo tarp centrinio ir grafinio procesoriaus galimybės, siekiant padidinti atliekamų užduočių vykdymo našumą priklausomai nuo algoritmo etapų vykdymo trukmės ir algoritmo struktūros:
   - nustatyta, kad didėjant duomenų signalo nuoskaitų skaičiui, grafinio ir centrinio procesoriaus užduoties įvykdymo sparta keičiasi. Nagrinėjant procesorių našumo santykį, pastebėtas ženklus vykdymo spartos skirtumas grafinio procesoriaus naudai.
   - smulkias užduotis centrinis procesorius įvykdo greičiau, nes prieš pirmąją operaciją CUDA architektūroje inicializuojama grafinė posistemė bendros paskirties užduotims spręsti žymiai sumažina skaičiavimų našumą.
   - centrinis procesorius reikalingas bet kuriai sistemai, net ir tai, kurioje didžioji ar visa algoritmo dalis pernešama grafinei posistemei, nes dauguma įrenginių palaiko duomenų apsikeitimą tik su centriniu procesoriumi.

Literatūra

[1] A. Alkholidi, A. Alfalou and H. Hamam. A new approach for optical colored image compression using the JPEG standards. *Sign. Proc.*, **87**(4):569–583, 2007.

[2] H. Bauke and Ch.H. Keitel. Accelerating the Fourier split operator method via graphics processing units. *Comp. Phys. Commun.*, **182**(12):2454–2463, 2011.

[3] Y. Chen, X. Cui and H. Mei. *GPU Computing Gems*, chapter 39 – Large-scale fast Fourier transform. 2011, pp. 629-642.

[4] R. Lacoste. *The Dark Side Practical Applications for Electronic Designs Concepts*, chapter 6 – The fast Fourier transform from *A* to *Z*. 2010, pp. 79-92.

[5] T. Larsen, G. Pryor and J. Malcolm. *GPU Computing Gems,* Jade Edition, chapter 28 – Jacket: GPU powered MATLAB acceleration. 2012, pp. 387-398.

[6] S. Li, R. Chang and V. Lomakin. *GPU Computing Gems,* Jade Edition, chapter 19 – Fast electromagnetic integral equation solvers on graphics processing units. 2012, pp. 243-266.

[7] X. Li, G. Bi, S. Stankovic and A.M. Zoubir. Local polynomial Fourier transform: A review on recent developments and applications. *Sign. Process.*, **91**(6):1370–1393, 2011.

[8] Y. Ling, M. Ehlers, E.L. Usery and M. Madden. FFT-enhanced IHS transform method for fusing high-resolution satellite images. *ISPRS J. Phot. Rem. Sens.*, **61**(6):381–392, 2007.

[9] H. Liu, H. Liu, X. Tong and Q. Liu. A Fourier integral algorithm and its GPU/CPU collaborative implementation for one-way wave equation migration. *Comp. Geosc.*, 2011.

[10] F.L. Marquezino, R. Portugal and F.D. Sasse. Obtaining the quantum Fourier transform from the classical FFT with QR decomposition. *J. Comp. Appl. Math.*, **235**(1):74–81, 2010.

[11] M. Parker. *Digital Signal Processing*, chapter 10 – Discrete and fast Fourier transforms (DFT, FFT). 2010, pp. 97-112.

[12] H.W. Press, A.S. Teukolsky, T.W. Vetterling and P.B. Flannery. *Numerical Recipes in C*, 2nd edition. pp. 504–521, 1992.

SUMMARY

Analysis of Fast Fourier Transformations algorithm for CUDA Architecture

B. Andziulienė, E. Žulkas, A. Kuprinavičius

In this work Fast Fourier transformation algorithm for general purpose graphics processing unit processing (GPGPU) is discussed. Algorithm structure and individual stages performance were analysed. With performance analysis method algorithm distribution and data allocation possibilities were determined, depending on algorithm stages execution speed and algorithm structure. Ratio between CPU and GPU execution during Fast Fourier transform signal processing was determined using computer-generated data with frequency. When adopting CPU code for CUDA execution, it not becomes more complex, even if stream procesor parallelization and data transfering algorith stages are considered. But central processing unit serial execution).

Keywords: FFT, general purpose GPU algorithms, CUDA.