VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

Julija Semenenko

GRAFINIŲ PROCESORIŲ PANAUDOJIMO SPRENDŽIANT INŽINERINES PROBLEMAS GALIMYBIŲ TYRIMAS

RESEARCH ON GPU USAGE FOR SOLVING ENGINEERING PROBLEMS

Baigiamasis magistro darbas

Informacinių technologijų studijų programa, valstybinis kodas 6211BX016

Duomenų gavybos technologijų specializacija

Informatikos inžinerijos studijų kryptis

Vilnius, 2019

ANOTACIJA

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Informacinių technologijų katedra

ISBN ISSN
Egz. sk. ……..
Data ….-….-….

Antrosios pakopos studijų **Duomenų gavybos technologijų** programos magistro baigiamasis darbas
Pavadinimas: **Grafinių procesorių panaudojimo sprendžiant inžinerines problemas galimybių tyrimas**
Autorius: **Julija Semenenko**
Vadovas: **Dmitrij Šešok**

Kalba: lietuvių

Anotacija

Baigiamajame darbe pateikiama skaičiavimų su grafiniais procesoriais taikomų sprendžiant skirtingus inžinerinius uždavinius apžvalga, GPU ir CPU skaičiavimų palyginimai. Išnagrinėtas šilumos laidumo uždavinys. Įgyvendintas dvimačio šilumos laidumo uždavinio sprendimas Jakobio iteracinio algoritmo pagrindu Python programavimo kalba, *tensorflow-gpu* bibliotekos ir NVIDIA CUDA technologijos pagalba.

Tyrimai atlikti su 6 CPU ir 4 GPU, greičiausias GPU atlieka skaičiavimus 19 kartų greičiau už lėčiausią CPU. Vidutinis GPU laiko laimėjimas 8 kartai. Žymus GPU laimėjimas pastebėtas pradedant nuo 500^2 matricos *float* tipo elementu.

Bendra darbo apimtis: 41 puslapiai be priedų, 16 paveikslų, 17 lentelių, 43 literatūros šaltinių.

Prasminiai žodžiai: CUDA, grafinis procesorius, Jakobio iteracinis algoritmas, lygiagretūs skaičiavimai, skaitiniai eksperimentai, šilumos laidumo uždavinys.

ANNOTATION

Vilnius Gediminas Technical University
Faculty of **Fundamental Sciences**
Department of Information Technologies

ISBN ISSN
Copies No. ……
Date ….-….-….

Master's degree Studies **Data Mining Technology** study programme Master's Graduation Thesis
Title: **Solving engineering problems using global optimization algorithms**
Author: **Julija Semenenko**
Academic supervisor: **Dmitrij Šešok**

Thesis language: Lithuanian

Annotation

In this final work overview of the GPU solutions for different engineering problems is provided, as well as the comparison between CPU and GPU computations and overview of thermal conductivity problem and its existing computational solutions. Python implementation of Jakobi iterative algorithm via tensorflow-gpu library and NVIDIA CUDA technology is provided with the execution time statistics for several GPU.

Experiments were conducted with 6 CPU and 4 GPU. Fastest GPU completed calculations 19 times faster than slowest CPU. In average, GPU were 8 times faster than CPU. Significant speed up in GPU calculations starts at 500^2 *float* type matrix elements.

Structure: introduction, parallel computing overview, thermal conductivity problem, Jakobi iteravite algorithm, computational experiments, conclusions, references.

Thesis consists of 41 pages without appendixes, 16 pictures, 17 tables, 43 references.

Keywords: computational experiments, CPU, CUDA, GPU, Jakobi iterative algorithm, parallel computing, thermal conductivity problem.

Turinys

Paveikslų sąrašas

Lentelių sąrašas

Santrumpų sąrašas

CC	Grafinės plokštės skaičiavimo pajėgumas (angl. *Compute Capability*)
CUDA	Lygiagrečių skaičiavimų architektūra, sukurta *NVIDIA* (angl. *Compute Unified Device Architecture*)
FLOPS / FLOP/s	operacijų su slankiojančio kablelio skaičiais kiekis per sekundę (angl. *FLoating-point Operations Per Second*).
GPGPU	Bendrosios paskirties skaičiavimai su GPU (angl. *General-purpose computing for graphics processing units*).
GPU	Grafinis procesorius (angl. *Graphics Processing Unit*)
ILP	Lygiagretinimas pagal instrukcijas (angl. *ILP, Instruction-Level Parallelism*)
LLS	Linijinių lygčių sistema
ML	Mašininis mokymas (angl. *Machine Learning*)
MLUPS	GPU spartos matavimo vienetas, taikomas Lattice-Boltzmann‘o metodui (angl. *Million Lattice site Updates Per Second*)
ROCm	„AMD“ sukurta lygiagrečių skaičiavimų platforma (angl. *Radeon Open Compute platform*)

Įvadas

Temos aktualumas

Didelės kompanijos, kurių pagrindinė veikla yra pagrįsta skaičiavimais, pavyzdžiui, „Adobe“, ANSYS, „Autodesk“, „MathWorks“, „Wolfram Research“, naudoja ir ragina kitus plėtoti GPU skaičiavimus (Owens, Davis, & Luebke).

Didelė GPU skaičiavimų dalis atitenka mašininio mokimo nišai. Įvarinių rūšių neuroniniams tinklams (Shin, et al., 2019) ir klasifikatoriams (Fambrini, et al., 2018) mokyti su GPU yra plėtojamos specializuotos bibliotekos, kaip, pavyzdžiui, *TensorFlow-GPU*. GPU kūrėjai kuria tokias bibliotekas palaikančią programinę įrangą (pavyzdžiui, *NVIDIA cuDNN*). Tačiau ne tik neuroniniai tinklai yra GPU naudotojai. Didelė dalis inžinerinių uždavinių yra praplečiama lygiagrečiais GPU skaičiavimais, kas leidžia apdoroti žymiai didesnius duomenų kiekius. Yra kuriamos detalios realaus pasaulio simuliacijos (Kuckuk & Köstler, 2018) perrašomi elektros, slėgio ir kitų potencialinių laikų skaičiavimo algoritmai (Ahamed & Magoulès, 2017). GPU skaičiavimai yra diegiami į saugumo sistemas (Filonenko, Hernández, & Jo, 2018) ir taikomi medicinoje (Lu, Ramachandra, Pham, Tu, & Cheng, 2019).

Mokslinis naujumas

Jeigu užduotį galima išskaldyti į paprastesnius veiksmus ir jeigu jiems atlikti nereikalingi prieš tai buvusių skaičiavimų rezultatai, t. y., veiksmus galima atlikti lygiagrečiai, naudojant GPU sprendimo paieškai rezultatą galima gauti žymiai greičiau negu įprastais CPU skaičiavimais. Ir atvirkščiai, realizuojant sunkius, išsišakojusius sprendimus, GPU rezultatas laiko atžvilgių bus labai panašus arba net lėtesnis negu CPU. Praktinė darbo dalis skirta GPU skaičiavimams įteraciniu Jakobio algoritmu. Šis algoritmas dažnai minimas, kaip turintis didelį potencialą lygiagretiems skaičiavimams (Cormie-Bowins, 2012).

Praktinė vertė

GPU skaičiavimai leidžia smarkiai pagreitinti užduotis, kurias galima spręsti lygiagrečiai, tačiau tam reikia kitų metodų. Pavyzdžiui, Krylovo metodas labai populiarus sprendžiant linijines lygčių sistemas (Anzt, et al., 2017), tačiau taikant GPU skaičiavimus, žymaus pagreičio galima laukti taikant iteracinius algoritmus, pvz. Jakobio (Saha, 2017). Šis darbas skirtas parodyti, kokio skaičiavimo pagreičio galima pasiekti su GPU.

Darbo tikslas – ištirti galimybę pagreitinti inžinerinių uždavinių sprendimą, dalį skaičiavimų atliekant naudojant grafinius procesorius.

Darbo uždaviniai:

1. Atlikti mokslinės literatūros analizę: išnagrinėti ankstesnių mokslinių tyrimus, kuriuose atliekami įvairių uždavinių sprendimai taikant grafinius procesorius, aprašyti pasiektus rezultatus.

2. Išnagrinėti šilumos laidumo uždavinį, jį aprašyti ir išspręsti Jakobio iteraciniu algoritmu.
3. Sukurti programinę įrangą skaitiniams eksperimentams atlikti.
4. Atlikti skaitinius eksperimentus, sprendžiant įvairios apimties uždavinius.
5. Palyginti gautus rezultatus.

Tyrimo metodai:

1. Teoriniai: įvairios literatūros nagrinėjama tema analizė.
2. Skaitiniai sprendimo metodai.

Darbo struktūra

Pirmame skyriuje yra pateikiama trumpa GPU istorijos apžvalga, nagrinėjami GPU, pagrinde CUDA technologijos taikymas įvairiems uždaviniams spręsti, duomenų prigimtis, skaičiavimo modeliai, taikymo sritys ir rezultatai. Antroje dalyje nagrinėjamas šilumos laidumo uždavinys, pateikiamas jo aprašymas, matematinis modelis, mokslininkų pakeikti sprendimo metodai, aprašomas Jakobio metodas. Trečioje dalyje yra aprašytas pagal Jakobio metodą sukurtas algoritmas, jo realizacija, naudoti įrankiai. Pateikiami skaitinių eksperimentų rezultatai įvairiems duomenų kiekiams su skirtingoms grafinėmis plokštėmis, rezultatai palyginami su CPU.

Darbo aprobacija

Dalis darbo rezultatų publikuota straipsnyje:

Semenenko, Julija; Šešok, Dmitrij. Lygiagretūs skaičiavimai su CUDA = Parallel computing with CUDA // Jaunųjų mokslininkų darbai = Journal of young scientists. Šiauliai: VšĮ Šiaulių universiteto leidykla. ISSN 1648-8776. 2017, Nr. 47(1), p. 87-93. DOI: 10.21277/jmd.v47i1.135. [CEEOL – Central and Eastern European Online Library; IndexCopernicus] [M.kr.: T007].

Galutinius darbo rezultatus planuojama pristatyti konferencijoje DAMSS 2019, kuri vyks Druskininkuose 2019.11.28 - 2019.11.30.

1. Lygiagretūs skaičiavimai

Pirmoji grafikos procesorių (toliau – GPU, angl. *Graphics Processing Unit*) karta, sukurta XX a. pabaigoje, buvo skirta atvaizduoti 3D vaizdą 2D plokštumoje realiu laiku. Pagrindiniais GPU rinkos motyvatoriais tapo vis didėjantys kompiuterinių žaidimų reikalavimai. Kadangi atvaizdavimo procesas labai imlus resursams, buvo pritaikytos optimizacijos: nematomų plokštumų šalinimas apribojant matomos zonos gylį, skaičiavimų lygiagretinimas. GPU nuskaitydavo 3D scenos aprašymą kaip trikampių ir viršūnių masyvą, bei naudotojo parametrus, pagal kuriuos buvo projektuojamas 2D vaizdas. Įmanoma buvo panaikinti nematomas plokštumas, nurodyti viršūnių spalvą bei atlikti interpoliuotą nudažymą. Taip pat, jau buvo įmanomas objektų tekstūrų nurodymas ir apšvietimo skaičiavimas be šešėlių. Tam, kad atvaizduoti šešėlius, reikėjo papildomų „šešėlių žemėlapio“ (angl. *shadow mapping*) (Williams, 1978) arba „šešėlių apimties“ (angl. *shadow volume*) (Crow, 1977) algoritmų.

Pirmajai GPU kartai priklauso *S3 ViRGE, ATI Rage - ATI Radeon 7500, Matrox Mystique, 3dfx Voodoo, NVIDIA GeForce 256 - GeForce 3* ir kiti. Jiems programuoti buvo plačiai naudojama *OpenCL* sąsaja, kiek mažiau *Direct3D*.

Antroji GPU karta leido naudotojui pačiam nustatyti apšvietimo ir viršūnių koordinačių transformavimo algoritmus. Pirmoji karta palaikė tik numatytuosius. Atsirado galimybė išskaičiuoti ekrano pikselio spalvą – GPU pradėjo vadinti šeideriais (ang. *shade* – šešėliuoti). Pirmąsias programas, irgi vadinamuosius šeiderius, sudarė vos po 20 GPU asemblerio eilučių komandų. Perėjimo iš CPU į GPU ir atgal komandų nebuvo, visi skaičiavimai buvo atliekami naudojant skaičius su fiksuotu kableliu. Augant šeiderių populiarumui atsirado aukštesnio lygio jiems skirtos programavimo kalbos, pavyzdžiui, *NVIDIA* pasiūlė *Cg, Microsoft* pasiūlė *HLSL*.

