https://doi.org/10.15388/vu.thesis.69
https://orcid.org/0000-0002-3760-6343

VILNIAUS UNIVERSITETAS

Mantas
VAITONIS

Didelio dažnio kompiuterizuotų prekybos strategijų inžinerija finansinėse rinkose

DAKTARO DISERTACIJA

Technologijos mokslai,
Informatikos inžinerija T 007

VILNIUS 2020

Disertacija rengta 2015-2019 metais Vilniaus universitete.

Mokslinis vadovas
doc. dr. Saulius Masteika (Vilniaus universitetas, technologijos mokslai, informatikos inžinerija – T 007).

SANTRAUKA

Elektroninėse vertybinių popierių biržose prekyba atliekama investuotojų ir informacinių technologijų specialistų kuriamais ir automatizuotais prekybiniais algoritmais. Pastaraisiais metais algoritmų kompiuterizavimas ir būtinybė būti greitesniam už konkurentus, atliekant investicinius sprendimus bei prekybinius sandorius, dienos prekybą perkėlė į nanosekundinių sandorių dažnį. Tobulėjant elektroninių biržų informacinių technologijų infrastruktūrai, greitėjant sandorių atlikimui bei spartėjant atliekamų sandorių dažniui, prekybos finansų rinkų veikla labiau susiduria su informatikos inžinerijos nei finansinių strategijų kūrimo problematika ir uždaviniais. Tampa aktuali ir būtina tiriamoji mokslinės eksperimentinės plėtros veikla, taikant informatikos inžinerijos žinias finansų sektoriaus dalykinėje srityje, kuriant, testuojant ir pasiūlant inovatyvius sprendimus, tinkamus spręsti didelio dažnio prekybos problematiką ir uždavinius. Šiuo darbu siekiama sukurti, pritaikyti ir adaptuoti didelio dažnio statistinio arbitražo prekybos algoritmo testavimo metodą, galintį analizuoti realaus laiko nanosekundžių tikslumo duomenų, gaunamų iš elektroninių biržų, srautą. Siekiant išsiaiškinti, kaip veikia didelio dažnio prekyba elektroninėse biržose, darbe nagrinėti jų šaltiniai ir elektroninės prekybos mokslo raida. Atlikta literatūros analizė atskleidė, kad trūksta ištestuoto formalizuoto metodo ir technologinio sprendimo, leidžiančio testuoti algoritmines didelio dažnio prekybos – DDP strategijas finansų rinkose. DDP strategijų testavimą numato Europos Sąjungos MiFID II direktyva, kuria siekiama DDP prekybos operatorius įpareigoti testuoti naudojamas algoritmines ir DDP strategijas. Nustatyta, kad nėra sprendimo, kaip testavimą formalizuoti ir atlikti technologiškai, todėl reikia sukurti DDP strategijų testavimo metodą. Siekiant įgyvendinti išsikeltą užduotį, buvo sukurtas DDP statistinio arbitražo strategijų testavimo algoritmo prototipas, kuris remiasi pasiūlytu metodu. Atlikus testavimą, buvo įrodyta, kad taikomas metodas, kuris naudoja GPU atmintį, kodo vektorizavimą, lygiagrečius skaičiavimus, daugiamates matricas ir lygiagrečius branduolius, pasiekia užsibrėžtą tikslą ir geba atlikti duomenų analizę bei priimti prekybos sprendimą greičiau nei atkeliauja nauji duomenys iš elektroninės biržos.

Reikšminiai žodžiai: didelio našumo skaičiavimai, didelio dažnio prekyba, algoritminė prekyba, GPU, daugiamatės matricos, statistinis arbitražas, CUDA.

ABSTRACT

Today people are almost becoming absolute when talking about trading in electronic markets, due to fact that now almost all trading is done by the trading algorithms. Since trading is done with the help of computers the speed of trading decision and placing orders has grown from daily trading to microsecond and even nanosecond time precision. Based on the speed that is necessary to make orders, which helps to beat other market members, became one of the most essential parts of any trading algorithm. New technological advancements with high performance computing in electronical markets and the need for fast trading decision making leads the market members to high frequency trading. The analysis of the literature showed the lack of a tested formalized method and technological solution that allows testing algorithmic high-frequency trading-HFT strategies in financial markets. Testing of HFT strategies is also provided in the European Union's MiFID II Directive, which aims to oblige HFT trading operators to test all trading algorithms and HFT strategies for possible errors. Also it enables the supervisory authorities to verify the algorithms as well. It was found that there is no formalize and technologically perform testing, thus it is necessary to develop a testing method for HFT strategies. In this paper we wished to create and implement the testing method for the automated high frequency statistical arbitrage trading that would be able to analyse big amount of tick-by-tick data received from electronic market in nanosecond time-stamp precision. The initial motivation of this paper was to prepare testing method for automated HFT statistical arbitrage trading that would work with big amount of high frequency data and suggest the hardware for this type of system. To achieve the above mentioned aim the HFT statistical arbitrage testing algorithm prototype was created based on the proposed method. The proposed method demonstrates how the use of the GPU memory, code vectorization, parallel calculations and multidimensional arrays brings impressive speedups in the HFT trading and analyzation of the high frequency data.

Key words: high performance computing, high frequency trading, algorithmic trading, GPU, multidimensional matrices, statistical arbitrage, CUDA.

ŽYMĖJIMAI IR SANTRUMPOS

- **CUDA** - (angl. Compute unified device architecture) technologija leidžia panaudoti GPU.
- **GPU** - (angl. Graphic processing unit) programuojamas loginis lustas, skirtas grafinių vaizdų atvaizdavimo funkcijoms.
- **DDP** – didelio dažnio prekyba.
- **DDD** – didelio dažnio duomenys.
- **CPU** - (angl. Central processing unit) atsakingas už įrenginių bei programų komandų analizavimą bei vykdymą.
- **FPGA** - (angl. Field-programmable gate array) programuojama loginė matrica.
- **ALU** – (angl. Arithmetic logic unit) aritmetinis loginis vienetas.
- **SIMD** – (angl. Single instruction, multiple data) viena instrukcija, daug duomenų, tai yra lygiagrečių kompiuterių klasė.
- **SISD** – (angl. Single instruction stream, single data stream) viena instrukcija, vienas duomenų šaltinis, tai yra kompiuterių architektūrą.
- **SM** – (angl. Streaming Multiprocessors) srautiniai daugiaprocesoriai, yra GPU CUDA dalis.
- **SP** – (angl. Streaming processor) srautiniai procesoriai, yra GPU CUDA dalis.
- **FPU** – (angl. Floating point unit) slankiojo kablelio vienetas, dar žinomas kaip matematinis koprocesorius arba skaitinis koprocesorius.
- **FI** – finansų inžinerija.
- **NMS** – (angl. National Market System) nacionalinės rinkos sistemos reglamentas.
- **TDP** – (angl. Temporal data partitioning) laikinas duomenų padalinimas yra lygiagretaus programavimo atskyrimo strategija.
- **SDP** – (angl. Spatial data partitioning) erdvinis duomenų padalinimas yra lygiagretaus programavimo atskyrimo strategija.
- **DOT** – (angl. Designated order turnaround) elektroninė sistema, kuri padidina prekybos platformos efektyvumą.
- **ADF** – (angl. Agmented Dickey Fuller test) išplėstinis Dickey Fuller testas.
- **FLOPS** – (angl. Floating point operations per second) tai kompiuterio našumo matas.

- **P(i,t)** – yra reali ateities sandorio kaina.
- **P()** – nurodo tikimybę.
- ε_t – J. Caldeira ir G. V. Moura pasiūlytoje strategijoje esantis kriterijus, kuris yra skirtumas tarp rastos finansinių instrumentų normalizuotų kainų poros.
- ***p(i,t)*** – yra normalizuota ateities sandorio kaina.
- z_t – J. Caldeira ir G. V. Moura pasiūlytoje strategijoje esantis kriterijus, kuriuo vadovaujantis nustatomi prekybos signalai.
- ***PL*** – prekybos strategijos pelnas/nuostolis iš pirkimo pozicijų.
- ***PS*** – prekybos strategijos pelnas/nuostolis iš pardavimo pozicijų.
- ***TP*** – prekybos strategijos bendras pelnas/nuostolis.
- $\mathcal{Y}_t$ – duomenų eilutė (vektorius).
- ***S*** – Šarpo rodiklis.
- $\mathbf{S_{i,j}}$ – kvadratinis normalizuotų kainų skirtumas.
- μ_{PnL} – pelno/nuostolio vidurkis.
- σ_{PnL} – pelno/nuostolio standartinis nuokrypis.
- $\mathbf{H_0}$ – nulinė hipotezė.
- **μ** – vidurkis.
- **σ** – standartinis nuokrypis.
- **i** – nurodo ateities sandorį.
- **j** – nurodo ateities sandorį.
- **t** – nurodo laiko eilutę.
- **ε** – vidurkio nulinė klaida.
- **ls** – finansinio instrumento kaina, kai jis buvo nupirktas.
- **lb** – finansinio instrumento kaina, kai jis buvo parduotas.
- **d** – M. S. Perlini darbe esantis kriterijus, pagal kurį nustatomi prekybos signalai.

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

TURINYS

ĮVADAS

Nuolat kyla reikalavimai įvairių kompiuterizuotų mokslo sričių (fizikos, matematikos, finansų ir kt.) skaičiavimo galiai. Didėjantys lygiagrečios architektūros poreikiai, kurie susiję su išaugusių skaičiavimų kiekiu, daugiabranduoliniai centriniai procesoriai (angl. *central processing unit, CPU*) ir grafikos apdorojimo įrenginiai (angl. *graphics processing unit, GPU*), tampa ne alternatyva, bet lygiagretaus programavimo būtinybe. Lygiagretus programavimas yra viena iš svarbiausių temų, kai kalbama apie programinę įrangą, leidžiančią vienu metu atlikti kelis ir daugiau skaičiavimų.

GPU tapo vienu iš pagrindinių elementų, leidžiančių atlikti lygiagrečias užduotis nuo mašinų mokymosi iki molekulinės dinamikos ir didelio dažnio prekybos (DDP) finansų rinkose. Tačiau pasiekti didelį programinės įrangos sistemų GPU našumą vis dar nėra lengva užduotis. Taip yra todėl, kad GPU neturi programinės įrangos, kuri reguliuotų duomenų perdavimą visoje sistemoje, ją nesudėtingai valdant per įvestį / išvestį (angl. *input / output, I / O*). Todėl daliai aplikacijų, kurios galėtų pasinaudoti GPU lygiagretinimu, reikia nemažų išlaidų ir žinojimo, kaip jomis naudotis, siekiant panaudoti GPU potencialą (Sangman K. et al., 2014, Nvidia, 2014, Foley D. ir Danskin J., 2017, Nvidia, 2019).

Viena iš sričių, kuri gali pasinaudoti lygiagrečiu programavimu ir skaičiavimais, yra DDP. Kompiuterinės infrastruktūros ir technologijų plėtra (Capgemini, 2012, Maureen O., 2015, Mackintosh P., 2019) padidino sandorių skaičių elektroninėse biržose, generuodama didelius finansinių duomenų kiekius (Dacorogna M. M., 2001, Mackintosh P., 2019, Virgilio G. P. M., 2019). Finansų rinkos pasikeitė įvairiais būdais, kurie atspindi technologinę pažangą ir reguliacinius pokyčius. 1980 metų pradžioje sukurtos prekybos programos ir prekyba buvo pradėta vykdyti kompiuteriais, mažinant ją atviros salės tipo biržose (Hasbrouck J. et al., 1993). Atsiradus kitoms, visiškai elektroninėms, prekybos vietoms, ypač elektroninių ryšių tinklams (ECN), dešimtojo dešimtmečio pabaigoje dar labiau pasikeitė prekyba NASDAQ ir Niujorko vertybinių popierių biržoje (NYSE) (Weston J. P., 2002). Šie veiksniai paskatino įdiegti pirmąją Niujorko vertybinių popierių biržos elektroninę sistemą DOT (angl. *Designed Order Turnaround*) (Kauffman R. J. et al., 2015). Dėl šių priežasčių rinkos dalyviams reikalavimai IT intensyvėja ir tampa būtinybe ieškant

optimaliausių technologinių sprendimų. NMS reglamentas (angl. *National Market System*) paskatino konkurenciją tarp prekybos vietų JAV; MiFID direktyva Europoje lėmė įvairesnę finansinę ekosistemą, dėl kurios pagerėjo rinkos kokybė (Gresse C., 2011, OHara M. ir Ye M., 2011). Tai nulėmė, kad per pastaruosius metus didelio dažnio prekybos algoritmų tyrinėjimas tapo itin aktualiu kiekybiniu įrankiu, kurį pasitelkia analitikai (Aldrige I., 2010, Durbin M., 2010, Zubulake P. ir Lee S., 2011). DDP yra algoritminis būdas, kuris dėl naujausių technologijų (daugiabranduolinių sistemų FPGA, GPU, CPU; duomenų apdorojimo įrankių Hadoop, Spark, HDF ir kt.) (Milutinovic V. et al., 2017), finansų inžinerijos ir programavimo žinių leidžia pateikti *pirk / parduok* sandorius elektroninėms biržoms, taikant kiekybinius modelius (Mariano R. S. ir Kuen Tse Y., 2008, Lai T. L. ir Xing H., 2008, Kantz H. ir Schreiber T., 2004). Kompiuterizuotoms prekybos sistemoms reikia spartesnių greičio skaičiavimų sprendimų nei generuojami duomenys elektroninėse biržose, negalimi duomenų srauto ir signalų apdorojimo vėlavimai (Kaya O., 2016).

Finansų inžinerija (FI) yra daugiadalykinė sritis, apimanti finansų teoriją, inžinerijos metodus, matematines priemones ir programavimą. Ji taip pat apibrėžta kaip techninių metodų, ypač matematinių ir skaičiavimo, taikymas finansų praktikoje (Wang Z. ir Zheng W., 2014). Plačiąja prasme kiekvienas, kuris finansuose naudojasi techninėmis priemonėmis, gali būti vadinamas finansų inžinieriumi. Kartais šis terminas susiaurinamas, kad apimtų tik sukurtus naujus finansinius produktus ir prekybos strategijas. Finansų inžinerijoje mokslo netyrinėta, bei technologiškai sudėtingiausia dalis siejama su informacinių technologijų integravimu į vertybinių popierių biržas, prekybinius algoritmus, investicinių sprendimų paramos sistemas. Sudėtingumas ypač išauga adaptuojant sistemas didelio dažnio prekybai (Wang Z. ir Zheng W., 2014, Lyuu Y., 2001). Šiame darbe finansų inžinerija taikoma formalizuojant didelio dažnio statistinio arbitražo prekybos strategijų algoritmus, sprendžiant didelio dažnio algoritmų skaičiavimo uždavinius. Tyrime naudojamas terminas „strategija“ apima techninės finansų rinkų analizės strategijos ir algoritminės prekybos IT inžinerijos derinimą. Siekiama apibrėžti strategijos erdvę – tam tikro strategijos tipo rinkinį, kuris perkeliamas į didelio dažnio prekybos algoritmą.

MiFID II direktyvoje nurodomos algoritminės prekybos finansinėmis priemonėmis taisyklės. Rinkos dalyviai privalo užtikrinti veiksmingą kontrolę, taisyklių palaipsnį adaptavimą savo investicinėje veikloje. Vienas žinomiausių reikalavimų yra sandorių stabdymo užtikrinimas, kad būtų

sustabdyta prekyba, jei kainų svyravimas tampa per didelis, viršijantis dienos prekybos ribas. Naudojami algoritmai turi būti išbandyti, sudarytos sąlygos priežiūros institucijoms juos patikrinti, patvirtinti jų teisėtumą, atitikimą reikalavimams. Pastebėtina, kad MiFID II nėra apibrėžta, kaip testuojamos algoritminės prekybos strategijos (Busch D., 2016). Atsižvelgiant į poreikį patikimus DDP strategijų testavimo metodus išbandyti naujoje, santykinai mažai ištirtoje ir daug duomenų turinčioje sudėtingoje sistemoje, darbe siekiama išsiaiškinti, ar į skaičiavimų lygiagretinimą orientuotas metodas yra tinkamas patikrinti didelio dažnio prekybos strategijų stabilumą ir tvirtumą, atsiradus nestabilumams rinkose. Nėra bendrai apibrėžta, kuris didelės apimties duomenų skaičiavimų lygiagretinimo metodas yra efektyvesnis ir pakankamas didelio dažnio prekyboje, todėl galėtų būti taikomas finansų tarpininkų ar prekiautojų kaip algoritminės prekybos (AP) ir DDP testavimo priemonė.

Įsigilinus į tematiką, atlikus literatūros analizę, identifikuota pagrindinė problema – neformalizuoti DDP strategijos ir algoritmai, neištirtas jų taikymo efektyvumas, nėra siūlomas testavimo metodas, taikantis technologinius sprendimus. Reikalingas kompiuterizuotas metodas, pagrįstas informatikos inžinerijos žiniomis, leidžiantis rinkos dalyviams ir finansų rinkų priežiūros institucijoms atlikti DDP algoritmų testavimus. Nustatyta ir tai, kad nėra vieningos nuomonės dėl DDP strategijų poveikio rinkoms, likvidumui (Bershova N. ir Rakhlin D., 2013, Morelli M., 2017, Linton O. ir Mahmoodzadeh S., 2018, Virgilio G. P. M., 2019), nėra metodo didelio dažnio prekybinės veiklos keliamoms rizikoms testuoti (Bush D., 2016). Dėl to, kad nėra metodo, neįmanoma detaliai tikrinti DDP strategijų naudos ar žalos rinkoms (Bush D., 2016). Būtina sukurti testavimo metodą, jungiantį skirtingus skaičiavimų lygiagretinimo metodus, kuriuo būtų galima testuoti DDP.

Mokslinių tyrimų ir eksperimentinių testavimų metu siekiama išsiaiškinti, ar, taikant kodo vektorizavimą, daugiamatės matricos ir branduolių lygiagretinimą, GPU aplinkoje padidės prekybos sprendimų priėmimo greitis, bus sukurtas metodas, skirtas testuoti DDP strategijas, patobulinti realaus laiko analitiką.

Darbo objektas

Šio darbo tyrimo objektas – didelio dažnio statistinio arbitražo algoritminės prekybos sistemos ir didelio dažnio duomenys elektroninėse finansų biržose.

Darbo tikslas ir uždaviniai

Darbo tikslas – sukurti didelio dažnio statistinio arbitražo prekybos testavimo metodą, priimantį algoritminės prekybos sprendimus greičiau, nei generuojami nauji duomenys elektroninėse biržose.

Darbo tikslui pasiekti iškelti uždaviniai:

1. Formalizuoti didelio dažnio algoritmines strategijas, nustatant technologines specifikacijas didelio dažnio duomenų apdorojimui, ištestuoti statistinio arbitražo DDP strategijų efektyvumą, pritaikyti įgytas žinias šio darbo tikslui pasiekti.
2. Pasiūlyti ir sukurti DDP testavimo ir duomenų apdorojimo realiu laiku metodą, kuris dirbtų su didelio dažnio duomenimis ir atliktų algoritminės prekybos skaičiavimus sparčiau nei gaunami didelio dažnio duomenys iš elektroninių biržų.
3. Ištestuoti sukurtą metodą, sukuriant DDP testavimo įrankio prototipą, leidžiantį atlikti skaičiavimus su didelio dažnio ir apimties duomenimis.
4. Aprašyti gautus rezultatus ir pastebėjimus, įvertinti sukurto didelio dažnio statistinio arbitražo prekybos algoritmų testavimo metodo taikymo galimybes ir galimus apribojimus.

Tyrimo metodika

Rengiant darbą, remtasi moksline literatūra, moksliniais straipsniais, šaltiniais internete. Darbe taikomi metodai: mokslinės literatūros analizė ir apibendrinimas, stebėjimas, skaitmeninis modeliavimas, atrankos metodas, sintezė, eksperimentai, koreliacinė analizė, statistinė analizė, kompiuterinis duomenų apdorojimas.

Sistemine darbų ir analitine tyrimų apžvalga siekiama išanalizuoti esamas didelio dažnio prekybos sistemas ir technologijas, išsiaiškinti jų taikymo metodus; taip pat siekiama teoriškai pagrįsti kuriamo didelio dažnio statistinio arbitražo prekybos algoritmo ir didelio dažnio duomenų apdorojimo metodo poreikį.

Teorinėje tyrimo dalyje aiškinamas sukurto modelio poreikis, leidžiantis įgyvendinti išsikeltus tikslus, taip pat aprašoma platformos architektūra. Toliau pateikiamas teorinis didelio dažnio statistinio arbitražo prekybos algoritmo prototipas ir didelio dažnio duomenų apdorojimo metodas, nagrinėjamas jų teorinis ir praktinis tinkamumas, pristatomas tyrimo eksperimentas, jame naudojami duomenys ir nurodomos nuostatos, pagal kokius kriterijus bus vertinamas eksperimentas.

Po teorinės dalies aptariamas atliktas eksperimentas, vertinamas sukurtas prototipas, pasirinkta platformos architektūra, argumentuojamas pasirinktas didelio dažnio statistinio arbitražo prekybos sistemos modelis.

Mokslinis naujumas

Nors didelio dažnio prekyba yra neretai aptariama verslo publikacijose, tačiau jos informatikos inžinerijos aspektas mažai nagrinėjamas. Labai mažai informacijos yra apie didelio dažnio prekybos sistemų taikymą (limitacijas ir rizikas) bei optimalius technologinius reikalavimus. Daugumoje mokslinių straipsnių, susijusių su didelio dažnio prekyba, tyrinėjamos optimalių strategijų parinkimo metodologijos, nusakančios, kaip matuoti didelio dažnio prekybos strategijų pelningumą, kokie yra optimalūs prekybos strategijos naudojami parametrai, kada pateikti sandorių informaciją elektroninėms biržoms, kaip matuoti didelio dažnio prekybos aktyvumą, taip pat tai, kas nėra susiję su optimalia didelio dažnio prekybos sistemų konfigūracija, architektūra, metodu, kaip taikyti didelio dažnio prekybos sistemas pasirinktai technologiniai platformai (Vaitonis M., 2018, Kearns M. et al. 2010, Anane M. ir Abergel F., 2014, Beckhardt B. et al., 2016). Todėl svarbu detaliau išnagrinėti didelio dažnio prekybos sistemas, jų taikymą, architektūrą ir galimus inžinerinius sprendimus, kurie padėtų įgyvendinti didelio dažnio prekybą dirbant su didelio dažnio ir apimties duomenimis.

Norint naudoti didelio dažnio prekybos algoritmus, reikia įvertinti, keliose biržose bus prekiaujama, kiek finansinių instrumentų ir skirtingų prekybos strategijų bus naudojama, koks bus naudojamų duomenų dažnumas (sekundiniai, milisekundiniai ir t. t.), koks bus lango, kuriame vyks prekyba, dydis. Priklausomai nuo pasirinktos konfigūracijos, gali paaiškėti, kad, norint įgyvendinti didelio dažnio prekybą, teks lygiagrečiai atlikti tūkstančius skaičiavimų.

Tyrimo metu naudojant statistinio arbitražo strategijas, reikia rasti koreliuotas finansinių instrumentų poras. Tai atlikti, naudojantis didelio dažnio duomenimis, tampa labai sudėtinga. Taip nutinka, nes skirtingi finansiniai instrumentai rinkose juda skirtingu dažnumu, todėl, norint palyginti didelio dažnio prekybos duomenis, juos reikia agreguoti. Dėl prieš tai jau minėtų priežasčių, pereinant į didelio dažnio prekybą su didelio dažnio duomenimis, prekyba vykdoma ne tam tikru laiku, bet pagal tai, kiek būna gaunama *tick-by-tick* duomenų. Todėl algoritmai seka ne paprastą laiką, bet pasirinkto GPU procesorių ciklų skaičių, t. y. prekybos langas yra ne tam tikras laikas, bet gautų duomenų kiekis ir procesoriaus ciklų skaičius.

Darbe iškelta ir patvirtinta hipotezė – kodo vektorizacijos naudojimas, kai statistinio arbitražo DDP algoritmas perkeliamas į GPU, algoritmo duomenys formuojami kaip daugiamatės matricos ir algoritmo skaičiavimai, vykdomi lygiagrečiai išskaidant juos tarp atskirų lygiagrečių branduolių (angl. *kernel*), leidžia priimti didelio dažnio prekybos sprendimus greičiau nei gaunamas didelio dažnio duomenų srautas iš elektroninių biržų.

Sukurtas didelio dažnio statistinio arbitražo prekybos testavimo metodas, kuriuo šio tipo prekybos algoritmas perkeltas į GPU aplinką, sujungus kodo vektorizaciją, daugiamates matricas ir branduolių lygiagretinimą, yra pakankamas priimti prekybinius sprendimus.

Darbo rezultatų praktinė vertė

Disertacijoje pasiūlyta technologija, būdas ir jais grįstas sukurtas metodas, kuris leidžia apdoroti didelio dažnio ir apimties duomenis didelio dažnio prekybos strategijose greičiau nei gaunami nauji duomenys iš elektroninių biržų. Tai leidžia aplenkti kitus rinkos dalyvius. Sprendimo esmė – ne tik greitas duomenų apdorojimas, bet ir greitas teisingo prekybos sprendimo priėmimas. Atsižvelgus į tai, kad dėl komercializacijos visi didelio dažnio prekybų sistemų sprendimai yra konfidencialūs, būtina moksliškai aprašyti jų veikimo principą ir parodyti, kaip šiose sistemose taikomi metodai gali padėti kitose mokslo srityse. Disertacijoje pasiūlytas metodas gali būti taikomas ne tik realiai prekybai, bet gali būti naudojamas kaip DD statistinio arbitražo strategijų testavimo metodas. Šio metodo reikia dėl MiFID II direktyvos, kuri reikalauja iš prekybos operatorių tikrinti visas algoritminės ir didelio dažnio prekybos sistemas, kurios veikia pas juos.

Atliktų tyrimų pagrindu gali būti kuriama sistema, kuri leistų, naudojantis GPU, atlikti didelio dažnio skaičiavimus. Šie skaičiavimai gali būti pritaikomi didelio dažnio prekyboje siekiant pelno arba elektroninėse biržose didinant likvidumą. Taip pat tie patys metodai gali būti pritaikomi kitose mokslo srityse, kuriose reikia greitai ir teisingai priimti sprendimus, pavyzdžiui dirbtinio intelekto srityje.

Ginamieji teiginiai

Disertacijos ginamieji teiginiai:

1. Didelio dažnio statistinio arbitražo prekybos strategijų formalizavimo metodologija, kuria remiantis galima nustatyti optimalų duomenų normalizavimo ir prekybos langą algoritminėje prekyboje mili ir nanosekundiniu dažniu, yra tinkama įvertinti strategijų efektyvumą.

2. DDP priimami sprendimai, naudojant didesnio dažnio duomenis, yra efektyvesni algoritminėje prekyboje ir nulemia skaičiavimų lygiagretinimo būtinybę.

3. Sujungus kodo vektorizaciją, daugiamates matricas, branduolių lygiagretinimą ir perkeliant formalizuotus algoritmus į GPU, pasiekiama duomenų apdorojimo sparta yra didesnė, nei generuojamas didelio dažnio duomenų srautas elektroninėse biržose.

Darbo aprobavimas

Disertacijos rezultatai pristatyti aštuoniose mokslinėse konferencijose:

1. 2015 metais BIS 2015 tarptautinėje mokslinėje konferencijoje „Business Information Systems Workshops“ Poznanėje, Lenkijoje skaitytas pranešimas „Quantitative Research in High Frequency Trading for Natural Gas Futures Market“.
2. 2015-12-03 – 2015-12-05 7-ojoje mokslinėje konferencijoje DAMSS 2015 „Duomenų analizės metodai programų sistemoms“ Druskininkuose pristatytas stendinis pranešimas „High frequency statistical arbitrage strategy engineering and algorithm for pairs trading selection“.
3. 2016-10-13 – 2016-10-15 ICIST 2016 22-ojoje „Tarptautinėje informacijos ir programų technologijų konferencijoje“ Druskininkuose skaitytas pranešimas „Research in high frequency trading and pairs selection algorithm with Baltic region stocks“.
4. 2016-12-01 – 2016-12-03 8-ojoje mokslinėje konferencijoje DAMSS 2016 „Duomenų analizės metodai programų sistemoms“ Druskininkuose pristatytas stendinis pranešimas „Computerized high frequency trading of nanoseconds in futures market“.
5. 2017-10-12 – 2017-10-14 ICIST 2017 23-ojoje „Tarptautinėje informacijos ir programų technologijų konferencijoje“ Druskininkuose skaitytas pranešimas „Statistical Arbitrage Trading Strategy in Commodity Futures Market with the Use of Nanoseconds Historical Data“.
6. 2017-11-30 – 2017-12-02 9-oje mokslinėje konferencijoje DAMSS 2017 „Duomenų analizės metodai programų sistemoms“, Druskininkuose, skaitytas pranešimas „Research in High Frequency Statistical Arbitrage Strategies Applied to Microsecond and Nanosecond Information“.
7. 2018-04-27 IVUS 2018 23-ojoje tarptautinėje mokslinėje konferencijoje Kaune „Information Society and University Studies“ skaitytas

pranešimas „CPU and GPU Implementations for High Frequency Trading in Algorithmic Finance“.

8. 2018-05-29 SYSTEM 2018 23-ojoje tarptautinėje mokslinėje konferencijoje Gliwice, Lenkijoje „Information Society and University Studies“ skaitytas pranešimas „Algorithmic trading and machine learning based on GPU“.

