VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Šilumos laidumo uždavinio lygiagretinimo tyrimas naudojant centrinius ir grafinius procesorius

Steady-state heat equasion parallization analysis on CPU and GPU

Bakalauro baigiamasis darbas

Atliko:	Mantas Petrikas	(parašas)
Darbo vadovas:	dr. Rokas Astrauskas	(parašas)
Darbo recenzentas:	lekt. Irus Grinis	(parašas)

Vilnius – 2022

Santrauka

Šiame nagrinėjamos šilumos laidumo uždavinio lygiagretino galimybes naudojant centrinius ir grafinius procesorius. Darbo metu buvo implentuoti centrinius procesorius naudojantys nuoseklusis ir lygiagretusis algorimtai bei grafinius procesorius naudojantis algoritmas. Algoritmus įgyvendinančios programos buvo testuojamos Vilniaus Universiteto Matematikos ir Informatikos fakulteto Skaitmeninių tyrimų ir skaičiavimų centro paskirstytų skaičiavimų tinkle. Centrinius procesorius naudojanti lygiagretųjį algoritmą įgyvendinanti programa buvo ištestuota naudojant 256 branduolius ir buvo gautas 98 kartų pagreitėjimas. Grafinius procesorius naudojanti lygiagrečioji programa buvo ištestuota naudojant NVIDIA Tesla V100 SXM2 grafinę plokštę ir buvo gautas 360 kartų pagreitėjimas lyginant su nuosekliuoju algorimtu. Nustatyta, kad grafinius procesorius naudojanti šilumos lygi sprendžianti programa naudoja mažiau elektros energijos resursų nei centrinius procesorius naudojanti programa.

Raktiniai žodžiai: Šilumos lygtis, Laplaso lygtis, Centrinis procesorius, OpenMPI, Grafinis procesorius , CUDA

Summary

This work examines the parallelization possibilities of the heat transfer problem using CPUs and GPUs. Serial and parallel algorithms using CPU and algorithm using GPU were implanted during the work. The program that implement the algorithms were tested in the distributed computing network of the Digital Research and Computation Center of the Faculty of Mathematics and Informatics of Vilnius University. A program implementing a parallel algorithm using CPUs was tested using 256 cores and speedup of 98 was observed . A parallel program using graphics processors was tested using an NVIDIA Tesla V100 SXM2 graphics card and the process was 360 times faster to the serial algorithm. Also, this work constitrutes that an application that solves heat equation uses a graphics processor uses less power than an application that uses a central processing unit.

Keywords: Heat equation, Laplace equation, CPU, OpenMPI, GPU, CUDA

TURINYS

Įvadas

Kompiuterinės simuliacijos yra svarbi modernių tyrimų dalis. Norint elektyviai atlikti sudėtingas simuliacijas, dažnai neužtenka vieno kompiuterio branduolio resursų, ir reikia paskirstyti užduoties sprendimo darbą kelioms procesoriaus gijoms, kurios sugebėtų vienu metu, lygiagrečiai atlikti skaičiavimus. Tai galima padaryti išnaudojant centriniam procesoriuje esančius branduolius ir gijas, naudojant grafinius procesorius, ar pasitelkiant superkompiuteriuose esančius sujungtus mazgus, taip išnaudojant daugiau nei vieną centrinį ar grafinį procesorių.

Viena tipinė lygiagretinimo užduotis yra šilumos lygties sprendimas. Šilumos lygtis yra vienas iš Puasono lygties (ang. Poisson's equation) pritaikymo galimybių. Šios dalinės diferencialinės lygtys plačiai naudojamos fizikoje, sprendžiant elektrostatikos [Hou08], gravitacijos, magnetizmo [Bla96], pastovios būsenos temperatūrų [BE01] ir hidrodinamikos [Kad85] problemas. Ši dalinė diferencialinė lygtis turi kelis sprendimų būdus, kurių vienas yra naudojant baigtinių skirtumų metodą (ang. finite difference method) [YM15]. Šis metodas leidžia apskaičiuoti šilumos pasiskirstymą tam tikrame apribotame plote, žinant ploto kraštinių taškų temperatūrą. Norint efektyviau apskaičiuoti temperatūros pasiskirstymą plote, galima pradinį plotą suskirstyti į dalis, ir vykdyti temperatūros skaičiavimus atskirtuose plotuose lygiagrečiai. Šis darbas nagrinėja šilumos lygties sprendimo galimybes, naudojant centrinius procesorius (ang. CPU) ir grafinius (ang. GPU) lygiagrečiam lygties sprędimui apribotame plote, ir palygina lygiagrečiųjų algoritmų pagreitėjimą ir efektyvumą.

Darbo tikslas - įvertinti ir palyginti šilumos laidumo uždavinio lygiagretinimo algoritmo galimybes naudojant centrinius ir grafinius procesorius.

Darbo uždaviniai:

- implementuoti nuoseklųjį algoritmą, sprendžianti šilumos pasiskirstymo uždavinį
- implementuoti nuseklu ir įvertinti šilumos laidumo uždavinio algoritmo pagreitėjimą naudojant centrinius procesorius
- suprojektuoti ir implementuoti šilumos laidumo uždavinio sprendimo algorimtą, naudojantį grafinių procesorių resursus
- įvertinti grafinius procesorius naudojančio algoritmo našumą ir praktiškumą lyginant su centrinius procesorius naudojančiu algoritmu
- palyginti gautus rezultatus rezultatus su kitais panašiais problemas nagrinėjančių mokslinių darbų rezultatais

Darbo rezultatai:

- Nuoskelioji šilumos laidumo uždavinio sprendimo implementacija

- Lygiagrečioji šilumos laidumo uždavinio algoritmo sprendimo naudojanti centrinius procesorius
- Lygiagrečioji laidumo uždavinio sprendimo implementacija naudojanti grafinius procesorius
- Algoritmų teorinių ir praktinių pagreitėjimų analyzė
- Grafinius ir centrinius procesorius naudojančių algoritmų pagreitėjimų palyginimas

1. Šilumos laidumo uždavinys

Šio darbo tyrimo objektas - šilumos laidumo lygties [BF11] sprendimo būdai.

Šilumos pasiskirstymą erdvėje aprašo Puasono lygtis dvimatei erdvei:

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y) \tag{1}$$

Kadangi šiame darbe bus nagrinėjamas erdvė be papildomų vidinių šilumos šaltinių, todėl $f(x,y) = 0$, o temperatūros pasiskirstymą lemia kraštinių taškų temperatūra.

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = 0 \tag{2}$$

Ši lygtis dar yra žinoma kaip Laplaso lygtis (ang. Laplace's equation). Norint šia lygtį pritaikyti praktikoje, naudodami baigtinių skirtingų metodą apibrėžiame erdvę - matricą, kurioje bus skaičiuojama temperatūra, pradinę sistemos temperatūra šoniniuose ir vidiniuose taškuose. Šiame rašto darbe nagrinėjamas temperatūros pasiskirstymą kvadratinėje erdvėje, kurią suskirstysime į vienodo dydžio kvadratinius taškus, turinčius savo temperatūrą.