2003 m. buvo įgyvendintas skaičiavimas naudojant skaičius su slankiojančiu kableliu, davęs pradžią bendrosios paskirties GPU skaičiavimams, nesusietiems su grafika – GPGPU (angl. *General Purpose GPU*). *Direct3D* pirmasis spėjo pridėti šeiderių palaikymą, tačiau *OpenCL* irgi tapo populiarus. GPU rinkoje išryškėjo *ATI* ir *NVIDIA* kompanijos. Lygiagrečių skaičiavimų su GPU tyrimai buvo tobulinami programas suskaldant į mažesnius srautus, vykdomus tuo pačiu metu. Antros kartos GPU pavyzdžiai: *NVIDIA GeForce 4 - 5, ATI Radeon 8500 - X800*.

Trečios kartos GPU leido programuoti šakojimus ir ciklus, o tai savo ruožtu leido kurti sudėtingesnius šeiderius. *OpenGL* 2.0 pradeda palaikyti *GLSL* šeiderinio programavimo kalbą. Aktyviai sprendžiami įvairūs uždaviniai taikant paskirstytus GPU skaičiavimus – GPGPU. Atsiranda sveikųjų ir dvigubo tikslumo (angl. *double*) skaičių palaikymas. Kuriamos GPU programavimo bibliotekos, pavyzdžiui, *RapidMind, Accelerator*, komerciniai GPU skaičiavimai.

3čios kartos GPU pavyzdžiai: *NVIDIA GeForce 6 - 9, ATI Radeon X1K - HD4K*. Grafinių programų kūrimo įrankiai: *OpenGL, DirectX*, šeiderinės kalbos *GLSL, HLSL* ir *CG*.

Tarp GPGPU įgyja populiarumą *RapidMind, NVIDIACUDA, AMD CTM-Brook.*

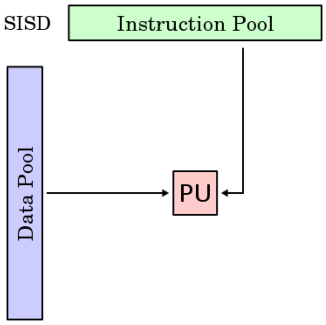
1.1. Flyno taksonomija

Maiklas Flynas dar 1966 m. pasiūlė skaičiuojančiųjų mašinų architektūros taksonomiją (klasifikaciją) pagal komandų ir duomenų lygiagretumą (Flynn, 1972). Klasifikaciją originaliai sudarė 4 klasės, vėliau jį buvo papildyta. Nors ir nevisos klasės turi realių pavydžių, ši taksonomija leidžia greitai sudaryti įspūdį apie kompiuterių architektūros raidos kryptis ir netgi jų raidos istoriją, nuo paprasčiausiu SISD variantų iki MIMD. GPU šioje taksonomijoje atitinka SIMD klasę. Šiuolaikiniai kelis branduolius turintys CPU irgi daugiau nebepriklauso SISD.

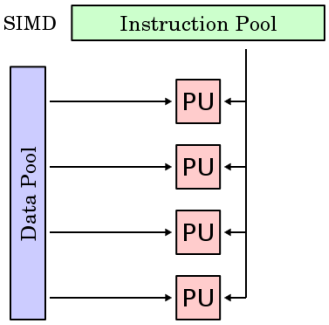
SISD

Viena instrukcija, vienas duomenų srautas (toliau – SISD, angl. *Single Instruction, Single Data*) (žr. 1.1 pav), kas atitinka fon Neimano architektūrą. Procesorius apdoroja duomenų srautą iš duomenų rinkinio (angl. *data pool*) pagal nurodytą vieną instrukciją iš instrukcijų rinkinio (angl. *instruction pool*). Čia jokio lygiagretumo, nei duomenų, nei veiksmų nėra.

$$y = f(x) \qquad (1)$$



1.1 pav. *SISD architektūra*[1]



1.2 pav. *SIMD architektūra*[2]

SIMD

Viena instrukcija, daug duomenų srautų (toliau – SIMD, angl. *Single Instruction, Multiple Data*) (žr. 1.2 pav). Procesorius apdoroja visus duomenų srautus iš duomenų rinkinio pagal nurodytą vieną instrukciją iš instrukcijų rinkinio ir kiekvienam srautui apskaičiuoja jo rezultatą.

$$y_i = f(x_i) \qquad (2)$$

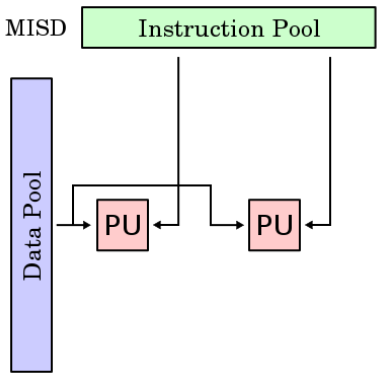
MISD

Daug instrukcijų, vienas duomenų srautas (toliau – MISD, angl. *Multiple Instruction stream, Single Data*) (žr. 1.3 pav.). Instrukcijų lygiagretumas, kai procesorius tuos pačius duomenis vienu

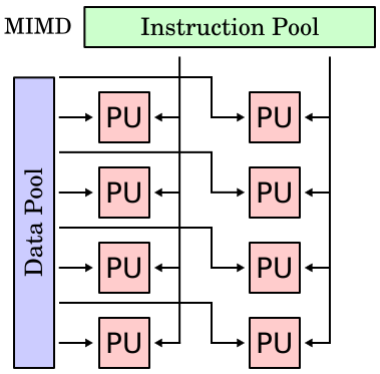
[1] [2] Autorius: I. Cburnett, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2233537

metu apdoroja pagal skirtingas instrukcijas. Ši architektūra beveik nenaudojama, kai kurie netgi mano, kad ši klasė yra tuščia ir gali būti naudojama naujoms architektūrinėms kryptims plėtoti. Labiausiai tinkami atstovai būtų konvejerinės mašinos.

$$y_j = f_j(x) \qquad (3)$$



1.3 pav. *MISD architektūra*[3]



1.4 pav. *MIMD architektūra*[4]

MIMD

Daug instrukcijų, daugialypis duomenų srautas (toliau – MIMD, angl. *Multiple Instruction stream, Multiple Data*) (žr. 1.4 pav). Dažniausiai tai daugiaprocesorinės sistemos, daugiabranduolinės arba klasterinės. Kaip ir SIMD, skirstomos į 2 poklasiuis, priklausomai nuo atminties tipo: bendrosios atminties multiprocesoriai ir NUMA sistemos bei paskirstytos atminties su MPI (angl. *Message Passing Interface*) realizacija.

$$y_{i,j} = f_j(x_i) \qquad (4)$$

GPU architektūra artimiausia SIMD architektūrai, tačiau galimi ir kiti variantai (išskyrus SISD). Pati *NVIDIA* naudoja SIMT sąvoką (angl. *Single Instruction Multiple Thread*). Dažniausiai matematiniai skaičiavimai naudoja tuos pačius duomenis, kuriuos reikėtų lygiagrečiai apdoroti (pritaikyti skirtingas funkcijas), o ne veiksmas po veiksmo. Matematinis aparatas leidžia apskaičiuoti stulbinančius dalykus ir taikomas nuo teorijos iki praktikos.

Kitų vėlesnių architektūrų pavyzdžiai:

- CISC (angl. *Complex Instruction Set Computer*), skaičiavimai su sudėtingų instrukcijų rinkiniu (100-200 komandų rinkinys);
- RISC (angl. *Reduced Instruction Set Computing*), sumažinto instrukcijų rinkinio skaičiavimai (50-100 komandų rinkinys, pašalintos retai naudojamos instrukcijos);

[3] [4] Autorius: I. Cburnett, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2233537

- VILW (angl. *Very long instruction word*), procesoriai su keliomis skaičiavimo mašinomis. Viena procesoriaus instrukcija turi savyje kelias komandas, kurios turi būti įvykdytos lygiagrečiai, todėl įgijo „ilgos instrukcijos“ vardą.

1.2. Rezultatų įvertinimas

Lyginant CPU ir GPU našumą galimi keli palyginimo variantai. Kadangi CPU ir GPU yra skirtingi įrenginiai, negalima lyginti jų vienas prie vieno. Norit įvertinti skaičiavimų pagreiti, dažniausiai imami panašių kartų CPU (dauguma CPU turi 4 branduolius ir palaiko tam tikrą lygiagretinimą) ir GPU. Yra matuojama, kiek užtrunka skaičiavimai su abejais įrenginiais ir pateikiamas santykis. GPU skaičiavimo laikas gali įtraukti duomenų kopijavimą į atmintį, arba neįtraukti.

Kita metrika, taikoma GPU skaičiavimams įvertinti yra FLOPS arba FLOP/s (angl. *FLoating-point Operations Per Second*). FLOPS parodo, operacijų su slankiojančio kablelio skaičiais yra atliekama per sekundę. FLOPS nepriklauso tarptautinei matavimu sistemai SI (pr. *Le Système International d'Unités*), tačiau yra ganėtinai dažnai sutinkama moksliniuose straipsniuose aprašant kompiuterių našumą. Metrikai galioja tie patys klasikiniai priešdėliai. Mega-, gigaflopsų našumas sutinkamas personaliniuose kompiuteriuose. Tera-, peta- ir t.t, dažniausiai naudojami aprašant superkompiuterius. Paskirstytų skaičiavimų sistema BOINC siekia daugiau kaip 21,4 PFLOPS. 2011 m. *Intel* paskelbė apie eksaflopsinio našumo kompiuterio sukūrimą apie 2020 m. Panašų ekstaflopinio kompiuterio sukūrimo laiką įvardija ir kitos įmonės bei įstaigos.

Trečia metrika yra MLUPS (angl. *Million Lattice site Updates Per Second*), milijonas atnaujintų tinklo mazgų per sekunde. Metrika yra gana specifiška, taikoma Lattice-Boltzmann'o metodui, vadinamu LBM. Priklauso nuo tinklo tipo ir rezoliucijos (duomenų išdėstymo) (Delbosc, Summers, Khan, Kapur, & Noakes, 2014).

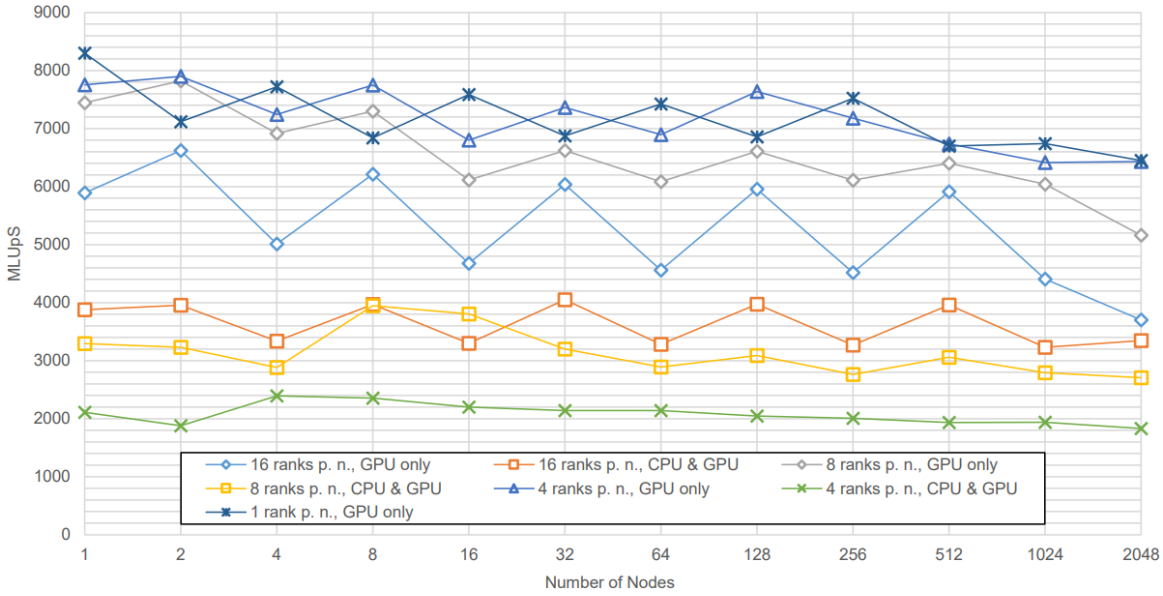
Visi budai įvertinti GPU našumą bus minimi toliau pateikiamuose straipsnių apžvalgose.