Disertacijos rezultatai pateikti šešiose mokslinėse publikacijose:

1. Masteika S., Vaitonis M., Quantitative Research in High Frequency Trading for Natural Gas Futures Market, Business Information Systems Workshops, Springer International Publishing, Vol. 228, pp. 29 – 35, 2015 m.
2. Vaitonis M., Masteika S. (2016). High frequency statistical arbitrage strategy engineering and algorithm for pairs trading selection. 7th International Workshop on Data Analysis Methods for Software Systems [abstracts book], Druskininkai, Lithuania, December 3 – 5, 2015. ISBN 978-9986-680-58-1. pp. 51.
3. Vaitonis M., Masteika S. (2016). Research in high frequency trading and pairs selection algorithm with Baltic region stocks. Information and Software Technologies. 22nd International Conference, ICIST 2016, Druskininkai, Lithuania, October 13 – 15, 2016, Proceedings. ISBN 978-3-319-46254-7, pp. 208 – 217.
4. Vaitonis M., (2017). Pairs Trading Using HFT in OMX Baltic Market. Baltic J. Modern Computing, Vol. 5(2017), No. 1, pp. 37 – 49.
5. Vaitonis M., Masteika S. (2017) „Statistical Arbitrage Trading Strategy in Commodity Futures Market with the Use of Nanoseconds Historical Data“, Information and Software Technologies: 23rd International Conference, ICIST 2017, Druskininkai, Lithuania, October 12 – 14, 2017, Proceedings. R. Damaševičius and V. Mikašytė (Eds.): ICIST 2017, CCIS 756, pp. 303 – 313, ISBN 978-3-319-67642-5.
6. Vaitonis M., Masteika S. (2018). Experimental Comparison of HFT Pair Trading Strategies using Microsecond and Nanosecond Future Commodity Contracts Data. Baltic J. Modern Computing, Vol. 6(2018), No. 2, pp. 195 – 216, ISSN 2255-8950.

Darbo apimtis ir struktūra

Disertaciją sudaro šeši skyriai ir literatūros sąrašas. Disertacijos skyriai: Įvadas, Didelio dažnio prekyba ir didelio našumo kompiuterija finansuose,

Tyrimui naudotų strategijų formalizavimas, Siūloma didelio dažnio prekybos strategijų testavimo metodologija, Reikalavimai tyrimui ir Bendros išvados.

Įvade aptariama darbo motyvacija, pateikiami išsikelti uždaviniai; nurodomas mokslinis aktualumas ir kt. Toliau aptariama algoritminė, didelio dažnio ir statistinio arbitražo prekyba, aiškinamasi, kaip jos veikia ir kada naudojamos. Antrajame skyriuje nagrinėjama tai, kaip taikoma didelio dažnio prekyba didelio našumo kompiuterijoje, ir kokios technologijos naudojamos norint pasiekti tokio dažnio prekybą. Pasirinkus GPU CUDA technologiją tolesniam tyrimui, trečiajame skyriuje analizuojamos statistinio arbitražo strategijos, kurios bus naudojamos tyrime. Ketvirtame skyriuje pateikiamas didelio dažnio statistinio arbitražo prekybos metodas, kuriame pasitelkiama GPU CUDA, kodo vektorizavimas, daugiamatės matricos ir branduolių lygiagretinimas. Kituose skyriuose aptariami tyrimo reikalavimai, nurodomi naudojami didelio dažnio duomenys ir pateikiami tyrimo rezultatai. Jie apibendrinami paskutiniajame skyriuje – Bendrose išvadose.

1 ALGORITMINĖ PREKYBA

Dabar rinkos yra sudarytos elektroninėje erdvėje su *pirk / parduok* eilėmis, kurios yra tvarkomos kompiuteriais, pasitelkiant prekybos algoritmus, naudojančius fundamentalią ir techninę analizę, siekiant priimti prekybos sprendimus.

1992 metais NASDAQ Niujorke pristatė FIX protokolą, skirtą pateikti sandorius. Vėliau jis buvo priimtas dar kelių biržų ir dabar šis protokolas tapo informacijos perdavimo standartu globalioje akcijų rinkoje. Naudojantis šiuo protokolu, galima atlikti visus sandorius elektroninėse biržose be žmogaus įsikišimo (Limaye S. S., 2014).

Vienas pagrindinių įnašų į elektroninę prekybą, vėliau ir didelio dažnio prekybą yra sumažintas biržos ir prekybos komunikavimo ir prekybos sprendimų priėmimo laikas. Seniau biržose prekybos sprendimus priimdavo brokeris, t. y. žmogus, kuris telefonu atlikdavo visus sandorius. Tokio tipo prekyba dažnai nukentėdavo dėl žmogiškojo faktoriaus klaidų ir užvilkintų sandorių. Todėl tapo aišku, kad tokio tipo sistema turi būti pakeista tada, kai tik pasitaikys toks standartas, kuris galės ją pakeisti. Toks standartas atsirado. Jis vadinamas paskirtu pavedimu (angl. *Designated Order Turnaround, DOT*). Pirmoji elektroninė sprendimų priėmimo sistema buvo pristatyta Niujorko akcijų biržoje (Leinweber, 2007). Tačiau pirmas sisteminis prekybos mechanizmas atsirado dar 1990 metais ir, remiantis *O'Hara* ir *Goodhart*, jo reikėjo taip ilgai laukti dėl dviejų pagrindinių priežasčių. Buvo labai mažai elektroninių sandorių, kurie tuo metu nebuvo populiarūs, ir asmeniniai kompiuteriai buvo labai brangūs. Pirmąjį silikono tranzistorių 1954 metais BELL laboratorijose pagamino Morris Tanenbaum. Šis išradimas reiškė esminį pokytį gaminant asmeninius kompiuterius. 1990 metų pabaigoje jau daug žmonių galėjo sau leisti turėti asmeninį kompiuterį namuose. Tuo metu jau buvo galima matyti hibridines komunikacines struktūras, skirtas komunikuoti elektroninėms biržoms ir toms, kuriose dirbo tik žmonės, pavyzdžiui, *Amerikos akcijų birža* (AMEX) (Aldrige, I., 2010).

Technologinės inovacijos pakeitė ne tik prekybos procesus, tačiau stipriai paveikė ir visą finansų rinką. I. Aldridge teigimu, būtinas išėjimas iš griežtai apibrėžtos hierarchinės struktūros, kuri egzistavo iki 1990 metų pabaigos. Tuo metu pagrindiniai rinkos veikėjai buvo brokeris ir pardavėjas, kurie gaudavo didžiausią komisinį mokestį. Dabar didžioji dalis rinkų tampa vis labiau decentralizuotos (Zubulake P. ir Lee S., 2011).

Algoritminė prekyba (automatinė prekyba, juodosios dėžės prekyba) yra procesas, kuriam pasitelkiami kompiuteriai su įdiegtomis specifinėmis programomis. Pagrindinis jų tikslas yra prekiauti skirtingose rinkose tokiu greičiu, kuris žmogui yra neįmanomas. Sukurtas algoritmas remiasi laiku, finansinių instrumentų kaina, skaičiumi arba matematiniais modeliais. Tokio tipo sistemos yra nuoseklesnės ir išvengia žmogiškojo faktoriaus, kuris prekiaujant remiasi emocijomis, o ne sudarytomis taisyklėmis (Jay Desai et al., 2012).

Algoritminės prekybos privalumai:

• užsakymų pateikimas geriausiomis kainomis;

• greitas ir efektyvus užsakymo pateikimas;

• sumažinamos išlaidos už komisinius mokesčius;

• automatiškai tikrinama rinka įvykus mažiausiems pasikeitimams;

• galima testuoti algoritmą turint seną ir naują informaciją;

• sumažinama klaidingo užsakymo pateikimo, kuris gali įvykti dėl žmogiškojo faktoriaus, rizika (Sipilä M., 2013, Aldridge I., 2013).

Algoritmine prekyba naudojasi daug prekybos ir investicinių organizacijų, kurios užsiima skirtinga prekyba:

• vidutinio ir ilgo laiko investuotojai (pensijų, investiciniai fondai, draudimo kompanijos), kurie perka akcijas dideliais kiekiais ir ilgam laikotarpiui;

• trumpo laikotarpio investuotojai (rinkos kūrėjai, spekuliantai ir arbitražo naudotojai), kurie naudojasi automatinės prekybos sistemomis;

• sisteminiai investuotojai (rinkos sekėjai, porų prekybos naudotojai, rizikos fondai), kurie patys kuria prekybos sistemas ir jas pritaiko skirtingoms rinkoms (Zubulake P. ir Lee S., 2011).

Visos strategijos, kurios vykdo algoritminę prekybą, ieško rinkoje situacijų, kuriose galima išlošti iš padidėjusios finansinio instrumento kainos ar sumažėjusių išlaidų. Toliau pateikiamos kelios pagrindinės algoritminės prekybos strategijos:

- Rinką sekančios strategijos – dažniausiai naudojama strategija, kuri remiasi vidurkio ar labai pakitusia informacija, kuri susijusi su finansinio instrumento kaina ar kitu techniniu indikatoriumi. Šio tipo strategijos nesistengia nuspėti ateities kainų judėjimo, bet remiasi istorine vidurkio informacija ir pagal ją priima sprendimus (Sipilä M., 2013).
- Arbitražinės strategijos – tokio tipo strategijos stebi tą patį finansinį instrumentą skirtingose rinkose. Atsiradus prekybos signalui, perkama toje

rinkoje, kurioje yra mažesnė kaina ir tuo pačiu metu parduodama kitoje rinkoje, kurioje yra kaina didesnė (Sipilä M., 2013).

- Strategijos, kurios remiasi matematiniais modeliais; daug matematinių modelių gali būti pritaikyti algoritminėje prekyboje. Pavyzdžiui, delta (santykis, kuriuo galima palyginti turto kainos pokytį) – neutrali strategija, kuri leidžia investuotojui kompensuoti teigiamas ir neigiamas deltas taip, kad bendra portfelio delta būtų sumažinta iki nulio (Sipilä M., 2013).
- Prekybos intervalo strategijos. Šio tipo strategijos remiasi idėja, kad aukščiausios ir žemiausios akcijos kainos yra laikinos, todėl jos periodiškai turi grįžti prie vidutinės kainos. Taikant šią strategiją, reikia nustatyti kainų intervalą, kurį būtina stebėti, ir pradėti prekybą tada, kai pasiekiamos šio intervalo ribos (Sipilä M., 2013).
- Akcijų skaičiaus vidutinės kainos strategijos. Jos aplenkia sukurtą didelį užsakymą tam, kad pačios sukurtų daug smulkių užsakymų, kurių skaičius neviršytų nurodyto laikotarpio skaičiaus. Tikslas yra sukurti užsakymus atsižvelgiant į kainas, apskaičiuotas pagal parduotus jų skaičius, ir nustatant vidutinę kainą (Sipilä M., 2013).
- Akcijos laiko vidutinės kainos strategija. Ji aplenkia sukurtą didelį užsakymą tam, kad pati sukurtų daug smulkių užsakymų, naudojančių vienodai išdalintus laiko intervalus nusirodytam laikotarpiui. Šios strategijos tikslas yra sukurti kuo daugiau užsakymų, prilygstančių artimai vidutinei kainai per nurodytą laikotarpį (Sipilä M., 2013).
- Procentinio kiekio strategijos. Šiuo atveju iki tol, kol užsakymas yra visiškai įvykdytas, algoritmas siunčia dalinius nurodyto kiekio užsakymus investuotojo pateiktu dažniu. Ši „žingsnių“ strategija siunčia vartotojo nurodytus užsakymus, kurie apskaičiuojami pagal procentinį kiekį nuo rinkos, taip didindama arba mažindama pateikiamų užsakymų kiekį, kol pasiekiama norima kaina (Sipilä M., 2013).

1.1 Didelio dažnio prekyba

Kiekvienas, kuris susiduria su finansinėmis rinkomis, yra bent kartą girdėjęs terminą „didelio dažnio prekyba“ (DDP). Tačiau tvirtą, tikslų ir išsamų šio termino apibrėžimą rasti sunku. Tikslios apibrėžties nebuvimas neleidžia aiškiai ir paprastai atskirti DDP nuo kitų prekybos kategorijų, pavyzdžiui, nuo unitarinės ar „tuščiosios“ prekybos (Harris L., 2003). P. Zubulake ir S. Lee (2011) teigia, kad terminas buvo apibrėžtas daug kartų, tačiau nė vienas apibrėžimas nėra visiškai pripažįstamas DDP

bendruomenės. IOSCO (2011), tarptautinis vertybinių popierių standartizavimo centras, nurodo, kad apibrėžti DDP yra sunku ir nėra bendro sutarto apibrėžimo. IOSCO šį reiškinį apibrėžia taip: DDP susideda iš išmaniųjų ir naujausių technologijų naudojimo, pasitelkiant skirtingas strategijas nuo rinkos kūrimo iki arbitražo (Gregoriou G. N., 2015).

Apžvelgtoje literatūroje autoriai taip pat nenurodo tikslaus DDP apibrėžimo. Vietoje to jie nurodo funkcijas ir tai, kokios charakteristikos apibrėžia tokio tipo prekybą. Tačiau įvairūs autoriai dažnai nurodo skirtingas funkcijas ir charakteristikas, nors tarp jų ir galima rasti ryšių. IOSCO (2011) taip pat nurodė bruožus, būdingus DDP:

- yra kiekybinis ir naudoja algoritmus;
- turi didelę dienos portfelio apyvartą ir užsakymų su prekyba santykį;
- lygios arba artimos pozicijos yra pateikiamos kiekvienos prekybos dienos pabaigoje;
- prekybos pozicijos yra laikomos labai trumpą laikotarpį, kartais iki sekundės ar tik dalį sekundės;
- jas dažniausiai naudoja komercinės prekybos įmonės;
- labai jautri vėlavimui, nes vykdo itin daug sandorių (Gregoriou, G. N., 2015).

D. Easley, M. L. de Prado ir M. O'Hara (2013), kitaip traktuodami DDP, pateikia tokį charakteristikų sąrašą:

- DDP remiasi mikrostruktūra ir ieško, kaip išnaudoti rinkose atsirandančius netikslumus;
- remiasi strategijomis ir yra sukurtos išnaudoti nuspėjamus rinkos veiksmus;
- naudoja naujo tipo informaciją.

P. Zubulake ir S. Lee (2011) apibrėžė DDP tiksliau, kadangi įtraukė ir pagrindinius DDP žaidėjus, kurie yra reguliuojamos rinkos kūrėjai, statistinio arbitražo rizikos fondai, mažai vėluojantys brokeriai ir kliringo namai. Pateikiamas toks jų charakteristikos sąrašas, kuris apibrėžia DDP ir prekiaujančias įmones:

- pirmiausia – technologijos; visos DDP firmos daugiausiai dėmesio skiria technologiniams sprendimams;
- svarbu ne tik greitis; šios firmos privalo turėti patikimą, atsparią, nenuspėjamą prekybos struktūrą;
- naudojami prekybos modeliai ir strategijos yra pagrindinis skiriamasis šių įmonių bruožas, kuris lemia jų veiklos pelningumą; dėl šios priežasties

daug pinigų investuojama į žmogiškuosius išteklius – samdomi srities ekspertai ir geriausi doktorantai;

- istoriškai siauras profilis, kuriame šios įmonės prekiauja savo pinigais.

I. Aldridge (2010) išskiria kitus DDP bruožus, naudojamus apibrėžti mažo dažnio ir didelio dažnio prekybos skirtumus:

- didelė kapitalo apyvarta;
- dienos metu vykdomų pozicijų atidarymas ar uždarymas, jų nelaikant atidarytų per naktį;
- prekybai naudojami algoritmai;
- greiti kompiuterių siunčiami atsakymai biržoms, kurie keičia rinkos sąlygas;
- naudojamos strategijos generuoja didelį skaičių sandorių su mažą grąžą.

Žvelgiant nuodugniau, galima teigti, kad šioje knygoje išskiriami DDP ir algoritminės, sistemingos, elektroninės ir mažo vėlavimo prekybos skirtumai. Teigiama, kad DDP reiškia greitą prekiaujamo kapitalo perskirstymą ir apyvartą. Siekiant užtikrinti įmanomą ir patikimą perskirstymą, būtinas elektroninėje erdvėje veikiančių algoritmų naudojimas.

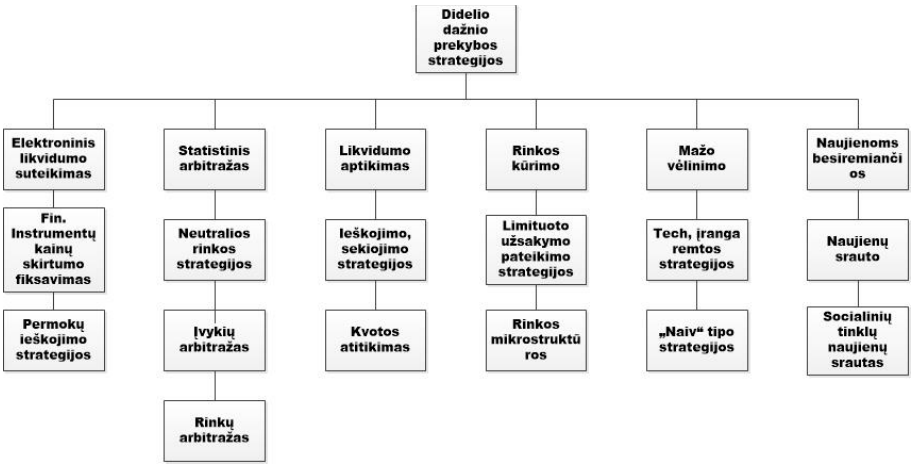
G. N. Gregoriou (2015) įžvelgia dar glaudesnį algoritminės prekybos ir DDP ryšį. Šiems dviem tipams būdingos struktūrinės ir operacinės charakteristikos, kurios yra bendros ir skirtingos. Bendras sudaro iš anksto parengtų prekybos strategijų naudojimas, automatinis sandorių sudarymas, joms nereikia žmogaus įsikišimo, pasitelkiamas tiesioginis ryšys su birža. Abu naudoja profesionalūs rinkos dalyviai, jie pasitelkia realaus laiko rinkos informaciją. Algoritminę prekybą išskiria tai, kad atidarytos pozicijos gali būti laikomos savaites ar net mėnesius. Jų tikslas – pasiekti iš anksto nustatytą prekybos pelną. Tik DDP būdinga: automatinis prekybos sprendimų priėmimas, tik tos įmonės taikomi prekybos metodai, labai daug sandorių, jie dažnai atšaukiami, trumpai laikomos atidarytos prekybos pozicijos, dėmesys skiriamas itin likvidiems instrumentams, reikalavimas mažai vėluoti, informacijai gauti naudojama kolokacija ir individualus ryšys.

Apibendrinant galima teigti, kad elektroninių algoritmų naudojimas yra būtinas DDP. Siekiant, kad visos naudojamos strategijos būtų efektyvios ir mažiausiai šališkos operaciniu atžvilgiu, jos turi būti įdiegtos į programinę įrangą, veikiančią kompiuteriuose, susietuose su elektroninėmis biržomis. Žvelgiant iš šios pozicijos, DDP yra labai naudingas kuriant technologines inovacijas (labai geras algoritmas nenaudingas, jei įdiegtas į silpną techninę

įrangą) ir investuojant į IT infrastruktūrą, pvz., projektas *Express by Hibernia Atlantic*.

Paminėtinas subtilus ir mažiau intuityvus bruožas – duomenys, kuriuos naudoja DDP. Įvairūs balansų dokumentai ar kita fundamentali informacija, kuri lėtai atsinaujina, nebus tinkama tokio tipo prekybai. Visi didelio dažnio prekybos algoritmai, kurie dirbtų tik su firmų dienos pabaigos kainomis, neturėtų pakankami didelio srauto duomenų, kad būtų efektyvūs. Dauguma DDP strategijų naudoja *tick-by-tick* duomenis, kurie yra susiję su pačiu prekybos procesu (Easley D. et al., 2013). Šie duomenys pasikeičia kiekvieną kartą, kai naujas *pirk / parduok* signalas, lemiantis itin didelį kiekį analizuotinų duomenų, nusiunčiamas į biržą.

Dar vienas bendras bruožas, apibrėžiantis DDP, yra didelė apyvarta, pozicijų, laikomų per naktį, nebuvimas ir trumpas atidarytų pozicijų laikymas su mažu pelnu per sandorį. P. Zubulake (2011) teigimu, nėra griežtai apibrėžto laiko, kiek turėtų būti laikomos atidarytos pozicijos, kai naudojama DDP. Tai tinka net ir tada, kai tikslas yra pasiekti reikšmę, kuo artimesnę nuliui, todėl pozicijos laikymo laikotarpis gali svyruoti. I. Aldridge aprašo šią dalį, teigdamas, kad atidarytos pozicijos gali būti laikomos minutes ar net valandas, bet vis tiek tai gali vadintis didelio dažnio prekyba. Galima teigti, kad, naudojant DDP, yra tikimybė turėti labai trumpus pozicijų laikymo terminus, esančius arčiau nulio (gali būti laikomos ir nanosekundes), tačiau, priklausomai nuo strategijos, tai nėra būtina daryti visą prekybos laikotarpį.



1 pav. Didelio dažnio prekybos strategijos

Didelio dažnio prekybos strategijas galima suskirstyti į šešias grupes: elektroninio likvidumo suteikimas, statistinis arbitražas, likvidumo aptikimas, rinkos kūrimas, mažas vėlavimas ir strategijos, kurios remiasi naujienomis. Visų šių strategijų principas nesiskiria nuo lėto dažnio naudojamų strategijų, tačiau didelio dažnio duomenų naudojimas leidžia jas išnaudoti skirtingai ir dažnai gaunami kitokie rezultatai, nei naudojantis lėto dažnio duomenis. Visų strategijų tyrimas užimtų daug laiko, todėl šiame darbe koncentruotasi ties statistinio arbitražo (porų prekybos) strategija.

Nors didelio dažnio prekyba atlieka svarbų vaidmenį šiandieninėje rinkoje, tačiau su ja susijusi literatūra nėra išsami. Pagrindinė problema – duomenys, pagal kuriuos sudėtinga identifikuoti šį fenomeną ir su juo dirbti. Dvi pagrindinės problemos kyla iš duomenų, naudojamų analizuoti DDP (Fabozzi F. J., 2011), žinomų kaip *tick-by-tick* arba didelio dažnio duomenys (DDD).

Pirmoji problema yra susijusi su rinkos mikrostruktūros efektu, kuris reiškia, kad, naudojant DDD, finansinio instrumento kaina elgiasi kitaip, nei ją palyginant su dienos pabaigos informacija, todėl ją analizuoti yra sunkiau.

Antra problema yra ta, kad sunku rasti kainų koreliaciją, nes skirtingi finansiniai instrumentai informaciją gali pateikti skirtingu greičiu. Dėl to šie duomenys kartais agreguojami vienoduose laiko languose, vedami jų vidurkiai ir taip gaunamas norimas rezultatas. Reikia naudoti skirtingus modelius. Literatūroje didelio dažnio duomenyse kainos seka šuolio – difuzijos (angl. *jump-diffusion*) procesą. Bet toks procesas neleidžia atsirasti nukrypimams ar kitiems kainoms būdingiems judėjimams (Fabozzi F. J., 2011).

F. J. Fabozzi, S. M. Focardi ir C. Jonas (2011) teigimu, nors didelio dažnio duomenys reiškia spartų žingsnį į priekį nustatant dalį finansinių rodiklių, tačiau tai neleidžia kurti universalių taisyklių. Kitos galimos tendencijos susijusios su duomenų, kurie gali konfliktuoti su nepertraukiamu laiko modeliu, diskretiškumu. Todėl daugiau negalima teigti, kad duomenys yra nepertraukiami. Dėl galimų skirtingų naudoti duomenų dydžių gali kilti papildomų kliūčių skaičiuojant kai kuriuos statistinius rodiklius.

Šiuo metu finansų rinkos yra visiškai automatizuotos, susidedančios iš prekybos algoritmų, todėl didžioji dalis rinkos yra užvaldyta didelio dažnio prekybos algoritmų (Fox G. R. et al., 2015). Didelio dažnio prekyba apima algoritminės prekybos programas, kurios išnaudoja sekundinius pasikeitimus rinkoje vykdyti prekybą. Šio tipo prekyba jau kurį laiką yra pakeitusi tradicinę prekybą, kai biržoje prekiaudavo pats žmogus (Fox G. R. et al.,

2015). Vienas pagrindinių tikslų didelio dažnio prekyboje yra būti pirmam, aplenkti kitus, t. y. pateikti pirkimą ar pardavimą greičiau už kitus. Šio tipo strategijų dydžio dalį sudaro vadinamosios rinkos kūrimo strategijos, kuriomis naudojantis, yra didinamas likvidumas esančiose elektroninėse rinkose. Rinkos kūrimo strategijos yra būtiniausios mažo likvidumo rinkoms, kad jos taptų patrauklesnės kitiems rinkos dalyviams. Būtent dėl šios priežasties rinkos kūrėjai moka mažus, o kartais ir jokių komisinių mokesčių (Herlemont D., 2013; Zubulake P. and Lee S., 2011; Brogaard J. et al., 2013; Jaramillo Th. C., 2016). Didžioji dalis ekonomistų ir akademinės finansų bendruomenės didelio dažnio prekybą laiko naudinga rinkai dėl likvidumo suteikimo, nes taip pritraukia kapitalą ir naujos rinkos dalyvius (Jaramillo Th. C., 2016).

Turint omenyje, kad prekyba turi būti vykdoma sekundiniu, o kartais ir nanosekundiniu greičiu, visa prekyba turi vykti tik naudojant didelio našumo kompiuterius, leidžiančius prekiauti tokiu greičiu. Didelio dažnio prekyba yra automatinė algoritminė prekyba, kurią galima apibūdinti taip:

a) Prekyba vykdoma kompiuteriais, žmogui kišantis labai mažai arba visai nesikišant.

b) Naudojama mažo vėlavimo technologija, kuria pagreitinamas prekybos sprendimų priėmimas ir sumažinamas laikas pateikti prekybos signalą elektroninėje biržoje.

c) Naudojamas didelio našumo prisijungimas prie elektroninės biržos.

d) Atliekamas didelis užklausų kiekis (pirkimai, pardavimai, pozicijų atidarymas, uždarymas ir atšaukimas) dėl didelio greičio.

Prekybos strategijos, naudojančios didelio dažnio prekybą, ieško sunkiai pastebimų ir trumpų rinkos pasikeitimų, kuriuos galėtų išnaudoti savo naudai, pasitelkdamos didelio našumo kompiuterius ir atlikdamos greitus sprendimus dideliais kiekiais. Šios prekybos galimybės yra labai smulkūs pokyčiai rinkose, kurie veikia nepelningas finansinių instrumentų kainas. Tačiau dėl esamos galimybės atlikti tūkstančius prekybos signalų per sekundę pelnas gali išaugti.

Didelio dažnio prekyba yra itin komercializuota, tad sužinoti, kokias strategijas naudoja įmonės ir kokia yra optimali konfigūracija, yra labai sunku; be to, dažnai ši informacija yra konfidenciali. Informacija apie optimalų DDP strategijų taikymą ir metodus, pasitelkiamus norint pasiekti didelius prekybos greičius, padėtų ne tik toliau plėtoti šias naudojamas strategijas ir techninę įrangą, bet ir galėtų būti taikoma kitose mokslo srityse, kuriose reikia per kuo trumpesnį laiką apdoroti didelius kiekius duomenų.

Daugumoje publikacijų pateikiamos metodologijos, siūlančios pasirinkti tinkamas DDP strategijas. Taikant šias metodologijas, taip pat siūlomas būdas matuti DDP strategijų pelningumą, nustatyti optimalius strategijų parametrus, kada reikėtų pateikti užsakymus biržoms, kaip matuoti šių strategijų aktyvumą ir kt. Tačiau metodologijose nepateikiama informacijos, rekomenduojančios teisingai realizuoti didelio dažnio prekybos sistemas, neaiškinama, kokias technologines platformas naudoti ir kaip jos turėtų būti konfigūruotos (Vaitonis M., 2018, Kearns M. et al. 2010, Anane M. ir Abergel F., 2014, Beckhardt B. et al., 2016). Todėl labai trūksta informacijos apie optimalią DDP strategijų konfigūraciją ir tai, kaip jos turėtų būti pritaikomos pasirinktai technologinei platformai. Dar mažiau yra informacijos apie techninę įrangą, kuri būtų optimali DDP. Šios informacijos trūkumas taip pat motyvavo tyrimo atlikimą.

Prieš pradedant ieškoti optimalios didelio dažnio prekybos konfigūracijos, būtina išsiaiškinti, kokią didelio dažnio prekybos strategiją reikėtų naudoti: ar statistinį arbitražą, naujienų srauto, mažo vėlavimo, rinkos kūrimo, įvykių arbitražą, ar kitas strategijas. Remiantis EUREX, kuri yra didžiausia Europos prekybos elektroninė birža, viena pagrindinių naudojamų didelio dažnio prekybos strategijų yra statistinis arbitražas, kuriuo rinkose kuriamas likvidumas ir vadinamos rinkos kūrimo strategijos (EUREXCHANGE, 2013). Dėl šios priežasties darbe buvo pasirinkta detaliau patyrinėti tokio tipo strategijas ir išsiaiškinti, kaip jas būtų galima taikyti didelio dažnio prekyboje.