Naudodami baigtinių skirtumų metodą galime apibrėžti antros eilės dalinės diferencialinės lygties apytikslias reikšmes:

$$\frac{\partial^2 u}{\partial x^2}(x,y) \approx \frac{u(x+h,y) - 2u(x,y) + u(x-h,y)}{h^2} \tag{3}$$

$$\frac{\partial^2 u}{\partial y^2}(x,y) \approx \frac{u(x,y+h) - 2u(x,y) + u(x,y-h)}{h^2} \tag{4}$$

kur h yra atstumas tarp taškų, o $u_{x,y}$ - taško temperatūra taške, x,y. Įstatydami šias reikšmes į Laplaso lygtį gauname:

$$\frac{u(x+h,y) - 2u(x,y) + u(x-h,y)}{h^2} + \frac{u(x,y+h) - 2u(x,y) + u(x,y-h)}{h^2} = 0 \tag{5}$$

$$\frac{u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h) - 4u(x,y)}{h^2} = 0 \tag{6}$$

$$\frac{u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h)}{h^2} = \frac{4u(x,y)}{h^2} \tag{7}$$

Kadangi atstumas šiame darbe nagrinėjame uždavinyje atstumas tarp erdvės taškų nėra lygus 0, galime lygties abi puses padauginti iš h^2:

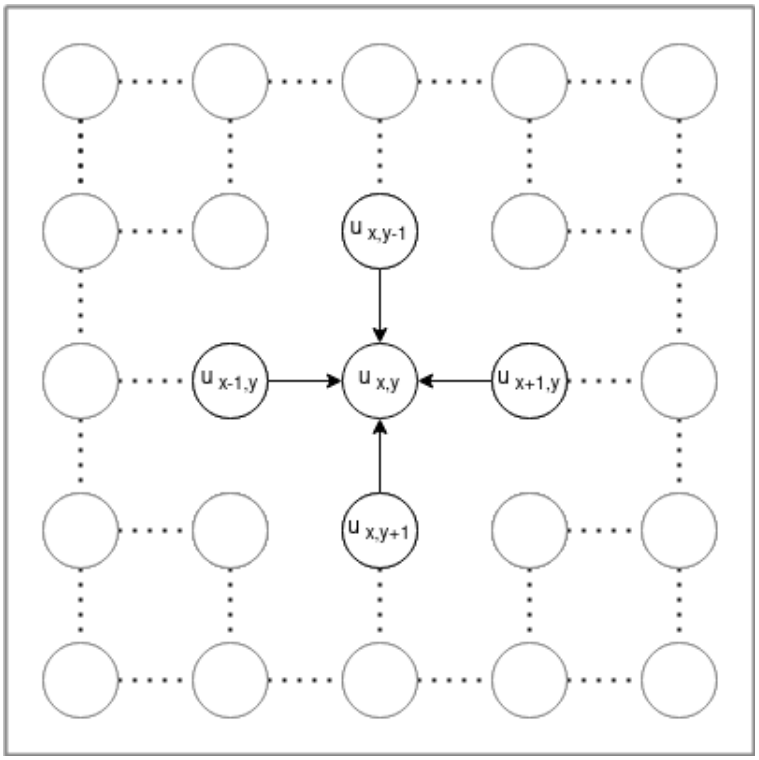
$$u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h) = 4u(x,y) \tag{8}$$

$$\frac{u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h)}{4} = u(x,y) \tag{9}$$

Šią formulę galime pritaikyti praktiškai, dvimatę erdvę projektuodami kaip matrica, padalinę ją į vienodo atstumo taškus:

$$u_{x,y} = \frac{u_{x+1,y} + u_{x-1,y} + u_{x,y+1} + u_{x,y-1}}{4} \tag{10}$$

Gauname, kad vidinio taško temperatūros reikšmė yra lygį jį supančių 4 taškų temperatūrų reikšmių vidurkiui (1 paveikslėlis).



1 pav. Temperatūros pasiskirstymas vidiniuose matricos taškuose

Norėdami išspręsti šias lygčių sistemas, naudosime Jakobi iteracinį metodą [BF11], kiekvienos iteracijos metu formulės kintamųjų reiškmes keisdami praėjusios iteracijos reikšmėmis. Laikysime, kad sistema pasiekė galutinę savo būsena, kai tų pačių taškų temperatūrų skirtumai skirtingų iteracijų metu bus mažesni už norima paklaida.

Kraštinių taškų temperatūros pasiskirstymas nulemia galutinį temperatūros pasiskirstymą erdvėje. Vidinių taškų pradinis temperatūros pasiskirstymas įtakos galiniam temperatūros pasiskirstymui neturi, tačiau gali turėti įtakos konvergencijos greičiui. Kraštinių taškų temperatūra šito

tyrimo metu aprašomą kaip:

$$f(x,0) = f(x,N-1) = |sin(x/N \times 2\pi \times 5/2)| = |sin(\frac{5\pi \times x}{N})|$$

,

kur N yra matricos kraštinės taškų skaičius. Ši formulė pirmajai ir paskutinei eilutei priskiria 5 sin bangų teigiamas reikšmes. Likusiems taškams priskiriama 0 temperatūros reikšmė. Šio darbo metu taškams priskiriamos temperatūros reikšmės nuo 0 iki 1 imtinai, taip leidžiant modeliuoti į realia objekto temperatūra taikant paprasta proporcija. Vizualiai pradinis temperatūrų pasiskirstymas matomas 2 paveikslėlyje. Juodi taškai atitinka 1 reikšmę, balti - 0. Uždavinio tikslas - nustatyti galutinį temperatūros pasiskirstymą plokštelėje, kai temperatūra plokštelėje nebekinta. Leistina paklaida (temperatūros skirtumas kuri pasiekus laikysime kad programa pasiekė galutinę savo būseną) šio tyrimo eksperimentų metu buvo 0.000001. Vizualiai galutinis temperatūros pasiskirstymas matomas 3 paveikslėlyje. Laikas per kurį temperatūra pasiskirsto erdvėje ir erdvė sudarančios medžiagų šiluminis laidumas šiame darbe nėra nagrinėjami.

2 pav. Pradinis temperatūros pasiskirstymas

3 pav. Galutinis temperatūros pasiskirstymas plokštelėje

2. Lygiagretiems skaičiavimams naudojama įranga

Šiame skyriuje apžvelgsime įvairius lygiagreties skaičiavimams naudojamus įrenginius. Skaičiavimo įrenginai dažnaiusiai skirtomi į 4 kategorijas: CPU - Centriniai procesoriai, GPU - Grafiniai procesoriai, FPGA - Vartotojų programuojamos loginių elementų matricos ir ASIC - Programos specifiniai grandynų grandinės [NSS+16]

Centriniai procesoriai (ang. central processing unit -CPU) - yra bendri procesorius, skirtas įvairioms užduotims atlikti. Populiariausias įrenginys turintis didžiausią programavimo technologų įvairove ir todėl leidžiantis lengvai įgyvendinti algoritmus Dauguma modernių centrinių procesorių savyje turi galimybes vykdyti užduotis skirtingomis gijomis, taip leidžiant atlikti kelias instrukcijas vienu metu.

Grafiniai procesoriai (ang. graphic processing unit - GPU) yra specializuotas įrenginys su patobulintomis matematinio skaičiavimo galimybėmis. Lyginant su centriniais procesoriais, šie grafiniai procesoriai turi didesnis slankaus kablelio skaičiavimo galimybes. Grafiniai procesoriai dažniausiai naudojami vykdyti vienai instrukcijos su daug duomenų (angl. single-instruction-multiple-data - SIMD) modelio programas [HN07].

Grafiniai procesoriai dėl savo architektūros apribotą galimų atlikti instrukcijų kiekį, Grafinių procesoriai naudojimi kompiuterinės grafikos, matematinių skaičiavimų ir mašininio mokymosi užduotyse [RM16].

Vartotojų programuojamos loginių elementų matricos (ang. Field Programmable Gate Array - FPGA) yra programuoji lustai, skirti programuoti loginius vartus [MC07]. FPGA leidžia programuotuojui pritaikyti lustą tam tikrai programai, taip pasiekiant didelį programos greitaveiką su energijos mažais suvartojimo kaštais. FPGA naudojami aviacijos, medicinos, dirbtinio intelekto, daiktų interneto, laidinio ir belaidžio tinklo, automobilių pramonėje [HSS+17; MC07]. Pastaruoju metu FPGA pradėjo populiarėti ir tarp debesų kompiuterijos pasiūlymų [SFJ+19]. Nors jų galimybės teoriškai yra identiškos centriniams ir grafiniams procesoriams [VDM21], dėl naudojamo žemo lygio programavimo kalbų instrukcijų jie naudojami paprastesnėms programoms.

Programos specifiniai grandynų grandinės - (ang. Application-Specific Integrated Circuit - ASIC) yra procesoriai, skirti atlikti vienai specifiniai užduočiai. Po procesoriaus pagaminimo, jo nebegalima perprogramuoti, todėl jų projektavimui ir testavimui dažniausiai naudojami kitokia aparatinė įranga (dažnai dėl projektavimo panašumo - FPGA). ASIC naudojami tada kai reikia didelės veikimo spartos, dažniausiai realaus laiko sistemose [GN13]. ASIC tipo lustai pasižymi geriausia greitaveika ir mažiausiais energijos sanaudomis, bet turi didelius proketavimo ir gaminimo kaštus [Gay93], todėl dažniausiai naudojami industrinėse sferose.