1.3. Taikymas

GPGPU taikymo sritis yra labai plati. Dažniausiai naudojami hibridiniai modeliai, kurie apjungia CPU atminties ir GPU lygiagrečių, spartesnių skaičiavimų privalumus. Šis darbas orientuotas į GPU skaičiavimus su CUDA, kadangi dauguma tyrimų bus atliekami su *NVIDIA* grafinėmis plokštėmis.

2018 m. buvo paviešintas vandenyno srautų tyrimas (Kuckuk & Köstler, 2018). Srautams esant šalia paviršiaus, galima taikyti sekliųjų vandenų lygtis (angl. SWE, *Shallow Water Equations*), kurios aprašo elipsines ir hiperbolines dalines išvestines. Kol lygčių sistemos mažos, išvestinės lengvai apskaičiuojamos, bet augant sistemai, atsiranda poreikis optimizuoti skaičiavimus juos lygiagretinant.. Skaičiavimai buvo atlikti *Piz Daint* superkompiuterio (Maltsev & Würsten, 2017)

GPU klasteryje (2018 m. 6ta vieta TOP500 galingiausių nepaskirstytų kompiuterinių sistemų). Sistemą sudaro 5 320 kompiuteriniai mazgai su *Intel Xeon E5-2690 v3* su 12 branduolių (2.60 GHz dažnio) ir *NVIDIA Tesla P100* su 16GB RAM. 2048 GPU apskaičiavo trilijoną nežinomųjų. Užduotis buvo sudalinta į fragmentus, toliau atliekant grynuosius GPU ir hibridinius CPU-GPU skaičiavimus. Hibridinis modelis leidžia dalinti problemą į didesnes dalis, kadangi CPU gali saugoti daugiau duomenų. Tačiau, kaip matyti iš 1.5 pav., grynieji GPU skaičiavimai yra iki 2 kartų greitesni.



1.5 pav. *Pasiektas MLUpS (skaičiavimų mazgui per sekunde) kiekis (Kuckuk & Köstler, 2018).*

(Filonenko, Hernández, & Jo, 2018) darbe CUDA buvo panaudota priešgaisrinės saugos stebėjimo kamerose dūmams atpažinti. Nors dūmų aptikimo metodų yra labai daug, iki šiol ne vienas negalėjo užtikrinti didelio kadrų kiekio apdorojimo. Aprašytas aptikimo metodas remiasi dūmų dalių forma bei spalva. Fiksuota kamera leidžia „pašalinti“ pastovias detales iš kadro ir susitelkti ties kintančiomis. Pagal spalvą nustatoma tikimybė, ar besikeičianti dalis yra dūmai.

Naudojant hibridinį skaičiavimų modelį, sujungiant CPU ir GPGPU (*Intel Core i7-4720HQ* CPU, *NVIDIA GeForce GTX 980M* GPU, 16 GB DDR3 RAM), leido autoriams išsaugoti 320x240 HD vaizdo įrašų apdorojimo laiką mažesniuose, nei 200 ms rėžiuose, kas atitinka stebėjimo sistemų reikalavimus. Kraštams aptikti buvo panaudota *OpenCV 3.0* biblioteka, tačiau ilgiausiai trunkantis spalvų apdorojimo procesas buvo sukurtas tiesiogiai su CUDA C++ programavimo kalba.

1.1 lentelė. *Vidutinės laiko sąnaudos atskiriems dūmų aptikimo vaizdo įraše procesams (Filonenko, Hernández, & Jo, 2018)*

Process	320 × 240 CPU	320 × 240 Hybrid
Background subtraction	10,2086	1,7377
Morphology transformations	0,8785	4,9470
Color probability calculation	15,7245	1,3468
Labeling	1,1860	1,7617
Edge detection	1,9202	2,0147
Color probability filtering	0,9113	0,9113
Boundary roughness	2,7419	2,7419
Edge density calculation	1,0604	1,0604
Filtering by area	0,9373	0,9373
Total	35,5688	17,4587

Kita CUDA taikymo sritis – skirtingo dydžio panašių paveikslų paieška (Feng, Zhang, & Gao, 2015), kuri taikant tik CPU užtrunka o rezultatai lengvai sugadinami esant paveikslų „triukšmui“. Lygiagretus GPGPU (CPU – *Intel (R) Xeon (R), E5620@2AOGHz; GPU – GT750Ti (750Ti), NVIDIA*) padeda pašalinti triukšmą, parinkti tinkamiausius pavyzdinius pikselių gabaliukus atitikmens paieškai, net jeigu triukšmingi yra ir paveikslai, su kuriais yra lyginama.

$$I_0 = DQ_kL + n_k;\ 1 \le k \le K \ (3)$$

Formulė 3 aprašo paveikslo formavimo procesą, kur L yra didelės raiškos originalių paveikslėlių vektorius, o I stebimi mažos raiškos gabaliukai. Q yra iškraipymas – pavyzdžiui, kontūrų ryškumo praradimas. n žymi triukšmą.

Paveikslėlis yra sudalinamas į mažas dalis, kurios yra nepriklausomai padidinamos n kartų. Toliau generuojamas CUDA skaičiavimo bloko vienetas, gijų kiekis apskaičiuojamas pagal (4) ir turi dalintis iš 32 (jeigu nesidalina, pridedama dar viena gija, kuri bus nepilnai užpildyta), taip pasiekiant optimalų CUDA resursų išnaudojimą.

$$N = global_x \times global_y \times block_x \times block_y \ (4)$$

Naudojama bendroji (angl. *shared*) atmintis vietoje globaliosios, nes nuskaitant pikselių kaimynus, patys duomenys yra nuskaitomi daugelį kartų.

1.2 lentelė. *CUDA, C++ ir Matlab skaičiavimo laikų palyginimas (Feng, Zhang, & Gao, 2015)*

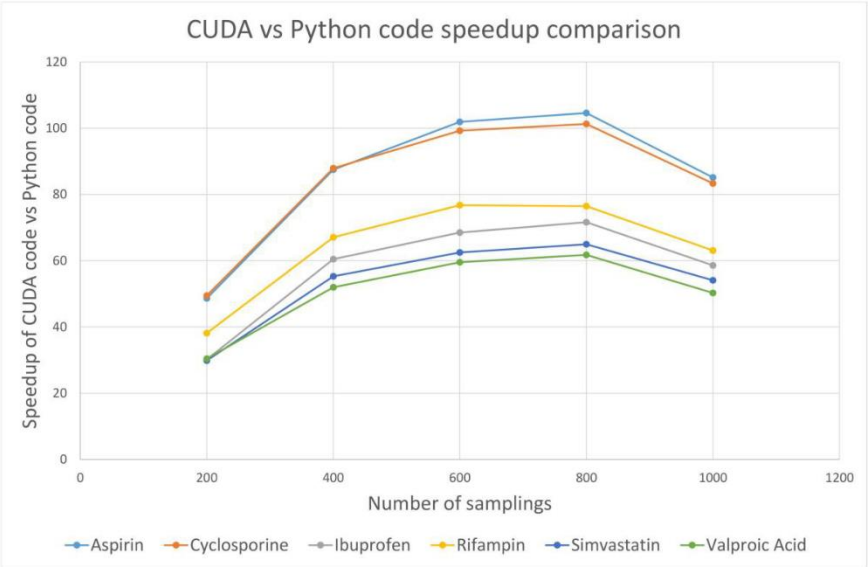
	CUDA	C++	Matlab
Image 1 (1920x1080)	179 ms	53 s	347,45 s
Image 2 (l920xl080)	185 ms	58 s	378,79 s
Image 3 (l920xl080)	164 ms	49 s	320,62 s

CUDA GPGPU skaičiavimai pradėti taikyti medicinos srityje. Kadangi vaistų tarpusavio sąveikas (toliau – DDI, angl. *Drug-Drug Interaction*) yra sunku apskaičiuoti, o vaistų deriniai yra

išrašomi gana dažnai. (Lu, Ramachandra, Pham, Tu, & Cheng, 2019) sukurtas įrankis CuDDI (angl. *Compute Unified Device architecture-based DDI searching*) naudojamas DDI informacijos surinkimui iš *PubMed* duomenų bazes, kurią sudaro daugiau negu 150.000 publikacijų.

Algoritmą sudaro 3 žingsniai: duomenų apie vaistus gavimas iš *PubMed*; DDI skirtų terminų nustatymas; sąryšių tarp DDI terminų atvaizdavimas. Palyginus įprasta CPU Python versiją su CUDA, buvo pastebėtas 30-105 kartų laimėjimas skaičiavimuose, priklausimai nuo imties dydžio ir ieškomų vaistų minėjimo dažnumo. CUDA naudojama atsitiktinių pavydžių išrinkimui, jų statistinei analizei ir ieškomo termino atitikimo dažnio nustatymui tiriamuose publikacijų grupėse.

Kaip intuityviai galima buvo nuspėti, kuo didesnė duomenų imtis vienam vaistui, tuo didesnis bus CUDA laimėjimas palyginus su CPU publikacijų patikrinimo metodu.



1.6 pav. *Skaičiavimo su CUDA pagreitis, palyginus su Python versija (A)*

CUDA naudojama ne tik tyrimams, bet ir kaip pagalbinė vizualizacijos priemonė. Baigtinio Skirtumo Laiko Sričių (angl. *Finite-Difference Time-Domain,* FDTD) metodas yra modeliavimo metodas taikytinas skaičiuojamoje elektromagnetikoje ir pasižymi aukštais reikalavimais kaip skaičiavimų pajėgumui, taip ir atminčiai. Berašant modeliavimo įrankį, (Warrena, et al., 2019) autoriai pažymėjo, kad pasiekė 1194 Mcells/s ir 3405 Mcells/s su *NVIDIA Kepler* ir *Pascal* architektūromis, kas yra 30 kartų greičiau, negu lygiagretūs CPU (Intel Core i7-4790K) skaičiavimai su *OpenMP*.

Taip pat autoriai greta atliko ir *NVIDIA* grafinių kortų palyginimą, pažymint, kad *GeForce* serija, skirta kompiuteriniams žaidimams daro CUDA prieinama daugeliui namuose, o į pramoninius duomenų kiekius orientuota *Tesla* (P100 GPU) serija pasiekia puikius rezultatus didelio atminties deka.

(Fambrini, et al., 2018) Optimizavo JSEG algoritmą CUDA pagalba. Šis algoritmas sudarytas iš 2 nepriklausomu žingsnių: spalvų kvantavimo, t. y. kai iš tęstinio duomenų srauto padaromos

diskrečios įvesties klasės, ir erdvės segmentavimo. Pakeitus paveikslą jo spalvų-klasių žemėlapiu yra surandamos spalvų ir tekstūrų ribos. Kaip vienas iš JSEG taikymo būdų yra dirbtinio intelekto panaudojimas stebint termografinės elektros tinklo nuotraukas. Dažniausiai jas stebi žmogus-operatorius ir, pastebėjus pavojingai įkaitusiais zonas, kviečia remonto brigadą. (Fambrini, et al., 2018) pasiūlė pakeisti žmogų-operatorių Giliojo mokymosi neuroniniu tinklu.

Su CUDA (*GeForce* GTX1080) segmentavimo žingsnis 9x9 tinkleliui sudarė 73 ms, o su CPU tokio pat dydžio tinkleliui – 970 ms. Segmentatoriui buvo parašytas klasifikatorius pasitelkiant CUDA, C kalbą ir YOLO algoritmą.

Artimiausių kaimynų (angl. *nearest neighbor partitioning*, NNP) klasifikavimo algoritmas yra plačiai žinomas ir naudojamas. (Wang, et al., 2018) sėkmingai paspartino algoritmo konstravimo procesą pasitelkiant lygiagretųjį skaičiavimo modelį. Labiausiai imlios skaičiavimams dalys – panašumo matricos sudarymas, tikslo funkcija ir pavyzdžių sužymėjimas. Kadangi kiekviena dalelė modelyje yra nepriklausomas ir savarankiškas neuroninis tinklas, skaičiavimus galima atlikti lygiagrečiai. Taikomas mišrus skaičiavimo modelis: dalelių spiečiaus optimizacija – svoriai ir slenksčiai (angl. *bias*) yra pateikiami CPU pusėje, o klasifikatoriaus įvertinimas – *a* normalizavimas, panašumo skaičiavimai ir tikslo funkcija atliekami GPU (*Tesla* K10). Jeigu CPU buvo pakeisti svoriai, GPU tenka perskaičiuoti viską iš naujo, bet tai daroma lygiagrečiai. CPU priverstas laukti GPU įverčio. Tačiau kiekvienas blokas atsakingas iš karto už kelis modelius. Buvo pasiektas iki 70 kartų geresni rezultatai laiko atžvilgiu palyginus su įprastąją CPU architektūra klasifikuojant cemento mikrostruktūras.