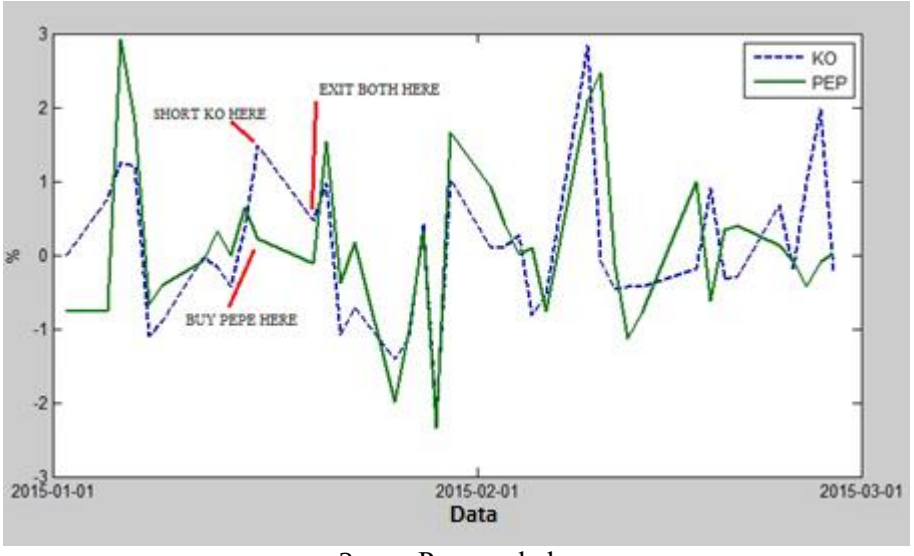
1.2 Statistinio arbitražo prekyba

Statistinis arbitražas atsirado kartu su pirmuoju rizikos fondu 1950 metais. Statistinis arbitražas veikė taikydamas matematinius modelius – ieškojo finansinių instrumentų kainų netikslumų, kai vienu metu atidarytos pirkimo ir pardavimo pozicijos sumažindavo prekybos riziką (Ferguson ir Laster, 2007). Statistinio arbitražo, arba kitaip – porų prekybos, strategijos gauna pelną, o pirkimo pozicijos uždirba daugiau arba pralošia mažiau nei pardavimo pozicijos. Prieš kompiuterizuojant šias strategijas, prekybos pozicijų valdymas pasižymėjo plačiomis nerizikingomis statistinio arbitražo prekybos galimybėmis, tačiau jos rėmėsi klaidingais kainų nustatymais arba kainų nustatymo vėlavimu skirtingose rinkose (Driaunys et al., 2014). Pastaruoju metu dėl technologinės pažangos ne tik galima prekiauti dideliais kiekiais, bet įmanus ir lygiagretus priėjimas prie koreliuotų rinkų. Visos

rinkos persikėlė į elektroninę erdvę, prekyba vykdoma dideliu greičiu ir dideliais kiekiais, o rinkos dalyviai turi priėjimą prie realaus laiko duomenų.

Porų prekyba yra viena dažniausių statistinio arbitražo strategijų ir nuo pat 1980 metų yra aktyviai naudojama profesionalių rinkos dalyvių, investuotojų, rizikos fondų ir kitų (Vidyamurthy G., 2004; Dunis Ch. et al., 2010; Gatev E. W. N et al., 2006; Hogan J. P. et al., 2004). Istoriškai porų prekybos strategijos remiasi finansinių instrumentų kainų nustatymo neatitikimais. Šio tipo strategijų tikslas yra rasti du susijusius finansinius instrumentus, kurie rinkoje juda panašiai. Tada reikia laukti, kol jų kainų judėjimas nukryps nuo istorinio vidurkio, ir atidaryti *pirk / parduok* pozicijas, vienu metu pasitelkus abu finansinius instrumentus, tikintis, kad kainos grįš prie istorinio vidurkio (Miao J., 2014; Elliot R. J. et al., 2005). Šio tipo strategijos yra vadinamos neutralios rinkos statistinio arbitražo strategijomis, kurios remiasi finansinių instrumentų kainų konvergencija. Labiausiai susijusios kainų poros parenkamos prekybai, tada atitinkamai parenkamos šiai porai *pirk / parduok* pozicijos – taip prekiaujama esant mažiausiai rizikai (Gatev E. W. N et al., 2006).

Galima pateikti porų prekybos strategijos pavyzdį: žemiau esančiame paveikslėlyje pavaizduotos dvi itin koreliuotos akcijos, atstovaujančios panašiam produktui ir labai panašiai reaguojančios į rinkos pokyčius, t. y. *Coca-Cola* (KO) ir *Pepsi* (PEP).



2 pav. Porų prekyba

Kaip matyti pateiktame paveikslėlyje, abi akcijos juda panašiai. Porų prekybos strategija stebi šį judėjimą, atstumą tarp abiejų akcijų ir laukia,

kada jis nukryps nuo istorinio vidurkio. Būtent toks nukrypimas ir leidžia šio tipo strategijoms pradėti prekybą. Pradėjus prekybą, toliau stebimos akcijos ir laukiama, kada jų kainos grįš prie istorinio vidurkio, o prieš tam įvykstant, uždaromos visos prekybos pozicijos.

1.3 Didelio dažnio prekybos panaudojimo atvejis

Pagrindinis didelio dažnio porų prekybos strategijos tikslas yra rasti du finansinius instrumentus, kurie judėtų kartu, t. y. būtų koreliuoti. Kai pora randama, prekybos strategija turi nuspręsti, kurį poros narį reikia pirkti ir kurį parduoti pagal tos strategijos prekybos taisykles. Remiantis atliktais tyrimais, išskirti šeši pagrindiniai porų prekybos strategijų naudojimo žingsniai:

1. Lango, skirto normalizuoti duomenis ir prekiauti, dydžio parinkimas.
2. Duomenų normalizavimas.
3. Koreliuotų porų parinkimas.
4. Prekybos taisyklių nustatymas.
5. Prekybos vykdymas.
6. Porų prekybos strategijos efektyvumo matavimas (Vaitonis M. ir Masteika S., 2016; Vaitonis M. ir Masteika S., 2017).

Pirmiausia reikia nustatyti prekybos parametrus – duomenų normalizavimo ir prekybos lango dydį. Remiantis prieš tai atliktais tyrimais, kuriuose buvo naudojami nanosekundiniai duomenys, nustatyta, kad 20 sekundžių prekybos ir duomenų normalizavimo langas buvo optimalus (Vaitonis M. ir Masteika S., 2017). Kai šių langų dydžiai yra parinkti, reikia normalizuoti ateities sandorių kainas, leidžiančias jas lyginti tarpusavyje. Vėliau turi būti taikomas porų parinkimo metodas ir ieškoma rastų porų prekybos galimybių. Rasti prekybos signalai siunčiami elektroninei biržai ir kiekvienos prekybos dienos pabaigoje įvertinamas taikytos strategijos efektyvumas.

1.4 Didelio dažnio prekybos duomenų normalizavimas

Duomenų normalizavimas naudojant didelio dažnio prekybą yra vienas iš svarbiausių punktų, kuriuo galima lyginti skirtingas ateities sandorių kainas; taip panaikinami trukdžiai ir galima aiškiau matyti kainų pokyčius. Šiame tyrime duomenų normalizavimas atliekamas taip: apskaičiuojamas kiekvienos ateities sandorio kainos $p_{i,t}$ vidurkis $\mu_{i,t}$ ir pasirinkto

normalizavimo lango standartinis nuokrypis $\sigma_{i,t}$, o toliau taikoma tokia formulė:

$$P_{i,t} = \frac{p_{i,t} - \mu_{i,t}}{\sigma_{i,t}} \quad (1)$$

$P_{i,t}$ yra normalizuota ateities sandorio kaina, kur *i* nurodo ateities sandorį ir *t* laikotarpį, už kurį skaičiuojama (Vaitonis M. ir Masteika S., 2017; Perlini M. S., 2009).

1.5 Prekybos porų parinkimas

Prieš pradedant ieškoti prekybos signalų, turi būti rastos visos galimos finansinių instrumentų poros. Tam yra naudojamas porų parinkimo algoritmas. Šiame tyrime išskiriami du pagrindiniai porų parinkimo metodai: mažiausių kvadratų metodas ir kointegracijos metodas. Prieš tai atliktuose tyrimuose buvo taikyti abu šie metodai (Vaitonis M. ir Masteika S., 2016; Vaitonis M. ir Masteika S., 2017; Vaitonis M. ir Masteika S., 2018; Vaitonis M., 2017).

Mažiausių kvadratų metodas

Mažiausių kvadratų (arba kitaip vadinamas atstumų) metodas reikalauja palyginti nedaug skaičiavimų ir linijinės algebros, kuriais būtų nustatyti du koreliuoti elementai (Binh D. et al., 2006; Miller S. J., 2006).

Remiantis Gatev (2006), randama ateities sandorių pora, kuri turi mažiausią kvadratinį normalizuotų kainų nuokrypį. Tai yra ganėtinai paprastas metodas, kurį taikant, reikia atlikti nedaug skaičiavimų, kuriam nereikia didelio techninės įrangos našumo. Todėl jis galėtų patikti didelio dažnio prekybininkams. Pagal atstumų metodą spėtina, kad yra statinis linijinis dviejų ateities sandorių ir jų kainų, kurios atskirai yra atsitiktinės, ryšys. Vienas iš šio metodo privalumų yra tas, kad nėra galimybės netinkamai specifikuoti ar įvertinti randamą porą, tačiau jis negali prognozuoti (Binh D. et al., 2006).

Kointegracijos metodas

Kointegracijos metodas leidžia nustatyti ilgalaikius dviejų kintamųjų ryšius tada, kai jie turi vienodą integracijos lygį. Vadinasi, dvi nestacionarios laiko eilutės yra kointegruotos, jei jų linijinis darinys yra stacionarus (Engle R. F. ir Granger C. W. J., 1987).

Šis metodas puikiai tinka poroms parinkti, kai norima išnaudoti trumpalaikius nukrypimus nuo ilgalaikio ryšio. Trumpalaikiai nukrypimai

panaikinami taisant klaidas, t. y. pagal Vidyamurthy (2004), koreguojant vieną arba abi laiko eilutes, kuriomis norima sukurti ilgalaikį ryšį. Tai reiškia, kad kitaip nei atstumų, taikant kointegracijos metodą, galima prognozuoti remiantis istorine informacija.

Norint rasti du kointegruotus elementus, reikia imtis šių veiksmų:

1. Identifikuoti ateities sandorių poras, kurios potencialiai galėtų būti kointegruotos.
2. Kai potencialios poros aptinkamos, tada reikia patikrinti hipotezę, leidžiančią tikėtis, kad ateities sandorių pora, besiremianti istoriniais duomenimis, iš tikrų yra kointegruota.
3. Patikrinti, ar rastos kointegruotos poros gali būti naudojamos prekybai (Vidyamurthy G., 2004).

Šio metodo tikslas yra rasti linijinio derinio, kuris turi reikšmingą nuspėjamą komponentą ir kuris yra nesuderintas su visais rinkos pokyčiais, poras. Pirmiausia matuojamas poros kainų skirtumo stacionarumas. Tai atliekama tikrinant, ar duomenų sekos yra integruotos ta pačia tvarka pasitelkiant Papildytą Dickey Fuller testą (ADF) (Caldeira J. ir Moura G. V., 2013). Dickey Fuller testas naudojamas nustatyti stacionarumą, kuris tikrina autoregresyvaus integruoto judėjimo vidurkio prieš stacionarumą ir alternatyvumą nulinę hipotezę. Nulinė hipotezė tikrinama aiškinantis, ar procesas turi šakninį vienetą (yra ne stacionarus). Jei pora yra kointegruota, tai jų reikšmių skirtumas turėtų būti stacionarus (Bogoev D. ir Karam A., 2016; Mushtaq R., 2011). Papildytas Dickey Fuller testas praplečia Dickey Fuller testą ir įtraukia praplėstą užlaikymą nuo priklausomų kintamųjų siekiant išspręsti autokoreliacijos problemą (Mushtaq R., 2011; Dickey D. ir Fuller W., 1979). Jei išlaikomas ADF testas, tada atliekamas kointegracijos testas su visomis galimomis porų kombinacijomis. Norint atlikti kointegracijos testą, taikomas Engle ir Granger dviejų žingsnių metodas bei Johansen testas (Dickey D. ir Fuller W., 1979; Mushtaq R., 2011). Engle ir Granger pristatė tokią teoremą: jei yra dvi ar daugiau duomenų eilučių y_t ašyje, kurios yra tarpusavyje kointegruotos, tai yra ir klaidų taisymo galimybė (Engle R. F. ir Granger C. W. J., 1987).

1.6 Skyriaus išvados

Šiame skyriuje apžvelgta algoritminė prekyba, jos veikimas, aptarta didelio dažnio prekyba, pateiktas didelio dažnio naudojimo atvejis, nagrinėtos porų prekybos (arba statistinio arbitražo prekybos) strategijos ir

metodai, skirti parinkti poras statistinio arbitražo prekybai. Atlikus literatūros analizę, buvo nustatyta, kad yra mažai informacijos apie optimalią didelio dažnio strategijų konfigūraciją. Išsiaiškinta, kokios strategijos turėtų būti taikomos ir kokį metodą derėtų pasitelkti siekiant jas realizuoti, priklausomai nuo prekybos algoritmo ir pasirinktos techninės įrangos. Remiantis viena didžiausių Europos prekybos elektroninių biržų, taip pat galima teigti, kad viena pagrindinių naudojamų didelio dažnio prekybos strategijų yra statistinis arbitražas, kurį naudojant rinkose kuriamas likvidumas. Tokio tipo strategijos gali būti vadinamos rinkos kūrimo, todėl buvo pasirinkta detaliau nagrinėti šio tipo strategijas.

2 DIDELIO DAŽNIO PREKYBA IR DIDELIO NAŠUMO KOMPIUTERIJA FINANSUOSE

Norint geriau suprasti didelio dažnio prekybą, reikėtų išsiaiškinti, kokie algoritmai naudojami prekiaujant dideliu greičiu. Remiantis efektyvios rinkos hipoteze, kaina eina atsitiktiniu keliu ir turėtų būti neįmanoma gauti pelno iš sisteminės prekybos. Didelio dažnio prekyba paremta koncepcija vadinama rinkos mikrostruktūros prekyba, ji tyrinėja kainos formavimo procesus. Tai yra rinkos dalyvių bandymų ir klydimo procesas. Toks procesas vyksta tada, kai rinkos dalyvis atmeta savo vertinimą ir kai gauna naujos informacijos. *Pirk / parduok* reikšmės ir dydis yra nuolat keičiamos remiantis tais vertinimais tol, kol iš lėto konverguoja į optimalų ryšį (Aldrige I., 2010). Lyons išskiria du pagrindinius rinkos mikrostruktūros modelius: inventorių modelis ir informacijos modelis. Žodis, kuris geriausiai išskiria šiuos du modelius, yra „naujienos“. Informacijos modelis stebi, kaip rinka juda, kai gaunama nauja informacija. Inventorių modelis, kitaip dar žinomas kaip rinkos likvidumo tiekėjas ar rinkos kūrėjas, aiškina kainos judėjimą, neatsižvelgdamas į naujienas (Aldrige I., 2010).

Pirmoji grupė strategijų arba algoritmų, kurie gali būti naudojami didelio dažnio prekyboje, yra pelningos rinkos kūrimo strategijos. Pirmoji tokio tipo strategija remiasi Gabler Ruin problema, teigiančia, kad tikimybę, jog lošėjas praloš, galima apskaičiuoti pagal šią formulę:

$$P(Failure) = (\frac{P(Loss)*Loss}{P(Gain)*Gain})^{W_0} \quad (2)$$

Kur P() nurodo tikimybę, *Loss* ir *Gain* yra dydis arba rezultatas ir W_0 yra pradinis turtas ar investicija (Aldrige I., 2010).

M. Avellaneda ir S. Stoikov (3) darbe kitaip traktuojamas rinkos kūrimas. Jie taiko matematinį modelį ir remiasi stochastinėmis diferencialinėmis lygtimis, kuriančiomis optimalų *pirkti* / *parduoti* signalą, kurį galima leisti į rinką (Avellaneda M. ir Stoikov S., 2008).

Paskutinis inventorinių modelių algoritmų rinkinys remiasi tuo, kaip rinka juda priklausomai nuo limituotų užsakymų disbalanso, kuris dažnai nukreipia prekybą kita kryptimi. Tai yra logiška, nes jei *pirk* kainos pusė yra daug didesnė už *parduok*, tai bus daug sunkiau užpildyti tuos užsakymus, tada kaina sunkiau kils. D. Easley, M. L. de Prado ir M. OHara aprašytas modelis, kuris remiasi bendru disbalanso rodikliu, vadinamu mikrokaina (Easley D. et al., 2013).

Pagrindinis inventorinių modelių trūkumas yra tas, kad jie nesiremia jokia kitų rinkos dalyvių motyvacija ir nauja informacija. Kita vertus, šios

motyvacijos, siekiai ir naujienos yra pagrindinis informacijos modelių aspektas. I. Aldridge teigimu, „informacijos modeliai naudoja žaidimų teorijas kaip atvirkštinę inžineriją kainoms ir prekybos kryptims nustatyti; tikslas – sužinoti, kokią informaciją turi rinkos kūrėjai“.

Informacijos modeliai taip pat pasitelkia stebėjimus arba užsakymo reikšmes, leidžiančias priimti pagrįstus prekybos sprendimus (Aldrige I., 2010). Dar vienas svarbus veiksnys – asimetrinės informacijos koncepcija. Visa tai nulemia netikslaus pasirinkimo fenomeną, kuris iš esmės leidžia informuotiems rinkos dalyviams išrinkti neinformuotus rinkos dalyvius.

Pirmoji informacinių modelių algoritmų grupė susideda iš techninės analizės, kuri pasitelkia statistinius įrankius, kuriais gali aptikti finansinių instrumentų ryšius. Pagrindinis tikslas – rasti statistinę priemonę (kaip koreliacija) ir labiausiai tarpusavyje koreliuotus finansinius instrumentus. Kai tik kainos nukrypsta nuo tam tikros reikšmės ar istorinio vidurkio, rinkos dalyvis turi pateikti priešingas pozicijas šiems finansiniams instrumentams. Šie metodai remiasi tik finansinių instrumentų kaina ir jokiais kitais kriterijais, pvz., nesiremia užsakymo knygos informacija ar kitomis fundamentaliomis reikšmėmis. I. Aldridge (2010) manymu, „aptiktas statistinis ryšys yra atsitiktinis ar klaidingas ir turi mažą nuspėjamąją galią“.

Kiti algoritmai ar strategijos, kurie naudoja matematinius ir fundamentalius įrankius, yra dalis statistinio arbitražo strategijų. Jos realizuoja didelio dažnio prekybą kaip būdą, leidžiantį greičiausiai aptikti rinkų neatitikimus ar nukrypimus nuo jų.

Didelio dažnio prekyba sparčiau augo nuo tos dienos, kai ji pirmą kartą buvo pristatyta rinkose. Jau 2010 metais DDP akcijų biržose šoktelėjo iš beveik 0 (dar 2005 metais) iki 40 % visų sudarytų sandorių Europoje. Pavyzdžiui, 2005 metais didelio dažnio prekyba buvo atsakinga tik už 20 % sandorių Amerikoje ir šoktelėjo iki 60 % 2009 metais. Tada ištiko finansinė krizė ir 2014 metais didelio dažnio prekybos dalis krito iki 35 % ir 50 % visų sandorių Europoje ir Amerikoje atitinkamai (Kaya O., 2016). Dabar DDP yra atsakinga už vidutiniškai 55 % sandorių Amerikos akcijų rinkose ir apie 40 % Europos akcijų rinkose (Krauss C., 2015). CFTC nustatė, kad nuo 2012 spalio iki 2014 spalio mėnesio algoritminės prekybos sistemos buvo atsakingos už beveik 80 % visų užsienio ateities sandorių, 67 % ateities palūkanų normos sandorių, 62 % ateities akcijų sandorių, 47 % ateities metalų ir energijos sandorių ir 38 % ateities žemės ūkio produktų sandorių (Miller R. S. ir Shorter G., 2016). Dabar apie 70 % visų sandorių, atliekamų Amerikos rinkose, priklauso algoritminei prekybai.

Visos didelio dažnio prekybos sistemos turi pagrindinius keturis komponentus:

1. Informacijos infrastruktūra – finansinių instrumentų informacijos procesorius (SIP). Tai technologija, leidžianti rinkti elektroninių biržų informaciją, ją įvertinti ir siųsti atgal kaip nuolatinį srautą su geriausiomis *pirkti* / *parduoti* kainomis. SIP turi dirbti labai greitai. Vidutiniškai Niujorko akcijų birža apdoroja apie 200 000 užklausų per sekundę, iš kurių 28 000 per sekundę yra konvertuojamos į sandorius. Prekybininkai su elektroninėmis biržomis bendrauja FIX protokolu, kurį sudaro informacija, užrašyta ASCII simboliais, ir naudojamas XML plėtinys, kuris dar vadinamas FIXML (Clark C., 2011).

2. Didelio greičio duomenų ryšiai. Prekybininko ir biržų ryšiai yra labai našūs, įprastai 1000 Gbps. Taip pat šie ryšiai turi mažą vėlavimą. Prekybininkai, siekdami turėti kuo didesnį šviesos greitį, naudoja žemos refrakcijos rodiklio gyslas. Vis daugiau prekybininkų naudoja iš lazerių mikrobangų sudarytus oro kanalus. Šios lenktynės vyksta visur, pavyzdžiui, mikrobangų ryšys tarp Londono ir Frankfurto sumažino informacijos perdavimo greitį, palyginti su optiniu kabeliui, iki 40 % (Krauss C., 2015).

3. Didelio našumo skaičiavimo techninė įranga. Yra padidėjęs didelio našumo skaičiavimo platformų, veikiančių su GPU ir FPGA, poreikis (Krauss C., 2015).

4. Didelio dažnio prekybos algoritmai. Jie yra greiti, lygiagretūs ir specialiai sukurti tam, kad išnaudotų pačius mažiausius kainų pokyčius didžiausiu galimu greičiu (Krauss C., 2015).

Norint visiškai išnaudoti didelio dažnio prekybą, techninės įrangos akceleracija turi būti įtraukta į bendrą sprendimą. Ši akceleracija ir aukštesni skaičiavimų našumai gali būti pasiekti naudojant tikslinę techninę įrangą. Tam tikslui pasiekti gali būti naudojamos skirtingos platformos, susidedančios iš CPU, GPU ir FPGA. Norint pasirinkti tinkamiausią konfigūraciją, reikia žinoti visų sistemų pliusus ir minusus.

Pirmasis, supratęs, kad GPU galima naudoti ne tik grafikams, bet ir bendro naudojimo programoms, buvo Markas Harris. Nuo tada GPU programavimo metodai evoliucionavo ir dabar yra keli bendro naudojimo programų būdai: CUDA (angl. *Compute Unified Device Architecture*) pristatyta *NVIDIA* ir *APP (Stream)* pristatyta *AMD*. Naujas standartas *OpenCL* (angl. *Open Computing Language*) skirtas sujungti skirtingus GPU pritaikymo metodus ir pateikti vieną bendrinę sistemą, skirtą rašyti

programoms heterogeninėse sistemose, naudojančiose GPU ir CPU (Bogdan O. et al.,2012).

2.1 Didelio našumo kompiuterija naudojant CPU, GPU ir FPGA

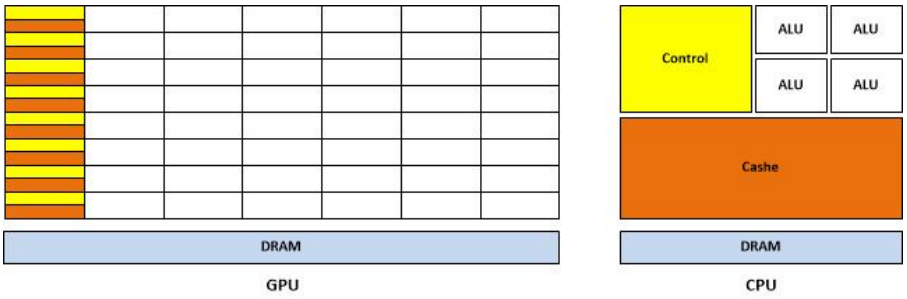
Mikroprocesorių architektūros, kuri yra naudojama CPU, tikslas yra vykdyti vieną loginę instrukciją, t. y. aritmetinę užduotį (ALU): per ciklą, kuris yra valdomas vieno kontrolerio ir su viena duomenų struktūra. Pagal *Flynn's Taxonomy* (Flynn M., 2011), tokios instrukcijos ir duomenų perdavimas vadinamas viena instrukcija, vienais duomenimis (SISD). Atmetus logines instrukcijas, esančias CPU, likusi jo dalis įprastai yra skiriama spartinamajai atminčiai. Ši atmintis yra labai maža ir apribota, todėl mikroschema ir naudojama tik laikinai laikyti duomenis, su kuriais dirbama. Naudojant šio tipo atmintį, yra mažinamas atminties vėlavimas, nes duomenys, kurie yra spartinamojoje atmintyje, greičiau pasiekiami, nei duomenys, kurie laikomi kitose atmintyse (Flynn M., 2011).

1990 metais CPU ir GPU vadovavosi Moore taisykle. Tačiau kiekviena jų gavo skirtingos naudos iš Moore taisyklės. CPU našumas buvo padidintas didinant ciklų skaičių ir naudojant papildomus tranzistorius naujesnėms ir greitesnėms instrukcijoms, o GPU galia buvo didinama didinant ne ciklų, o ALU skaičių. Tačiau 1998 metais nutiko taip, kad GPU aplenkė CPU tranzistorių skaičiumi ir iki šiandien jų turi daug daugiau (Flynn M., 2011).

GPU, palyginti su CPU, pasiūlė visai priešingą operacijų filosofiją. GPU tikslas yra atlikti tą pačią instrukciją su kiekvienu paveikslėlio pikseliu. Pagal *Flynn's Taxonomy* (Flynn M., 2011), tokio tipo instrukcijos ir duomenų srautai yra vadinami viena instrukcija, turinčia daug duomenų (SIMD). Kitaip nei CPU, GPU didžiąją savo dalį yra paskyrusi ALU ir tik maža dalis skirta spartinamajai atminčiai. Taip yra todėl, kad GPU spartinamoji atmintis buvo skirta atlikti buferio funkciją, o ne mažinti atminties vėlavimą.

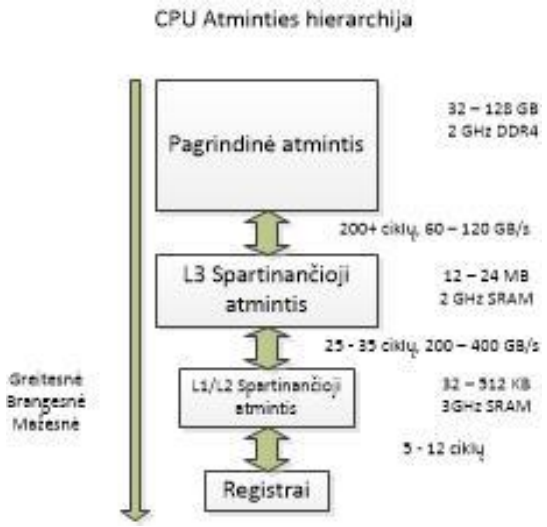
Skirtumas tarp CPU ir GPU yra toks, kad GPU yra labiau specializuotas darbui su skaičiais ir aritmetiniais veiksmais; tai yra labai svarbu apdorojant grafiką tada, kai reikia milijonų, o kartais ir bilijonų skaičiavimų per sekundę. Priklausomai nuo gamintojo, branduolių kiekis, esantis GPU, skiriasi. GPU, kuris turi 100+ branduolių ir gali apdoroti tūkstančius užklausų, gali tam tikras programines įrangas paspartinti 100+ kartų. Palyginti su CPU. GPU šią akceleraciją pasiekia energetiškai efektyvesnis ir

kaina per branduolį, palyginti su CPU, yra daug mažesnė (Asaduzzaman A. et al., 2014; Nambia P. P. et al., 2014).



3 pav. GPU ir CPU

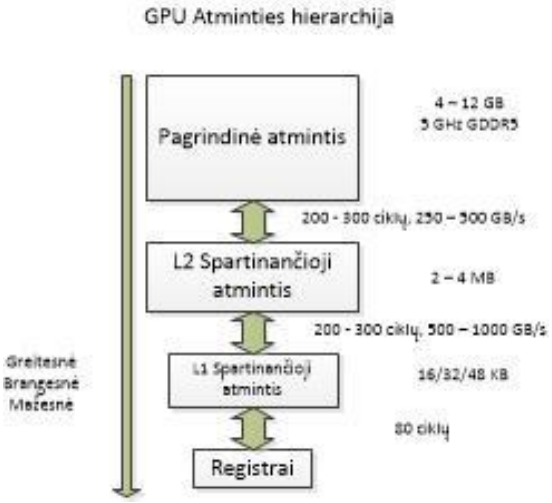
Atminties greičiams palyginti pateikiamos CPU ir GPU atminčių hierarchijos, kuriose atsispindi atskirų atminčių greičiai, pralaidumas ir tai, kiek užklausų per sekundę galima apdoroti.



4 pav. CPU atminties hierarchija

Pati sparčiausia atmintis yra registruose, kuriuose saugomos įvairios instrukcijos. CPU registrai gali apdoroti 5–12 užklausų vienu metu su L1/L2 spartinamąja atmintimi. Pagal spartą šios spartinamosios eina po registrų. Jų talpa gali būti nuo 32 Kb iki 512 Kb, dirba 3GHz SRAM sparta ir vienu metu gali apdoroti 25–35 užklausas iš L3 spartinamosios atminties. L3 yra žemiausio lygio, lėčiausia ir didžiausios talpos spartinamoji atmintis. Jos dydis svyruoja nuo 12 iki 24 Mb, veikia 2GHz SRAM sparta, gali apdoroti iki 200 užklausų iš pagrindinės atminties ir šių atminčių duomenų

pralaidumas gali būti nuo 200 iki 400 Gb/s. Ši atmintis yra pati talpiausia, tačiau ir pati lėčiausia. Jos dydis gali būti nuo 32 iki 128 Gb, veikimo dažnis gali siekti 2GHz DDR4 ir duomenų pralaidumas 60–120 Gb/s. Būtent atminčių pralaidumas ir atskleidžia didžiausią GPU privalumą (Asaduzzaman A. et al., 2014; Nambia P. P. et al., 2014).