Šiame darbe šilumos pasiskirstymo uždavinys yra sprendžiamas naudojant CPU ir GPU įrenginius. Naudojant pagrininius procesorius įgyvenintas nuseklusis ir lygiagretusis algoritmai, o dėl skaičiavimo naudojamos slankaus kablelio aritmetikos lygiagretusis algortimas taip pat bus įgyventintas naudojas grafinius procesorius.

3. Algoritmų testavimo metodai

Siekiant išlaikyti vienodas sąlygas ir ištestuoti algoritmą turint didelį centinių procesorių kiekį, visi šiame darbe aprašyti eksperimentai buvo vykdomi Vilniaus Universiteto Matematikos ir Informatikos fakulteto Skaitmeninių tyrimų ir skaičiavimų centro paskirstytų skaičiavimų tinkle.

Šiame darbe testuoti CPU resursams buvo naudojamas paskirtytų paskirstytų skaičiavimų tinko „beta“ telkinys, kurį sudaro 56 (testavimo metu praktiškai buvo pasiekiami tik 42) mazgai turinys po 2 Intel Xeon X5650 procesorius, kurių kiekvienas turi po 6 branduolius. Kiekviename šio telkinio mazge yra 24 GB operatyviosios atminties (ang. RAM) ir jie turi prieigą prie 20Gbit/s infiniband tinklo. Skaičiavimų tinkle buvo instaliuota Debian operacinė sistemą, lygiagrečiam algoritmui buvo naudojamas OpenMPI bibliotekos 2.1.0 versija. Programoms paleisti buvo naudojama sistemoje esanti „Slurm“ užduočių valdymo sistema ir „short“ užduočių eilė, turinti 2 valandų ir 2 GB vienam naudojamam branduoliui limitą.

Testuoti GPU resursams buvo naudojamas „hpc“ klasterio „gpu“ telkinys, turintis 3 Tesla V100-SXM2 grafinius procesorius, kiekvienas iš kuriu turi po 32GB operatyviosios atminties, turintis 2560 dvigubos slankiojo kablelio CUDA branduolius. Skaičiavimų tinkle buvo instaliuota Qlustar 11 operacinė sistemą, sukurta Ubuntu 18.04 LTS pagrindu, gpu testavimui buvo naudota CUDA įrankių 10.1 versija. GPU programos paleisti taip pat buvo naudojama sistemoje esanti „Slurm“ užduočių valdymo sistema ir „gpu“ užduočių eilė, turinti 2 valandų ir 12 GB vienam naudojamam branduoliui limitą. GPU testavimui buvo naudojams tik 1 grafinis procesorius.

Visi programos veikimo laiko testai buvo atliekami po 3 kartus, ir tada buvo parenkamas geriausias programos vykdymo laikas. Tai buvo daroma norint išvengti kitų skaičiavimo tinklo naudotojų, sistemos pasileidimo ar laikinų tinklo sutrikimo poveikio testavimo rezultams.

Matricos kraštinė	Vykdymo laikas (s)	Iteracijų skaičius	10000 iteracijų vykdymo laikas
128	3.747	17000	2.204
256	42.596	49000	8.693
512	432.985	122000	35.491
1024	2155.811	156000	138.193
2048	> 7200	155000	543.031

1 lentelė. Nuoseklaus algoritmo vykdymo laikas

4. Nuoseklusis algoritmas

Nuoseklusis algoritmas įgyvendintas C++ kalba. Algoritmo pradžioje alokuojama atmintis dviems N*N dydžio masyvams, kur N yra matricos kraštinės ilgis: viename saugoma dabartinė matricos būsena, kitas yra pildomas naujomis reikšmėmis iteracijos metu. Naudojami vienmačiai masyvai, nes didžioji dalis OpenMPI bibliotekos funkcijų, kuri buvo naudotos lygiagretinant algoritmą, tikisi vienmačių masyvų duomenų perdavimui, o nuoseklaus algoritmo veikimui vienmačių ir dvimačių masyvų naudojimas nedaro didelės įtakos algoritmo veikimo laikui. Pirmasis masyvas užpildomas pradine matricos būsena, tada pradedamas iteracinis procesas, kai kiekvienos iteracijos metu, kiekvienam vidiniam matricos taškui yra priskiriama reikšmė, lygi jį supančių 4 taškų vidurkiui, o kraštinės matricos reikšmės nekinta. Taip pat iteracijos metu fiksuojamas maksimalus temperatūros pasikeitimas, skirtas nustatyti ar matrica pasiekė galutinė savo būseną. Ši reikšmė kiekvienos iteracijos palyginama su nustatyta paklaidos reikšme ir jei temperatūros pasikeitimas yra mažesnis už leistina paklaida, laikome kad matrica pasiekė savo galutinį temperatūrų pasiskirstymą.

Testuojant nuoseklųjį algoritmą su skirtingais matricos kraštinės ilgio reikšmėmis (1 lentelė), nepastebėta tiesioginės sasajos tarp vykdymo laiko ir kraštinės ilgio. Tačiau fiksuojant iteracijų skaičių, pastėbėta logoritminė priklausomybė tarp programos iteracijų vykdymo laiko ir kraštinės ilgio. Padidinus kraštinės ilgį beveik 2 kartus, vykdymo laikas padidėja beveik 4 kartus. Tai galima sieti su tuo, kad algoritmas nagrinėja šilumos pasiskirstymą dvimatinėje erdvėje, todėl padidinus matricos kraštinės ilgį 2 kartus, duomenų kiekis padidėja 4 kartus. Matricos kraštinės ilgiai pasirinkti dvejeto laipsniai, kad lygiagretinant algoritmą vidinės matricos reikšmes būtų padalinti skirtingiems procesams vienodais kiekiais. Į ši skaičių neieina papildomos dvi eilutės, kurios šiame skaičiavime atitinka matricos kraštines - eilutes ir stulpelius kurių temperatūrų reikšmės nekinta. Testuojant nuoseklųjį algoritmą, kai matricos kraštinę sudaro 2048 taškai, buvo pasiektas sistemos užduočių vykdymo laiko limitas (2 valandos), todėl 10000 iteracijų skaičiavimo reikšmė nustatyta pagal pirmų 10000 iteracijų vykdymo laiką..

5. Lygiagretusis algoritmas, naudojantis centrinius procesorius

Pradžiai apibrėšime kriterijus, pagal kurios bus vertinamas lygiagretus algoritmas. Vienas iš pagrindinių vertinimo kriterijų yra algoritmo pagreitėjimas, nusakantis kiek kartų greičiau lygiagretus algoritmas, naudojantis p branduolių, įvykdo užduotį už nuoseklųjį algoritmą [EZL89]. Algoritmo pagreitėjimas vertinamas kaip:

$$S(p) = \frac{t_s}{t_p},$$

kur p yra procesorių skaičius, t_s - geriausias nuoseklaus algoritmo vykdymo laikas, t_p - lygiagretaus algoritmo, naudojančio p procesorių, vykdymo laikas.

Maksimalus įmanomas teorinis pagreitėjimas gali būti tiesinis. Tai galėtų būti pasiekta kai duomenys yra padalinami į vienodas dalis, o komunikacijos tarp procesų nevyksta. Tuomet pagreitėjimas yra lygus procesorių skaičiui:

$$S_{max}(p) = \frac{t_s}{t_s/p} = p.$$

Jei lygiagretaus algoritmo pagreitėjimo reikšmė yra didesnė nei procesorių skaičių - $S(p) > p$ - pasiekiamas supertiesininis (ang. superlinear) pagreitėjimas. Tai gali atsitikti dėl neoptimaliai įgyvendinto nuoseklaus algoritmo, arba netiksliai įgyvendinto lygiagretaus algoritmo, kai yra praleidžiami žingsniai.