(Bohacek, et al., 2019) pritaikė CUDA atvirkščiai šilumos laidumo problemai spęsti. Per 3 dešimtmečius ištobulintas algoritmas vis tiek reikalavo ilgo skaičiavimo laiko. 3 studentai-doktorantai pasiūlė skirtingus sprendimus naudojant GPU ir palygino juos su standartiniais sprendimo kodais (*OpenFOAM (FDIC)* ir *ANSYS Fluent (AMG)*). Geriausius rezultatus davė CUDA C sprendimas simetrinėms teigiamai apibrėžtoms matricoms, saugomoms DIA formatu. Lyginant su prieš tai sukurtais GPU sprendimais, buvo pasiektas 46 kartų laimėjimas mažam tinklui ir 16 tūkstančių kartų dideliam. Lyginant su klasikiniais metodais, tik *OpenFOAM* gali varžytis su GPU: esant mažam tinklui GPU 15% lėtesnis, negu *OpenFOAM*. Tačiau, esant dideliam tinklui, GPU laimi iki 15 kartų.

Norit efektyviai naudoti GPU, reikia išmanyti GPU architektūrą. (Bell & Garland, 2008) atliko CUDA taikymo sprendžiant išretintas matricas tyrimus. Autoriai sudaugino įvairaus formato išretintas matricas naudojant *GTX 200* serijos plokštę. Tyrimas buvo atliktas su įstrižu formatu (kai ne nulinės reikšmės yra išdėliotos ties įstrižainėmis), ELL formatu (esant M x N matricai, kai nenulinių įrašų kiekis eilutėje neviršija K, duomenys yra saugomi M x K tankiojoje matricoje), koordinačių formatu (saugoma duomens pozicija ir reikšmė), suspaustu išretintų eilučių formatu,

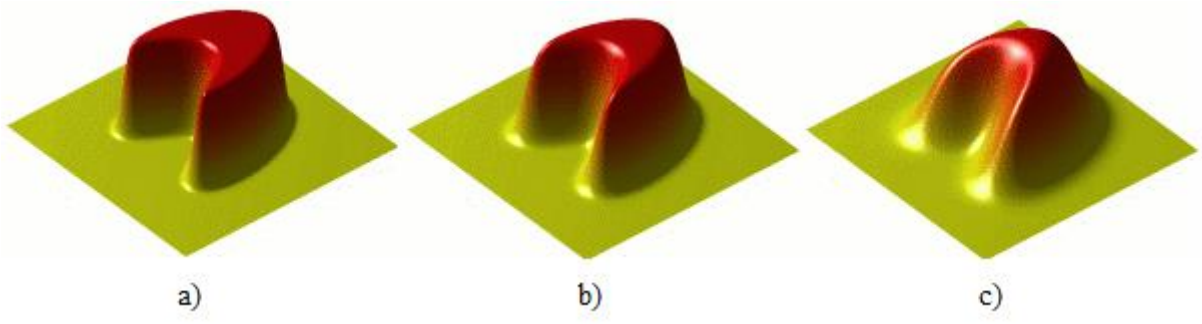
hibridiniu formatu (koordinačių ir EEL derinys), paketo formatu (matrica išskaldoma į paketus). Buvo pasiektas 36 GFLOP/s našumas viengubam tikslumui ir 16 GFLOP/s *double* formatui. Skaičiavimu sparta pagreitėjo 10 kartų lyginant su 4 branduolių *Intel Clovertown*.

2. Šilumos laidumo uždavinys

Kietųjų kūnų šilumos laidumo reiškinys yra stebimas, kai kūnas yra nevienodai įkaitęs (be šilumos spinduliavimo). Kūno temperatūra skirtingai kinta skirtinguose taškuose priklausomai nuo laiko. Uždavinių tipai:

- Vienmačiai: nagrinėjamas įkaitintas strypas, atitinkamai naudojama tik x strypo koordinatė ir laikas, $T = f(x,t)$;
- Dvimačiai: nagrinėjama plokštuma, formulė papildoma y plokštumos koordinate (2.1 pav.), $T = f(x,y,t)$;
- Trimačiai: nagrinėjami erdviniai kūnai, $T = f(x,y,z,t)$.

Čia x, y, z yra kūno taško koordinatės, t – laikas, T – temperatūra.



2.1 pav. *Dvimačio šilumos laidumo skaitinio sprendimo pavyzdys. Čia spalva ir aukštis atvaizduoja temperatūrą tame plokštumos taške: raudona ir aukšta zona įkaitinta, žalia ir žema vesesnė. Matome, kaip figūra „lydosi“ nuo a) iki c) būsenos – vesta kraštai, lengvai įkaista šalia esantys taškai*[5].

Dar šilumos laidumo uždaviniai gali būti:

- Stacionarūs – kai nusistovi nuolatinis temperatūrų skirtumas erdvėje. T. y., kūnas gauna tiek pat šilumos per laiko vienetą, kiek ir praranda. Tokių uždavinių tikslas – rasti temperatūrą kiekviename taške.
- Nestacionarūs – temperatūra nėra pastovi ir reikia nustatyti jos kitimo tendenciją kiekvienam kūno taškui.

2.1. LLS

Iš esmės šilumos laidumo uždavinys susiveda į linijinių lygčių sistemos sprendimą. Priklausomai nuo kvadratinio tinklo dydžio, sistemą sudaro $(N-1)^2$ lygčių. Linijinių lygčių sistemą (LLS) galima užrašyti kaip:

$$Ax = b$$

Kur A tai $n \times n$ matrica, b – atsakymų matrica, o x yra sprendimų vektorius. Linijinių lygčių sistemos sprendimas naudojamas taikant daugelį metodų, pavyzdžiui, baigtinių skirtumų, baigtinių tūrių,

[5] Kadrai iš O. Aleksandrovo MATLAB sukurto modelio: https://en.wikipedia.org/wiki/Heat_equation#/media/File:Heat_eqn.gif

baigtinių elementų metodus ir t.t. Nors CPU sprendimai yra optimizuoti beveik iki tobulumo, jie vis tiek yra apribuoti sistemos dydžio ir atminties ribojimų. Tačiau LLS sprendimas gali būti puikiai lygiagretinamas. GPU įgalina tokių sprendimų pagreitį.

2.2. LLS sprendimo metodai

Šiame skyriuje pateikiama kelių LLS sprendimo metodų apžvalga, kaip ir hibridiniai šitų modelių sprendimai. (Corso, Gullí, & Romani, 2007) nagrinėjo stacionarių ir nestacionarių metodų naudingumą interneto matricoms, naudojamoms paieškos varikliuose. Stacionarūs metodai parodo geresnius rezultatus esant griežtai apibrėžtai sąlygai.

2.1 lentelė. *Sąrašas stacionarių, Krylovo poerdvio ir priešsąlyginių metodų (Corso, Gullí, & Romani, 2007)*

Stationary Methods	Krylov Subspace Methods	Preconditioned Methods
POWER	GMRES(n)	GMRESP(n)
JACOBI	BiCG	BiCGP
DIRECT GS	BiCGStab	BiCGStabP
REVER SEGS	CGS	CGSP
	CGNR	CGNRP
	QMR	QMRP

Krylovo metodas

(Anzt, et al., 2017) išnagrinėjo Krylovo metodų paspartinimą taikant GPU skaičiavimus naudojant skirtingą, jau sukurtą, Krylovo sprendimo įrangą: *BiCGSTAB*, *CGS*, *QMR*, ir *IDR(s)*. Tačiau esant dideliam duomenų kiekiui autoriai pasinaudojo Jakobio algoritmu taip pagerinant „laiko-sprendimo“ santykį.

Krylovo-Schwartz‘o metodas

(Abhyankar, Constantinescu, Smith, Flueck, & Maldonado, 2016) pasiūlė Krylovo-Schwartz‘o metodo realizaciją dinaminėms jėgos simuliacijoms. Šis metodas apjungia Krylovo erdvės linijinių lygčių sprendėją GMRES (angl. *Generalized Minimal Residual Method*) ir persidengiančią, apribotą, prisitaikančią Schwartz‘o sąlygą. Esminis privalumas – metodas yra pritaikytas lygiagretiems skaičiavimams taip juos paspartinant.

Gausso-Sidelio metodas

(Saha, 2017) pasiūlė bendrą nekvadratinių LLS sprendimą Gausso-Seidelio ir Jakobio metodo principu. Naudojant iteracinį metodą laikas, per kurį gaunamas aproksimuojamas sprendimas, sutrumpėjo lyginant su tos pačios LLS sprendimų SOR metodu. (Ramakrishna Tipireddy, 2019) pasitelkė Gausso-Seidelio metodą kartu su Jakobio metodu stochastinei Galerkino dalinių išvestinių

lygčių diskretizacijai. Tie patys metodai buvo panaudoti kaip sąlygos Krylovo metodams ir įrodė, kad šie metodai galimi naudoti kaip sąlyga GMRES sprendėjui.

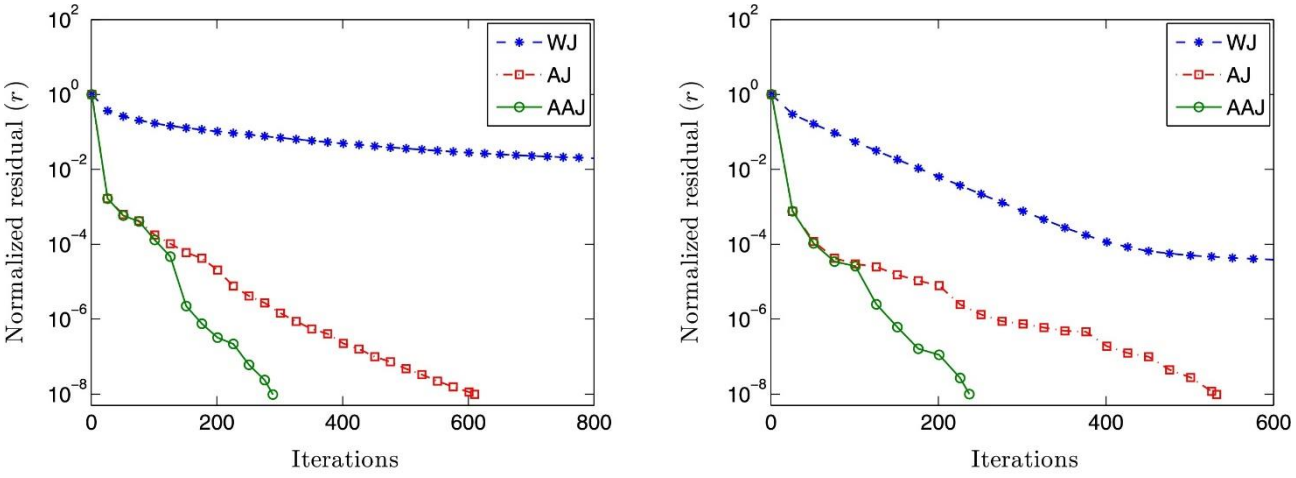
Jakobio metodas

Jakobio metodas konverguoja lėčiau (Amador & Gomes, 2012), negu kiti, pavyzdžiui, Gausso-Siedelio arba Krylovo metodas, jis iš prigimties yra lygiagretus (Margaris, Souravlas, & Roumeliotis, 2014) ir todėl taikytinas GPU skaičiavimams. Kai kuriais atvejais, pavyzdžiui, esant išretintai matricai, Jakobio metodas yra net greitesnis už Krylovo metodą (Cormie-Bowins, 2012).

Kadangi Jakobio metodas dažnai naudojamas kaip pagalbinis apjungiant jį su kitais LLS sprendimo metodais ir turi didelį potencialą lygiagretinimui, būtent jis buvo pasirinktas realizacijai ir eksperimentų atlikimui. Toliau pateikiamas detalus Jakobio metodo aprašymas.

Jakobio iteracinis algoritmas pavadintas vokiečio matematiko ir mechaniko Karlo Gustavo Jokūbo Jakobio (1804 – 1851 m.) vardu, pasiūliusio šį algoritmą (Jacobi, 1846). Tačiau dėmesio algoritmas sulaukė tik po šimtmečio atsiradus kompiuteriams.

Taikomojoje matematikoje iteraciniu metodu vadinama procedūra, kuriai paduodamas pradinis atsitiktinis sprendimas (spėjimas). Šis spėjimas naudojamas aproksimuotų sprendimų sekos kūrimui, kur n-tasis sprendimas yra išskaičiuojamas pagal prieš jį buvusius. Papildžius šį metodą terminavimo sąlyga yra gaunamas iteracinis algoritmas.