5 pav. GPU atminties hierarchija

Palyginti su CPU pagrindine atmintimi, GPU pagrindinės atminties pralaidumas yra beveik keturis kartus didesnis ir siekia 250–500 Gb/s, tačiau jos dydis mažesnis 4–12 G, 5GHz GDDR5. Pagrindinė atmintis, kartu su L2 spartinamąja atmintimi, gali vienu metu atlikti 200–300 užklausų. GPU L2 atminties dydis yra kelis kartus mažesnis, palyginti su CPU L2, ir yra 2–4 Mb, tačiau GPU gali turėti nuo kelių iki šimtų šių atminčių kiekvienam multiprocesoriui. Duomenų L2 ir L1 atminčių pralaidumas taip pat yra daug didesnis už CPU ir gali pasiekti 500–1000 Gb/s. Nors ši atmintis mažesnė 16/32/48 Kb už CPU L2, tačiau, kaip ir L1, jų gali būti šimtai, skirtų kiekvienam multiprocesoriui. GPU registrai pranašesni, nes jie gali vykdyti iki šešių kartų daugiau užklausų vienu metu, palyginti su CPU registrais (Asaduzzaman A. et al., 2014; Nambia P. P. et al., 2014).

Iki pat 2007 metų bet koks GPU naudojimas skaičiavimams, išskyrus grafikos atvaizdavimą, būdavo labai retas ir galimas tik naudojant grafines API kaip *OpenGL* ir *DirectX* (Group K., 1992; Microsoft, 1997). Pokytis įvyko tada, kai *Nvidia* 2007 metų liepos 23 dieną pristatė CUDA. Ši platforma leido bendrojo naudojimo grafines *Nvidia* plokštes naudoti skaičiavimams su *Tesla* architektūra.

Dabar *Nvidia* CUDA, nors ją galima naudoti tik su *Nvidia* produktais, yra pirmaujanti sistema, skirta naudoti grafines vaizdo plokštes skaičiavimams (GPU). Kita sistema, skirta atlikti tuos pačius skaičiavimus su grafinėmis vaizdo kortomis, yra *OpenCL* (atvira skaičiavimo kalba), kuri buvo sukurta Khronos grupės (K. Group, 2009). *OpenCL* tikslas yra sukurti standartizuotą lygiagretaus skaičiavimo platformą, kuri būtų nepriklausoma nuo techninės įrangos. Iš esmės identišką *OpenCL* kodą turėtų būti galima paleisti CPU (*Intel, AMD, ARM, IBM* ir kiti) ir GPU (*Nvidia* ir *ATI*). Vis dėlto CUDA turi patogesnę vartotojui aplinką, palaikančią programavimo kalbas, kurios yra naudojamos atliekant mokslinius skaičiavimus (*C/C++*, *Python* ir kitas) (Nvidia, 2007; PGI, 2009; Klöckner A. et al., 2012; Nvidia, 2014).

GPU ir FPGA palyginimas

Palyginus šias dvi technologijas, atkreiptinas dėmesys, kad FPGA efektyviau išnaudoja energiją, o GPU – efektyvesnis ekonomiškai. FPGA technologijos tikslas yra sukurti konkuruojančias fiksuoto kablelio operacijas su artimu techninei įrangai programiniu sprendimu, o GPU yra labiau pritaikyta lygiagretiems procesams su slankiojančiu kableliu, naudojant tūkstančius mažų procesoriaus branduolių.

1 lentelė. GPU ir FPGA palyginimas (Minhas U. I. et al., 2014)

Funkcija	**Analizė**	**Nugalėtojas**
Slankiojančio kablelio apdorojimas	Maksimalus slankiojančio kablelio operacijų kiekis per sekundę yra didesnis naudojant GPU su maksimaliu DSP palaikymu	GPU
Laiko vėlinimas	Perkelti algoritmai į FPGA demonstruoja deterministinį laiką, kuris yra su šiek tiek mažesniu vėlinimu nei lyginant su GPU.	FPGA
Apdorojimas/vatai	Matuojant GFLOPS per vatą, FPGA yra apie 3 kartus geresnis. Tačiau GPU kiekvienais metais vis rodo geresnius rezultatus ir artėja prie FPGA.	FPGA
Sąsaja	GPU yra jungiamas naudojant PCIe jungtį, FPGA yra lankstesnis ir leidžia prisijungti prie kito įrenginio naudojant praktiškai bet kokią jungtį.	FPGA
Atgalinis suderinamumas	Programinės įrangos sukurtos darbui senesnėms GPU gali dirbti ir su naujomis. FPGA HDL gali būti perkeltas ir į naujesnę versiją, tačiau turi būti modifikuotas.	GPU
Lankstumas	FPGA neturi lankstumo modifikuojant sintezuojamą kodą ir perkeliant į techninę	GPU

	įrangą, GPU tai nesukuria papildomų trikdžių.	
Dydis	Dėl mažesnio energijos sunaudojimo FPGA reikalauja mažiau priemonių šilumos išskaidymui, kai yra montuojamas mažesnėje aplinkoje.	FPGA
Plėtojimas	Dauguma algoritmų yra pritaikyti darbui su GPU, o FPGA plėtojimas yra sudėtingas ir brangus.	GPU
Apdorojimas/kaina	Nors FPGA sunaudoja mažiau energijos, tačiau skaičiuojant pinigines sąnaudas per vieną GFLOPS, GPU yra daug pigesnis už FPGA.	GPU

Vienas iš didžiausių FPGA privalumų yra tas, kad FPGA mikroschemos yra techninis algoritmo variantas, o techninė įranga yra greitesnė už programinę įrangą, taip pat FPGA naudoja mažiau energijos. GPU seniau reikėdavo daug energijos įvairiems skaičiavimas, todėl sprendimai, jautrūs energijos suvartojimui, negalėjo naudoti GPU, tačiau dabar naujosios GPU plokštė sunaudoja daug mažiau energijos ir šiuo klausimu tampa dideliu konkurentu FPGA. GPU veikia su programine įranga, todėl daugiau laiko užtrunka paleisti algoritmą. Visos instrukcijos pirmiausia turi būti paleidžiamos ir sustatomos į eilę: turi būti atliekami matematiniai skaičiavimai ir rezultatai turi būti grąžinami į atmintį. Tačiau GPU konstrukcija sudaro sąlygas vykdyti daug skaičiavimų lygiagrečiai ir algoritmas atlieka skaičiavimus daug greičiau, nei tai vyko iš eilės procesoriuje. Skirtingai nuo standartinių CPU algoritmų, GPU algoritmas yra paleidžiamas labai arti programinės įrangos ir todėl padidina šio sprendimo efektyvumą. GPU pranašumas paaiškėja naudojant slankiojančio kablelio operacijas. GPU branduoliai yra vietinės techninės įrangos slankiojančio kablelio procesoriai. Pavyzdžiui, 384 branduolių GPU gali atlikti 384 slankiojančio kablelio matematinius skaičiavimus kiekvieno ciklo metu. Papildomai GPU yra sukurti kartu su labai sparčia atmintimi ir tiesioginiu atminties pasiekiamumu (DMA): taip GPU branduoliai, nenaudodami CPU ciklų, gali pasiekti duomenis. Yra daug paprasčiau atnaujinti GPU ar perrašyti algoritmą, skirtą GPU, pakeitus techninę įrangą, palyginti su FPGA, kurioje tą padaryti yra sudėtinga. GPU vartotojai turi prieigą prie nemokamų bibliotekų ir programinės įrangos kūrimo įrankių (Minhas U. I. et al., 2014).

Specifinės taikomosios programos užtikrina didelį našumą ir mažą lankstumą, o bendro naudojimo procesoriai suteikia didelį lankstumą ir

vidutinį našumą. Žinoma, kad bendro naudojimo procesoriai yra sukurti atlikti daugybę operacijų, tačiau jie nėra tinkami intensyviai apdoroti grafinį vaizdą, nes naudoja serijinius apdorojimo blokus. GPU skiria daug daugiau tranzistorių apdoroti duomenis nei juos spartinti ir kontroliuoti jų perdavimą. GPU yra sukurti taip, kad vykdytų skaičiavimus lygiagrečiai; jie turi didelį kiekį branduolių, kurie sukurti apdoroti daug užduočių vienu metu. GPU yra palyginti lankstūs ir juos nėra sudėtinga programuoti naudojant aukšto lygio programavimo kalbas, taip pat jie jau yra pritaikyti naudoti ir nešiojamuose įrenginiuose. Jie taip pat naudojami kaip galingi pagalbininkai dirbant su specifinėmis programomis, kurioms reikia didelio našumo kompiuterijos. Įterptinėse sistemose skaitmeniniai signalų procesoriai (DSP) ir vartotojo programuojamos loginių elementų matricos (FPGA) yra pirmaujančios procesorių technologijos. Iš esmės FPGA, skirtingai nuo procesoriaus, yra perkonfigūruojamas įrenginys. Kalbant neprofesionalia kalba, galima teigti, kad FPGA yra energetiškai efektyvesnis sprendimas, o GPU yra itin efektyvus ekonomiškai. Individualizuotos konfigūruojamos architektūros naudojamos apdoroti didelius kadrų skaičius tada, kai dirbama su aukštos raiškos grafiniais vaizdais; tai leido sukurti efektyvų jų apdorojimą realiu laiku ir išlaikė mažą energijos suvartojimą, palyginti su CPU ir GPU sprendimais (Chelva M. S. ir Sharanappa V. H., 2016).

Remiantis atlikta literatūros analize (Véstias M. P. ir Horácio C. N., 2014; Jones D. H. et al., 2010; Grozea C. et al., 2010; Minhas U. I. et al., 2014) GPU, FPGA ir CPU yra apibendrinti žemiau pateiktoje lentelėje:

2 lentelė. FPGA, GPU ir CPU procesorių palyginimas

FPGA	**CPU**	**GPU**
Lygiagretus	Nuoseklus	Lygiagretus
Labai greitais duomenų apdorojimas realiu laiku	Duomenų apdorojimas skiriasi nuo priklausomybių	Greitais duomenų apdorojimas realiu laiku
Veikia prastai su slankiojančio kablelio operacijomis	Universalus	Veikia puikiai su slankiojančio kablelio operacijomis
Reikia sąsajos su programine įranga	Nereikia sąsajos su programine įranga	Nereikia sąsajos su programine įranga
Palyginti mažiau lankstus, bet pasiekiamas aukštas našumas	-	Labai didelis programavimo lankstumas

Galima daryti išvadą, kad FPGA yra energetiškai efektyvesnis, o GPU ekonomiškai yra itin efektyvus. FPGA yra labiau pritaikyti atlikti fiksuoto

taško operacijas tuo pačiu metu su artimu techninei įrangai programavimo sprendimu, o GPU yra optimizuotas lygiagrečioms plaukiojančio taško operacijoms, naudojančioms tūkstančius mažų branduolių ir nereikalaujančioms specialios sąsajos. Dirbant su didelio dažnio prekyba, neįmanoma išvengti plaukiojančio taško operacijų, nes reikia apdoroti kainas ir jas normalizuoti. Aukščiau išvardyti faktoriai lėmė tai, kad toliau šiame tyrime bus naudojama GPU architektūra, kuri bus skirta adaptuoti didelio dažnio prekybos sistemą.

2.2 GPU pasirinkimas (CUDA)

Bendrosios paskirties grafikos apdorojimo įrenginiai (GPU) paskutiniais metais yra dažnai tyrinėjami dėl didelio našumo skaičiavimų baigtinių skirtumų, dalelių, Boltzmann, baigtinių elementų ir metodų ribinių elementų (Owens J. D. et al., 2007). Norint paspartinti skaičiavimus, pasitelkiant GPU, buvo sukurta nauja API, kuri vadinasi CUDA (vieninga įrenginio architektūra). CUDA programuotojui leidžia kurti aukštesnės paradigmos kodą, kuris veikia GPGPU (Owens J. D. et al., 2007).

Tipiška CUDA programa gali būti vykdoma dviem būdais – sinchronizuotai ir asinchronizuotai. Naudojant sinchronizuotą būdą, CPU paleidžia pagrindinį programos kodą, kuris yra vykdomas iš eilės. Paleidus procedūrą, kuri skirta vykdyti skaičiavimus GPU, GPU perims programos valdymą, taip sukurdamas didelį kiekį gijų ir jų blokų, lygiagretindamas skaičiavimus. CPU bus sustabdytas tada, kai GPU baigs skaičiavimus. Antrojo būdo atveju, CPU paleidus procedūrą, kuri leidžia GPU skaičiavimus, procedūra veiks toliau net ir vykstant skaičiavimams GPU (Lou X., 2012).

Kiekvienos naujos kartos *Nvidia* įrenginiai, kurie palaiko CUDA, turi unikalų versijos pavadinimą (CC), arba skaičiavimo pajėgumą. Skaičiavimo pajėgumas yra sudarytas iš dviejų skaičių, kurie nurodo fizinius ir loginius *Nvidia* GPU parametrus. Pirmasis skaičius nurodo bendrą GPU kartą (apibūdina instrukcines galimybes), o antras skaičius nurodo įrenginio branduolius. 3 lentelėje nurodytos visos *Nvidia* GPU kartos iki 2019 metų.

3 lentelė. *Nvidia* GPU versijos ir jų skaičiavimo galimybės (CC)

GPU architektūra	**Išleidimo metai**	**CC**
Nvidia Tesla	2007	1.x
Nvidia Fermi	2010	2.x
Nvidia Kepler	2012	3.x
Nvidia Maxwell	2014	5.x
Nvidia Pascal	2016	6.x
Nvidia Volta	2018	7.x

Pagrindinis GPU privalumas yra jo galimybė lygiagretinti procesus. Norint atlikti grafinius skaičiavimus, nebūtina itin kontroliuoti ir daug tarpusavyje komunikuoti, palyginus su atliekamų skaičiavimų kiekiu (Owens J. D. et al., 2007). GPU yra taip sukurti, kad spręstų problemas, kurias galima perorganizuoti lygiagrečiai su dideliu aritmetiniu intensyvumu. Tipinis GPU yra sudarytas iš srautinių procesorių (SP) masyvų, kurie yra išdėlioti srautiniuose multiprocesoriuose (SM) (Kirk D. B. ir Hwu W. M., 2010).

GPU susideda iš daugiabranduolinių skaičiavimų, šie bendrosios paskirties grafikos apdorojimo įrenginiai programuojami viena pagrindine daugiabranduole programavimo paradigmos instrukcija (SIMT). GPU jau kurį laiką yra naudojami spręsti įvairias skaičiavimų problemas įvairiose aplikacijose (Labaki J. et al., 2011).

CUDA iš esmės yra C programavimo kalbos plėtinys. Ji yra pritaikoma visoms *Nvidia* sukurtoms grafinėms plokštėms, turinčioms jos architektūrą. Procesorių branduolių, branduolių blokų ir jų tinklo koncepcijos yra trys abstrakcijos, kurios dažnai minimos kaip CUDA programavimo paradigmos. Procesoriaus branduolys yra vienas iš daugelio komponentų, atsakingų už jam duotos instrukcijos vykdymą tik su vienu duomenų paketu. Grupė procesoriaus branduolių, remiantis SIMT paradigma, dirba lygiagrečiai vykdydama tą patį operacinės sistemos procesą tai pačiai duomenų grupei. Procesoriaus branduoliai yra suskirstyti į blokus. Kiekvienas iš jų paleidžiamas kiekvieno SM, esančio GPU: branduoliai, esantys bloke, gali dalytis duomenimis per SM bendrą atmintį ir jie gali būti sinchronizuojami tam tikru proceso metu. Branduolių blokai yra supinti į tinklą, kuris išskirsto juos per visus SM, esančius GPU. Šis blokas gali būti organizuojamas kaip vienadimensis, dvidimensis ir tridimensis masyvas; šiuo atveju CUDA suteikia kintamąjį, kuris indeksuoja kiekvieną branduolį, esantį bloke. Analogiškai ir branduolių blokų tinklai gali būti vienadimensiai,

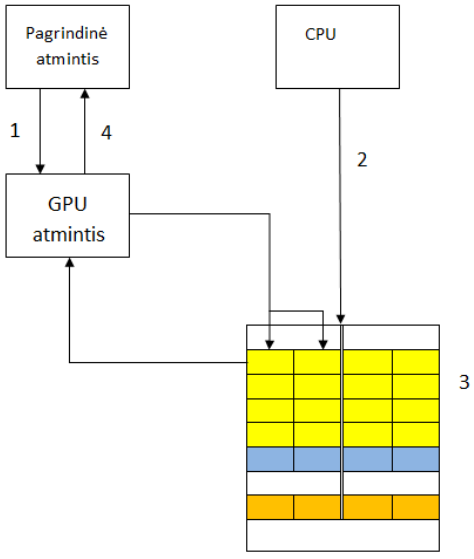
dvidimensiai ir tridimensiai masyvai. Branduolių blokai, esantys tinkle, taip pat gali būti indeksuojami (Labaki J. et al., 2011).

Procesų inicijavimas ir serijiniai skaičiavimai yra atliekami CPU. Tada duomenys ir kodas, skirtas lygiagretinti, yra siunčiamas į grafinę kortą. Žemiau esantis paveikslėlis iliustruoja tipinį CPU – GPU organizavimą. Kiekvienam lygiagrečiam skaičiavimui priskiriami procesoriaus branduoliai. GPU turi šias skirtingas atmintis:

a) globaliąją, kuri yra pati didžiausia ir matoma visiems branduolių blokams, esantiems tame pačiame tinkle;
b) bendrąją, kuri yra branduolių blokų tinkle ir matoma visiems branduoliams, esantiems bloke.

Bendroji atmintis yra daug spartesnė už globaliąją, tačiau yra daug mažesnė. GPU bendroji atmintis padeda paspartinti skaičiavimus, nes yra skirta CUDA blokams ir yra arčiau branduolių. Kai skaičiavimai yra baigti, jų rezultatai siunčiami atgal į CPU (Adil S. H. ir Qamar S., 2009).

Žemiau esančiame paveikslėlyje pavaizduota, kaip veikia CUDA. Duomenys, reikalingi lygiagretiesiems skaičiavimams, yra gauti iš vartotojo naudojant CPU. Atmintis šiems duomenims yra išskiriama CPU ir GPU. Kai reikalingi duomenys yra gauti, jų kopijos iš CPU atminties keliauja į GPU atmintį. Vėliau CPU siunčia instrukcijas į GPU, kad atliktų lygiagrečius skaičiavimus.



6 pav. Procesų judėjimas CUDA aplinkoje (Adil S.H. ir Qamar S., 2009)

1. Duomenų kopijavimas iš CPU į GPU atmintį.
2. CPU siunčia instrukcijas į GPU.
3. GPU vykdo lygiagrečius skaičiavimus.
4. Skaičiavimo rezultatai keliauja iš GPU į CPU (Nambia P. P. et al., 2014).

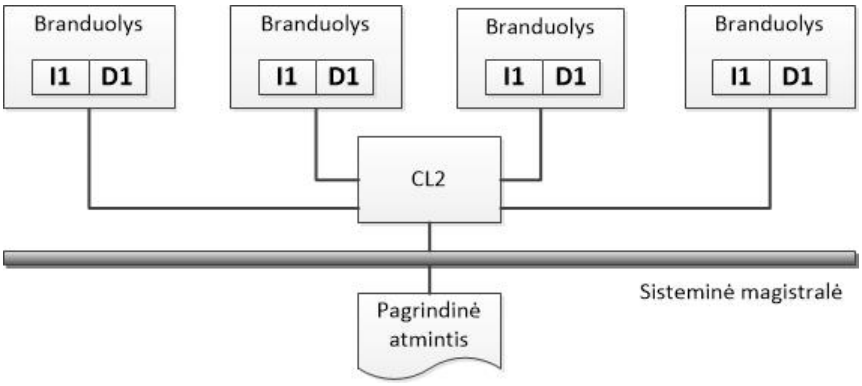
Duomenis, esančius CPU globaliojoje atmintyje, ilgiau trunka apdoroti nei duomenis, esančius GPU globaliojoje atmintyje. Norint visiškai išnaudoti GPU ir jos atminties galimybes, plėtoti didelio našumo prekybos strategijas, šiame tyrime buvo pasirinkta naudoti CUDA CPU-GPU sistemos architektūrą.

Prieš pradedant kalbėti apie techninę įrangą, skirtą CUDA, šioje dalyje bus aptarta, kaip konfigūruoti CPU ir GPU sistemas, kad jos sąveikautų tarpusavyje. Dalis, kurią šioje sistemoje valdo CPU, yra vadinama *host*. Prie jos prijungtas GPU yra vadinamas *device* (įrenginys). Kiekviena tokia sistema privalo turėti šiuos komponentus:

- pagrindinę plokštę,
- CPU lustą,
- operatyviąją atmintį, skirtą CPU,
- GPU.

Dabartiniai CPU prie įvesties / išvesties centro yra jungiami atitinkama CPU gamintojo jungtimi: *Intel* jungimas QPI (*QuickPath Interconnect*) ir *AMD* jungimas HT (*HyperTransport*). Abi technologijos leidžia dalytis CPU spartinamąja atmintimi, kuri leidžia paspartinti skaičiavimus ir didina užklausų greitį (Foley D. ir Danskin J., 2017).

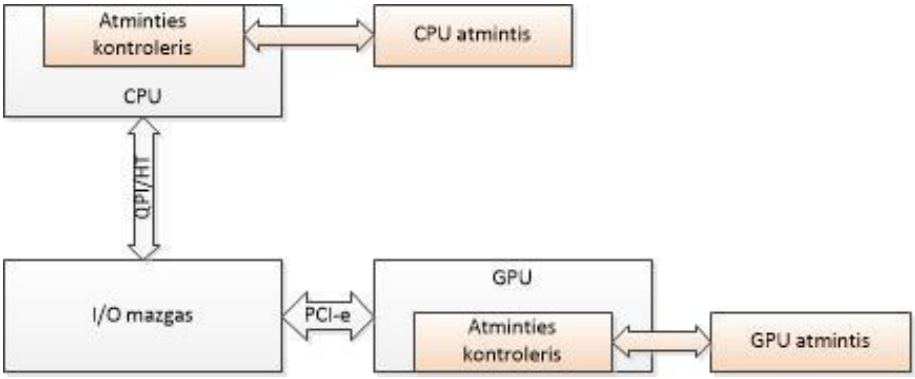
Dauguma dabartinių CPU (*Intel, AMD* ir *IBM*) naudoja multiprocesorių architektūrą ir kiekvienas procesoriaus branduolys turi savo pirmo lygio spartinamąją atmintį (CL1). Tokio tipo multiprocesorių architektūra taip pat turi antro lygio spartinamąją atmintį (CL2) ir pagrindinę atmintį RAM. CL1 dažniausiai yra suskaidytas į instrukcijų ir duomenų spartinamąją atmintį ir CL2 dažniausia yra sujungta. Keturių branduolių *Intel* procesoriaus spartinamosios atminties pavyzdys pateiktas 7 paveikslėlyje.



7 pav. *Intel* tipo CPU spartinamoji atmintis

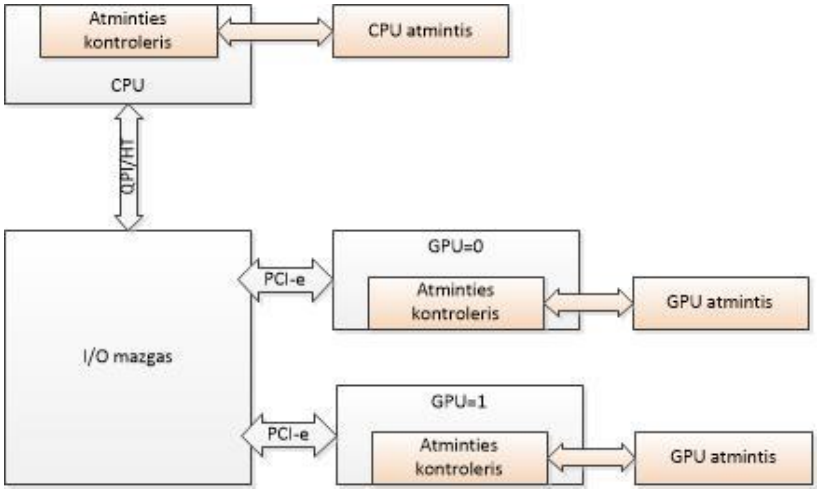
Sistemoje, naudojančioje multiprocesorių CPU ir daugiaprocesorių GPU, kurios palaiko CUDA, algoritmas visada pradedamas CPU. Algoritmo inicializavimas ir nuoseklios jo dalys yra vykdomos CPU. Vėliau duomenys ir lygiagrečios algoritmo dalys siunčiamos į GPU plokštę. GPU naudoja skirtingo tipo atmintis: globalioji atmintis yra didžiausia prieinama visiems skaičiavimo blokams ir matoma visiems GPU procesoriams ir jų tinklui; bendroji atmintis yra naudojama skaičiavimo bloko ir pasiekiama tik jam priklausantiems branduoliams. Bendroji atmintis yra labai greita, tačiau daug mažesnės talpos už globaliąją atmintį (Asaduzzaman A. et al., 2014). GPU bendroji atmintis padidina našumą, nes yra arčiau procesorių branduolių ir dedikuota tam tikram CUDA blokui.

Atskiros GPU vaizdo plokštės turi savo operatyvinę atmintį, kuri yra pačioje plokštėje. GPU taip pat naudoja savo integruotus į kiekvieną lustą kontrolerius. Tokia paprasta sistema, kurioje veikia vienas CPU ir vienas GPU, pavaizduota 8 paveikslėlyje.



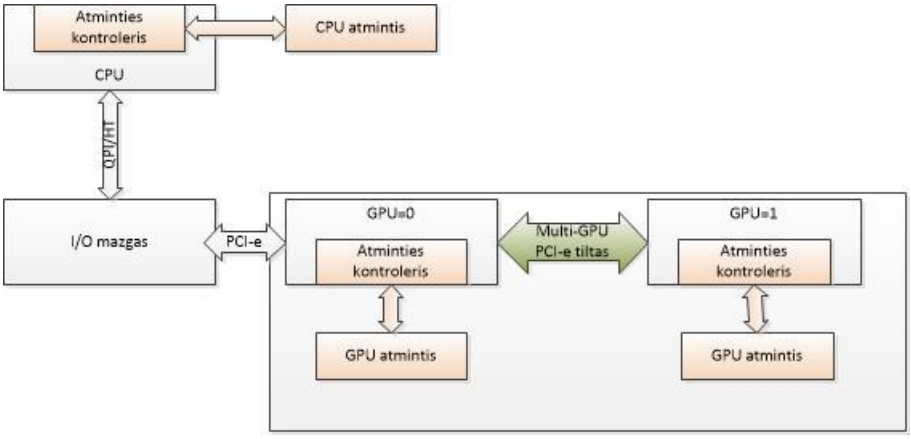
8 pav. CPU ir vieno GPU sujungimas per pagrindinę plokštę PCI-e jungtimi

Galimybė naudoti daugiau nei vieną GPU sistemoje galima tik per pagrindinę plokštę, nes dažniausiai turi daugiau nei vieną PCI jungtį. Taip pat serveriniai sprendimai, skirti naudoti daug GPU, dažniausiai jau būna paruošti pačių GPU gamintojų. Tinkamas sistemos su daug GPU panaudojimas priklauso nuo to, kaip jie prijungti prie sistemos. Yra du galimi prijungimo būdai. Pirmasis – kai GPU jungiami PCI jungtimis kaip atskiri įrenginiai (Foley D. ir Danskin J., 2017).



9 pav. CPU ir kelių GPU sujungimas per pagrindinę plokštę PCI-e jungtimi

Antras variantas yra turėti fizinį tiltą, kuris jungtų atskirus GPU. Taip prijungti GPU veikia kaip vienas ir yra efektyvesni negu sujungiami atskiromis PCI jungtimis.



10pav. CPU ir kelių GPU sujungimas per pagrindinę plokštę PCI-e tilto jungtimi

Du pagrindiniai GPU gamintojai *Nvidia* ir *ATI* siūlo savo sprendimus, skirtus sujungti daugiau negu vieną GPU. *ATI* siūlomas sprendimas yra *CrossFire,* o *Nvidia* yra *Scalable Link Interface* (SLI) ir leidžia prijungti iki keturių GPU vaizdo plokščių.

Šitaip prijungtos GPU plokštės komunikuoja su CPU per vieną PCI jungtį taip, kaip pateikta aukščiau esančiame paveikslėlyje (žr. 10 pav.). Dėl to perduodamų duomenų kiekis yra 16 Gb per sekundę, nes tai yra didžiausias PCI 3.0 pralaidumas. Būtent ši dalis yra silpnoji tokios sistemos vieta. Todėl tokią sistemą galima naudoti, jei naudojamos programos reikalauja mažo CPU ir GPU duomenų perdavimo ir nereikalauja didelio atskirų GPU tarpusavio bendravimo. Pagrindinis SLI tikslas yra sinchronizuoti atskirų GPU duomenų buferius. Kai reikia perduoti daug duomenų tiek iš CPU į GPU, tiek tarp pačių GPU, reikia naudoti CPU ir GPU sistemą, kuri ir pasirinkta šiame tyrime (Foley D. ir Danskin J., 2017).

2.2.1 CUDA vykdymo modelis

Pirmiausia reikėtų pradėti nuo CUDA vykdymo modelio apibūdinimo. Pagrindinis šio modelio komponentas yra branduolio funkcija. CUDA sistemoje, kurioje naudojama C/C++ programavimo kalba, branduolys yra kaip C funkcija, kuri vykdoma N kartų lygiagrečiai N CUDA branduolių (Klöckner A. et al., 2012). Vykdant kiekvieną branduolio funkciją, kiekvienas branduolys yra pažymimas pagal indeksą, vadinamą *threadIdx*. Kiekvienas branduolio indeksas dažnai nurodo tam tikrą užduotį, kurią jis turės įvykdyti. Gilinantis į modelį, nurodytina, kad branduoliai yra skirstomi į branduolių blokus. Kaip ir branduoliai, kiekvienas branduolių blokas turi savo indeksą, vadinamą *blockIdx*. Kad būtų patogiau, branduolių blokai gali būti vienos (x), dviejų (x, y) ir trijų (x, y, z) dimensijų. Papildomai branduolių blokai dar gali būti prijungti prie tinklų, kurie taip pat gali būti vienos, dviejų ir trijų dimensijų. Tokio tipo hierarchija sukuria patogią sistemą, galinčią lengvai apdoroti vektorius, matricas ir daugiamates matricas (Nvidia, 2007).