Maksimalus praktinis algoritmo pagreitėjimas priklauso nuo algoritmo dalies, kuri gali būti išlygiagretinamas. Jei algoritmo dalį, kuri negali būti išlygiagretinama, pažymėsime f ($0 \leq f \leq 1$), trumpiausias lygiagrečios programos vykdymo laikas bus lygus $f \times t_s + (1-f) \times t_s/p$, o maksimalus pagreitėjimas apibrėžiamas kaip Amdahl'o dėsnis [Amd67]:

$$S(p) = \frac{t_s}{f \times t_s + (1-f) \times t_s/p} = \frac{p}{1 + (p-1)f}.$$

Šiame darbe nagrinėjamo lygiagretaus algoritmo nuosekliojoje dalyje (t.y. dalyje kuri nėra išlygretinama) vyksta tik konfiguracinių parametrų nuskaitymas. Taip pat nėra lygiagretinamas paketų užkrovimas ir kitos prieš pagrindinės (ang. main) programos funkcija iškvietimą vykstančios operacijos, tačiau tai nedaro įtakos algoritmo tyrimui, nes laiko matavimas pradedamas tik iškvietus pagrindinę programos funkciją. Matricos kraštines suskaidžius į 1048 taškų, nuoseklaus algoritmo konfiguracijos nuskaitymas vidutiniškai truko 0,002 sekundes (žymėsime t_{np}), kai tuo

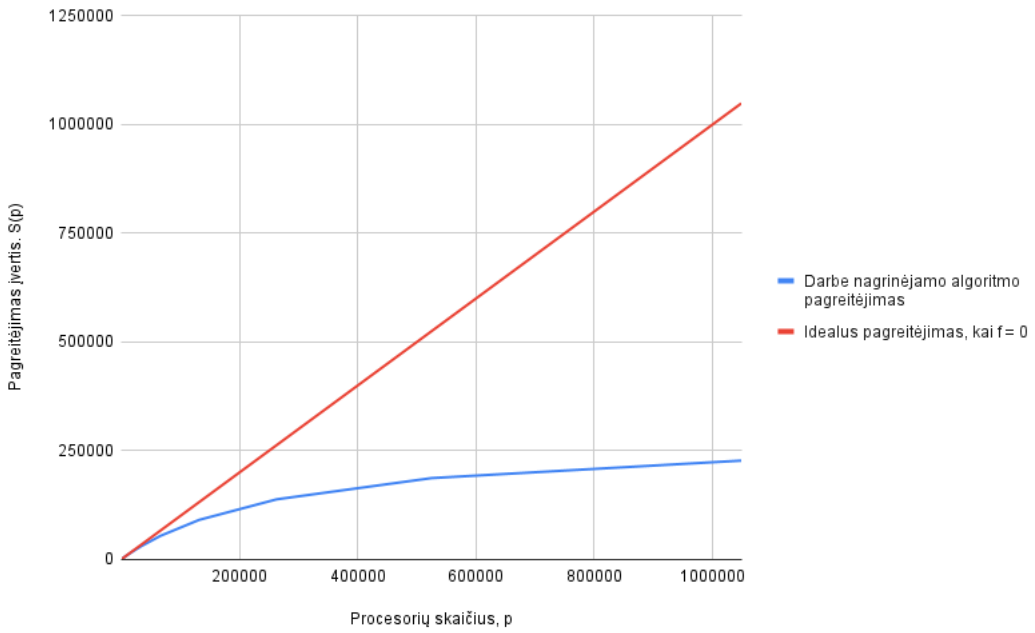
metu likusieji skaičiavimai vidutiniškai užtruko 2155.811 sekundes. Toks laiko santykis reikia kad nelygiagretinama algoritmo dalis yra:

$$f = \frac{t_{np}}{t_p} = \frac{0.002}{0.002 + 2155.811} \approx 0.00000093.$$

Šis įvertis reiškia, kad maksimalus algoritmo pagreitėjimas naudojant begalinį skaičių procesorių bus lygus:

$$\underset{p \to \text{inf}}{S(p)} = \frac{1}{f} = \frac{1}{0.00000093} \approx 1075268;$$

o tai reiškia neišlygegretinama algoritmo dalis turės įtakos lygiagretaus vykdymo laikui tik naudojant labai didelį procesų kiekį.



4 pav. Maksimalus lygiagretaus algoritmo pagreitėjimo įverčio priklausomybė nuo procesorių skaičiaus

Maksimalaus pagreitėjimo reiškmes naudojant skirtus procesorių kiekius matomos 4 grafike. Šio darbo ribose algoritmas nebus tiriamas naudojant labai didelį procesorių kiekį, todėl net ir naudojant maksimalų procesorių kiekį (256), maksimalus pagreitėjimo reikšme bus artima procesorių skaičiui ($S(256) \approx 255.939$), todėl neišlygegretinta algoritmo dalis neribos algoritmo pagreitėjimo.

Taip pat lygiagretiems algoritmams vertinti naudojama efektyvumo metrika [EZL89]. Darant prielaidą, kad nuoseklusis algoritmas visada efektyviai išnaudoja jam priskirto procesoriaus resursą, efektyvumo metrika parodo kokia laiko dalį nuoseklusis sprendimas išnaudoja jam priskirtus

procesorius. Efektyvumas, dažniausiai žymimas reide E, matematiškai išreiškiamas kaip nuoseklaus algoritmo ir lygiagretaus algoritmo padauginto iš naudotų branduolių skaičiaus vykdymo laiko santykis:

$$E = \frac{t_s}{t_p * p}$$

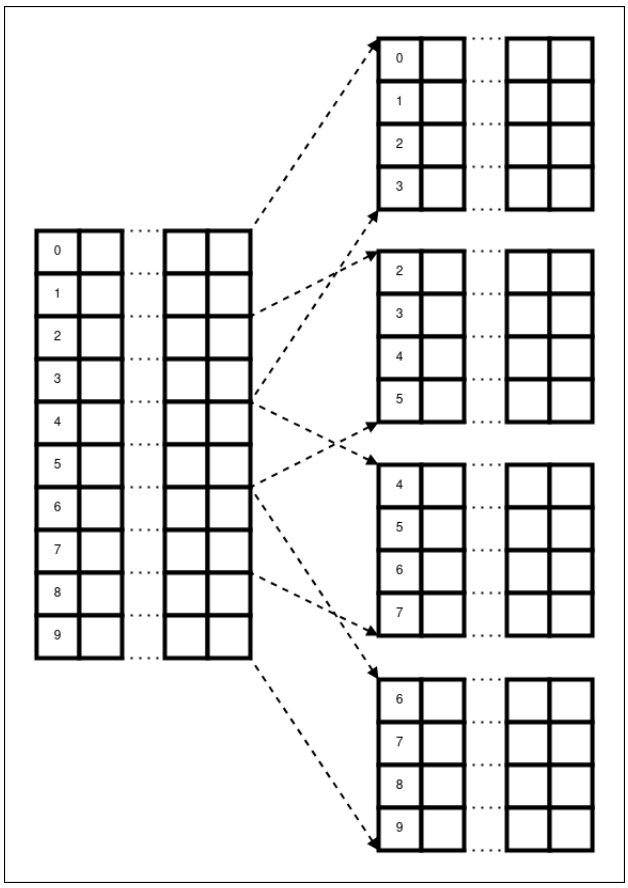
Taip pat efektyvumą galime išreikšti per algoritmo pagreitėjimą:

$$E = \frac{S(p)}{p}$$

Įdealiu atveju, esant maksimaliam algoritmo pagreitėjimui $S(p) =$" algoritmo efektyvumas bus lygus 1. Tai pasiekiama kai visi procesoriai visa laiką yra išnaudojami skaičiavimo užduotims.