(a) P1a1 (b) P2a1

2.2 pav. *Konvergencijos grafikai Weighted Jacobio (WJ), Andersono–Jacobio (AJ) ir Alternuojančio Andersono–Jacobio (AAJ) metodams (P.Pratapa, Suryanarayana, & E.Pask, 2016)*

Iteracinis metodas konverguoja, jeigu atitinkama seka konverguoja su duotais pradiniais duomenimis. Tiesioginiai metodai stengiasi išspręsti problemą per baigtinį operacijų skaičių ir, jeigu nėra apvalinimo klaidų, pateikia tikslų atsakymą. Iteraciniai metodai dažniausiai taikomi esant nelinijinėms lygtims. Tačiau, jeigu linijinę problemą sudaro daug kintamųjų ir tiesioginiai

skaičiavimai paprasčiausiai yra per brangus laiko atžvilgiu ar net neįmanomi (Amritkar, de Sturler, Świrydowicz, Tafti, & Ahuja, 2015), praverčia ir iteraciniai metodai.

(P.Pratapa, Suryanarayana, & E.Pask, 2016) Atliko kelių iteracinių metodų palyginimą lygiagretiems didelių, išretintų sistemų sprendimams. (2.2 pav.) galima matyti, kaip konverguoja Weighted Jacobio (WJ), Andersono–Jacobio (AJ) ir Alternuojantis Andersono–Jacobio (AAJ) metodai. LLS buvo gauta diskretizuojant Laplaso lygtį su nulinėmis Dirichlet ir Neimano kraštinėmis sąlygomis. Autoriai nurodo, kad esant dideliam duomenų kiekiui, būtini iteraciniai algoritmai (Richardsono, Gausso–Seidelio, SOR). Šitie metodai blogai prisitaiko prie sistemos dydžio, todėl Krylovo poerdvio (angl. *subspace*) metodai, pavyzdžiui, GMRES, linksniuojamo gradiento (angl. *conjugate gradient*), išlieka labai populiarūs. Jakobio iteracinis algoritmas išsiskiria paprastumu ir didelio lygiagretinimo potencialu, todėl ieškomi būdai paspartinti jo konvergenciją išlaikant metodo paprastumą.

3. Skaitiniai eksperimentai

Šio darbo praktinę dalį sudaro:

1. Jakobio iteracinis algoritmas;
2. Algoritmą realizuojanti programinė įranga;
3. Skaitiniai eksperimentai.

Eksperimentai atlikti su šiais GPU modeliais:

- *AMD* RX VEGA 56;
- *NVIDIA* GTX1060 (pagrindinis GPU įrenginys, jeigu nenurodyta kitaip, būtent jis buvo naudojamas aprašytiems skaičiavimams);
- *NVIDIA* GTX860M;
- *NVIDIA Tesla* M40;

Eksperimentai atlikti su šiais CPU modeliais:

- *Intel* i7-3630QM;
- *Intel* i7-4720HQ;
- *Intel* i7-6700K;
- *Intel* i7-7700 (pagrindinis CPU įrenginys, jeigu nenurodyta kitaip, būtent jis buvo naudojamas aprašytiems skaičiavimams);
- *Intel* i7-7820HQ.

GPU ir CPU specifikacijos pateikiama 3.5 skyriuje.

3.1. Jakobio iteracinis algoritmas

Šiame darbe nagrinėjamas stacionarus dvimatis šilumos laidumo uždavinys:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y), \qquad (x,y) \in (0,1) \times (0,1), \quad (5)$$

$$u(x,y) = \mu(x,y), \qquad (x,y) \in \gamma, \quad (6)$$

čia *u(x, y)* yra temperatūra srities taške *(x, y)*, γ žymi srities kontūrą, *μ(x, y)* pateikta temperatūra srities kontūre, o *f(x, y)* apibrėžia šilumos šaltinį.

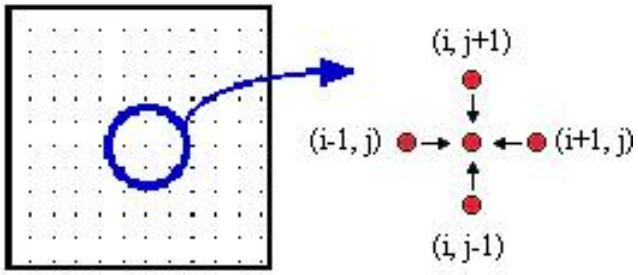
Ši lygtis yra vadinama Puasono (*Poisson*) vardu ir yra plačiai taikoma fizikoje. Jos pagalba aprašomi įvairiausi potencialiniai laukai, pvz., elektros, slėgio ir t.t.

Uždavinys sprendžiamas skaitiškai – baigtinių skirtumų metodu. Norint naudoti baigtinių skirtumų aproksimaciją, reikia apibrėžti diskretųjį tinklą, kuris tinka sprendžiamam uždaviniui:

$w_h = \{(x_i, y_j):\ x_i = ih, y_j = jh, 0 \le i,j \le N\}$, kur $h = \frac{1}{N}$ - tinklo žingsnis.

Ieškomas diskretusis sprendinys $U_{ij} = U(x_i, y_j)$, apibrėžtas tinklo taškuose. Kraštiniuose tinklo taškuose reikšmes galima rasti iš kraštinių sąlygų (6) srities kontūre. Kiekvieno vidinio taško

reikšmei gauti, (5) diferencialinė lygtis tame taške yra pakeičiama į algebrinę lygtį aproksimuojant išvestines baigtiniais skirtumais. Naudojamas penkių gretimų taškų šablonas (3.1 pav.)



3.1 pav. *Diskretusis tinklas ir schemos šablonas.*

Taip gaunama tiesinė lygčių sistema:

$\frac{U_{i-1,j}-2U_{i,j}+U_{i+1,j}}{h^2}+\frac{U_{i,j-1}-2U_{i,j}+U_{i,j+1}}{h^2}=f_{i,j}$, kai $1 \le i,j \le N-1$.

Sistemą sudaro $(N-1)^2$ lygčių.

3.2. Programinė įranga

Jakobio iteracinis algoritmas buvo realizuotas Python programavimo kalba.

Naudotos bibliotekos:

- *Numpy*;
- *TensorFlow-GPU*.

Kodas prieinamas *GitHub* repozitorijoje:

https://github.com/kolesov93/tf_jacobi/blob/master/main.py

TensorFlow-GPU biblioteka

TensorFlow yra atviro kodo biblioteka skirta skaitiniams skaičiavimams su duomenų grafais. Pagrinde, naudojama mašininiam apmokymui (toliau – ML, angl. *machine learning*). Leidžia lengvai kurti ML modelius, pateikiama kartu su *Keras API*, suteikia daug abstrakcijų lygių, lieka tik pasirinkti tinkamiausia. Kaip teikia kūrėjai (Why TensorFlow, 2019), apmokyti modelius ir juos įdiegti bus paprasta, nesvarbu, ar serveryje, ar internete, ar įrenginyje.

Gali būti naudojama *Python* programavimo kalba arba *TensorFlow.js*. Yra išplėstiniai variantai (*TensorFlow Extended, TFX*) ir lengvesni su žemesniais reikalavimais (*TensorFlow Lite*), skirti *Android, iOS, Raspberry PI* ir kitiems įrenginiams.

Šiame darbe yra naudojamas *TensorFlow* bibliotekos plėtinys, *TensorFlow-GPU*. Kaip ir jo pirmtakas, šis plėtinys buvo sukurtas norint naudoti GPU skaičiavimo pajėgumus mašininiam apmokymui, kuris ilgai užtrunka, bet gali būti vykdomas lygiagrečiai. Tačiau tai yra biblioteka, kuriai reikės prieigos prie GPU, kurie yra skirtingų modelių ir skirtingų kūrėjų (pavyzdžiui, *NVIDIA, AMD* ir t.t.), todėl įdiegimui reikės pasiruošti.

Norint įdiegti *TensorFlow-GPU* su *NVIDIA GPU* reikės:

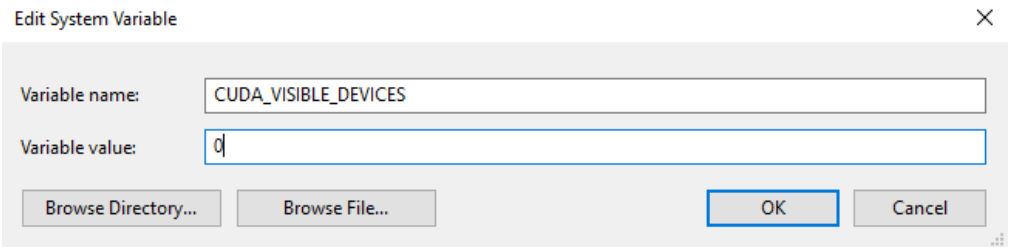
1. Patikrinti, ar GPU CC (angl. *Compute Compability*) yra 3.0 ar didesnis. Pavyzdžiui, dauguma eksperimentų šiame darbe atliekami su GTX 1060 plokšte, kurios CC yra 6.1 (3.7 lentelė).
2. Įdiegti CUDA. Naujausia versija yra 10.1, šiame darbe naudojama 10.0 versija.
3. Įdiegti cuDNN *(NVIDIA CUDA Deep Neural Network*) (NVIDIA cuDNN, 2019) biblioteką. Šiame darbe naudojama naujausia esama 7.5 versija.
4. Papildomai Windows sistemoje gali tekti išeksportuoti CUDA_VISIBLE_DEVICES kintamąjį nurodant norimus naudoti GPU įrenginius. Jie numeruojami nuo 0 (3.2 pav.).

Po šitų žingsnių galima suinstaliuoti pačią *TensorFlow-GPU* biblioteką per *pip* komandą (angl. *Python Package Index*). *Numpy* biblioteka papildomų reikalavimų neturi.

```
pip install tensorflow-gpu

pip install numpy
```

Šiame darbe naudojama *TensorFlow-GPU* 1.13.1 versija.



3.2 pav. *Nurodomas Windows sistemos kintamasis CUDA_VISIBLE_DEVICES=0. Bus naudojama tik viena grafine plokštė. Jeigu nurodyti kintamojo reikšmę kaip -1, GPU įrenginiai sistemoje matomi nebus. Esant keliems įrenginiams, jie nurodomi per kablelį: 0, 1, 2.*

Kaip matyti iš cuDNN pavadinimo, biblioteka buvo sukurta gyliams neuroniniams tinklams (angl. *deep neural networks*) mokyti su GPU. Biblioteką sudaro aktyvacijos sluoksnių, normalizavimo, susukamų (angl. *convolutional*) neuroninių tinklų ir kitų realizacijos. cuDNN palaiko daugelį giliojo apmokymo (angl. *deep learning*) bibliotekų, pavyzdžiui, *Caffe, Chainer, Keras, MATLAB, PyTorch*. Tarp jų ir šiam darbui reikalinga *TensorFlow*.

Kodo su *TensorFlow-CPU* privalumas tame, kad jis veiks ne tik su CUDA. Esant ne *NVIDIA*, bet *AMD* plokštei, reikės kitos šios bibliotekos versijos, *TensorFlow-ROCm*. Iš pradžių įdiegiamas *ROCm* (angl. *Radeon Open Compute*), tada *TensorFlow-ROCm* biblioteka (TensorFlow, 2019).

```
pip install tensorflow-rocm
```

3.3. Uždavinys be šilumos šaltinio

Pagal realizuoto Jakobio algoritmo aprašymą (skyrius 3.1), buvo sukurta funkcija, priimanti 5 parametrus:

1. kvadratinio tinklo šono ilgis N (tinklo dydis N^2),
2. kraštinę sąlygą $\mu(x, y)$,
3. šilumos šaltinį $f(x, y)$,
4. aproksimacijos paklaidą ε (sustojimo sąlygą),
5. skaičiavimo įrenginį (CPU arba GPU).