Nvidia CUDA vykdymo modelis yra labai panašus į SIMD architektūrą. Kiekviena instrukcija, kuri yra vykdoma CUDA, yra SIMD tipo instrukcija. Tai reiškia, kad CUDA visą laiką naudojasi vektorine informacija. Šio vektoriaus dydis vadinamas SIMD pločiu. Jis priskiria tam tikrą kiekį branduolių, kurie lygiagrečiai veikia pagal tą pačią instrukciją. SIMD plotis yra 32, todėl jis dirba su 32 bitų informacija, CUDA dirba su tokio pločio

vektoriais. Jei gaunama sudėties instrukcija, CUDA dirba su masyvais, kurie sudaryti iš 32 elementų. Pagal šią instrukciją CUDA prie registro A 32 elementų prideda 32 B elementus ir jų rezultatą išsaugo C registro 32 elementuose (Nvidia, 2007).

2.2.2 CUDA atminties modelis

Norint visiškai išnaudoti GPU galimybes, turi būti užtikrintas tinkamas GPU atminties ir jos branduolių duomenų našumas. Kad tai būtų pasiekta, pirmiausia reikia išsiaiškinti CPU ir GPU sistemos atminties hierarchiją. CPU ir GPU sistema veikia su dviem pagrindiniais atminties tipais:

a) *host* atmintis, kuri yra paprastai valdoma CPU, tipiškai ji yra DRAM atmintis;

b) *device* atmintis, kuri yra GPU vaizdo plokštėje. Ši atmintis turi savo atminčių hierarchiją, kuri bus aptarta kituose šio darbo skyriuose.

Žiūrint iš atminties perspektyvos, tipiškas GPGPU vykdymas susideda iš šių žingsnių:

1) duomenų perkopijavimas iš *host* į *device* atmintį;
2) paskirtos GPU užduoties vykdymas;
3) gauto rezultato nukopijavimas iš *device* į *host* atmintį.

Kaip jau minėta prieš tai šiame skyriuje, visi duomenys perduodami per PCI, kuri tampa šios sistemos silpnąja vieta. Remiantis GPGPU programavimo bendruomene, teisingam duomenų išdėstymui yra išskirtos trys empirinės taisyklės (Minhas U. I. et al., 2014):

1) gavus duomenis GPU, juos ten ir laikyti;
2) duoti GPU pakankamai darbo;
3) visą dėmesį sutelkti tam, kad būtų galima pakartotinai panaudoti turimus duomenis GPU.

Jei duomenys yra naudojami GPU, tai jie bus laikomi mažo vėlavimo ir mažos apimties atmintyje. Priešingu atveju, jei duomenys saugomi naudoti ateityje, jie laikomi didelio vėlavimo ir didelės apimties atmintyje. Prieš atliekant nuodugnesnę GPU atminties hierarchijos analizę, pateikiamos visos galimos atmintys, jų fiziniai ir loginiai parametrai (Farber R., 2011).

Globali atmintis

Globalioji atmintis yra labiausiai naudojama GPU atmintis. Dalis autorių net visą DRAM atmintį laiko globaliąja. Ši atmintis gali būti pasiekta bet kurio GPU branduolio ir jos gyvavimo laikas yra toks, koks yra CUDA

programos veikimo laikas. Ši atmintis priskiriama ir išvaloma pačio *host*, taikant *cudaMalloc* ir *cudaFree* metodus. Įprastai šioje atmintyje yra laikoma dalis duomenų, kuriuos tikimasi vėliau panaudoti arba perrašyti programos veikimo metu. Kadangi globaliąją atmintį gali pasiekti bet kuris branduolys iš branduolių grupės, problemų kyla dėl to, kad branduolių blokai nėra sinchronizuojami branduolių blokų tinkluose. Toks atminties skaitymų užklausų skaičius iš branduolių (dažniausiai tūkstančiai ar net daugiau) gali būti labai padrikas ir dėl to gali itin kristi našumas. Tačiau didžiausia problema dėl branduolių nenuspėjamumo susiklosto tada, kai didelis jų skaičius siunčia rašymo užklausą į labai mažą globaliosios atminties dalį. Šios globaliosios atminties problemos ir jų sprendimo būdai bus aptarti kituose skyriuose (Farber R., 2011).

Kaip jau buvo minėta, globalioji atmintis yra GPU hierarchijos stuburas ir joje yra laikoma didžioji dalis duomenų. Todėl maksimalus atminties apkrovimo efektyvumas ir optimizavimas yra vienas svarbiausių dalykų. Globaliosios atminties pasiekiamumas ir efektyvumas nulemia visos sistemos našumą. Esant prastam globaliosios atminties efektyvumui, visi kiti spartinimai, naudojant spartinamąsias atmintis ar kt., netenka prasmės (Farber R., 2011).

2.2.3 CUDA techninės įrangos modelis

Ankstesniame skyriuje buvo aptartas CUDA programavimo modelis. CUDA SIMT vykdymo ir CUDA atminties modeliai buvo aptarti iš programuotojo perspektyvos. Buvo paminėti tik keli pastebėjimai apie programavimo modelio ir pačios techninės įrangos ryšį. Kitame skyriuje bus siekiama išsamiau aptarti patį *Nvidia* CUDA techninės įrangos modelį. CUDA SIMT vykdymo modelis sujungia į vieną giją visus 32 loginius branduolius. Ši gija vėliau konvertuojama į branduolių blokus, kurie prižiūri srautinius multiprocesorius (SM), esančius GPU mikroschemose.

Šimtai ar net tūkstančiai CUDA branduolių, arba tiesiog branduolių, yra pagrindiniai darbininkai, atliekantys didžiąją dalį GPU darbo. Šie branduoliai yra panašūs į ALU (aritmetinius loginius elementus) ir FPU (plaukiojančio taško elementus), esančius CPU. Šiuolaikiniai GPU turi savo plaukiojančio taško ir sveikų skaičių elementus. Todėl GPU gali lengvai atlikti operacijas su 32 bitų plaukiojančio taško ar 16 arba 32 bitų sveikųjų skaičių (IU) tipo duomenimis. Kiekvienas CUDA branduolys turi savo siuntimo kanalą, kuriuo gauna jam paskirtas vykdyti instrukcijas iš SM. Taip

pat kiekvienas branduolys turi ir savo surinkimo kanalus, skirtus gauti informacijai iš registrų. Kai FPU arba IU (sveikojo skaičiaus elementas) atlieka su jam skirtais duomenimis paskirtą instrukciją, tai rezultatas yra laikinai laikomas branduolio eilėje ir laukia tol, kol galės būti perduotas į registrą (Nvidia, 2018).

Praktikoje vienas CUDA branduolys yra fizinis CUDA programavimo modelio branduolio atitikmuo. Instrukcijos, kurios yra paskirtos 32 CUDA programavimo modelio branduoliams, yra paskiriamos būti atliktos 32 fiziniuose CUDA branduoliuose (Nvidia, 2018).

2.3 GPU pritaikymas įprastiniams lygiagretiesiems skaičiavimams

Lygiagretieji skaičiavimai yra skaičiavimų forma, kuria daugybė skaičiavimų atliekama vienu metu, vadovaujantis principu, kad sudėtingiausios problemos gali būti išskaidytos į daug mažesnių uždavinių, kurie gali būti išsprendžiami nepriklausomai vieni nuo kitų. Šis skaičiavimų tipas jau daug metų naudojamas daugiausia dirbant su superkompiuteriais, tačiau pastaraisiais metais vis dažniau pritaikomas namų kompiuteriuose. Šis išpopuliarėjimas siejamas su fiziniais procesorių taktinio dažnio apribojimais dėl didelių energijos sąnaudų ir naujų kompiuterių procesorių, turinčių po keletą šerdžių (angl. *core*), jie atsirado maždaug 2004 metais. Yra keletas skirtingų paskirstytųjų skaičiavimų architektūros klasių, skirstomų pagal tai, kokiame technikos lygmenyje yra realizuotas skaičiavimų lygiagretumas. Galima išskirti tokias grupes: daugiaprocesorės sistemos, daugiabranduolės sistemos, kompiuterių klasteriai, keleto kompiuterių sistemos ir kt. (Nvidia, 2018). Šiame darbe bus nagrinėjamas ganėtinai naujas lygiagrečiųjų skaičiavimų tipas – skaičiavimas naudojantis grafinių vaizdo kortų procesoriais (GPU).

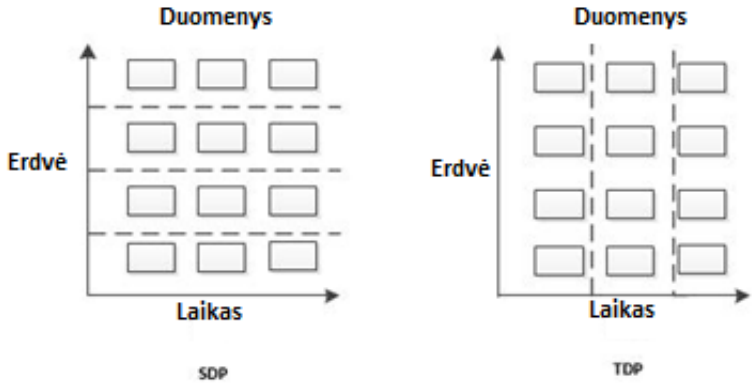
Šiuo metu NVIDIA yra viena iš pirmaujančių GPU gamintojų. Jų sukurta bendrinė skaičiavimo įrangos architektūra (CUDA) leidžia programuotojams išnaudoti visas jų GPU galimybes didelės apimties problemoms spręsti. CUDA sprendimai galimi su visomis NVIDIA vaizdo plokštėmis, kurios sukurtos remiantis *Tesla* architektūra. NVIDIA *Tesla* architektūra yra sukurta naudojant kintamus masyvus, sudarytus iš keičiamo srauto multiprocesorių (SM). Vienas *Tesla* multiprocesorius yra sudarytas iš aštuonių procesoriaus branduolių (SP), dviejų specialiųjų funkcijų, procesorių instrukcijų rinkinio ir vidinės atminties. SM kuria, valdo ir vykdo funkcijas skirtinguose branduoliuose su nuliniu planavimo apkrovimu. Šiuo

būdu kiekvienas branduolys dirba tik su jam skirtu duomenų paketu. GPU galima išskirti dvi skirtingas atminties rūšis: vidinę ir įrenginio. Bendroji atmintis yra viena iš vidinių atminčių, kurią naudoja visi multiprocesoriuje esantys branduoliai. Šios atminties sparta prilygsta L1 spartinamajai atminčiai, esančiai CPU. Įrenginio atmintis yra didelio greičio DRAM atmintis su didesniu vėlavimu nei vidinė atmintis (dažniausiai iki šimto kartų lėtesnė). Įrenginio atmintis dar dalijama į rašymo ir skaitymo, spartinamąją atmintį (globalioji ir lokalioji) ir skirtą tik skaityti. Naudodamiesi CUDA sistemų kūrėjai gali programas papildyti užklausomis lygiagretiems branduoliams. Branduolio kodas tokiu atveju yra vykdomas grupės branduolių, susietų su GPU multiprocesoriumi. Branduoliai yra sugrupuoti į 1D, 2D ir 3D blokus, kurie dar yra sujungti į 1D arba 2D tinklą (Spampinato D. G. ir Elstery A. C., 2009).

Taikant CUDA programavimo metodą, labai svarbu yra teisingai padalyti atskiriems branduolių blokams ir tinklams naudojamus duomenis. GPU atminties architektūra yra sudėtinga, nes yra daug atminties tipų ir lygių GPU kortoje. GPU esanti (GGDR) DRAM dažniausiai įvardijama kaip globalioji ir pati svarbiausia atmintis. Į globaliąją atmintį dedami duomenys, kuriuos turės pasiekti visi aktyvūs GPU procesoriai. Skirtingų blokų procesoriai negali tarpusavyje komunikuoti ar būti sinchronizuoti, tačiau to paties bloko procesoriai gali naudotis jiems priklausančia bendrąja atmintimi, kuri yra mažesnė už globaliąją, tačiau yra greičiau pasiekiama. Dar yra papildoma spartinamoji atmintis, skirta tik skaityti konstantas ir tekstūras bei specifines GPU užduotis. GPU taip pat turi komunikuoti su standartine CPU RAM atmintimi taip, kaip ir bet kuri kita sistema. Efektyvus GPU programų naudojimas reikalauja teisingo visų atminčių naudojimo. Pirmiausia visi duomenys yra talpinami RAM atmintyje, kuri pasiekiama CPU ir GPU. Paskui dalis duomenų, kuri turi būti apdorojama GPU, keliauja į GPU globaliąją atmintį. Branduoliai gali būti vykdomi tik su duomenimis, esančiais tam tikrose GPU atmintyse. Galutinis skaičiavimų rezultatas išsaugomas globaliojoje GPU atmintyje ir keliauja į CPU atmintį tam, kad būtų atlikti paskutiniai funkcijos veiksmai. Šie duomenų perdavimai iš GPU atminties į CPU atmintį ir atvirkščiai užima dalį procesoriaus ciklų, todėl reikia atsižvelgti į tai ir skirti papildomo laiko šiems veiksmams atlikti.

Duomenų lygiagretinimas yra svarbi lygiagretinimo proceso technikos dalis, kuri pasitelkia lokalumo principą. Lygiagretinant duomenis, programa suskaidoma į atskiras dalis, kurios vykdo tą pačią instrukciją pagal skirtingus

duomenis. Masačusetso technologijų instituto (MIT) tyrėjai pateikia dvi duomenų lygiagretinimo strategijas: erdvinis duomenų skaidymas (SDP) ir laikinasis duomenų dalinimas (TDP). Taikant SDP strategiją ir erdvinį indeksavimą, duomenys padalijami procesams (11 pav.). Šio tipo strategija gali būti naudojama tada, kai turimi didelių dimensijų erdviniai duomenys su mažai priklausomybių. Norint sudaryti sąlygas duomenų komunikacijai ir sinchronizacijai, būtinos papildomos instrukcijos. Duomenų padalijimas su lygiagretintu algoritmu tampa paprastas tol, kol algoritmas atlieka tą pačią funkciją branduolių blokuose su visais erdviniais indeksais (Asaduzzaman A. et al., 2014; Hoffmann H. et al., 2009).



11 pav. Dvi galimos lygiagretinimo strategijos

Taikant TDP strategiją, pasitelkiami laikini indeksai, dalijantys duomenis procesams. Atliekami kiekvieno proceso ir visų erdvinių indeksų, susietų su jų laikinais indeksais, kaip pavaizduota 11 pav., skaičiavimai. Šiuo atveju komunikacija tarp duomenų bus priklausoma nuo naudojamos lygiagrečios aplikacijos. Tipiniu TDP atveju algoritmas duomenis naudoja nuo priskirto laikino indekso, o procesas vykdo visas priskirtas instrukcijas su tais duomenimis. Šio tipo strategija naudojama tada, kai yra didelės apimties laikinos dimensijos duomenys su mažai priklausomybių. Eksperimentai atskleidė, kad, taikant TDP strategijas, pasiekiamas geriausias pralaidumas, o pasitelkus SDP strategijas, pasiekiamas mažiausias vėlavimas, tačiau prarandama kokybė (Asaduzzaman A. et al., 2014).

Ankstesniuose skyriuose buvo aptarta sudėtinga, daugiasluoksnė sistema, kuri atspindi CUDA GPU sistemos architektūrą. Buvo nustatyta, kad daugeliu atvejų programos, veikiančios šioje sistemoje, priklauso nuo programuotojo ir nuo jo požiūrio į:

- CUDA programavimo modelį, t. y. koks lygiagretinimo lygis pasiektas ir iki kokio lygio yra išvengta skirtingų kodų vykdymo vienoje SIMD grandyje;
- CUDA atminties modelį, t. y. kaip efektyviai yra išnaudojama DRAM atmintis, kaip pasiekiama ir išnaudojama L1 / L2 spartinamoji atmintis, ar yra naudojama ir kaip efektyviai pasitelkiama bendroji atmintis ir t. t.;
- CUDA techninės įrangos modelį, t. y. kaip įsivaizduojamas branduolių blokų tinklas paskirsto darbą tarp visų SM, esančių GPU.

Nors yra rekomenduojami universalūs būdai, kaip programuotojui pasiekti didelį sistemos efektyvumą (pvz., veiksmingas GPU atminties panaudojimas), tačiau vienam sprendimui įgyvendinti yra daug būdų, todėl žinoti jų efektyvumo iš anksto praktiškai neįmanoma. Net skirtingų architektūrų GPU gali vykdyti tą patį kodą su skirtingais našumais ir tai nepriklauso nuo paties programuotojo. Taip paaiškėjo, kad kiekvieno sprendimo įgyvendinimas ir optimizavimas labai priklauso nuo naudojamų įrenginių.

2.4 Kodo vektorizacija, daugiamatinės matricos ir lygiagretūs branduoliai

Vektorizavimas yra procesas, kai algoritmas yra transformuojamas iš darbo su pavieniais duomenų elementais į programą darbui su duomenų masyvais. Kompiuterių moksle vektorizacija yra procesas, kuriame algoritmas konvertuojamas iš skaliarinės būsenos. Joje jis atlieka operaciją su viena pora duomenų, į vektorinę būseną, kurioje viena instrukcija gali būti vykdoma su duomenų vektoriumi. Šios transformacijos metu, įterpiant papildomas instrukcijas ir saugiklius, reikia kontrolės perdavimą paversti į duomenų perdavimą. Taip yra lygiagretinama programa, kurioje viena instrukcija taikoma daugeliui duomenų. Baigus vektorizaciją, skaičiavimai tampa efektyvesni ir daug greitesni – taip padidinamas programos našumas (Vaughan C. T. et al., 2018; Li P. et al., 2015).

Tokio tipo programa gali veikti tokioje sistemoje, kuri palaiko vektorines instrukcijas. Norėdamas atlikti tokią transformaciją, programuotojas gali pats parašyti vektorizuotą kodą arba naudoti pusiau automatinius kompiliatorius, kuriuose tik nurodo vektorizuotinas dalis. Programos našumas ir efektyvumas priklauso nuo kodo struktūros (Li P. et al., 2015).

Paruošus vektorizuotą kodą, jį galima perkelti į GPU. Prieš perkeliant kodą į GPU, reikia jį atidžiai patikrinti, rasti butelio kaklelius ir kuo labiau juos optimizuoti. Taip pat kodas turėtų dirbti su pakankamai dideliais

duomenų masyvais, nes tik tokiu atveju GPU ir lygiagretinimo efektyvumas išnaudojamas visiškai (Perino F., 2014).

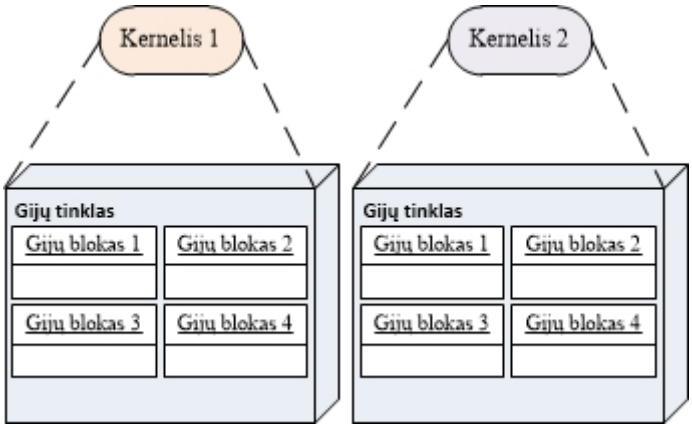
Toliau norint dar labiau optimizuoti ir dar labiau padidinti našumą, reikia sumažinti duomenų perdavimų laiką arba duomenų perdavimų kartų skaičių. Yra keli būdai, kaip tą pasiekti:

- kuo labiau naudoti GPU palaikomas funkcijas;
- kintamuosius kurti iš karto GPU viduje (Bogdan O. et al., 2012).

Visa tai įgyvendinus, labai svarbu saikingai naudoti duomenų masyvų indeksavimą, nes kiekvienas ciklas, indekso paieška, kuri pradedama GPU, po to turi būti siunčiama į CPU ir atgal. Taip yra todėl, kad tik CPU atlieka šiuos veiksmus. Taip vykdomas papildomas duomenų perkėlimas, kuris užima daug laiko.

Aplikacijos yra sudarytos iš gamintojo – vartotojo branduolių sujungimo (Sugerman J., 2009). Lygiagrečiai esantys branduoliai yra semantinis pagrindinių blokų rinkinys, sugrupuotas laikinai, pasikartojantis daug kartų ir gali būti vykdomas vienu metu neiškraipant rezultato. Jį sudaro įvairios programinės struktūros, apimančios ciklus, rekursijas ar bibliotekų iškvietimus. Šie branduoliai sudaro didžiąją programos vykdymo laiko dalį (Uhrie R. et al., 2020).

Lygiagretus programavimas su CUDA apima tiesioginį SAXPY rutinos, apibrėžtos BLAS tiesinės algebros bibliotekoje, įgyvendinimą tiek nuosekliai, tiek lygiagrečiai. Atsižvelgiant į vektorius x ir y, turinčius n slankiojo kablelio skaičių, jis atnaujina y kas $\alpha x + y$. Serijinis įgyvendinimas yra paprasta kilpa, apskaičiuojanti vieną y elementą kiekvienoje iteracijoje. Lygiagretusis branduolys veiksmingai vykdo kiekvieną iš šių nepriklausomų iteracijų lygiagrečiai, priskirdamas atskirą giją kiekvienam y elementui apskaičiuoti (Garland M. et al., 2008).



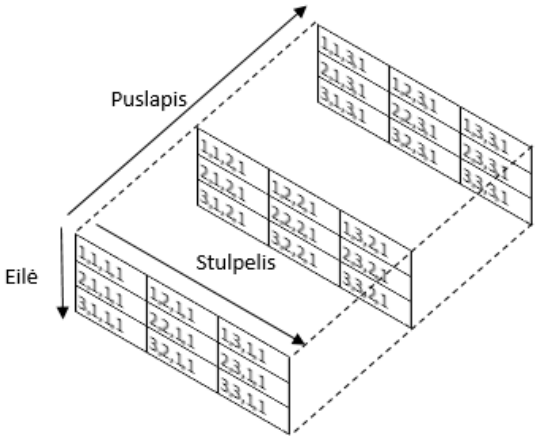
12 pav. Lygiagretūs branduoliai

CUDA programa yra perorganizuota į pagrindinę programą, susidedančią iš vieno ar daugiau nuosekliųjų gijų, veikiančių CPU, ir vieno ar daugiau lygiagrečių branduolių, tinkamų vykdyti lygiagrečiame apdorojimo įrenginyje, pavyzdžiui, GPU. Branduolys vykdo skaliarinę sekos programą lygiagrečių gijų rinkinyje. Programuotojas šias gijas gali sujungti į gijų blokų tinklą. Gijos, esančios gijų bloke, gali būti tarpusavyje sinchronizuojamos ir naudotis bendra, didelės spartos GPU mikroschemos atmintimi. Gijos iš skirtingų blokų tame pačiame tinkle gali būti koordinuojamos tik atliekant operacijas bendroje GPU atmintyje, kuri yra matoma visoms gijoms. CUDA reikalauja, kad gijų blokai būtų nepriklausomi, tai reiškia, kad branduolys turi būti teisingai vykdomas, nesvarbu, kokia tvarka vykdomi blokai, net jei visi blokai vykdomi nuosekliai savavališkai, be jokių išimčių. Šis apribojimas dėl gijų blokų, skirtų branduoliui, tarpusavio priklausomybės suteikia galimybę plėsti skaičiavimų kiekius. Jis taip pat reiškia, kad, skirstant lygiagretųjį darbą į atskirus branduolius, svarbiausia atsižvelgti į globalų ryšį ar sinchronizavimą tarp gijų (Garland M. et al., 2008, Uhrie R. et al., 2020).

Algoritmas, kuris naudojamas šiame tyrime, prieš tai buvo naudotas kituose tyrimuose, kuriuose buvo pasitelkti CPU ir GPU, tačiau nebūdavo išvengiama ciklų, masyvų indeksavimo ir jų paieškų. CPU gerai dirba su ciklais ir juos jame galima lygiagretinti tarp branduolių. Tačiau dėl to, kad CPU turi mažą skaičių branduolių, lygiagretinimas yra apribotas. Kaip jau minėta, norint visiškai išnaudoti GPU, reikia kuo mažiau naudoti ciklus, masyvų indeksavimą. Vienas iš būdų, leidžiantis išvengti ciklų, yra naudoti trijų dimensijų masyvus. Anksčiau buvo aptarta, kad GPU atmintis ir jos

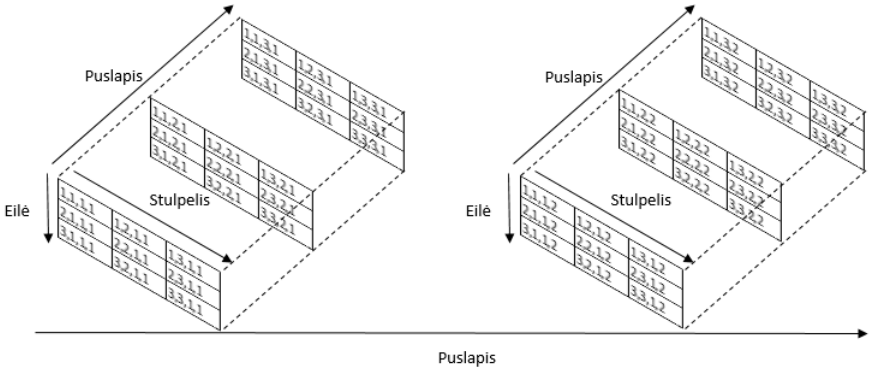
branduoliai gerai dirba su trijų dimensijų informacija. Tyrimo metu buvo naudojamas masyvų konvertavimas į tris dimensijas.

Trijų dimensijų masyvai susideda iš trijų duomenų ašių, kur *x* ir *y* ašys žymi duomenis ir jų kiekius, o *z* ašis žymi šių duomenų puslapių skaičius, t. y. kiek yra to tipo duomenų. Pavyzdžiui, norint tyrimo metu normalizuoti ateities sandorių kainas, jas reikia normalizuoti ir šiuos skaičiavimus galima lygiagretinti. Normalizacijos metu kiekvienas sandoris normalizuojamas apskaičiuojant ir kitų sandorių kainų vidurkius: tokiu atveju *x* ašis atspindi sandorius, *y* ašis – jų kainas pasirinktame lange, *z* ašis atspindi tai, kiek yra sandorių ir kiekvienos *z* ašies pirmas *x* ašies elementas yra vis kitas sandoris. Kitaip tariant, yra dvimačiai masyvai, kurie išdėlioti trečioje ašyje, taip sukuriant trimačius masyvus.



13 pav. Trijų dimensijų GPU masyvas

Konvertuojant duomenis į trimačius masyvus, jie perkeliami į GPU todėl, kad su jais galėtų atlikti skaičiavimus visi CUDA branduoliai; šie duomenys sukuriami naudojant *gpuArray* funkciją. Vienoje iš algoritmų vietų teko pasitelkti keturmačius masyvus. GPU, norėdamas dirbti su šiais keturmačiais masyvais, kiekvieną jo ketvirtos dimensijos elementą, kuris yra trimatis masyvas, turės apdoroti atskirai, nes GPU neveikia su didesniais nei trijų dimensijų masyvais. Nors tai užima šiek tiek laiko, tačiau šis procesas yra greitesnis už ciklų naudojimą. Keturmačiai masyvai naudojami tik porų parinkimo dalyje tada, kai poroms parinkti taikomas kointegracijos metodas.



14 pav. Keturių dimensijų GPU masyvas

Keturmatis masyvas sukuriamas kointegracijos porų parinkimo metodu, kuriuo atliekamas augmentuotas Dickey Fuller testas. Funkcija lygina visas ateities sandorių kainas prekybos lange tada, kai kainos jau yra normalizuotos. Tačiau visus keturmačius masyvus GPU turi perkonvertuoti ir kiekvieną peržiūrėti kaip atskirą trimatį masyvą. Tai lėmė, kad šis porų parinkimo metodas užtrunka šiek tiek ilgiau nei mažiausių kvadratų metodas.

Šie daugiamačiai masyvai buvo vienos svarbiausių dalių (po kodo vektorizacijos) optimizuojant didelio dažnio porų prekybos algoritmą.

2.5 Loginė architektūra

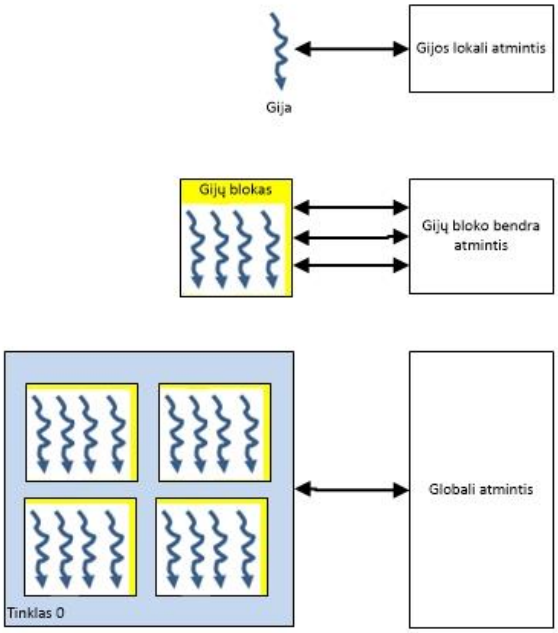
Daugiabranduolinių CPU ir GPU atsiradimas lėmė, kad procesoriai tapo lygiagrečiomis sistemomis. Tada iškilo pagrindinis iššūkis sukurti taikomąją programinę įrangą, kuri padidintų jos lygiagretumą. Šis lygiagretumas turi būti didinamas taip, kad būtų galima panaudoti didėjantį procesoriaus branduolių skaičių – taip, kaip 3D grafikos programose. Jos praplečia savo lygiagretumą naudodamos daugiabranduolinius GPU su kintančių branduolių skaičiais.