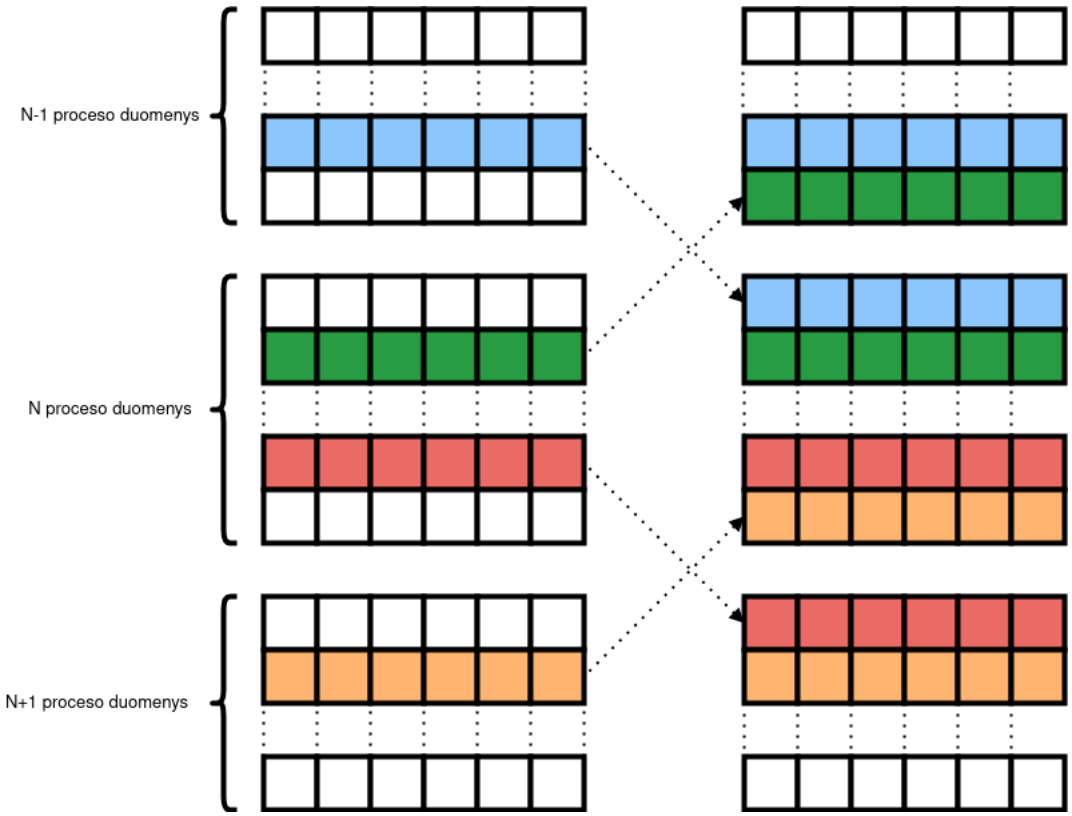
5.1. Duomenų dalinimas eilutėmis

Viena iš duomenų paskirstymo lygiagrečiajam algoritmui strategijų yra duomenis paskirstyti eilutėmis (arba stulpeliais). Norint $N * N$ dydžio matricos duomenis padalint duomenis p procesams, kiekvienam procesui programos vykdymo pradžioje priskiriama po $(\frac{N-2}{p} + 2) * N$ taškų. $N - 2$ atitinka vidinių matricos stulpelių kiekį, kuris kinta laikui bėgant (2 atitinka išorinius matricos stulpelius turinčius pastovią temperatūrą). Ši reikšmė padalinama iš procesų skaičiaus, ir kiekvienas procesui priskiriamos po 2 papildomas eilutes kurios yra naudojamos vidinių reikšmių skaičiavimui (5 paveikslėlis).



5 pav. Duomenų padalinimas. 10 eilučių padalinamos 4 procesams.

Naudojant Jakobi iteracinį metodą, kiekvienos iteracijos metu, pirmos ir paskutinės padalintų eilučių reikšmės turi būti sinchronizuojamos tarp gretimas matricos dalis apdorojančių procesų 6. Pirmasis ir paskutinis procesas šonines reikšmės sinchronizuoja tik su vienu iš procesu, nes viena iš jas sudarančių matricos eilučių atitinka pradinės matricos kraštines eilutes, turinčias pastovias reikšmes.



6 pav. Duomenų sinchronizacija tarp skirtingų procesų

Norint nustatyti, kada didžiausias temperatūrų iteracijų skirtumas pasiekė norimą reikšmę ir reikia nustoti vykdyti programą, kiekvienoje proceso iteracijos pabaigoje suskaičiuoja didžiausią lokalų temperatūros pokytį ir jį siunčia pagrindiniam procesoriui, kuris gavęs visas reikšmes, įvertiną ar bent vienas procesoriaus lokalus skirtumas viršija norima paklaidą, ir nusprendžia ar tęsti iteravimą. Norint išvengti visuotinio sinchronizavimo tarp visų branduolių ir pagrindinio proceso kiekvienos iteracijos metu, maksimalaus pokyčio reikšmių siuntimas ir palyginimas vykdomas tik kas 1000 iteracijų.

5.2. Lygiagretaus algoritmo, kai duomenys padalinimo eilutėmis, analyzė

Lygiagretaus algoritmo veikimo laiko įverčiai, kai matricos kraštinę sudaro 1048 taškų pateikiami 2 lentelėje. Prieš testuojant lygiagretųjį algoritmą pakeitus branduolių skaičių, buvo atliekamas testinis paleidimas. Testinio paleidimo yra skirtas užtikrinti kad visi užduočiai reikalingi mazgai nebūtų miego būsenoje (ang. Hibernation) ir būtų pasiruošę vykdyti algoritmą.

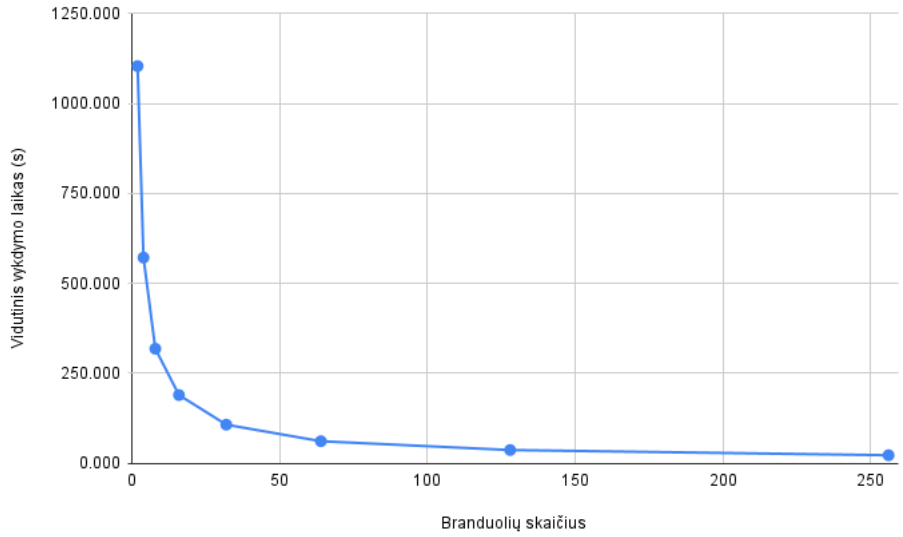
Kaip matoma 7 grafike, vykdymo laikas atvirkščiai priklauso nuo naudojamų branduolių skaičiaus.

Branduolių skaičius	Vykdymo laikas (s)
128	4027.416
256	2236.912

3 lentelė. Lygiagretaus algoritmo vykdymo laikas, kai matricos kraštinę sudaro 4096 taškai

Branduolių skaičius	Vykdymo laikas (s)
2	1104.232
4	571.615
8	318.093
16	189.300
32	106.901
64	60.894
128	36.370
256	21.911

2 lentelė. Lygiagretaus algoritmo vykdymo laikas, kai matricos kraštinę sudaro 1048 taškų

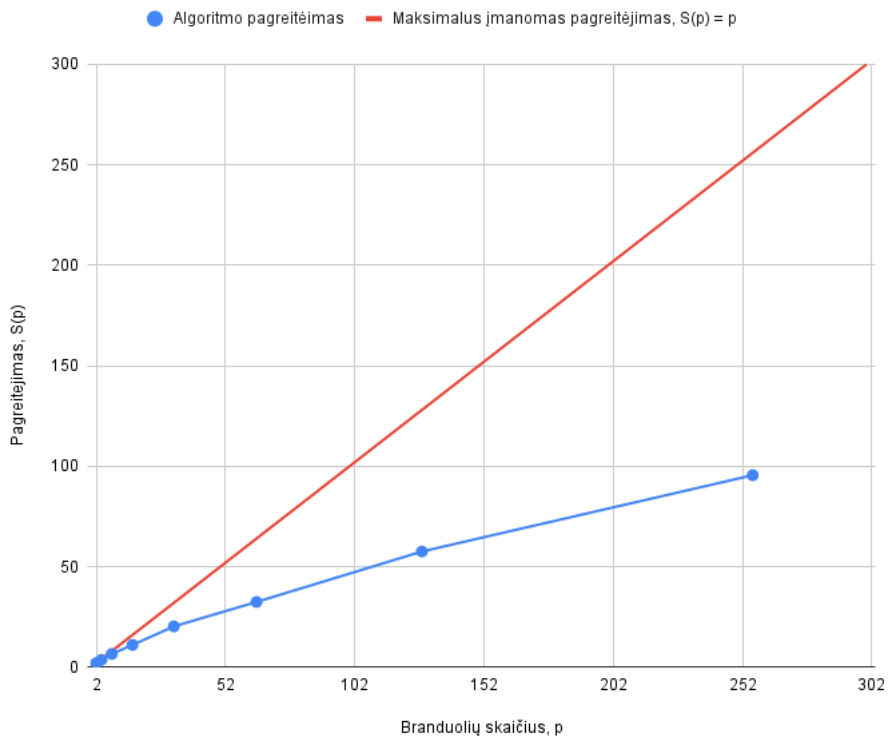


7 pav. Lygiagretaus algoritmo vykdymo laiko priklausomybė nuo naudojamų branduolių skaičiaus, kai matricos kraštinę sudaro 1048 taškų