Eksperimentai bus atliekami su CPU ir GPU.

```
solve(1000, constant_one, constant_zero, 2e-5, device='/gpu:0')
```

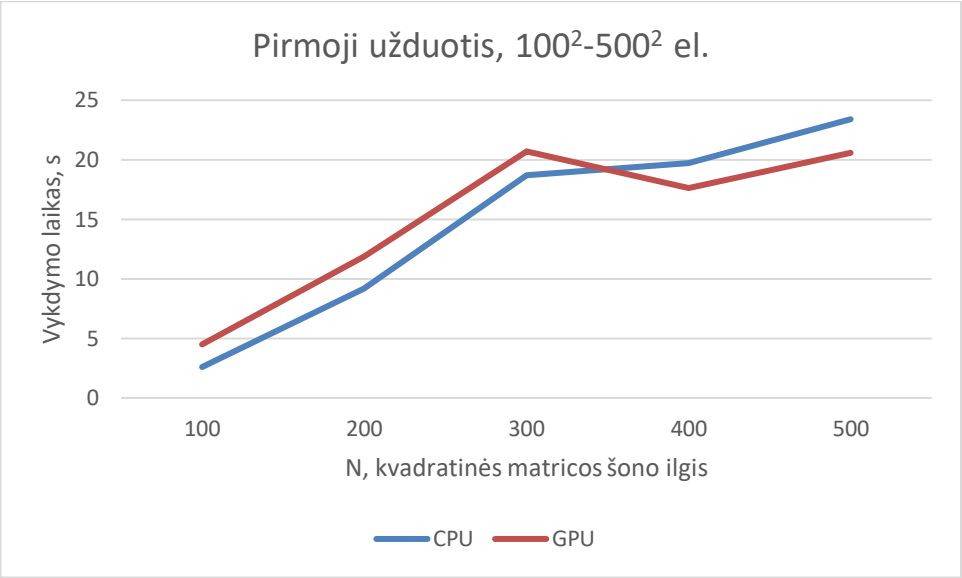
Pirmosios užduoties parametrai:

1. kvadratinio tinklo šono ilgis N kinta nuo 100 iki 1.000, žingsnis 100, vėliau nuo 1.000 iki 10.000 su žingsniu 1.000,
2. su vienetine kraštine sąlyga: $\mu(x, y) \equiv 1$,
3. be šilumos šaltinio: $f(x, y) \equiv 0$,
4. aproksimacijos paklaida $\varepsilon = 2e\text{-}5$,
5. skaičiavimo įrenginys (CPU arba GPU).

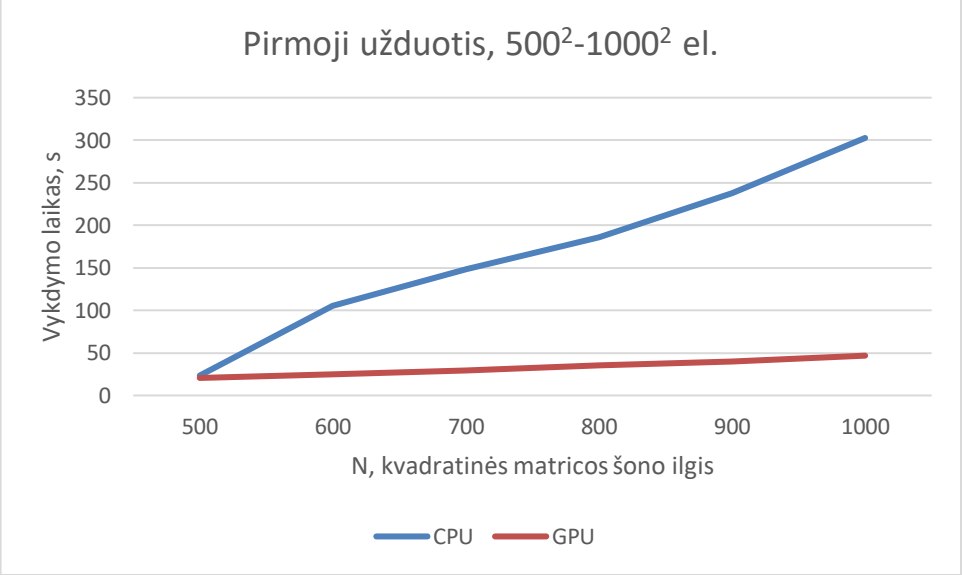
3.1 lentelė. *Pirmoji užduotis. GTX1060 grafinė plokštė ir Intel i7-7700 CPU*

N	CPU, s	GPU, s
100	2,6	4,5
200	9,2	11,9
300	18,7	20,7
400	19,7	17,6
500	23,4	20,6
600	105,7	24,9
700	147,9	29
800	186,2	35,3
900	237,7	40,2
1000	302,64	46,82

Lyginant GPU ir CPU matyti, kaip greitai GPU pradeda lenkti 4 CPU branduolius. Iki N = 400 (3.3 pav.), kur N – kvadratinės matricos šono ilgis, CPU veikia greičiau, jam nereikia įkrovinėti duomenų į atminti, skaičiavimų yra nedaug, gaunamas iki 2 kartų CPU laimėjimas, tačiau tai sudaro vos kelias sekundes (3.1 lentelė). Didėjant N, tai yra, didinant elementų kiekį, GPU pasiekia iki 6,4 karto laimėjimą. 3.4 pav. matyti, kaip greitai auga CPU vykdymo laikas nuo 500^2 elementų.



3.3 pav. *GPU ir CPU pirmosios užduoties sprendimo laikas priklausomai nuo LLS dydžio esant nedideliam, iki 500^2 elementų kiekiui*



3.4 pav. *GPU ir CPU pirmosios užduoties sprendimo laikas priklausomai nuo LLS dydžio nuo 500^2 iki 1.000^2 elementų*

3.2 lentelė ir 0 parodo CPU ir GPU pirmosios užduoties sprendimo laiką, kai N = 1000. Detalios naudotų CPU ir GPU specifikacijos pakeikiamos 3.5 skyriuje. Tačiau visi CPU yra 4 branduolių išskyrus vieną, kuriame skaičiavimais buvo aprauta 12.

Lentelėse matyti, kad net lėčiausias GPU, GTX 860M, užduotį išsprendžia 1,3 karto greičiau už sparčiausią CPU, 6700K. Imant visų CPU ir GPU rezultatų vidurkius, GPU išsprendžia 8,3 kartų greičiau už CPU. Greičiausias GPU lenkia lėčiausią CPU 19,6 karto.

3.2 lentelė. *CPU laiko sąnaudos pirmajam uždaviniui su N = 1000*

CPU	6700K	3630Q	12 branduolių CPU[6]	7700	7820HQ	4720HQ
Laikas, s	113,04	237,56	250	302,64	378,92	823,1

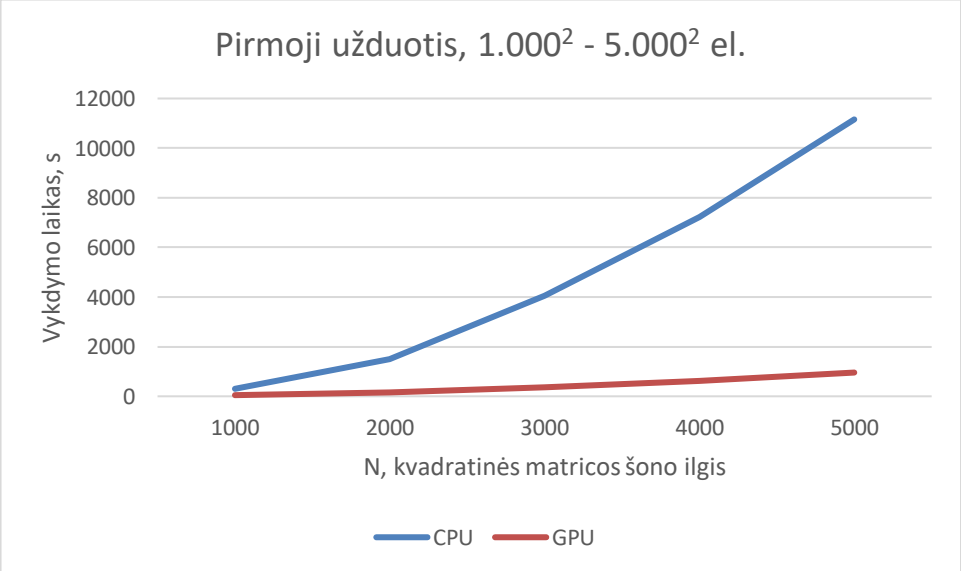
3.3 lentelė. *GPU laiko sąnaudos pirmajam uždaviniui su N = 1000*

GPU	Tesla M40	GTX 1060	RX VEGA 56	GTX 860M
Laikas, s	42	46,82	79,12	85

Didinant elementų kiekį dar daugiau, kartu didėja ir GPU pagreitis (3.5 pav.). Turint 5.000^2 elementų matricoje, GPU išsprendžia pirmąjį uždavinį 11 kartų greičiau, kas, suapvalinus, yra 3 val. CPU sprendimo ir 16 min. GPU (3.4 lentelė).

3.4 lentelė. *Pirmoji užduotis. GTX1060 grafinė plokštė ir Intel i7-7700 CPU didelės N reikšmėms (nuo N = 1000)*

N	CPU, s	GPU, s
1000	302,64	46,82
2000	1507,1	166,5
3000	4039,6	356,3
4000	7215,7	617,8
5000	11157,3	957



3.5 pav. *GPU ir CPU pirmosios užduoties sprendimo laikas priklausomai nuo LLS dydžio nuo 1.000^2 iki 5.000^2 elementų*

[6] Eksperimentas atliktas su 56 branduolių mašina, tačiau apkrauti pavyko tik 12. Detalios mašinos specifikacijos nėra.

Matricos dydis apribotas kelių dalykų. Visų pirma, tai GPU atmintis. Pagrindinėje šių skaičiavimų plokštėje yra 6 GB, kas reiškia, kad maksimalus matricos dydis būtų ~38.500^2 naudojant 32 bitų *float* duomenų tipą. Tačiau kitas parametras – *TensorFlow-GPU* ribojimai.

```
Cannot create a tensor proto whose content is larger than 2GB.
```

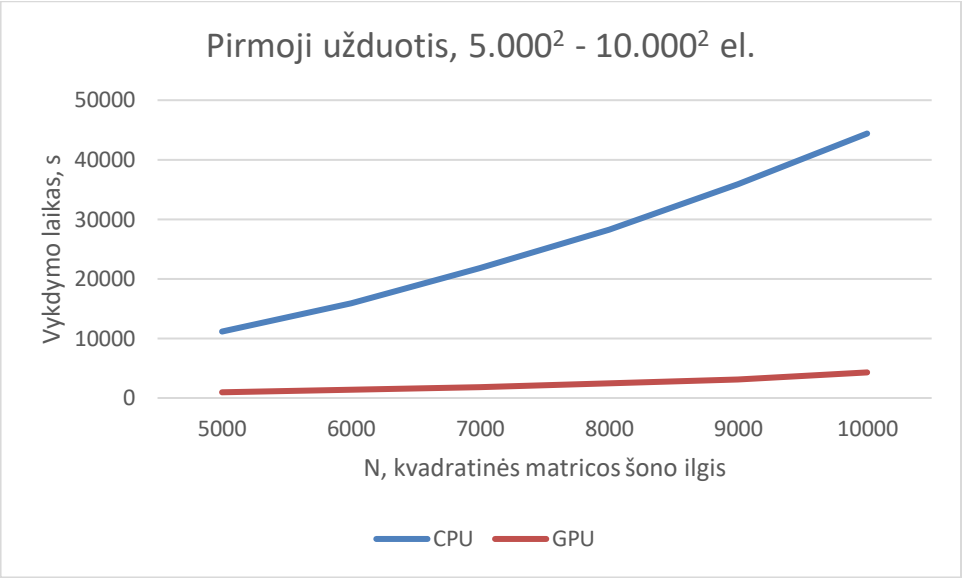
Taip, kaip kodas parašytas dabar, matrica įkraunama vienu metu ir privalo tilpti į 2 GB, kas yra apie 15.800^2 *float* elementų. Tačiau dėl papildomų veiksmų ir naudojamos atminties, maksimali matrica yra apie 10.000^2 elementų (skaičius suapvalintas).

Perrašius kodą, kad *TensorFlow proto* neturėtų įsiminti visų reikšmių (jos vis tiek perrašomos po inicializavimo), galima bus spręsti uždavinį ir su didesnėmis matricomis.

3.5 lentelė. *Pirmoji užduotis. GTX1060 grafinė plokštė ir Intel i7-7700 CPU didelės N reikšmėms (nuo N = 6000)*

N	CPU, s	GPU, s
6000	15920,6	1384,1
7000	21837,9	1867,9
8000	28295,8	2445
9000	35861,9	3091,2
10000	44406,1	4297,4

Pirmajam uždaviniui didžiausias laiko laimėjimas (19 kartų) pasiektas esant $N = 1000$, lyginant CPU *4720HQ* ir GPU *Tesla M40* (skaičiavimai didesnėms *N* reikšmėms su šitais įrenginiais atlikti nebuvo). *7700* ir *GTX 1060* porai didžiausias laimėjimas pasiektas 11,5-11,7 karto, kai $N = \{4000; 10.000\}$. 10.000^2 elementų matricą CPU išsprendė per 12 val., GPU – per valandą. Atlikus skaičiavimus didesniam *N* pavyzdžiui, su *Tesla M40,* galima laukti dar didesnio laiko laimėjimo.