CUDA programavimo modelis yra sukurtas taip, kad išnaudotų didėjantį GPU branduolių skaičių. Šis modelis turi atsižvelgti į tris pagrindines abstrakcijas – gijų grupių hierarchiją, bendrąsias atmintis ir barjerų sinchronizaciją. Programuotojas, atsižvelgdamas į šias abstrakcijas, gali praktiškai bet kokią užduotį išskaidyti į smulkesnes dalis. Jas galima spręsti atskirai ir lygiagrečiai, naudojant gijų blokus, tas pačias užduotis galima labiau išskaidyti į dar mažesnes dalis, kurios lygiagrečiai apskaičiuojamos gijose, esančiuose blokuose. Šis užduočių išskaidymas išsaugo užduoties

struktūrą, nes gijos gali bendradarbiauti tarpusavyje, spręsdamos kiekvieną papildomą užduotį ir taip sudaro sąlygas keisti ir didinti / mažinti automatinį gijų tinklą. Kiekvienas gijų blogas gali būti priskirtas bet kuriam multiprocesoriui (SM) esančiame GPU, bet kokia tvarka (vienu metu ir iš eilės). Taip CUDA programa gali naudoti nepriklausomą kiekį multiprocesorių ir paleidimo sistemai reikia žinoti tik absoliutų multiprocesorių kiekį, esantį GPU (Nvidia, 2019).

Kiekviena programa, kuri vykdoma GPU viduje, yra padalyta į daug gijų. Visos atskiros gijos vykdo tą patį programos kodą, tik su skirtingais duomenimis. Gijų blokas yra sudarytas iš grupės gijų, kurios, naudodamos bendrą bloko atmintį, tarpusavyje bendrauja ir sinchronizuoja savo darbą, kurį galėtų koordinuoti per bendrąją atmintį. Gijų blokai dar gali būti sujungiami į tinklą, kuris sudarytas iš grupės gijų blokų. Atskiros procedūros yra vykdomos sujungiant ir sukuriant gijų tinklą (Ploskas N. ir Samaras N., 2016).

Kiekviena atskira CUDA gija gali pasiekti duomenis, esančius skirtingose atmintyse taip, kaip pavaizduota žemiau esančiame 15 pav. Visos gijos turi tik joms priklausančią lokaliąją atmintį. Visi gijų blokai turi bendrąją atmintį, kurią pasiekia kiekviena gija, esanti tame bloke, o ši atmintis yra pasiekiama tol, kol egzistuoja nurodytas blokas. Taip pat kiekviena gija, esanti gijų blokų tinkle, gali pasiekti bendrą GPU įrenginio globaliąją atmintį.

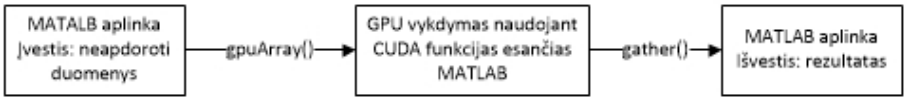


15 pav. GPU atminties hierarchija gijų atžvilgiu (Nvidia, 2019).

Naudodamas CUDA modelį, programuotojas gali pasiekti skirtingas atmintis, esančias GPU, kurias mato veikiančių procedūrų gijos. Gijos, kurios yra sugrupuotos į blokus, gali tarpusavyje bendrauti naudodamos bendrąją atmintį, kurią mato tik tas blokas. Kiekviena gija, priklausanti procedūrai, turi savo vietinę atmintį. Yra trečio tipo globalioji atmintis, kurą pasiekia visos gijos, nepriklausomai nuo gijų blokų. Šių atminčių pasiekiamumas gijoms leidžia itin paspartinti procedūrų skaičiavimus ir užduočių vykdymą (Horrigue L. et al., 2018).

Vienas iš programinės įrangos paketų, kuris palaiko GPU ir CUDA, yra MATLAB. Kiekybinių finansų srityje MATLAB visada buvo viena pagrindinių kalbų (Jacquier A., 2017). Šiame darbe pasiūlyto didelio dažnio statistinio arbitražo strategijų testavimo metodo algoritmas buvo sukurtas būtent naudojant MATLAB, kuris taip pat yra viena dažniausiai naudojamų testavimo (angl. *backtesting*) platformų finansų ir didelio dažnio prekybos srityse (Josephine A. ir Fransson L., 2016). Nors MATLAB nebūdingas mažas vėlinimas, tam labiau tinka tokios programavimo kalbos, kaip *C++*, *C#* ar *Java*, tačiau testavimui ji yra tinkamesnė. Sukurtas MATLAB algoritmas gali būti kompiliuotas į C kalbą arba C vykdomuosius failus, taip pasiekiant mažą algoritmo vėlinimą (Saikia M. J. et al., 2014).

MATLAB turi daug integruotų funkcijų, kurios palaiko *gpuArray* kintamuosius, leidžiančius veikti GPU CUDA funkcijoms. Naudojant *gpuArray*, duomenys yra perkeliami iš CPU arba sukuriami GPU viduje, o *gather* funkcija pasitelkiama duomenis perkelti atgal iš GPU į CPU atmintį. Naudojant šias dvi funkcijas, galima numatyti, kaip duomenis perkelti iš MATLAB aplinkos į GPU atmintį, ar kurti juos tiesiai GPU viduje, taip sumažinant duomenų migravimo tarp skirtingų įrenginių laiką.



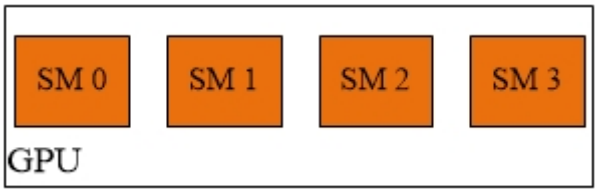
16 pav. Duomenų apsikeitimas MATLAB aplinkoje tarp CPU ir GPU

Yra galimybė naudoti ne tik integruotas funkcijas, esančias MATLAB, bet ir atskiras, kurias gali parašyti programuotojas. Šias funkcijas galima apsibrėžti MATLAB aplinkoje. Taikant šį metodą, net sudėtingas procedūras galima išskaidyti į smulkesnes užduotis ar skaičiavimus. Perkėlus jas į GPU ir pasitelkus CUDA, yra aprašomos atskiros funkcijos, kurios leidžia pasiekti didesnį našumą. Norint paleisti savo parašytą funkciją MATLAB aplinkoje, kuri leistų atlikti GPU daugiabranduolinius skaičiavimus, reikia naudoti vieną iš integruotų funkcijų, kuri sudarytų sąlygas naudoti GPU. Keli šių funkcijų pavyzdžiai yra *arrayfun, pagefun, bsxfun*.

Bsxfun skirta atlikti skaičiavimus su dviem *gpuArray* tipo masyvais. Aprašyta funkcija pritaikoma kiekvienai duomenų masyvo elementų porai, esančiai GPU atmintyje atskirai. Taip skaičiavimai gali būti lygiagretinami, juos vykdant kiekvienoje gijoje atskirai tuo pačiu metu. Taip pat sumažinamas laikas, skirtas perkelti duomenis tarp skirtingų atminčių ir procedūrų iškvietimo (He F. et al., 2015).

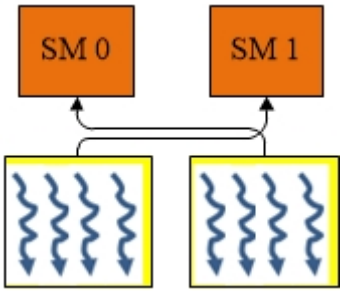
Bsxfun funkcija sukuria procedūrą, kuri turi po vieną giją per išvesties masyvo elementą, tokio dydžio, koks buvo įvesties masyvas. Kai kartu su *bsxfun* yra naudojamas *gpuArray*, sistema žino, jog yra norima paleisti procedūrą GPU viduje. Vadovaujantis šiuo metodu, darbe perkeliami skaičiavimai į GPU, pasitelkiant CUDA, tačiau vietoj įprastų duomenų masyvų naudojami daugiamačiai masyvai, su kuriais galima pasiekti didesnį turimos sistemos našumą.

MATLAB aplinkoje esančios integruotos *arrayfun, pagefun* ir kitos funkcijos veikia kaip procedūros, kurias galima aprašyti tokiu formatu: *f(x)=bsxfun(x)*. Paleidus procedūrą į GPU, paduodami gijų blokai, o jau pats GPU juos paskirstys tarp SM (multiprocesorių). *Bsxfun* (procedūra) pagal kintamųjų dydį nustato, kokio dydžio bus gijų blokas.



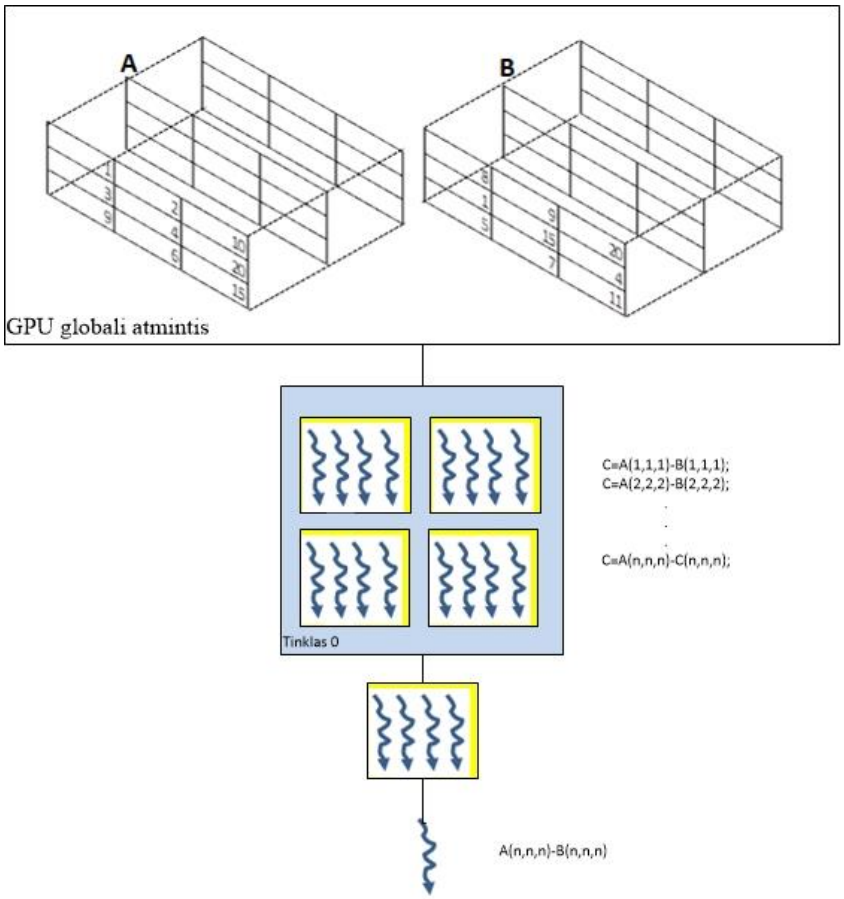
17 pav. GPU SM (multiprocesorių) loginė architektūra

SM gali paleisti daug lygiagrečių gijų, o GPU jau pats paskiria gijų blokus kiekvienam SM.



18 pav. GPU SM ir gijų blokų tarpusavio sąveika

Šiame darbe pasitelkiamos daugiamatės matricos, kuriomis spartinamas didelės apimties duomenų apdorojimas. Daugiamačių matricų perkėlimo į GPU ir jų skaičiavimo loginę architektūrą galima aprašyti taip: tarkime, turime funkciją, kuri yra *f(A,B)=bsxfun(@minus, A(3,3,3), B(3,3,3))*, kur *A* ir *B* yra daugiamačiai *gpuArray* masyvai. Čia *bsxfun* yra procedūra, kuri įjungia funkciją *A-B*. Abu *gpuArray* daugiamačiai masyvai *A* ir *B* pirmiausia yra sukuriami GPU globaliojoje atmintyje.



19 pav. Procedūros ir gijų loginė architektūra naudojant GPU

Nors daugiamačiai masyvai yra trimačio formato, tačiau pačiame GPU jie yra saugomi kaip *column-major*. Paleidus MATLAB aplinkoje procedūros funkciją *f(A,B)=bsxfun(@minus, A(3,3,3), B(3,3,3))*, sukuriami gijų blokai, kuriuose kiekviena gija atlieka tą patį skaičiavimą *A(n,n,n)-B(n,n,n)*. Taip skaičiavimai vykdomi lygiagrečiai tuo pačiu metu visose gijose, esančiose gijų blokuose, kurie sujungti tinkle.

2.6 Skyriaus išvados

Išanalizavus didelio dažnio prekybą ir didelio dažnio kompiuterių technologijas CPU, GPU ir FPGA, buvo nustatyta, kad GPU labiausiai tinka pasirinktam tyrimui dėl savo našumo, ekonomiškumo ir efektyvumo, dirbant su slankiojančio kablelio skaičiais. Atsižvelgus į esamą literatūrą, buvo

pasirinkta GPU CUDA technologija. Ši technologija yra palaikoma ir MATLAB programinės įrangos, pasirinktos šiam tyrimui, kadangi ji tinkama testuoti DDP strategijas. Atrinkti trys lygiagretinimo metodai: kodo vektorizacija, lygiagretūs branduoliai ir daugiamatės matricos. Juos taikant, galima įgyvendinti didelio dažnio prekybą. Išanalizavus GPU CUDA architektūrą ir jos galimus prijungimo prie sistemos metodus, nuspręsta, kad šiame tyrime bus naudojama CPU ir kelių GPU sujungimas per pagrindinę plokštę PCI-e jungtimi. Kadangi tyrime bus naudojamos dvi GPU CUDA vaizdo plokštės ir bus jungiamasi tiesiai prie pagrindinės plokštės per PCI-e jungtį, o ne per PCI-e bridge, informacija tarp GPU ir CPU bus perduodama sparčiau.

3 TYRIMUI NAUDOTŲ STRATEGIJŲ FORMALIZAVIMAS

Kai yra surastos prekybos poros duotam prekybos langui, tada pradedama ieškoti prekybos signalų. Šių signalų parinkimo strategija skiriasi pagal naudojamas strategijas. Pirmoji porų prekybos strategija pirmą kartą buvo pristatyta J. Caldeira ir G. V. Moura darbe (Caldeira J. and Moura G. V., 2013). Jų pasiūlytoje strategijoje pirma reikia nustatyti kriterijų ε_t – rastos finansinių instrumentų normalizuotų kainų poros skirtumą:

$$\varepsilon_t = P(i,t) - p(i,t) \quad (3)$$

Šioje lygtyje ε_t yra vieno finansinio instrumento normalizuotos kainos $P(i,t)$ ir jo poros, kito finansinio instrumento normalizuotos kainos $p(i,t)$ laiku t skirtumas. Nustačius šį skirtumą, skaičiuojamas kriterijus z_t:

$$z_t = \frac{\varepsilon_t - \mu_t}{\sigma_t} \quad (4)$$

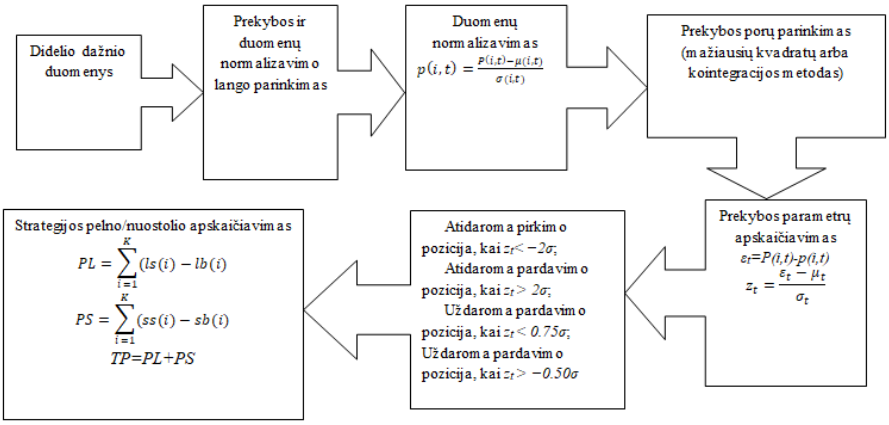
Čia μ_t yra vidurkis, σ_t yra rasto poros skirtumo ε_t praėjusio prekybos lango metu standartinis nuokrypis. Kai kriterijus z_t yra nustatomas, strategija gali pradėti ieškoti prekybos signalų (Caldeira J. and Moura G. V., 2013):

atidaroma pirkimo pozicija, kai $z_t < -2\sigma$;

atidaroma pardavimo pozicija, kai $z_t > 2\sigma$;

uždaroma pardavimo pozicija, kai $z_t < 0.75\sigma$;

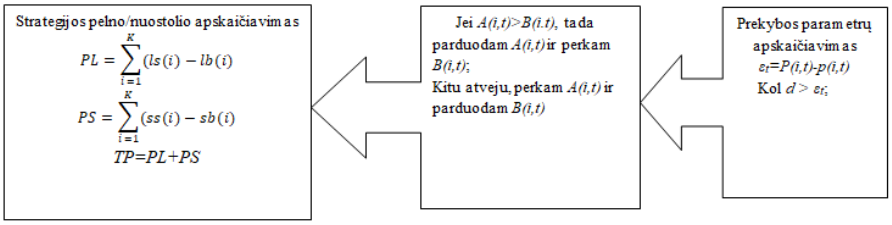
uždaroma pardavimo pozicija, kai $z_t > -0.50\sigma$ (Caldeira J. and Moura G. V., 2013).



20 pav. J. Caldeira ir G. V. Moura porų prekybos strategija

Tam, kad pozicijos nebūtų laikomos atidarytos labai ilgą laiką, yra nustatomas laikotarpis, kiek maksimaliai galima laikyti atidarytą poziciją. Palyginus prekybos strategijas, matyti, kad pradiniai parametrai yra

nustatomi vienodi (prekybos langas, maksimalus periodas laikyti pozicijas, komisinis mokestis). Antroji strategija, kuria remtasi šiame darbe, buvo pateikta M. S. Perlini tyrime (Perlini M. S., 2009). Kaip ir prieš tai buvusioje strategijoje, pirmiausia reikia nustatyti normalizuotos poros kainos skirtumą ε_t ir, atsižvelgus į jį, ieškoti prekybos signalų, lyginant šį skirtumą su prekybos pradžioje paties rinkos dalyvio nustatytu parametru *d*.



21 pav. M. S. Perlini porų prekybos strategija

Nustačius prekybos pozicijas, jos atidarytos laikomos tol, kol sumažėja normalizuotų poros kainų skirtumas arba pasiekiamas laikotarpis laikyti atidarytas pozicijas. Detalesnis prekybos signalų radimas pateikiamas žemiau:

kol $d > \varepsilon_t$,

jei *A(i,t)>B(i.t)*, tada parduodam *A(i,t)* ir perkam *B(i,t).*

Kitu atveju perkama *A(i,t)* ir parduodama *B(i,t)* (Perlini M. S., 2009).

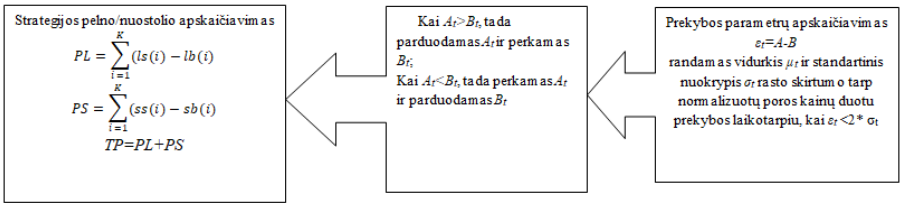
Pagrindinė M. S. Perlini prekybos strategijos logika yra stebėti skirtumą tarp normalizuotų poros kainų ir laukti, kol bus pasiektas kriterijus *d*, o tada pradėti prekybą. Tikimasi, kad kainos grįš į istorinį vidurkį ir skirtumas grįš į buvusią reikšmę tarp kainų. Prekybos strategija išnaudoja tokį skirtumo judėjimą (Perlini M. S., 2009).

Paskutinė strategija remiasi D. Herlemont tyrimu, kuriame jis naudojo porų prekybos strategiją su dienų uždarymo kainomis (Herlemont D., 2013).

Naudojant šią strategiją, pirmiausia nustatomas vidurkis μ_t ir nustatyto normalizuotų poros kainų skirtumo standartinis nuokrypis σ_t duotu prekybos laikotarpiu. Kai abu šie kriterijai žinomi, tai strategija gali pradėti ieškoti prekybos signalų. Kai skirtumas tarp rastos finansinių instrumentų normalizuotų kainų poros *A* ir *B* yra $<2*\sigma_t$:

kai $A_t > B_t$, tada parduodamas A_t ir perkamas B_t.

kai $A_t < B_t$, tada perkamas A_t ir parduodamas B_t (Herlemont D., 2013).



22 pav. D. Herlemont porų prekybos strategija

Pozicijos laikomos atidarytos tol, kol normalizuotų poros kainų A_t ir B_t skirtumas yra < μ_t, arba periodas laikyti atidarytas pozicijas yra pasiektas (Herlemont D., 2013).

Pasibaigus prekybos laikotarpiui, svarbu nustatyti kiekvienos strategijos efektyvumą, kuris sužinomas pagal *pelno / nuostolio* dydį laikotarpio pabaigoje. Siekiant nustatyti *pelną / nuostolį*, apskaičiuojamas kiekvieno prekybos signalo kainų skirtumas, t. y. uždirbta ar ne uždarius kiekvieną poziciją. Šis skaičiavimas detalizuojamas lygtimi (Perlini M. S., 2009; Vaitonis M. ir Masteika S., 2016):

$$PL = \sum_{i=1}^{K} (ls(i) - lb(i) \quad (5)$$

$$PS = \sum_{i=1}^{K} (ss(i) - sb(i) \quad (6)$$

Kintamasis *PL* rodo pelną iš pirkimo pozicijų, o kintamasis *PS* – iš pardavimo pozicijų, kintamais *i* atstoja atskirus prekybos signalus ir laiką, kada pozicijos atidaromos ir uždaromos. Pirkimo pozicijų pelnas apskaičiuojamas nustačius finansinio instrumento kainos, kai jis buvo nupirktas *ls* ir parduotas *lb*, skirtumą Šis skirtumas padauginamas iš komisinio mokesčio *K*, jei jis yra taikomas. Pardavimo pozicijų pelnas apskaičiuojamas, nustačius finansinio instrumento kainos, kada jis buvo parduotas *ss* ir nupirktas *sb*, skirtumą, ir jį padauginus iš komisinio mokesčio *K*. Pabaigoje suskaičiuojamas bendras *pelnas / nuostolis* (Perlini M. S., 2009; Vaitonis M. ir Masteika S., 2016):

$$TP=PL+PS \quad (7)$$

Atlikus visus skaičiavimus, pateikiamas kiekvienos strategijos rezultatas.

3.1 Duomenų normalizavimas

Pirmas šio darbo žingsnis buvo nustatyti, ar didesnio dažnumo duomenys, t. y. nanosekundinio tikslumo, yra efektyvus ir gali būti taikomi didelio dažnio prekyboje. Siekiant tai patikrinti, buvo panaudoti penki ateities sandoriai, atlikti nuo 2015-08-01 iki 2015-08-31, kuriuos pateikė *NANOTICK* organizacija. Norint turimus duomenis taikyti paruoštuose

algoritmuose, reikia juos normalizuoti tam, kad būtų galima lyginti tarpusavyje. Pirmas žingsnis yra suvienodinti visų duomenų laiko eilutes. Pavyzdžiui, jei yra vieno finansinio instrumento duomenys su laiko eilute 17:00:00.869053009 ir kitas finansinis instrumentas su laiko eilute 17:00:00.825207610, tai šios laiko eilutės turi atsispindėti abiejuose finansiniuose instrumentuose. Galimi duomenų variantai su laiko eilutėmis pateikti 4 ir 5 lentelėse.

4 lentelė. Nanosekundinių duomenų pavyzdys

Data	Laikas	Simbolis	Fin. instrumentas	Tipas	Kaina
20150809	17:00:00.869053009	NGF6	NG	A	3227
20150809	17:00:00.869053009	NGF6	NG	B	3221
20150809	17:00:00.930168164	NGF6	NG	A	3226
20150809	17:00:00.930168164	NGF6	NG	B	3221
20150809	17:00:01.017456320	NGF6	NG	A	3226
20150809	17:00:01.017456320	NGF6	NG	B	3219
20150809	17:00:01.059840559	NGF6	NG	A	3227
20150809	17:00:01.059840559	NGF6	NG	B	3219
20150809	17:00:01.156791713	NGF6	NG	A	3238
20150809	17:00:01.156791713	NGF6	NG	B	3216

5 lentelė. Nanosekundinių duomenų pavyzdys

Data	Laikas	Simbolis	Fin. instrumentas	Tipas	Kaina
20150809	17:00:00.825207610	HOF6	HO	A	16040
20150809	17:00:00.825207610	HOF6	HO	B	15950
20150809	17:00:00.826021615	HOF6	HO	A	16035
20150809	17:00:00.826021615	HOF6	HO	B	15950
20150809	17:00:00.838609766	HOF6	HO	A	16040
20150809	17:00:00.838609766	HOF6	HO	B	15950
20150809	17:00:00.865890817	HOF6	HO	A	16040
20150809	17:00:00.865890817	HOF6	HO	B	15945
20150809	17:00:00.866430043	HOF6	HO	A	16040
20150809	17:00:00.866430043	HOF6	HO	B	15944

Jei atsiranda nauja turimo finansinio instrumento duomenų laiko eilutė, tai ji užpildoma prieš tai buvusiomis pirkimo pardavimo kainomis. Taip

užpildomos visos naujai atsiradusios visų finansinių instrumentų duomenų laiko eilutės (Masteika S. ir Vaitonis M., 2015; Vaitonis M. ir Masteika S., 2016).

Užpildžius šias naujai atsiradusias eilutes, kitas žingsnis yra suvienodinti visas finansinių instrumentų kainas. Atskiri finansiniai instrumentai turi skirtingas kainas ir, norint juos lyginti, ieškoti tarpusavio koreliacijos, prekybos signalų, reikia kainas suvienodinti taip, kad būtų galima atlikti šiuos veiksmus. Imamas kiekvieno turimo finansinio instrumento *P(i,t)* suskaičiuojamas vidurkis $\mu(i,t)$ ir pasirinkto prekybos lango standartinis nuokrypis $\sigma(i,t)$, pritaikant juos pagal lygtį (Perlini M. S., 2009):

$$p(i,t) = \frac{P(i,t) - \mu(i,t)}{\sigma(i,t)} \quad (8)$$

Gauta reikšmė *p(i,t)* yra normalizuota finansinio instrumento kaina *i* laikotarpiu *t* (Perlini M. S., 2009). Šis normalizavimo metodas dar vadinamas *z-score*. Yra ir kitų duomenų normalizavimo metodų: *mini-max*, dešimtainis dalinimas, slenkantis langas ir kt. Tačiau atliekant šį tyrimą, remiamasi duomenų svarba, todėl buvo pasirinktas normalizavimo metodas, kuris yra dažniausiai naudojamas statistiniame arbitraže (Bogoev D. ir Karam A., 2016).

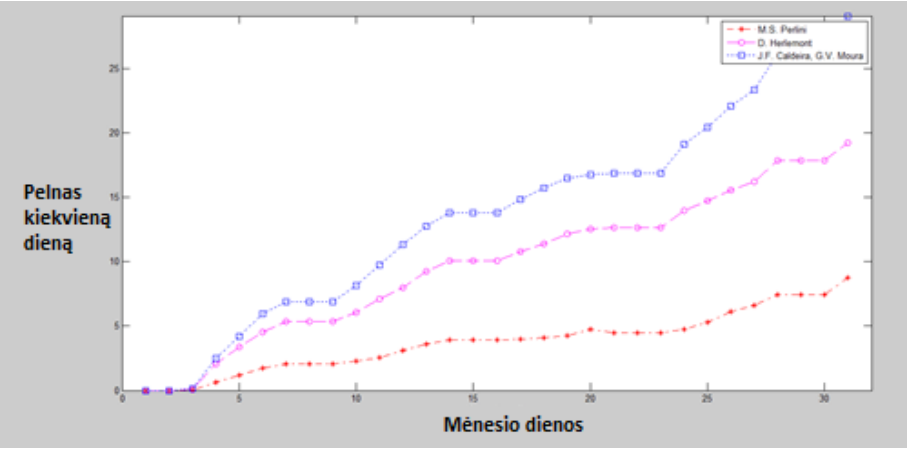
3.2 Porų prekyba naudojant nanosekundinius duomenis

Visos trys didelio dažnio porų prekybos strategijos remiasi neutralios rinkos strategijos pozicija. Jos atidaro pirkimo ir pardavimo pozicijas tuo metu, kai tik atsiranda prekybos signalas. Visi pirkimai ir pardavimai buvo atlikti naudojant rinkos kainas, o tai reiškia, kad pirkimai buvo atlikti pasiūlos, o pardavimai – pirkimo kainomis.

Vykdant porų prekybą, vienas finansinis instrumentas negali priklausyti daugiau negu vienai porai ir jis gali būti tik perkamas arba parduodamas tol, kol nėra uždaroma pozicija iki kito prekybos signalo atsiradimo. Šiame tyrime naudoti ateities sandorių nanosekundiniai duomenys ir nebuvo taikomi komisiniai mokesčiai todėl, kadangi stebimas tik pačių strategijų efektyvumas.