Taip pat lygiagretus algoritmas buvo testuojamas kai matricos kraštinę sudarė 4096 taškai, vykdymo laikas matomas 3 lentelė. Testuojant naudojant 64 ar mažiau branduolių, programos veikimo laikas viršijo 2 valandas, ir programa buvo sustabdyta. Naudojant šią matricos kraštinės reikšmę nuosekliojo algoritmo programa viršydavo sistemos atminties limitą (2GB 1 branduoliui).

Branduolių skaičius	Vykdymo laikas	Pagreitėjimas, $S(p)$	Efektyvumas, $E(p) = \frac{S(p)}{p}$
1	2155.811	1	1
2	1104.232	1.952	0.98
4	571.615	3.771	0.94
8	318.093	6.777	0.85
16	189.300	11.388	0.71
32	106.901	20.166	0.63
64	60.894	35.402	0.55
128	36.370	59.275	0.46
256	21.911	98.390	0.38

4 lentelė. Lygiagretaus algoritmo pagreitėjimas ir efektyvumas.



8 pav. Lygiagretaus algoritmo pagreitėjimo priklausomybė nuo naudojamų branduolių skaičiaus, kai matricos kraštinę sudaro 1048 taškų

8 grafike matomas lygiagretaus algoritmo logaritminis pagreitėjimas, tačiau Amdahl'o dėsnio [Amd67] apibrėžiamas maksimalus pagreitėjimas nėra pasiekiamas - naudojant tokį branduolių kiekį pagreitėjimas turėtų būti artimas idealiam pagreitėjimui $S_{max}(p) = p$. Tai galimai nutinka dėl to, kad Amdahl'o dėsnis neatsižvelgia į komunikacijos tarp skirtingų procesorių kaštus. Tikslų programos skaičiavimų ir komunikacijos santykį įvertinti yra sunku, nes komunikacijos laikas gali skirtis laikas priklausomai ar komunikuojantys procesai naudoja to pačio ar skirtingų centrinių procesorių branduolius, ir ar centriniai procesoriai yra toje pačioje ar skirtinguose motininėse plokštelėse. Tas iš dalies matoma 4 lentelėje, kad algoritmo efektyvumas naudojant 2 ir 4 branduo-

lius yra artimas 1, o naudojant daugiau branduolių lygiagretaus sprendimo efektyvumas mažėja. Tai galima sieti su tuo kad testavimui naudoto paskirstytų skaičiavimų tinklo mazguose naudojami procesoriai, turinys 6 branduolius, todėl naudojant 2 ir 4 branduolius didelė tikimybė, kad bus naudojamas 1 fizinis procesorius, taip išvengiant didelių komunikacijos kaštai tarp atskirų procesų. Taip pat, 4 lentelėje matomas algoritmo efektyvumo mažėjimas didinant branduolių skaičių, iš to galime daryti prielaidą, kad pradinius duomenis skaidant į vis mažesnes dalis, vis didesnę algoritmo vykdymo laiką užima komunikacija tarp skirtingus duomenis apdorojančių procesų. Nors testuojant su vis didesnių branduolių skaičiumi algoritmo efektyvumas mažėja, lygiagretaus algoritmo pagreitėjimo metrika didinant naudojamų branduolių nenustoja didėti, iš to galima daryti prielaidą šių testų metu nebuvo rastas branduolių skaičius, kurį naudojant komunikacijos kaštai nusvertų gautą algoritmo pagreitėjimą, kai užduotis yra padalinama į mažesnes dalis.

6. Lygiagretusis algoritmas, naudojantis grafinius procesorius

Lygiagrečiam algoritmui sukurti naudojančiam grafinius procesorius buvo naudota C++ kalbos CUDA biblioteka [YZP08]. Algoritmo pradžioje alokuojama atmintis trims N*N dydžio masyvams, kur N yra matricos kraštinės ilgis. Dviejuose masyvuose, kaip nuosekliąjį algoritmą įgyvendinančiojoje programoje, saugoma dabartinė matricos būsena ir pildomos naujomis reikšmėmis iteracijos metu, o trečiasis yra skirtas saugoti temperatūrų skirtumui tarp iteracijų ir yra naudojamas galutinės pakaidos skaičiavimui. Programos vykdymo pradžioje, kaip ir nuosekliajame algortime, pirmasis masyvas užpildomas pradine matricos būsena. Kadangi naudojamas grafinis procesorius, kuris yra skirtas operuoji savo atmintyje esančia infromacija, visų tris masyvams reikia išskirti atminties vieta GPU atmintyje ir ten nukopijuoti duomenis iš bendrosios paskirties atminties.

Kaip ir nusekliajame algoritme tada pradedamas iteracinis procesas, kai kiekvienos iteracijos metu, kiekvienam vidiniam matricos taškui yra priskiriama reikšmė, lygi jį supančių 4 taškų vidurkiui, o kraštinės matricos reikšmės nekinta. Taip pat iteracijos metu į atskirą matricą fiksuojamas maksimalus temperatūros pasikeitimas, skirtas nustatyti ar temperatūros pasiskirstymas pasiekė galutinė savo būseną. Kas 1000 iteracijų šios matricos reikšmės yra nukopijuojamos iš grafinės prokštės atminties į bendrąją atmintį, taip gražinant jos reikšmes į pagrindinį procesą. Tada pagrindinis procesas, naudodamas centrinio procesoriaus resursus, randa didžiausia matricos reikšmė ir ji yra palyginama su konfiguracijos faile nustatyta paklaidos reikšme. Jei temperatūros pasikeitimas yra mažesnis už numatytąją paklaida, laikome kad matrica pasiekė savo galutinį temperatūrų pasiskirstymą, ir temperatūros matricos reikšmės yra nukopijuojamos iš grafinės prokštės atminties į bendrąją atmintį.

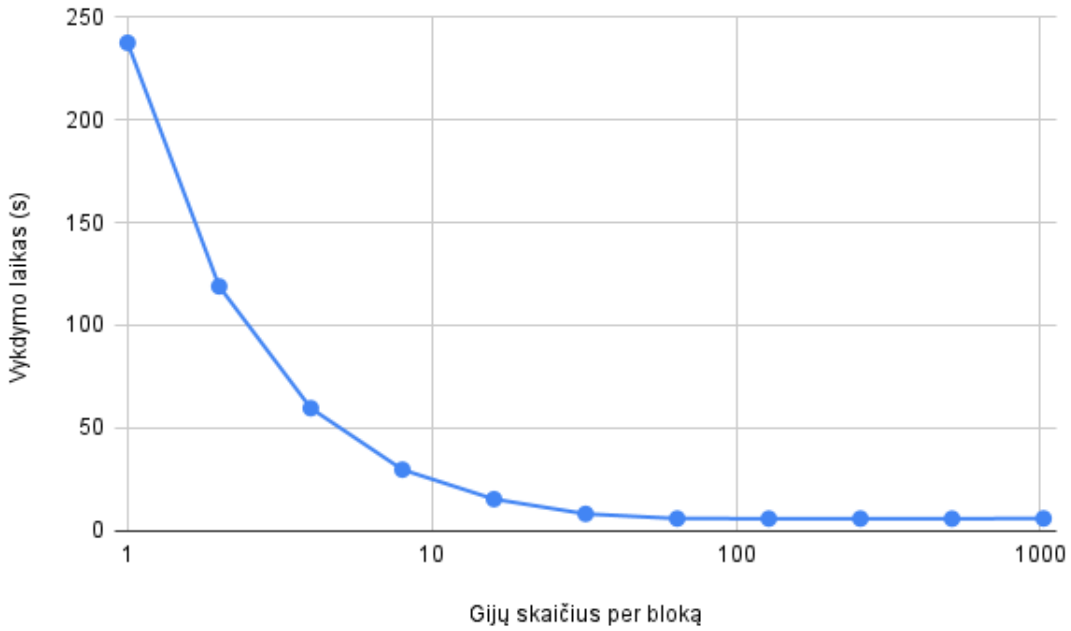
Prieš paleidžiant kodą, konfiguraciame faile nurodomas duomenų pasiskirstymas. CUDA programoms duomenų paskirtymui naudojami du parametrai blokų kiekis ir gijas per bloką. Kiekvieno bloko duomenys yra apdorojimi vieno iš GPU srautinių apdorojimo proceriaus (ang. streaming multiprocessor - SM). Kadangi kiekvienos iteracijos metu aprodojami turi būti visi duomenys, bendras gijų kiekis turi atitikti matricos taškų kiekį:

$$Bloku\_skaicius \times giju\_skaicius\_per\_bloka = N \times N,$$

kur N yra matricos dimensija.