3.6 pav. *GPU ir CPU pirmosios užduoties sprendimo laikas priklausomai nuo LLS dydžio nuo 5.000² iki 10.000² elementų*

3.4. Uždavinys su šilumos šaltiniu

Antrosios užduoties parametrai:

1. kvadratinio tinklo šono ilgis *N* kinta nuo 100 iki 1000, žingsnis 100,
2. su nuline kraštine sąlyga: $\mu(x, y) \equiv 0$,
3. šilumos šaltinis: $f(x,y) = -exp(-10((x-0{,}5)^2 + (y-0{,}5)^2))$. Tokia šilumos šaltinio funkcija pateikia labai mažus žingsnius po matricą ir, atitinkamai, žingsnių ir veiksmų reikės atlikti žymiai daugiau.
4. aproksimacijos paklaida ε = *1e-7*, dėl mažesnės paklaidos reikės atlikti daugiau veiksmų,
5. skaičiavimo įrenginys (CPU arba GPU).

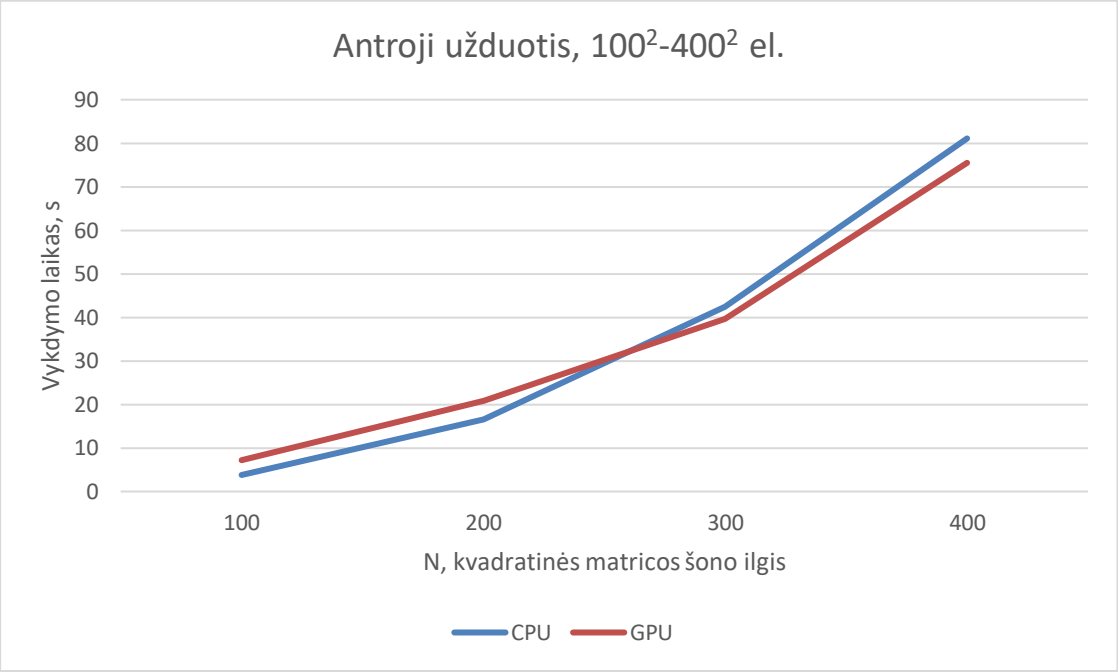
Kaip ir pirmojoje užduotyje, GPU greitai pradeda lenkti 4 CPU branduolius. Tačiau čia CPU veikia greičiau tik iki *N* = 300 (3.6 lentelė) (3.7 pav.). Laimėjimas sudaro vos kelias sekundes. Didinant elementų kiekį GPU vėl pasiekia iki 6,3 karto laimėjimą. 3.8 pav. matyti, kaip greitai auga CPU vykdymo laikas.

Kaip ir buvo minėta, šilumos šaltinio funkcija įgalina labai mažus žingsnius po matrica, veiksmų ir padidėja iteracijų skaičius. Ties N = 100, CPU skaičiuoja 1,5 karto lėčiau lyginant su pirmosios užduoties CPU sprendimu, GPU sprendimas pailgėjo 1,6 karto lyginant su pirmosios užduoties GPU sprendimu. Taip pat, dėl veiksmų kiekio GPU greičiau pradeda lenkti CPU. Tačiau drastiško pagreičio lyginant su pirmąją užduotimi nėra. Laiko santykis tarp CPU ir GPU lieka panašus esant tai pačiai *N*.

Ir pirmajam, ir antrajam uždaviniui pagreitis yra 6,3-6,4 karto, kai *N* = 1000 *7700-GTX1060* įrenginių porai). Tačiau bendras vykdymo laikas lyginant su pirmuoju variantu pailgėjo beveik iki 10 kartų.

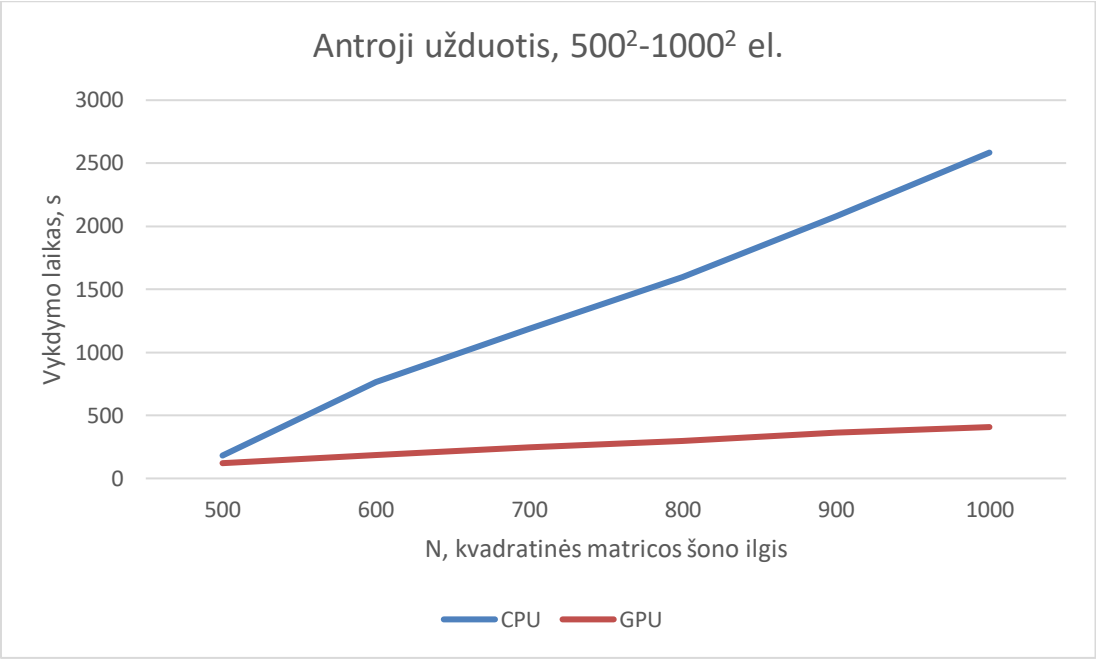
3.6 lentelė. *Antra užduotis. GTX1060 grafinė plokštė ir Intel i7-7700 CPU*

N	CPU, s	GPU, s
100	3,8	7,2
200	16,5	20,9
300	42,5	39,6
400	81,1	75,5
500	181,6	121
600	765,7	187,5
700	1185,3	248,5
800	1598,2	297,7
900	2077,8	362,9
1000	2584,5	407,4



3.7 pav. *GPU ir CPU antrosios užduoties sprendimo laikas priklausomai nuo LLS dydžio esant nedideliam, iki 400² elementų kiekiui*

3.8 pav. atrodo panašiai, kaip ir tų pačių *N* reikšmių pirmosios užduoties 3.4 pav., tik su žymiai ilgesniu vykdymo laiku.



3.8 pav. *GPU ir CPU antrosios užduoties sprendimo laikas pradedant 500^2 elementų matrica*

3.5. Įrangos specifikacijos

Šiame skyriuje pateikiamos darbe naudotų 4 GPU ir 7 CPU specifikacijos.

GPU

GTX 1060 yra pagrindė šiame darbe skaičiavimams naudota GPU. Kaip ir visos *NVIDIA GTX* serijos plokštės, ji yra skirta žaidimams tačiau, ją galima taikyti ir CUDA skaičiavimams. *GTX* serija patvirtina, kad lygiagretūs skaičiavimai gali būti atliekami „naminėmis“ sąlygomis.

3.7 lentelė. *GTX 1060 grafinės plokštės charakteristikos (NVIDIA) (GeForce GTX 1060, 2019)*

CC	6.1
Mikroarchitektūra	Pascal
Grafinis dažnis	1506
Procesoriaus dažnis	1708
Atminties sąsaja	32
Atminties konfigūracija	GDDR5
CUDA branduoliai	1280
Atminties sąsajos plotis	1024
Atminties pralaidumas	192 GB/s

GTX 860M irgi yra skirta žaidimams, tačiau tai yra nešiojamiems kompiuteriams skirta *NVIDIA* grafinė plokštė.

3.8 lentelė. *GTX 860M grafinės plokštės charakteristikos (NVIDIA) (GeForce GTX 860M, 2019)*

CC	5.0
CUDA branduoliai	1152 arba 640
Mikroarchitektūra	Maxwell
Atminties dažnis	Iki 2500 MHz
Atminties konfigūracija	GDDR5
Atminties sąsajos plotis	128-bit
Atminties pralaidumas	80.0 GB/s

Tesla M40 yra *NVIDIA* pagaminta grafinė plokštė, skirta duomenų centrams ir taikoma ML. Gamintojai teigia, kad išleidimo į rinką metu, tai buvo viena greičiausių giliajam mokymui skirtų plokščių (TESLA M40, 2019).

3.9 lentelė. *Tesla M40 grafinės plokštės charakteristikos (NVIDIA) (TESLA M40, 2019)*

CC	5.2
Mikroarchitektūra	Maxwell
Atmintis (RAM)	12 GB
SM procesoriai	3072
Atminties dažnis	6 GHz
Atminties konfigūracija	GDDR5
Didžiausias *float* operacijų kiekis per sekundę	7 TFLOPs
Didžiausias *double* operacijų kiekis per sekundę	0,21 TFLOPs

RX Vega 56 yra vienintelė šiame darbe ne *NVIDIA*, o *AMD* pagaminta plokštė. Kodas tinkamas ir jai naudojant *TensorFlow-ROCm*, o ne *TensorFlow-GPU* biblioteką.

3.10 lentelė. *Radeon RX Vega 56 grafinės plokštės specifikacija (AMD) (Radeon™ RX Vega⁵⁶ Graphics, 2019)*

Maksimalus atminties dydis	1,6 Gbps
Atminties greitis	8 GB
Atminties tipas	HBM2
Atminties sąsaja	2048-bit
Skaičiuojamieji vienetai (angl. *compute units*)	56
Bazinis dažnis	Iki 1156 MHz
Didžiausias pusės tikslumo (angl. *half precision*) operacijų kiekis per sekundę	21 TFLOPs
Didžiausias vienetinio tikslumo (angl. *single precision*) operacijų kiekis per sekundę	10,5 TFLOPs

CPU

3.11 lentelė. *i7-7700 procesoriaus charakteristikos (Intel® Core™ i7-7700 Processor, 2019)*

Litografija	14 nm
Branduolių kiekis	4
Gijų kiekis	8
Dažnis	3,6 GHz
Maksimalus Turbo Dažnis	4,2 GHz
Maksimalus atminties dydis (priklausomai nuo tipo)	64 GB

3.12 lentelė. *i7-7820HQ procesoriaus charakteristikos (Intel® Core™ i7-7820HQ Processor, 2019)*

Litografija	14 nm
Branduolių kiekis	4
Gijų kiekis	8
Dažnis	2,9 GHz
Maksimalus Turbo Dažnis	3,9 GHz
Maksimalus atminties dydis (priklausomai nuo tipo)	64 GB

3.13 lentelė. *i7-6700K procesoriaus charakteristikos (Intel® Core™ i7-6700K Processor, 2019)*

Litografija	14 nm
Branduolių kiekis	4
Gijų kiekis	8
Dažnis	4 GHz
Maksimalus Turbo Dažnis	4,2 GHz
Maksimalus atminties dydis (priklausomai nuo tipo)	64 GB

3.14 lentelė. *i7-3630QM procesoriaus charakteristikos (Intel® Core™ i7-3630QM Processor, 2019)*

Litografija	14 nm
Branduolių kiekis	4
Gijų kiekis	8
Dažnis	2,4 GHz
Maksimalus Turbo Dažnis	3,4 GHz
Maksimalus atminties dydis (priklausomai nuo tipo)	32 GB

Rezultatai

1. Atlikta literatūros analizė. Pateikti GPU skaičiavimų pavyzdžiai, naudoti metodai ir taikymo sritys. Dėmesys skirtas CUDA technologijos panaudojimui.
2. Išnagrinėtas šilumos laidumo uždavinys, jo sprendimui parinktas ir įgyvendintas Jakobio iteracinis algoritmas.
3. Eksperimentas atlikti sukurtas Jakobio iteracinis algoritmas Python programavimo kalba naudojant *TensorFlow-GPU* biblioteką.
4. Atlikti skaitiniai eksperimentai:
    a. Su ir be šilumos šaltinio;
    b. Su nuline ir vienetine kraštine sąlyga;
    c. Keičiant kvadratinės matricos šono dydį nuo 100 iki 1000 elementų su žingsnių 100;
    d. Keičiant ε reikšmę.
5. Palyginti CPU ir GPU rezultatai, skirtingų grafinių plokščių rezultatai.