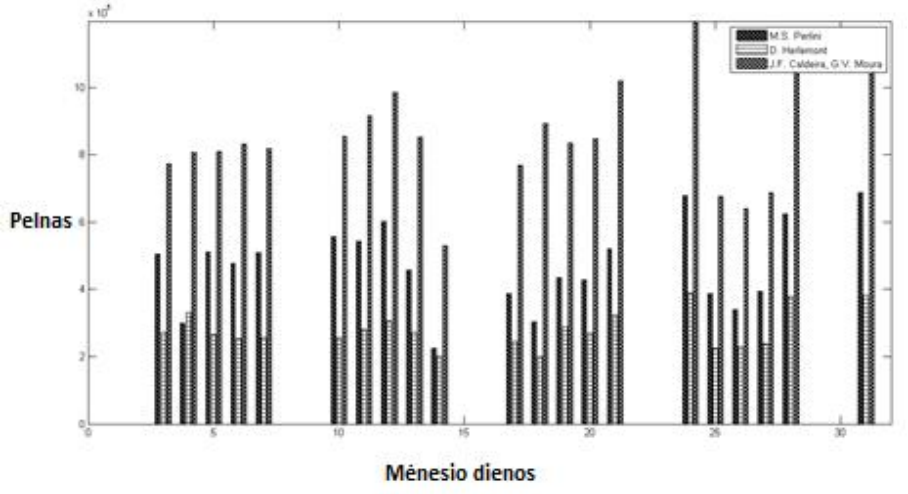
Tyrimo pabaigoje buvo pastebėta, kad visos trys didelio dažnio prekybos strategijos buvo pelningos, neatsižvelgus į komisinius mokesčius.

Grafike iliustruojamas visas nanosekundinis prekybos laikotarpis nuo 2015-08-01 iki 2015-08-31 ir nurodoma, kokius pelnus generavo kiekviena didelio dažnio porų prekybos strategija:



23 pav. Gautas pelnas, naudojant tris strategijas ir nanosekundinius duomenis prekybos laikotarpio pabaigoje

Atkreiptinas dėmesys į tai, kad taip pat galima matyti, kiek sandorių atliko kiekviena didelio dažnio porų prekybos strategija kiekvieną prekybos dieną atskirai, naudojant nanosekundinius duomenis.



24 pav. Atliktų sandorių kiekis kiekvieną prekybos dieną, naudojant visas tris strategijas

Sandorių kiekis kiekvieną dieną priklauso nuo prekybos strategijos taisyklių, skirtų rasti prekybos signalus: kuo porų prekybos strategijos taisyklės jautresnės, t. y. pastebi ir smulkius rinkos pokyčius, tuo daugiau prekybos signalų bus rasta ir atlikta daugiau sandorių. Tačiau ne visada didelis sandorių kiekis reiškia, kad prekybos strategija bus efektyvi ir

pelninga. Taip pat buvo atliktas to paties prekybos laikotarpio palyginimas naudojant milisekundinio ir nanusekundinio tikslumo duomenis. Tai buvo atlikta siekiant patikrinti, ar didesnio dažnio duomenys yra efektyvesni ir geba atlikti ne tik daugiau pardavimų / pirkimų, bet ir ar yra pelningesni.

6 lentelė. Tyrimo rezultatai, gauti naudojant milisekundinius ir nanosekundinius duomenis

	M. S. Perlini	**D. Herlemont**	**J. Caldeira ir G. V. Moura**
Pelnas naudojant nano. duomenis	8.74%	19.27%	29.11%
Atliktų sandorių kiekis	9878628	5869860	18051372
Pelnas naudojant mili. duomenis	4.01%	3.39%	4.75%
Atliktų sandorių kiekis	1491576	2135360	2538979

6 lentelėje galima matyti procentinius pelningumus ir tai, kiek sandorių atliko kiekviena didelio dažnio prekybos strategija. M. S. Perlini strategija laikotarpio pabaigoje turėjo 8,74 % pelną ir atliko 9 878 628 sandorius, D. Herlemont strategija atliko mažiausiai sandorių – 5 869 860 ir gavo 19,27 % pelną ir trečioji J. Caldeira ir G. V. Moura porų prekybos strategija atliko didžiausią kiekį sandorių – 18 051 372 ir uždirbo didžiausią pelną – 29,11 %. 7 lentelėje galima matyti vidutinį kiekvieno sandorio pelną, naudojant nanosekundinius ir milisekundinius duomenis.

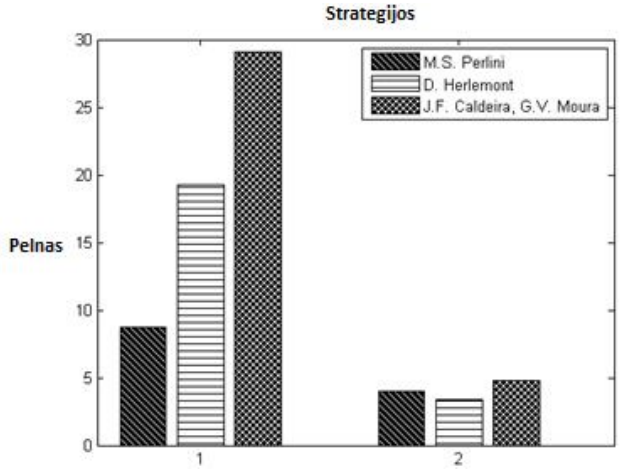
Tačiau vidutinis sandorio pelnas ne visada nurodo efektyviausią prekybos strategiją. Siekiant rasti efektyviausią strategiją, buvo skaičiuojamas visų didelio dažnio porų prekybos strategijų Šarpo rodiklis su nanosekundėmis ir milisekundėmis.

7 lentelė. Procentinis pelnas kiekvieną prekybos dieną

Data	**M. S. Perlini nano.**	**M. S. Perlini mili.**	**D. Herlemont nano.**	**D. Herlemont mili.**	**J. Caldeira ir G. V. Moura nano.**	**J. Caldeira ir G. V. Moura mili.**
2015.08.3	0.2991	0.2861	0.9131	0.1344	1.246	0.0482
2015.08.4	0.329	0.0951	1.1628	0.2393	1.257	0.0274
2015.08.5	0.5558	0.2684	1.3029	0.3997	1.6732	0.0153
2015.08.6	0.5296	0.2605	1.1581	0.3367	1.81	0.2466
2015.08.7	0.3334	0.1725	0.8016	0.254	0.911	0.1888

2015.08.10	0.2338	0.2397	0.7086	0.2693	1.251	0.1669
2015.08.11	0.2551	0.291	1.0476	0.3316	1.616	0.2252
2015.08.12	0.5766	0.3343	0.8812	0.107	1.596	0.3285
2015.08.13	0.4855	0.2346	1.272	0.1013	1.403	0.3322
2015.08.14	0.3569	0.1116	0.8347	0.0929	1.06	0.2173
2015.08.17	0.0275	0.0753	0.7157	0.0425	1.026	0.1978
2015.08.18	0.1219	0.0597	0.6141	0.037	0.866	0.1121
2015.08.19	0.1439	0.1856	0.7636	0.0425	0.812	0.2317
2015.08.20	0.4791	0.1352	0.3591	0.0443	0.239	0.1909
2015.08.21	-0.25	0.0917	0.118	0.0754	0.11	0.1201
2015.08.24	0.2562	0.2031	1.3034	0.1552	2.273	0.2941
2015.08.25	0.5519	0.086	0.7722	0.0504	1.335	0.1795
2015.08.26	0.8523	0.1502	0.8247	0.0539	1.613	0.2987
2015.08.27	0.5036	0.1992	0.6949	0.1271	1.285	0.3636
2015.08.28	0.7844	0.2974	1.6071	0.319	2.786	0.5674
2015.08.31	1.3203	02296	1.4236	0.1759	2.944	0.3922

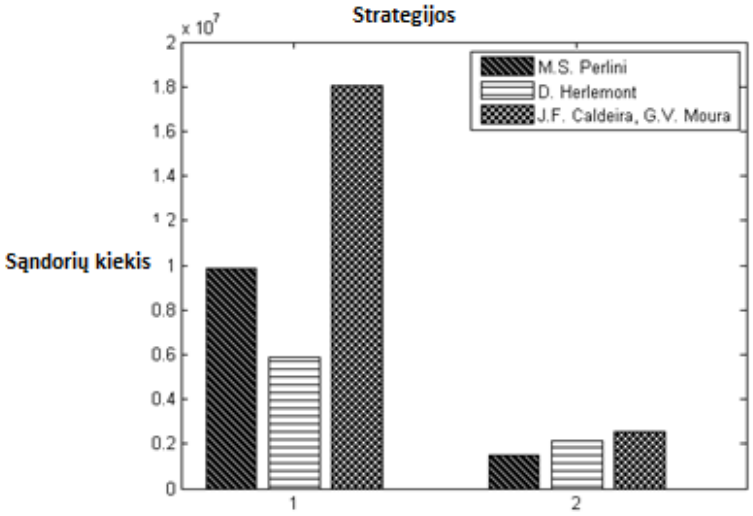
Prieš skaičiuojant Šarpo rodiklį, reikia nustatyti, kokia buvo kiekviena didelio dažnio prekybos strategija kiekvieną dieną atskirai su abiejų tipų duomenimis. 7 lentelėje atskleidžiamas visų prekybos strategijų pelningumas procentais kiekvieną dieną su milisekundiniais ir nanosekundiniais duomenimis. Galima pastebėti, kad nei viena diena nebuvo nuostolinga. 25 paveiksle esančiame grafike atskirai parodytas strategijų galutinio prekybos laikotarpio pelningumas, naudojant milisekundinius (2) ir nanosekundinius (1) duomenis.



25 pav. Procentinis pelnas, naudojant visas tris strategijas, milisekundinius ir nanosekundinius duomenis

Grafike atskleista akivaizdi duomenų naudojimo dažnio ir pelno sąsaja – didesnio dažnio duomenys duoda didesnį pelną. Tačiau, prieš priimant galutines išvadas, reikėtų apskaičiuoti Šarpo rodiklį, siekiant tiksliau nustatyti, kurio tipo duomenys efektyvesni ir kuri strategija buvo geriausia.

Taip pat vertėtų palyginti sandorių skirtumą naudojant skirtingo dažnio duomenis; jų palyginimą galima matyti grafike (žr. 26 pav.). Konstatuotina, kad nanosekundinių duomenų atveju įvyko daugiau sandorių nei milisekundinių duomenų atveju.



26 pav. Bendras sandorių kiekis, naudojant tris strategijas, išskiriant milisekundinius ir nanosekundinius duomenis

Surinkus būtiną informaciją apie visas tris didelio dažnio porų prekybos strategijas su milisekundiniais ir nanosekundiniais duomenimis, galima skaičiuoti Šarpo rodiklį. Jis skaičiuojamas naudojant formulę $S=\frac{\mu_{PnL}}{\sigma_{PnL}}$, kur μ_{PnL} yra pelno ar nuostolio vidurkis ir σ_{PnL} yra pelno ar nuostolio *PnL* standartinis nuokrypis.

8 lentelė. Visų strategijų Šarpo rodiklis, naudojant milisekundinius ir nanosekundinius duomenis

	Šarpo rodiklis M. S. Perlini strategijos	**Šarpo rodiklis D. Herlemont strategijos**	**Šarpo rodiklis J. Caldeira ir G. V. Moura strategijos**
Milisek. duomenys	1.3380	1.4240	1.7651
Nanosek. duomenys	1.3144	2.6388	2.0442

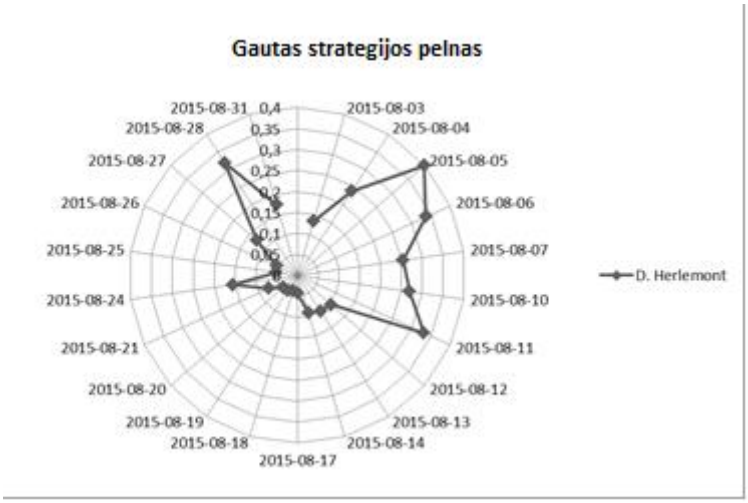
Kuo didesnis Šarpo rodiklis, tuo geresnė porų prekybos strategija. Lentelės viršuje matyti, kad J. Caldeira ir G. V. Moura didelio dažnio porų prekybos strategija, naudojanti milisekundinius duomenis, buvo geriausia. Tačiau kai reikėjo naudoti nanosekundinius duomenis, šią strategija aplenkė D. Herlemont didelio dažnio porų prekybos strategija. Rezultatai atskleidžia, kad vien didelis sandorių kiekis nereiškia, kad prekybos strategija yra efektyvi. Taip pat matyti, kad būtent naudojant nanosekundinius duomenis, gauti Šarpo rodikliai didesni dviejų strategijų atveju iš trijų, tai žymi didelio dažnio duomenų naudingumą. Didelio dažnio duomenys leidžia rinkos dalyviui dirbti su naujausia informacija, o dėl gaunamo didelio duomenų kiekio galima parinkti nedidelius prekybos langus. Taip lengviau pastebėti net smulkiausius rinkos nukrypimus. Kitas svarbus didelio dažnio duomenų privalumas yra tas, kad rinkos dalyvis gali aplenkti tuos, kurie dirba su lėtesnio dažnio duomenimis, arba jis gali priimti sprendimus daug greičiau už kitus.

3.3 Porų prekyba su nanosekundiniais duomenimis, naudojant CPU ir GPU

Du pagrindiniai algoritminės prekybos kriterijai yra greitis, kuriuo tie patys skaičiavimai gali būti atliekami naudojant didelį kiekį skirtingų duomenų, bei galimybė juos programuoti. Dėl šios priežasties tradicinė techninė įranga yra netinkama, pvz., centrinis procesorius. Centrinis kompiuterio procesorius yra sukurtas taip, kad atliktų veiksmus vieną po kito. Tačiau norint spartinti skaičiavimus, juos reikia lygiagretinti ir atlikti daug vienodų skaičiavimu tuo pačiu metu.

Šio tyrimo metu buvo naudojamas dviejų branduolių CPU *Intel i5*-3230M 2,6 GHz ir GPU su 96 CUDA branduoliais *GeForce* 710M. Pirmiausia prekybos strategijai buvo naudojamas tik centrinis procesorius. Pasitelkus *parfor* funkciją MATLAB aplinkoje, galima lygiagretinti skaičiavimus ir pačiam CPU padalyti juos branduoliams. Šioje dalyje buvo nustatyti skaičiavimai, kuriuos galima lygiagretinti CPU ir taip optimizuoti porų prekybos strategiją.

Šio eksperimento metu buvo lygiagretintos porų aptikimo, prekybos signalų nustatymo ir pelno skaičiavimo funkcijos. Šias funkcijas buvo galima lygiagretinti, kadangi jos, naudodamos vis kitus duomenis, atlikdavo tuos pačius veiksmus. Eksperimento pabaigoje buvo apskaičiuotas strategijos pelnas, pateiktas grafike (žr. 27 pav.).



27 pav. Kiekvieną prekybos dieną gautas strategijos pelnas

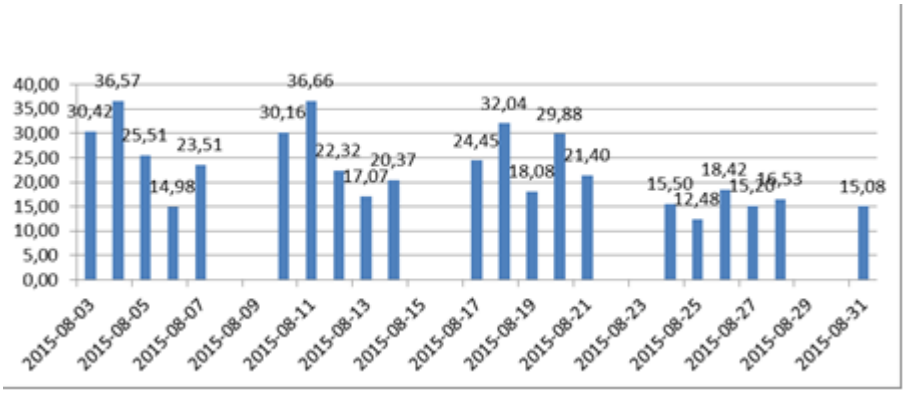
Eksperimento esmė nebuvo nustatyti strategijos pelningumą, bet, nustačius galimybes, lygiagretinti skaičiavimus CPU ir, vėliau juos perkėlus į GPU, pagreitinti prekybos algoritmo veikimą. 9 lentelėje vaizduojama, kiek duomenų buvo apdorota kiekvieną dieną ir kiek užtruko didelio dažnio porų prekybos strategija juos apdorodama, kai naudojo CPU ir GPU.

9 lentelė. CPU ir GPU palyginimas

Data	***Intel i5* - 3230M 2,6 GHz, 2 cores (sekundėmis)**	***GeForce* 710m, 96 CUDA Cores (sekundėmis)**	**Apdorotų duomenų kiekis**
2015-08-03	2991,80	2081,60	6096505
2015-08-04	2208,10	1400,50	4579465
2015-08-05	2393,70	1783,10	5793525
2015-08-06	3040,90	2585,3	5595770
2015-08-07	2650,10	2027,1	5586360
2015-08-10	4410,80	3080,70	5732355
2015-08-11	4980,30	3154,50	6249980
2015-08-12	2769,20	2151,20	6758875
2015-08-13	4122,60	3419,00	5666900
2015-08-14	1325,90	1055,80	4227335
2015-08-17	1550,00	1171,10	4879990
2015-08-18	1912,10	1299,50	4364540

2015-08-19	4002,30	3278,70	5666700
2015-08-20	4449,00	3119,43	5411145
2015-08-21	4311,70	3389,10	5946205
2015-08-24	4809,40	4064,00	7710745
2015-08-25	3960,20	3466,10	5105175
2015-08-26	3187,60	2600,40	5119660
2015-08-27	5004,90	4244,20	7963320
2015-08-28	5287,10	4413,10	7721975
2015-08-31	5409,70	4594,10	8613445

9 lentelėje vaizduojama, kiek sekundžių užtruko algoritmas, naudodamas skirtingų tipų techninę įrangą. Grafike, pavaizduotame 28 pav., procentais nurodyta, kiek paspartėjo skaičiavimai, naudojant GPU.



28 pav. Algoritmo vėlinimo sumažinimas, jį pritaikius GPU

Grafike matyti, kad algoritmą pritaikius dirbti su GPU, kuris turėjo 96 CUDA branduolius, skaičiavimai pagreitėjo nuo 12 % iki 36 % procentų, palyginti su CPU. Skaičiavimų pagreitėjimo skirtumas atsiranda dėl to, kad skirtingomis dienomis nustatomas skirtingas kiekis porų ir prekybos signalų. Kuo didesnį kiekį duomenų skaičiavimai gali lygiagretinti, tuo greičiau vyksta skaičiavimai. Eksperimentas parodė ne tik didelio dažnio duomenų svarbą, bet ir tai, kaip technologiniai pakeitimai gali paveikti algoritmo veikimą.

Toliau buvo siekiama išsiaiškinti, kaip būtų galima dar labiau paspartinti turimų statistinio arbitražo prekybos strategijų skaičiavimus, kad jie galėtų veikti didelio dažnio prekybos aplinkoje. Šioje tyrimo dalyje naudotas CPU *Xeon E5-2650* su aštuoniais branduoliais ir 20 MB spartinančiąja atmintimi, bei GPU *Nvidia GTX* – 1060 6GB su 1280 CUDA branduoliais. Tyrimo

metu tikrinta, kiek algoritmas laiko užtrunka atlikdamas dienos skaičiavimus, kai jis naudoja tik CPU, vėliau naudoja GPU ir naudojant GPU tada, kai turimas kodas yra vektorizuojamas siekiant išvengti skaičiavimų ciklų, dėl kurių gali atsirasti vėlinimas.

10 lentelė. CPU ir GPU palyginimas

M. S. Perlini			D. Herlemont			J. Caldeira ir G. V. Moura		
CPU	Nevektor. GPU	Vektor. GPU	CPU	Nevektor. GPU	Vektor. GPU	CPU	Nevektor. GPU	Vektor. GPU
3082,01	2212,58	7,56	2991,80	2234,69	7,06	3188,30	2175,46	7,13
2312,21	1866,77	5,92	2208,10	1625,87	5,52	2300,20	1836,70	5,55
2548,00	2275,70	7,41	2393,70	2033,04	6,80	2577,20	2238,40	6,87
3157,23	1980,72	7,02	3040,90	1796,88	6,51	3025,80	1741,62	6,56
2757,20	2020,74	7,01	2650,10	1617,00	6,67	2736,30	2010,04	6,75
4930,11	1723,71	7,12	4410,80	2397,89	6,72	4850,30	2409,41	6,79
5182,32	2513,59	7,58	4980,30	2833,62	7,22	5144,90	2215,78	7,22
2900,12	2174,64	8,21	2769,20	2007,54	7,68	2837,10	2075,58	7,73
4211,10	2082,78	7,08	4122,60	2287,30	6,55	4222,80	2591,05	6,65
1398,00	1208,22	5,52	1325,90	1204,28	5,24	1483,00	1284,10	5,21
1602,56	1491,32	6,21	1550,00	1342,17	5,87	1597,30	1333,63	5,86
2102,70	1455,73	5,66	1912,10	1695,52	5,33	2074,60	1570,07	5,40
4179,40	1633,95	6,92	4002,30	1620,07	6,61	4154,70	1495,48	6,63
4982,20	2130,43	6,65	4449,00	1948,76	6,41	4591,40	2098,70	6,42
4591,80	1883,19	7,26	4311,70	2111,37	7,22	4421,70	1941,67	6,89
5080,60	3185,21	9,07	4809,40	3193,18	8,61	4982,50	3306,48	8,74
4229,10	1810,58	6,44	3960,20	2200,31	5,94	4074,30	2154,72	6,06
3377,20	2266,08	6,38	3187,60	2564,24	6,03	3269,70	2261,87	6,05
5305,40	3110,35	9,42	5004,90	3494,92	8,87	5238,00	3089,66	9,09
5698,10	3186,38	9,07	5287,10	3642,83	8,59	5534,70	3769,10	8,71
5671,80	3835,31	10,08	5409,70	3033,25	9,51	5583,40	3617,43	9,64

10 lentelėje matyti, kad procentinis pagreitėjimas nevektorizuoto metodo, kuris naudoja ciklus ir dažną duomenų perkėlinėjimą į ir iš CPU ir GPU atminčių, perkėlus iš CPU į GPU, nanosekundinių duomenų apdorojimas pagreitėjo nuo 10 % iki 60 %. Siekiant įvertinti visą galimą GPU efektyvumą, algoritmą reikėjo perrašyti taip, kad jis galėtų visiškai panaudoti visus GPU privalumus – tam reikėjo pasitelkti kodo vektorizacijos metodą.

Dėl šio metodo taikymo nepavyko visiškai išvengti ciklų, tačiau jų sumažėja iki vieno, kuris skirtas perdavinėti tam tikro dydžio duomenų paketus. Procentinis pagreitėjimas iš nevektorizuoto į vektorizuotą GPU metodą siekia apie 99 %. Tai parodo, kiek kartų GPU yra efektyvesnis apdorojant didelio kiekio duomenis, kuriems reikalingas mažas vėlavimas.

3.4 Skyriaus išvados

Šiame skyriuje aptartos trys statistinio arbitražo prekybos strategijos. Jas pasitelkus, buvo atliktas tyrimas, taikant kointegracijos metodą prekybos porų parinkimui, siekiant patikrinti, ar nanosekundiniai duomenys yra efektyvesni už milisekundinius ir ar jie turi įtakos pelningumui. Pabaigus tyrimą, paaiškėjo, kad nanosekundiniai duomenys iš tiktųjų yra efektyvesni ir, juos naudojant, statistinio arbitražo strategijos veikia pelningiau. Taip yra dėl to, kad, naudojant didesnio dažnio duomenis, pačių duomenų yra daug daugiau, stebima ir tikrinama pati naujausia informacija, pastebimi net mažiausi rinkos ir finansinio instrumento kainos pokyčiai.

Antroje šio skyriaus dalyje siekta patikrinti, ar, vektorizavus didelio dažnio statistinio arbitražo strategijos algoritmo kodą ir perkėlus jį iš CPU į GPU, jis veiks greičiau, t. y. aptiks ir priims prekybinius sprendimus greičiau. Buvo matuojama, kaip greitai algoritmas apdoros kiekvienos dienos didelio dažnio duomenis, vykdydamas prekybą. Tyrimo pabaigoje paaiškėjo, kad algoritmo perkėlimas į GPU tikrai paspartino prekybos sprendimų priėmimą, kuris svyravo nuo 12 % iki 36 % skirtingomis dienomis. Greičio skirtumas susidarė todėl, kad prekybos dienos yra skirtingos, nustatomas skirtingas kiekis duomenų ir sukuriamas nevienodas skaičius prekybos sandorių: kuo jų daugiau, tuo ilgiau tai užtrunka. Nepaisant to, paaiškėjo, kad perkėlimas iš CPU į GPU, vektorizavus turimą didelio dažnio statistinio arbitražo prekybos algoritmą, yra efektyvus sprendimas.

4 SIŪLOMA DIDELIO DAŽNIO PREKYBOS STRATEGIJŲ TESTAVIMO METODOLOGIJA

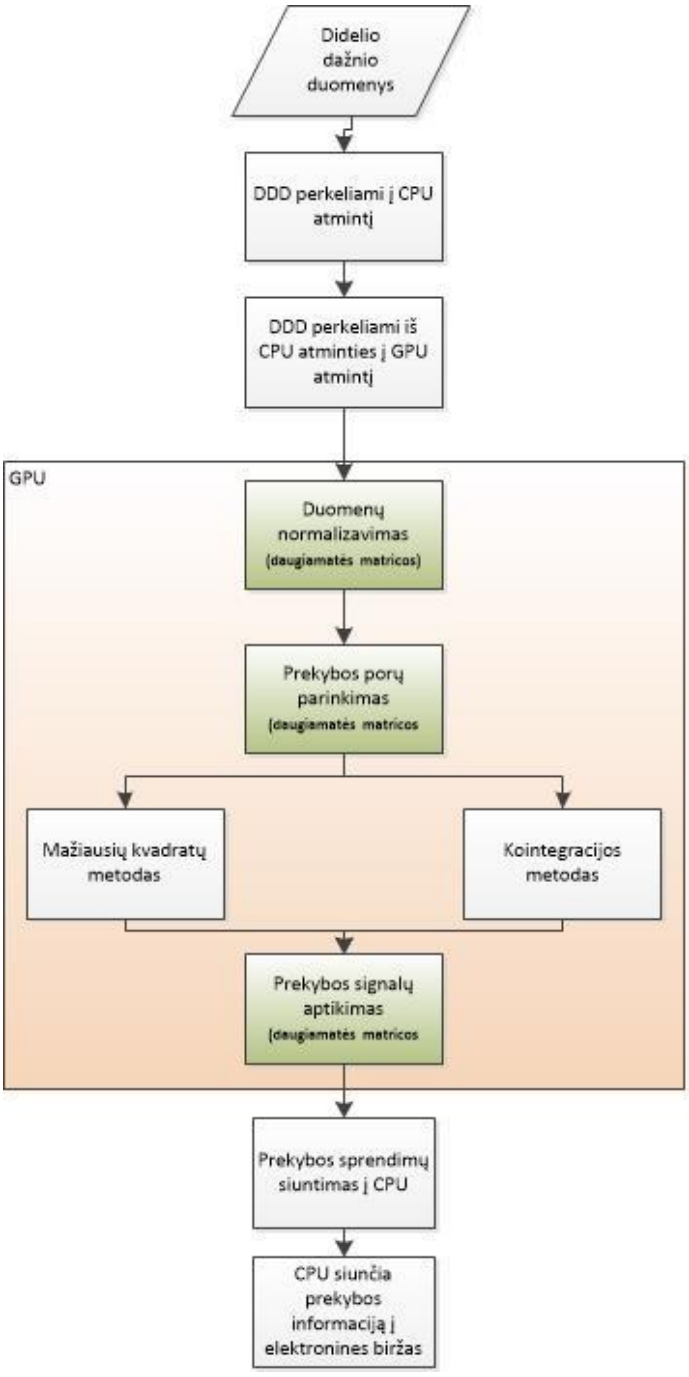
Prieš nuodugnesnę pasiūlyto sprendimo metodologijos analizę, svarbu aprašyti, kaip veikia CUDA. Pagrindinis CUDA komponentas – branduolys, kuris yra kaip C programavimo kalbos funkcija, paleidžiama N kartų lygiagrečiai N CUDA branduolių *threads*. Šie branduoliai yra dažnai grupuojami į blokus ir vadinami branduolių blokais. Toliau šie branduolių blokai gali būti vienos (x), dviejų (x, y) ir trijų (x, y, z) dimensijų. Šie blokai dar gali būti sujungiami į vienos, dviejų ar net trijų dimensijų tinklus. Tokio tipo CUDA branduolių hierarchija leidžia patogiai apdoroti duomenis, kurie yra vektoriniai, matricos arba daugiamatės matricos. CUDA vykdymo modelis yra labai panašus į SIMD (viena instrukcija, daug duomenų) architektūrą. Kiekviena instrukcija, kuri yra paleidžiama CUDA procedūroje, yra SIMD tipo. Tai reiškia, kad ji visą laiką dirba tik su vektoriniais duomenimis (Nvidia, 2007; Coon B. W. ir Lindholm J. E., 2008; Coon B. W. ir Lindholm J. E., 2009; Coon B. W. et al., 2010; Coon B. W. et al., 2011).