Gijų skaičius per bloką	Blokų skaičius	Vykdymo laikas, s	Pagreitėjimas, kartais
1	1048576	237.355	9.1
2	524288	118.921	18.1
4	262144	59.709	36.1
8	131072	29.8363	72.3
16	65536	15.492	139.2
32	32768	8.325	259.0
64	16384	6.05	356.3
128	8192	6.026	357.8
256	4096	6.03	357.5
512	2048	5.984	360.3
1024	1024	6.097	353.6

5 lentelė. Lygiagretaus algoritmo naudojančio GPU vykdymo laikas naudojant skirtingus blokų ir gijų per bloką parametrus, kai matricos kraštinę sudaro 1048 taškų



9 pav. Lygiagretaus algoritmo naudojančio GPU vykdymo Lygiagretaus algoritmo naudojančio GPU vykdymo laiko priklausomybė nuo gijų per bloką skaičiaus

Priešingai nei testuojant lygiagretųjį algorimtą naudojantį centrinius procesorius, naudojamų gijų skaičius nebuvo ribojamas - buvo naudoja visi grafinio procesoriaus pajėgumai. Šilumos uždavinio lygiagretinimo rezultatai naudojant grafinį procesorių su skirtingas blokų ir gijų per bloką parametrus, o matricos kraštinė yra lygi 1024 taškams, yra pateikiami 5 lentelėje. Matoma, kad naudojant GPU pasiektas pagreitėjimas iki 360 kartų lyginant su nuosekliuoju CPU naudojančiu algoritmų.

Kaip matoma 9 grafike, lygiagretusis algoritmas veikia daug lečiau jei yra naudojami mažiau nei 32 gijos per bloką. Tai galima sieti su tuo kad CUDA architektūroje kiekvienas procesorius

Matricos kraštinės ilgis	Vykdymo laikas, s	Iteracijų skaičius	10000 iteracijų vykdymo laikas
128	0.063	17000	0.037
256	0.175	49000	0.035
512	0.783	122000	0.064
1024	5.643	156000	0.361
2048	21.463	155000	1.384
4196	94.393	168000	5.618
8392	463.99	210000	22.094

6 lentelė. Lygiagretaus algoritmo naudojančio GPU vykdymo laikas naudojant 512 gijas per bloką

vykdo vieną komandą 32 gijoms iš karto, todėl naudojant mažiau nei 32 gijas nėra pilnai išnaudojamos procesoriaus galimybes. Naudojant daugiau gijų nėra pasiekiamas geresnis rezultatas, nes šio uždavinio GPU implementacijos skaičiavimuose yra naudojama tik bendra atmintis, todėl nėra išnaudojamas blokuose esanti greitesnė dedikuota atmintis. Naudoti daugiau nei 1024 gijas per bloką neleidžia CUDA architetūra.

Taip pat grafinių procesorių greitaveika buvo tiriama naudojant skirtingus matricų kraštinių reikšmes. Kaip matoma 6 lentelėje, naudojant mažas matricų kraštinių ilgio reikšmes (iki 512), 10000 iteracijų vykdymo greitis nekinta, tačiau toliau didinant matricos kraštinių reikšmės 10000 iteracijų vykdymo laikas didėja logoritmiškai, kaip ir nuosekliajame algortime. Iš to galima daryti išvadą, kad GPU skaičiavimuose naudojant mažą duomenų kiekį, didelę programos vykdymo laiko dalį užima GPU branduolių ir duomenų kopijavimo inicijavimas.

Šiame darbe implementuotą grafinius procesorius naudojantį algoritmą galime palyginti su kitame šilumos laidumo lygties sprendimą nagrinėjančiu darbe [BCB+21] aprašytu algoritmu. Kitame darbe [BCB+21] algoritmas buvo testuojamas naudojant 5120×5120 ir 20000×20000 dydžio matricas, tačiau programa buvo vykdoma tik ribota iteracijų skaičių - 5000 ir algoritmo vykdymas užtruko atitinkamai 8.85 ir 141.11 sekundžių. Testuojant šio darbo rašymo metu implementuotą algoritmą su tokias pačiais parametrais, algoritmo vykdymo laikas užtruko atitinkamai 4.142 ir 62.31 sekundes. Pagal lyginamajame darbe [BCB+21] aprašytą grafinę plokštę ir naudojamų CUDA branduolių skaičių galime, nustatyti kad jame buvo naudojami viengubo tikslumo slankaus kablelio - 32 bitų skaičiai. Šiame darbe atliktuose testuose yra naudojami dvigubo tikslumo slankaus kablelio - 64 bitų skaičiai. Perrašius šio darbo algoritmą, kad jis naudotų viengubo tikslumo slankaus kablelio skaičius, programos vykdymo laikai atinkamai sutrumpėjo iki 2.152 ir 32.871 sekundžių, ką galime sieti su tuo kad V100 vaizdo plokštė turi dvigubai daugiau 32 bitu slankaus kablelio skaičius procesorių. Matoma kad šio darbe atliktų testų metu grafines plokštes naudojantis algoritmas veikia daugiau nei 4 kartus greičiau. Ši laiko skirtumą galima sieti kad šio darbo metu buvo naudotas naujesnis ir galingesnis grafinis procesorius turintis 4 kartus daugiau CUDA branduolių,

taip pat skirtumą galėjo paveikti ir algoritmų implementacijų skirtumai. Atsižvelgdami į tai, galime tvirtinti kad darbe nagrinėjamas grafinius procesorius naudojantis algoritmas veikia korektiškai.

7. Grafinius ir centrinius procesorius naudojančių algortimų palyginimas

Norint palyginti GPU ir CPU reikia įsivesti vertinimo kriterijus.

Vienas vertinimo kriterijus yra vykdymo laikas. Jis yra svarbus kai vykdoma užduoties sprendimas yra jautrus laikui [YWB+19]. Vienas tokio vertinimo minusų yra neatsižvelgimas į skaičiavimo kaštus - naudojant didesnį skaičiavimo resursų kiekį dažniausiai bus pasiekiamas greitesnis rezultatas.

Dažnai GPU ir CPU algoritmo palyginuose vykdymo laiko metrija yra CPU ir GPU branduolių skaičiumi [BCB+21; GLN+08]. Toks palyginimas nėra visiškai korektiškas nes CPU ir GPU branduoliai skiriasi savo architektūra, veikimo greičiu ir paskirtimi, o ir skirtingų kartų įrenginiai skiriasi skaičiavimo galimybėmis (ang. compute capability) ir veikimo greičiu.

Dar vienas būdas lyginti centrinių ir grafinių procesorių veikimo efektyvumą yra matuoti algortimo veikimo metu naudojama energiją. Tai leistų praktiškai įvertini sunaudotos elektros energijos kaštus duomenų centrui. Šiame darbe atliktų eksperimentų metu naudotos elektros energijos kaštų informacija nėra prieinama, todėl kaštai bus vertinami teoriškai.

Ekperimentų metu naudojamas V100 grafinis procesorius esant pilnai apkrovai gali naudoti 300W energijos. Centrinių procesorių naudojamos energijos kiekis priklauso atliekamos užduoties [vKBB+16], todėl vertinimui naudosime gamintojo pateikiama TDP vertę, kuri Xeon X5650 atveju yra 95W. Skaičiamu metu laikysime kad likusieji mazgų komponentai skaičiavimų metu naudoja 100W energijos, o rezultatą norime gauti maksimaliai greitai.