Išvados

1. Atlikus mokslinės literatūros analizę paaiškėjo, kad GPU skaičiavimai plačiai taikomi neuroniniams tinklams mokyti, vaizdo įrašams su dideliu kadrų kiekiu apdoroti realiu laiku, objektų paieškai atlikti, dideliems duomenų kiekiams apdoroti, simuliacijoms kurti ir t.t. Dauguma iteracinių metodų tinka vykdyti su GPU. Dažniausiai taikomas hibridinis CPU ir GPU skaičiavimo modelis.
2. Šilumos laidumo uždavinys gali būti skirtas strypui, plokštumai arba erdviniam kūnui. Gali būti stacionarus arba ne stacionarus. Sprendžiant uždavinį galiausiai sudaroma linijinių lygčių sistema, kuri gali būti sprendžiama įvairiais metodais: Krylovo, Gausso-Sidelio ir kitais, tačiau mokslininkai siūlo problemas, susijusias su lygčių sistemos dydžiu, spręsti Jakobio metodu.
3. Jakobio metodo pagrindu šiame darbe buvo realizuotas Jakobio iteracinis algoritmas. Algoritmas parašytas *Python* programavimo kalba, naudojant *Numpy* ir *TensorFlow-GPU* bibliotekas. Programa yra universali, tinkama vykdyti ir su CPU, ir su skirtingų gamintojų GPU (*NVIDIA* ir *AMD*) naudojant skirtingas *TensorFlow* šeimos bibliotekas.
4. Atlikus skaitinius eksperimentus, kai yra keičiamas matricos dydis ir tinklo žingsnis nustatyta, kad kuo daugiau veiksmų reikia atlikti procesoriui, tuo trumpesnis bus GPU vykdymo laikas palyginti su CPU. Esant didesniam duomenų kiekiui (nuo 400^2 elementų), verta atlikti skaičiavimus su GPU, net jeigu tai bus nespecializuota, žaidimams skirta grafinė korta. Turint prieigą prie didesnio pajėgumo GPU (pavyzdžiui, *Tesla*), laimėjimas bus dar didesnis.
5. Esant mažam (iki 400^2) elementų kiekiui pasiekti šiek tiek, iki kelių sekundžių geresni CPU užduoties įvykdymo laiko atžvilgiu rezultatai. Esant daugiau elementų, pasiektas 11,7 kartų GPU pranašumas *7700 - GTX 1060* įrenginių porai. Lyginant skirtingus įrenginius, greičiausiai skaičiavimus atliko *Tesla P40* GPU. Nustatyta, kad naudojant blogiausių parametrų GPU pasiektas 1,33 karto geresnis rezultatas nei naudojant geriausių parametrų CPU. Geriausių parametrų GPU lenkia blogiausių parametrų CPU 19 kartų. Vidutiniškai GPU lenkia CPU 8,3 karto.

Literatūros sąrašas

Abhyankar, S., Constantinescu, E. M., Smith, B. F., Flueck, A. J., & Maldonado, D. A. (2016). Parallel Dynamics Simulation Using a Krylov-Schwarz Linear Solution Scheme. *Shrirang Abhyankar ; Emil M. Constantinescu ; Barry F. Smith ; Alexander J. Flueck ; Daniel A. Maldonado*, 1378-1386.

Ahamed, A.-K. C., & Magoulès, F. (2017). Efficient implementation of Jacobi iterative method for large sparse linear systems on graphic processing units. *Supercomputing*.

Amador, G., & Gomes, A. (2012). Linear Solvers for Stable Fluids: GPU vs CPU.

Amritkar, A., de Sturler, E., Świrydowicz, K., Tafti, D., & Ahuja, K. (2015). Recycling Krylov subspaces for CFD applications and a new hybrid recycling solver. *Journal of Computational Physics*.

Anzt, H., Gates, M., Dongarra, J., Kreutzer, M., Wellein, G., & Köhler, M. (2017). Preconditioned Krylov solvers on GPUs. *Parallel Computing*, 32-44.

Bell, N., & Garland, M. (2008). Efficient Sparse Matrix-Vector Multiplication on CUDA. *NVIDIA Technical Report NVR-2008-004*.

Bohacek, J., Kharicha, A., Ludwig, A., Wu, M., Holzmann, T., & Karimi-Sibaki, E. (2019). A GPU solver for symmetric positive-definite matrices vs. traditional codes. *Computers & Mathematics with Applications*.

Cormie-Bowins, E. (2012). A Comparison of Sequential and GPU Implementations of Iterative Methods to Compute Reachability Probabilities.

Corso, G. M., Gullí, A., & Romani, F. (2007). Comparison of Krylov subspace methods on the PageRank problem. *Journal of Computational and Applied Mathematics*, 159-166.

Crow, F. (1977). Shadow algorithms for computer graphics. *ACM SIGGRAPH Computer Graphics, 11*(2), 242-248.

Delbosc, N., Summers, J. L., Khan, A. I., Kapur, N., & Noakes, C. J. (2014). Optimized implementation of the Lattice Boltzmann Method on a graphics processing unit towards real-time fluid simulation. *Computers & Mathematics with Applications*, 462-475.

Fambrini, F., Iano, Y., Caetano, D. G., Rodriguez, A. A., Moya, C., Carrara, E., . . . Saito, J. H. (2018). GPU Cuda JSEG Segmentation Algorithm associated with Deep Learning Classifier for Electrical Network Images Identification. *Procedia Computer Science*, 557-565.

Feng, C., Zhang, X., & Gao, Z. (2015). An improved image super resolution and its parallel implementation based on CUDA. *2015 Tenth International Conference on Digital Information Management (ICDIM)*.

Filonenko, A., Hernández, D. C., & Jo, K.-H. (2018). Fast Smoke Detection for Video Surveillance Using CUDA. *IEEE Transactions on Industrial Informatics*, 725-733.

Flynn, M. J. (1972). Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, 948 - 960.

GeForce GTX 1060. (2019 m. 05 16 d.). Nuskaityta iš GeForce: https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-1060/specifications

GeForce GTX 860M. (2019 m. 05 16 d.). Nuskaityta iš GeForce: https://www.geforce.com/hardware/notebook-gpus/geforce-gtx-860m

Glaskowsky, P. N. (2009). *NVIDIA's Fermi: The First Complete GPU Computing Architecture.*

Intel. (be datos). *Technical Specifications.* Paimta 2017 m. 05 22 d. iš http://www.intel.com/content/www/us/en/products/processors/core/i7-processors/i7-7700.html

Intel® Core™ i7-3630QM Processor. (2019 m. 05 16 d.). Nuskaityta iš Intel: https://ark.intel.com/content/www/us/en/ark/products/71459/intel-core-i7-3630qm-processor-6m-cache-up-to-3-40-ghz.html

Intel® Core™ i7-6700K Processor. (2019 m. 05 16 d.). Nuskaityta iš Intel: https://ark.intel.com/content/www/us/en/ark/products/88195/intel-core-i7-6700k-processor-8m-cache-up-to-4-20-ghz.html

Intel® Core™ i7-7700 Processor. (2019 m. 05 16 d.). Nuskaityta iš Intel: https://ark.intel.com/content/www/us/en/ark/products/97128/intel-core-i7-7700-processor-8m-cache-up-to-4-20-ghz.html

Intel® Core™ i7-7820HQ Processor. (2019 m. 05 16 d.). Nuskaityta iš Intel: https://ark.intel.com/content/www/us/en/ark/products/97496/intel-core-i7-7820hq-processor-8m-cache-up-to-3-90-ghz.html

Jacobi, C. G. (1846). Über ein leichtes Verfahren, die in der Theorie der Säkularstörungen vorkommenden Gleichungen numerisch aufzulösen. *Crelle's Journal*, 51-94.

Kuckuk, S., & Köstler, H. (2018). Whole Program Generation of Massively Parallel Shallow Water Equation Solvers. *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 78-87.

Lu, Y., Ramachandra, A. C., Pham, M., Tu, Y.-C., & Cheng, F. (2019). CuDDI: A CUDA-Based Application for Extracting Drug-Drug Interaction Related Substance Terms from PubMed Literature. *Molecules*.

Maltsev, A., & Würsten, F. (2017 m. 06 19 d.). *ETH Zurich*. Nuskaityta iš Piz Daint: https://www.ethz.ch/en/news-and-events/eth-news/news/2017/06/piz-daint-is-a-world-leader.html

Margaris, A., Souravlas, S., & Roumeliotis, M. (2014). Parallel Implementations of the Jacobi Linear Algebraic Systems Solve.

NVIDIA cuDNN. (2019 m. 05 13 d.). Nuskaityta iš NVIDIA CUDA: https://developer.nvidia.com/cudnn

Owens, D. J., Davis, U., & Luebke, D. D. (be datos). *Intro to Parallel Programming.* Paimta 2017 m. 04 28 d. iš http://www.nvidia.com/object/cuda_home_new.html

P.Pratapa, P., Suryanarayana, P., & E.Pask, J. (2016). Anderson acceleration of the Jacobi iterative method: An efficient alternative to Krylov methods for large, sparse linear systems. *Journal of Computational Physics*, 43-54.

Radeon™ RX Vega⁵⁶ Graphics. (2019 m. 05 16 d.). Nuskaityta iš AMD: https://www.amd.com/en/products/graphics/radeon-rx-vega-56

Ramakrishna Tipireddy, E. T. (2019). Solvers and precondtioners based on Gauss-Seidel and Jacobi algorithms for non-symmetric stochastic Galerkin system of equations. *Numerical Analysis*.

Saha, M. (2017). Generalized Jacobi and Gauss-Seidel Method for Solving Non-Square Linear Systems. *Numerical Analysis*.

Semenenko, J., & Šešok, D. (2017). Lygiagretūs skaičiavimai su CUDA. *Jaunųjų Mokslininkų Darbai*, 87-93.

Shin, S., Jo, Y., Choi, J., Venkataramani, S., Srinivasan, V., & Sung, W. (2019). Workload-aware Automatic Parallelization for Multi-GPU DNN Training. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

TensorFlow. (2019 m. 05 13 d.). Nuskaityta iš ROCm, a New Era in Open GPU Computing: https://rocm.github.io/tensorflow.html

TESLA M40. (2019 m. 05 16 d.). Nuskaityta iš PNY: http://www.pny.eu/en/professional/explore-all-products/legacy-products/603-tesla-m40

TESLA M40. (2019 m. 05 16 d.). Nuskaityta iš PNY: http://www.pny.eu/en/professional/explore-all-products/legacy-products/603-tesla-m40

Wang, L., Zhu, X., Yang, B., Guo, J., Liu, S., Li, M., . . . Abrahamb, A. (2018). Accelerating nearest neighbor partitioning neural network classifier based on CUDA. *Engineering Applications of Artificial Intelligence*, 53-62.

Warrena, C., Giannopoulos, A., Gray, A., Giannakis, I., Patterson, A., Wetter, L., & Hamrah, A. (2019). A CUDA-based GPU engine for gprMax: Open source FDTD electromagnetic simulation software. *Computer Physics Communications*, 208-218.

Why TensorFlow. (2019 m. 05 13 d.). Nuskaityta iš TensorFlow: https://www.tensorflow.org

Williams, L. (1978). Casting curved shadows on curved surfaces. *ACM SIGGRAPH Computer Graphics, 12*(3), 270-274.