Lygiagrečiajam algoritmui, naudojančiam grafinius procesorius, vykdyti bus reikalingas vienas mazgas su dviems centriniais procesoriams ir vienu grafiniu procesoriumi. Darydami prielaidą kad sistema ekperimento metu bus pilnai apkrauta, galime apytiksliai numatyti ši sistema per vieną skaičiavimų sekundę sunaudotos $100 + 300 + 95 \times 2 = 590$ Vatų energijos. Kadangi geriausiais algoritmas buvo vykdomas 6s sekundes, tai sistema su GPU skaičiavimu metu sunaudos $590 \times 6 = 3540$ Džaulius energijos.

Lygiagrečiajam algoritmui, naudojančiam centrinius procesorius, vykdyti bus reikalingas 22 mazgai su dviems centriniais procesoriams. Ši sistema apytiksliai per vieną skaičiavimų sekundę sunaudotos $(100 + 95 \times 2) \times 22 = 6160$ Vatų energijos. Kadangi geriausiais algoritmas buvo vykdomas 21s sekundes, tai sistema naudodama tik centrinius procesorius skaičiavimu metu sunaudos $6160 \times 21 = 129360$ Džaulius energijos.

Pagal šiuos skaičiavimus galima matyti kad grafiniai procesoriai šilumos pasiskirtymo užda-

vinio sprendimui sunaudos mažesnį energijos kiekį.

Rezultatai

- Implementuoti nuoseklusis ir lygiagretusis centrinius procesorius naudojantys algoritmai spendžiantys šilumos pasiskirstymo užduotį Jacobi iteraciniu metodu naudojant baigtinių skirtumų schemą.
- Atliktas lygiagretaus algoritmo, naudojančio centrinius procesorius, efektyvumo ir teorinio ir praktinio pagreitėjimo analyzė.
- Implementuotas lygiagretusis šilumos pasiskirstymo uždavinį sprendžiantis algortimas naudojantis grafinius procesorius.
- Algoritmai buvo ištestuoti naudojant Vilniaus Universiteto Matematikos ir Informatikos fakulteto Skaitmeninių tyrimų ir skaičiavimų centro paskirstytų skaičiavimų tinklo resursus. Naudojant 256 centrinius procesorius buvo pasiektas 98 kartų pagreitėjimas, o naudojant 1 Nvidia Tesla V100-SXM2 32GB grafinį procesorių buvo pasiektas 360 kartų pagreitėjimas.
- Atliktas algoritmo, naudojančio grafinius procesorius, vykdymo laiko palyginimas su tokį pati uždavinį sprendžiančiu darbu [BCB+21] ir gautas ekvivalentus rezultatas.

Išvados

- Sprendžiant šilumos pasiskirstymo uždavinį pasiskirtytų skaičiavimo tinkle naudojant centrinius procesorius komunikacija tarp skirtingu tinklo mazgų lėtiną algoritmo veikimą.
- Norint gauti geresnius centrinius procesorius naudojančio lygiagretaus algoritmo pagreitėjimo ir efektyvumo rezultatus, būtų verta pabandyti kitokius duomenų dalinimo būdus (pvz. kvadratais), didinti procesorių skaičių mazguose.
- Grafiniai procesoriai leidžia greičiau ir naudojant mažiau energijos resursų išspresti šilumos pasiskirstymo uždavinį nei to šio uždavinio sprendimas naudojant centrinius procesorius.
- Norint padidinti grafinius procesoriaus naudojančio algoritmo veikimo greitį, būtų verta pabandyti naudoti grafinių plokščių blokuose esančią dedikuotą atmintį, naudoti daugiau nei vieną vaizdo plokštę.

Literatūra

[Amd67] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the April 18-20, 1967, spring joint computer conference*, p. 483–485, 1967.

[BCB+21] Safa Belhaous, Soumia Chokri, Sohaib Baroud ir Mohamed Mestari. Comparative study of the execution time of parallel heat equation on cpu and gpu. *Journal of Communications Software and Systems*, 17(4):350–357, 2021.

[BE01] Fredrik Berntsson ir Lars Eldén. Numerical solution of a cauchy problem for the laplace equation. *Inverse Problems*, 17(4):839, 2001.

[BF11] Richard L. Burden ir J. Douglas Faires. *Numerical analysis*. Cengage learning, 2011. 714 psl.

[Bla96] Richard J Blakely. *Potential theory in gravity and magnetic applications*. Cambridge university press, 1996.

[EZL89] Derek L Eager, John Zahorjan ir Edward D Lazowska. Speedup versus efficiency in parallel systems. *IEEE transactions on computers*, 38(3), 1989.

[Gay93] Rick Gayle. The cost of quality: reducing asic defects with i/sub ddq/, at-speed testing, and increased fault coverage. *Proceedings of IEEE International Test Conference-(ITC)*, p. 285–292. IEEE, 1993.

[GLN+08] Michael Garland, Scott Le Grand, John Nickolls, Joshua Anderson, Jim Hardwick, Scott Morton, Everett Phillips, Yao Zhang ir Vasily Volkov. Parallel computing experiences with cuda. *IEEE micro*, 28(4):13–27, 2008.

[GN13] Mariangela Genovese ir Ettore Napoli. Asic and fpga implementation of the gaussian mixture model algorithm for real-time segmentation of high definition video. *IEEE transactions on very large scale integration (VLSI) systems*, 22(3):537–547, 2013.

[HN07] Pawan Harish ir Petter J Narayanan. Accelerating large graph algorithms on the gpu using cuda. *International conference on high-performance computing*, p. 197–208. Springer, 2007.

[Hou08] MG House. Analytic model for electrostatic fields in surface-electrode ion traps. *Physical Review A*, 78(3):033402, 2008.

[HSS+17] Brian Hill, Jaclyn Smith, Gans Srinivasa, Kemal Sonmez, Ashish Sirasao, Amit Gupta ir Madhubanti Mukherjee. Precision medicine and fpga technology: challenges and opportunities. *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, p. 655–658. IEEE, 2017.

[YM15] Gangjoon Yoon ir Chohong Min. Analyses on the finite difference method by gibou et al. for poisson equation. *Journal of Computational Physics*, 280:184–194, 2015.

[YWB+19] Ming Yang, Shige Wang, Joshua Bakita, Thanh Vu, F Donelson Smith, James H Anderson ir Jan-Michael Frahm. Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: addressing an industrial challenge. *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, p. 305–317. IEEE, 2019.

[YZP08] Zhiyi Yang, Yating Zhu ir Yong Pu. Parallel image processing based on cuda. *2008 International Conference on Computer Science and Software Engineering*, tom. 3, p. 198–201. IEEE, 2008.

[Kad85] Leo P Kadanoff. Simulating hydrodynamics: a pedestrian model. *Journal of statistical physics*, 39(3):267–283, 1985.

[MC07] Eric Monmasson ir Marcian N Cirstea. Fpga design methodology for industrial control systems—a review. *IEEE transactions on industrial electronics*, 54(4):1824–1842, 2007.

[NSS+16] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan ir Debbie Marr. Accelerating recurrent neural networks in analytics servers: comparison of fpga, cpu, gpu, and asic. *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, p. 1–4. IEEE, 2016.

[RM16] Christopher Root ir Todd Mostak. Mapd: a gpu-powered big data analytics and visualization platform. *ACM SIGGRAPH 2016 Talks*, p. 1–2. 2016.

[SFJ+19] Rym Skhiri, Virginie Fresse, Jean Paul Jamont, Benoit Suffran ir Jihene Malek. From fpga to support cloud to cloud of fpga: state of the art. *International Journal of Reconfigurable Computing*, 2019, 2019.

[VDM21] Stepan Vyazigin, Anuar Dyusembaev ir Madina Mansurova. Emulation of x86 computer on fpga. *2020 IEEE 8th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, p. 1–6. IEEE, 2021.

[vKBB+16] Jóakim von Kistowski, Hansfried Block, John Beckett, Cloyce Spradling, Klaus-Dieter Lange ir Samuel Kounev. Variations in cpu power consumption. *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, p. 147–158, 2016.