Remiantis anksčiau paminėtais faktais, galima teigti, kad, norint perkelti didelio dažnio prekybą į GPU, reikia vektorizuoti turimą algoritmą naudojant vektorius, daugiamates matricas ir branduolių lygiagretinimą. Kitame skyriuje yra pateikiamas siūlomas testavimo metodas, skirtas taikyti DDP strategijas, naudojant GPU techninę įrangą, pasitelkiant GPU globaliąją atmintį, algoritmo vektorizaciją, daugiamates matricas ir lygiagrečius branduolius.

4.1 Didelio dažnio prekybos testavimo metodas naudojant GPU CUDA

Pagrindinis pasiūlyto metodo tikslas yra gauti didelio dažnio duomenis iš elektroninės biržos. Pirmiausia šie duomenys keliauja į CPU atmintį, o iš jos – į GPU atmintį, kurioje lygiagretinami skaičiavimai ir šie duomenys padalijami CUDA branduoliams. Gavus duomenis, CPU iš savo atminties siunčia juos į GPU atmintį kartu su kodu, kurį reikia vykdyti. Remiantis pasiūlytu metodu, *backtestingas* bus vykdomas su pasiūlytu statistinio arbitražo didelio dažnio prekybos sistema. Kaip jau buvo minėta, tyrime naudojami didelio dažnio *tick-by-tick* duomenys ir, norint ištestuoti turimas DDP strategijas, užtektų vieno mėnesio duomenų, tačiau šiame tyrime naudojami trijų mėnesių duomenys. Algoritminė prekyba skiriasi nuo kito

tipo investavimo dėl jo galimybės tiksliau nusakyti rinkos ir finansinių instrumentų judėjimą, remiantis istoriniais duomenimis – dėl jų didelio kiekio ir galimybės juos greitai apdoroti. Šiame tyrime *backtestingas* vykdomas turimai statistinio arbitražo didelio dažnio prekybos strategijai duodant apdoroti istorinius trijų mėnesių didelio dažnio duomenis, kuriuos naudojant, bus nustatomos galimos ateities sandorių poros ir sukuriami prekybos signalai. Kiekvienas prekybos signalas, atidarius ir uždarius poziciją, generuoja pelną arba nuostolį, taip pat skaičiuojamas greitis, per kiek laiko prekybos sistema priėmė prekybos signalą bei atidarė ir uždarė poziciją.



29 pav. Siūlomas DDP GPU sistemos metodas

Naudojamas algoritmas padeda lygiagretinti skaičiavimus, esančius GPU, tarp visų CUDA branduolių užpildant globaliąją atmintį informacija, taip spartinant pačio prekybos algoritmo veikimą. 29 paveikslėlyje pavaizduoti porų prekybos strategijos skaičiavimai, kuriuos galima lygiagretinti:

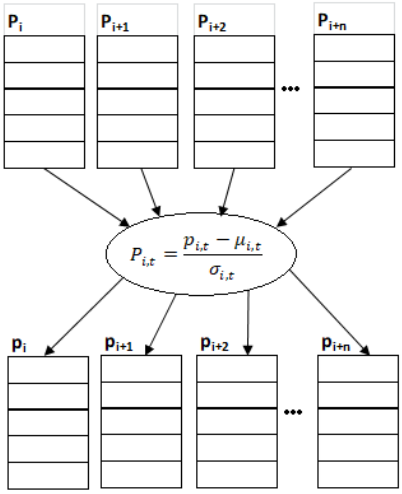
1. Duomenų normalizavimas.
2. Prekybos porų parinkimas.
3. Prekybos signalų paieška.
4. Prekybos signalų ir pozicijos uždarymo siuntimas.

Pirmiausia buvo galima lygiagretinti duomenų normalizavimą. Šioje didelio dažnio prekybos sistemos dalyje vidurkis ir standartinis nuokrypis yra skaičiuojamas visiems 130-čiai ateities sandorių. Tai lėmė, kad buvo galima iš viso lygiagretinti 16 900 skaičiavimų. Kita algoritmo dalis, kurioje galima taikyti lygiagrečius skaičiavimus, yra prekybos porų paieška. Norint nustatyti, kiek yra galimų porų, visi ateities sandoriai turi būti lyginami vienas su kitu. Todėl iš viso yra n(n-1) galimų prekybos porų ir šiame tyrime jų gali būti 16 770. Tyrime galima naudoti vieną iš pasirinktų arba abu porų parinkimų metodus vienu metu, dėl šios priežasties iš viso gali būti 16 770*2=33 400 galimų porų variantų, apskaičiuotų lygiagrečiai. Toliau yra naudojami trijų statistinio arbitražo strategijų prekybos signalų aptikimo metodai ir galima naudoti vieną, du ar visus tris metodus lygiagrečiai. Galiausiai, taikant visus tris metodus, yra 33 400*3*2*1=200 400 galimų lygiagrečių skaičiavimų. Viską susumavus, šiame tyrime maksimaliai buvo galima lygiagrečiai atlikti 217 300 skaičiavimų.

Tyrimo metu reikėjo rasti būdų, kaip paspartinti visą duomenų apdorojimą, porų aptikimą, prekybos signalų radimą ir jų perdavimą į elektroninę biržą. Tai reiškė, kad visas prieš tai naudotas prekybos kodas turi būti perrašytas išvengiant ciklų, jį vektorizuojant, kur įmanoma, ir panaudojant daugiamates matricas, taip išnaudojant visus GPU pajėgumus, t. y. jo globaliąją ir kitą atmintį, CUDA branduolius ir branduolių lygiagretinimą. Prekybos sistemai radus prekybos signalus ir nusprendus atidaryti ar uždaryti poziciją, ši informacija iš GPU atminties yra siunčiama į CPU atmintį ir tada keliauja į elektroninę biržą.

Prieš tai atliktuose tyrimuose (Vaitonis M. ir Masteika S., 2018; Vaitonis M., 2018) buvo bandomi taikyti jau anksčiau metodologijoje paminėti metodai (vektorizacija ir GPU atminties naudojimas), kuriais buvo didinamas statistinio arbitražo prekybos našumas ir didinamas sprendimų priėmimo greitis. Anksčiau atliktų tyrimų metu buvo naudojama *NVIDIA GeForce GTX* 1060 6GB kartu su MATLAB programine įranga, skirta

simuliuoti prekybos algoritmą ir pačią prekybą. Simuliacijos metu buvo naudojami penkių ateities sandorių didelio dažnio duomenys nuo 2015-08-01 iki 2015-08-31. Tačiau tada dar nebuvo pavykę visiškai vektorizuoti viso kodo ir panaudoti daugiamates matricas bei lygiagrečius branduolius. O tai lėmė, kad didžiausia pasiekta sprendimų priėmimo sparta buvo 84,95 mikrosekundės. Atsižvelgus į šių tyrimų metu naudotus duomenis, paaiškėjo, kad tokio greičio neužteks ir reiks ieškoti kitų paspartinimo būdų. Šiam tikslui pasiekti ir buvo pasitelkti visi metode aprašyti būdai, juos įgyvendinant MATLAB aplinkoje.



30 pav. Didelio dažnio duomenų normalizavimas

Ateities sandorių kainoms normalizuoti panaudota algoritme aprašyta *parallel* funkcija, ja normalizuojamos kainos *p* ir gaunama nauja normalizuota kaina *P*.

Kitas žingsnis yra koreliuotų porų ieškojimas, kurios būtų vėliau naudojamos ieškant prekybos signalų. Šiame žingsnyje naudojamos tik normalizuotos ateities sandorių kainos. Tyrimo metu yra galimi du porų parinkimo metodai:

1) mažiausių kvadratų metodas;
2) kointegracijos metodas.

Kai naudojamas mažiausių kvadratų metodas, yra ieškoma ateities sandorių poros, kurios kvadratinis normalizuotų kainų skirtumas *S* yra mažiausias. Taip kiekvienam ateities sandoriui yra nustatoma pora su

mažiausiu vienas nuo kito atstumu; tai reiškia, kad šie ateities sandoriai tarpusavyje koreliuoja (Huck N. ir Afawubo K., 2015).

$$S_{i,j=}\sum_t^T (P_{i,t} - P_{j,t})^2 \quad (9)$$

Formulėje $P_{i,t}$ ir $P_{j,t}$ normalizuotos ateities sandorių *i* ir *j* kainos laiku *t*, o *T* yra pasirinkto duomenų normalizavimo ir prekybos lango dydis (Huck N. ir Afawubo K., 2015).

Antras prekybos porų parinkimo metodas yra kointegracija. Taikant šį metodą, algoritmas pirmiausia patikrina, ar visos duomenų eilutės yra integruotos ta pačia tvarka, pasitelkiant papildytą Dickey Fuller testą (ADF). Tada atliekamas kointegracijos testas su visomis galimomis poromis, naudojant dviejų žingsnių Engle ir Granger, vėliau Johansen testą (Huck N. ir Afawubo K., 2015).

Papildyto Dickey Fuller testo metu tikrinama nulinė hipotezė:

$$y_t = c + \phi y_{t-1} \beta_1 \Delta y_{t-1} + \cdots + \beta_p \Delta y_{t-p} + \varepsilon_t \quad (10)$$

Čia Δ yra skirtumo operatorius, kuris yra lygus $\Delta yt = yt - yt\text{-}1$. Kintamasis *p* yra nustatomas empiriškai, kad vidurkio nulinė klaida ε_t būtų nekoreliuota. Tada nulinė hipotezė yra tokia:

$$H_o: \phi = 1 \quad (11)$$

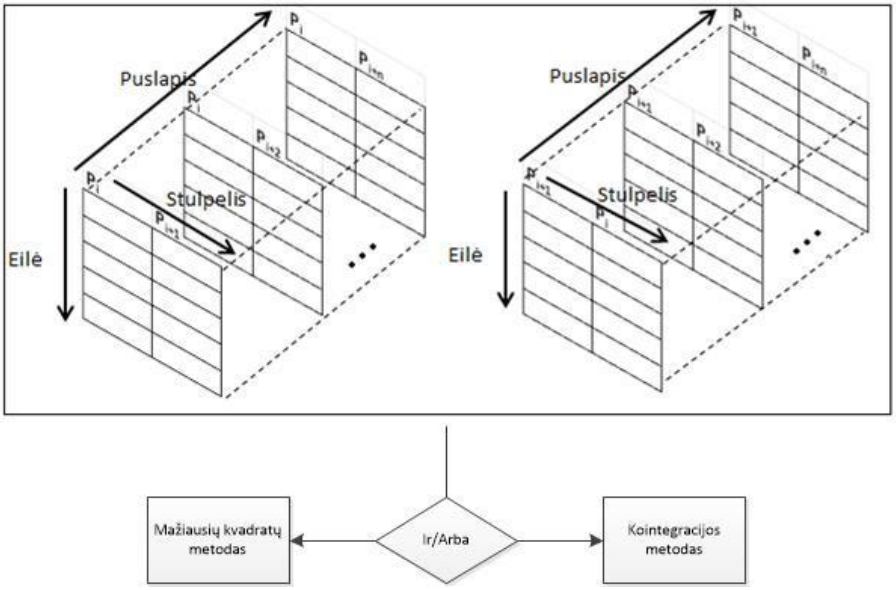
O jos alternatyvi hipotezė yra ϕ<1. Jei ADF testas atliekamas sėkmingai, tada seka dviejų žingsnių Engle ir Granger bei Johansen testai. Engle ir Granger testas susideda iš dviejų žingsnių. Jei $P_{1,t}$ ir $P_{2,t}$ yra dvi ateities sandorių kainos laiku *t*, tai pirmas žingsnis reikalautų, kad būtų $P_{1,t}$ regresija prieš $P_{2,t}$:

$$P_{1,t} - \beta P_{2,t} = \mu - \varepsilon_t \quad (12)$$

Formulėje μ žymi perėjimą. Kointegracija tarp dviejų ateities sandorių patikrinama analizuojant sekos integraciją liekanos ε_t kartu su ADF testu. Ateities sandoriai yra kointegruoti, jei regresijos liekanos yra stacionarios. Paskutinis žingsnis yra patikrinti, ar pora gali būti naudojama prekybai, kai visi testai su ateities sandorių poromis yra atlikti. Šiame etape patikrinama hipotezė *r* neribojamo kointegracinio ryšio neribojamame vektorių autoregresijos modelyje (VAR). Poros su aukščiausia statistine žyma bus naudojamos kaip galimos poros prekybai. Tada dar patikrinamas šios poros ryšio nuokrypis $P_{1,t} - \beta P_{2,t}$ nuo istorinio vidurkio μ. Tik tada galima naudoti porą tolesnei didelio dažnio prekybai (Huck N. ir Afawubo K., 2015).

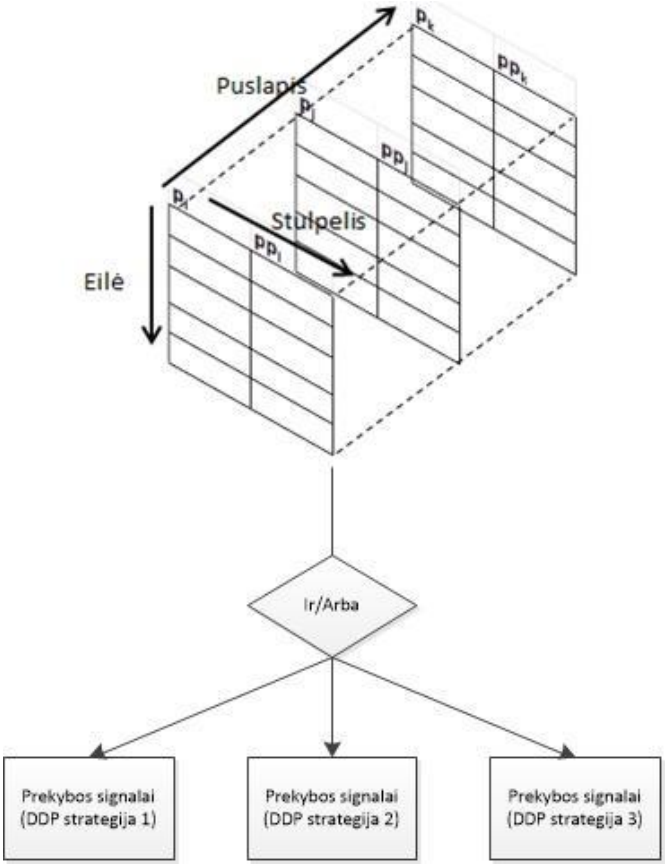
Tam, kad būtų galima ieškoti porų ir paspartinti šį procesą, jis atliekamas GPU sukuriant trimates matricas, kurių kiekviena bus skirtingas ateities sandoris. Kiekvienos iš trimačių matricų pirmas stulpelis yra vienodas, atitinkantis lyginamo ateities sandorio kainas *Pi*, antras stulpelis yra vis kito

ateities sandorio kainos. Ir toliau kiekvienas trimatės matricos puslapis yra skirtinga galima pora su tuo pačiu ateities sandoriu. Kai šios matricos yra paruoštos, tada galima pradėti ieškoti koreliuotų porų. Porų atrinkimui naudojama algoritme aprašyta *pair* funkcija.



31 pav. Didelio dažnio duomenų konvertavimas į 3D GPU masyvą ir perdavimas į pasirinktą porų prekybos strategiją

Radus visas galimas poras, jos taip pat išsaugomos trimatėje matricoje tam, kad, priklausomai nuo konfigūracijos, būtų galima ieškoti prekybos signalų. Taip galima vykdyti signalų paiešką su kiekvienu trimatės matricos puslapiu lygiagrečiai. Prieš išsaugant visas galimas poras į trimates matricas, yra panaikinami porų dublikatai, kuriuose yra AB ir BA bei AA ir BB poros. Naujoje trimatėje matricoje P_i yra ateities sandoris ir PP_i yra jo pora, o matricoje saugomos jų normalizuotos kainos. Atskiri trimatės matricos puslapiai reiškia skirtingas poras.



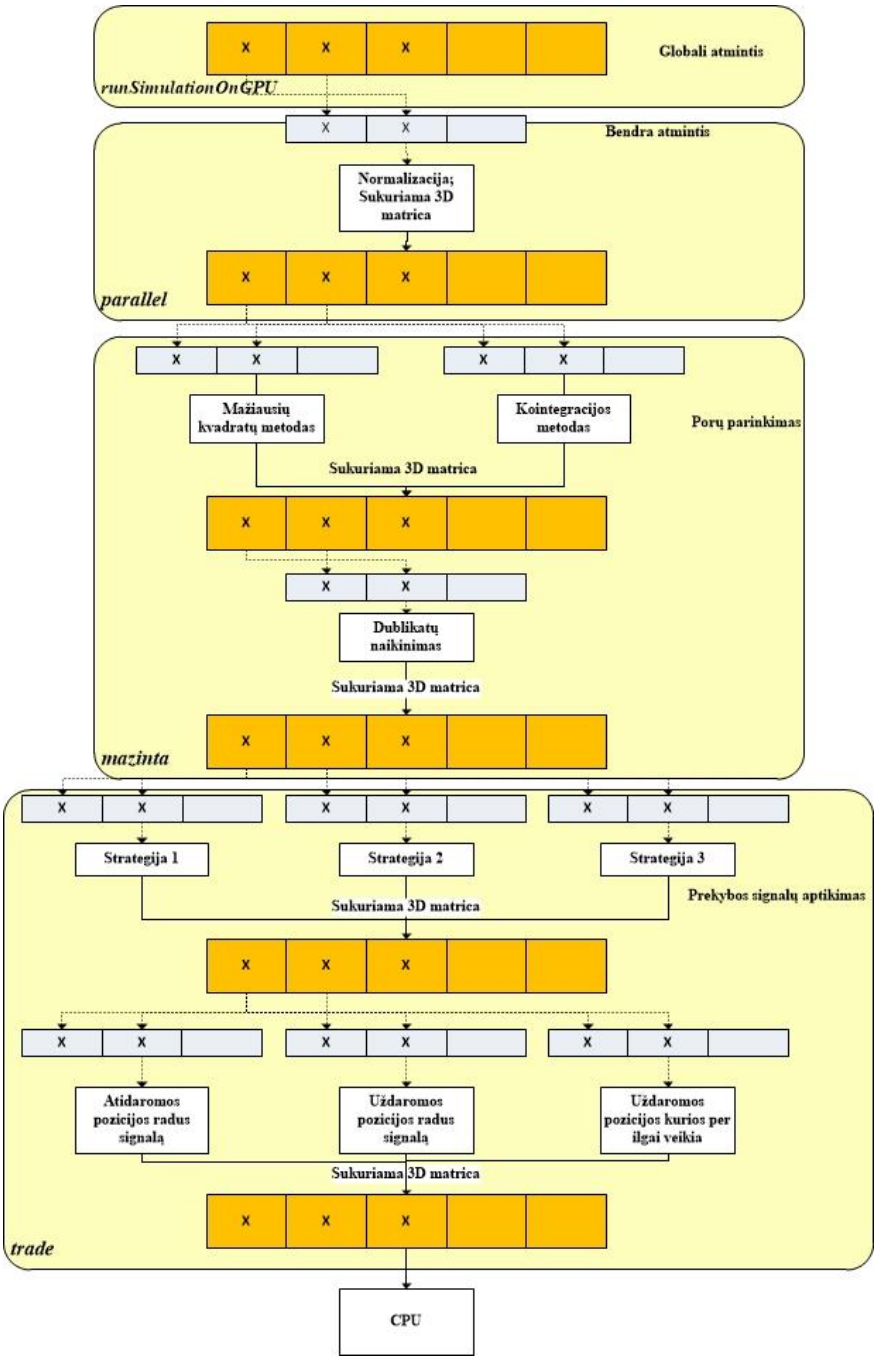
32 pav. 3D GPU masyvas su pasirinktomis ateities sandorių poromis, perduodamas prekybos signalų parinkimui

Paruošus šias trimates matricas, galima ieškoti prekybos signalų. Tam pasitelkiama *trade* funkcija. Priklausomai nuo naudojamos strategijos, signalų paieškos metodai skiriasi. Pagal algoritmo parametrus galima naudoti vieną iš trijų, bet kuriuos du arba visus tris prekybos signalų paieškos metodus. Naudojant daugiau nei vieną signalų paieškos metodą ir, prieš siunčiant informaciją atgal biržai bei norint išvengti dubliuotų signalų parinkimų, jie yra patikrinami ir dublikatai yra naikinami. Aptikus prekybos signalą ir patikrinus, ar nėra dublikatų, informacija apie norimą atidaryti poziciją siunčiama iš GPU į CPU atmintį: iš čia ji turi keliauti į elektroninę biržą. Pozicijos yra laikomos atidarytos tol, kol baigiasi maksimalus periodas jas laikyti atidarytas arba prekiaujama pora tapo nebekoreliuota.

Algoritmo kodo vektorizavimas parodė, kad sprendimų priėmimo greitis buvo nepakankamas – lygus 2,77 milisekundės. Papildomai pritaikius

daugiamačius masyvus metode, GPU skaičiavimams DDP paspartinti, greitis pagerėjo iki 76,88 mikrosekundės, tačiau tai buvo nepakankamas greitis, palyginti su gaunamų duomenų greičiu iš elektroninių biržų, kuris lygus 32,27 mikroekundės.

Išskaidžius duomenis, naudojant daugiamates matricas, ir norint pasiekti didesnį prekybos sprendimų priėmimo greitį, būtina lygiagretinti visus galimus skaičiavimus. Norint geriau išnaudoti GPU branduolius ir atmintis, buvo panaudotas branduolių lygiagretinimas. naudojant kodo vektorizaciją. Perkėlus statistinio arbitražo DDP algoritmą į GPU, algoritmo duomenys formuojami kaip daugiamatės matricos ir algoritmo skaičiavimai, vykdomi lygiagrečiai išskaidant juos tarp atskirų lygiagrečių branduolių, leidžia priimti didelio dažnio prekybos sprendimus greičiau.



33 pav. DDP statistinio arbitražo strategijos branduolių lygiagretinimas GPU aplinkoje

Atlikus naudojamo algoritmo analizę, buvo išskirtos DDP statistinio arbitražo strategijos dalys, dar kitaip vadinamos branduoliais, kurias galima vykdyti lygiagrečiai. Jas sudaro:

a) duomenų normalizacija, generuojant daugiamates matricas;
b) prekybos porų parinkimas, naudojant mažiausių kvadratų ir kointegracijos metodus;
c) parinktų prekybos porų dublikatų naikinimas;
d) prekybos signalų aptikimas;
e) prekybos pozicijų uždarymas / atidarymas.

Perkėlus gautą didelio dažnio duomenų srautą iš elektroninių biržų į GPU globaliąją atmintį, jie formuojami kaip daugiamatės matricos ir išskirstomi bendroms gijų blokų atmintims. Šie duomenys gali būti lygiagrečiai normalizuojami, juos apskaičiuoti duodant kiekvienai gijų blokų gijai. Gauti normalizuoti duomenys formuojami kaip daugiamatės matricos. Jie grąžinami į globaliąją atmintį ir vėl perkeliami į gijų blokų bendrąsias atmintis. Šiame žingsnyje galima lygiagretinti du prekybos porų parinkimo branduolius, kuriuos galima vykdyti vienu metu. Grąžinus gautus duomenis atgal į globaliąją atmintį, panaikinami gauti prekybos porų dublikatai. Tyrime naudotos trys statistinio arbitražo prekybos strategijos, kurių prekybos signalų parinkimus galima vykdyti lygiagrečiai. Paskutiniame žingsnyje lygiagrečiai atliekami pozicijų uždarymai / atidarymai ir per ilgai laikomų pozicijų uždarymai.

Darbe taikomo testavimo metodo pradžioje buvo gaunami duomenys iš elektroninės biržos, kai CPU paruošia duomenis ir jie perkelimi į GPU. Duomenų perkėlimas vykdomas algoritme aprašyta *runSimulationOnGPU* funkcija: paleidžiamos dvi grafinės plokštės ir sukuriami GPU kintamieji. Vėliau paleidžiama funkcija *parallel*, kuria lygiagretinami skaičiavimai ir visas prekybos algoritmas perkeliamas į GPU. Šiuo metodu išvengiama ciklų, kuriuos paleidus, įsijungia CPU ir pailgėja skaičiavimai. Norint vykdyti skaičiavimus sparčiai, jie turi vykti tik GPU viduje. Skaičiavimai vykdomi kiekvieno CPU CUDA branduolio ciklo metu. Skaičiavimai nelaikomi eilėje, o vykdomi vienu metu. *Parallel* funkcija leidžia sukurti pirmiausia trimatę matricą, kurioje sudaroma 130x62000 matrica su ateities sandorių kainomis, nurodant pirkimo ir pardavimo kainas. Suformavus šią matricą, paleidžiama kita funkcija *mazinta,* kuri lygiagretina porų parinkimą, aptinka prekybos signalus ir atidaro / uždaro sandorius, taip pat uždaro ilgai laikomas atidarytas pozicijas. GPU globaliosios atminties sąskaita yra kuriamos papildomos daugiamatės matricos. Pirma, kuri sukuriama, yra

visos galimos poros kiekvienam ateities sandoriui. Lygiagrečiai ieškoma porų, taikant kointegracijos ir mažiausių kvadratų metodus. Tačiau kointegracijos metode taikant 4D tipo matricas, skaičiavimai pailgėja, nes dalis duomenų turi būti laikoma eilėje, kadangi jie netelpa turimose GPU atmintyse. Todėl, atlikus tyrimą ir nustačius metodą, kuris ne tik greitesnis, bet ir patikimesnis parenkant poras, pasirinktas mažiausių kvadratų metodas. Paleidus porų parinkimo funkciją *pair*, sukuriama nauja tik jau atrinktų porų 3D matrica. Joje yra poros, su kuriomis galėtų būti vykdoma prekyba. Tuo pačiu metu algoritmas naikina galimus porų dublikatus. Tada paleidžiama funkcija *trade*, kuri vykdo prekybos signalų aptikimą visomis trimis prekybos strategijomis. Gaunami rezultatai išlaikomi 3D matricoje ir, nors vienai jų radus prekybos signalą, formuojamas užsakymas. Tuo pačiu metu yra tikrinamos esamos pozicijos, ar jų dar nereikia uždaryti, ar neatsirado uždarymo signalas arba gal jau pasiektas maksimalus laikas laikyti pozicijas atidarytas. Čia baigiasi prekybos algoritmas, kuris suformuotus užsakymus kelia atgal į CPU, kuris pagal gautą informaciją jau siunčia prekybos užsakymus ar atšaukimus biržoms.

4.2 Skyriaus išvados

Šiame skyriuje aptarta kodo vektorizacija, nagrinėta, kaip ją taikyti, norint perkelti turimą algoritmą, kuris pritaikytas CPU, į GPU aplinką. Pateiktas naujas metodas, kuriuo galima perkelti didelio dažnio statistinio arbitražo prekybos strategijos algoritmą į GPU CUDA aplinką, nurodytos daugiamatės matricos ir lygiagretūs branduoliai. Apibendrintina, kad, naudojant minėtus metodus, skaičiavimai vykdomi lygiagrečiai su visais CUDA branduoliais. Naudojamas algoritmas padeda lygiagretinti skaičiavimus, esančius GPU, tarp visų CUDA branduolių užpildant globaliąją atmintį informacija, taip spartinant paties prekybos algoritmo veikimą. Remiantis pasiūlytu DDP testavimo metodu, šias didelio dažnio statistinio arbitražo strategijos algoritmo dalis galima skaičiuoti lygiagrečiai: duomenų normalizavimą, prekybos porų parinkimą, prekybos signalų paiešką, prekybos signalų ir pozicijos uždarymo siuntimą.

5 REIKALAVIMAI TYRIMUI

Šio tyrimo metu naudoti 130 skirtingų ateities sandorių *tick-by-tick* duomenys, kuriuos *NANOTICK* kompanija pateikė 100 nanosekundžių tikslumu; ateities sandoriai susideda iš *ME* grupės, kuriai priklauso *NYMEX*, *COMEX* ir *CBOT*. Laikotarpis, kurio metu simuliuojama prekyba su naudojamo didelio dažnio algoritmu, yra nuo 2018-04-30 iki 2018-08-03, tai 69 prekybos dienos. Gavus visus duomenis apie šiuos 130 ateities sandorių, buvo pamatuotas vidutinis laikas, kokiu dažnumu iš biržų apie juos gaunama nauja informacija. Minėtas laikas lygus 32,27 mikrosekundėms. Tai reiškė, kad, norint prekiauti šioje biržoje, reikia aplenkti kitus rinkos dalyvius ir pateikti informaciją greičiau negu per 32,27 mikrosekundes. Taigi, vidutiniškai kiekviena prekybos diena turėjo 536 909 612 duomenų eilučių 130 ateities sandorių. Taikant pasirinktus metodus, buvo apdorota apytiksliai po 45 GB duomenų kiekvienai prekybos dienai, t. y. apie 3TB duomenų visam prekybos laikotarpiui arba 69 798 249 560 duomenų eilučių. Duomenys, kurių reikia šiam tyrimui, yra ateities sandorio kodas, pirkimo ir pardavimo kainos.

5.1 Reikalavimai tyrimo įgyvendinimui

Galimybė padidinti šio algoritmo našumą priklauso nuo paties algoritmo, jo kodo ir nuo techninės įrangos, kurioje jis veikia. Šio tyrimo metu naudotos dvi grafinės plokštės: *NVIDIA GeForce GTX* 1060 6GB ir *NVIDIA GeForce GTX* 1070 Ti 8GB.

Superkompiuterių našumas matuojamas pagal slenkančio taško operacijas, kurias sistema gali atlikti per sekundę, arba kitaip FLOPS. Todėl vertėjo apskaičiuoti tyrime naudojamos sistemos FLOPS reikšmę. Atlikus skaičiavimus, buvo nustatyta, kad sistemoje naudojami du GPU, *Intel Xeon* E5-2650 ir 24 GB RAM pasiekia 13,276 TFLOPS dydį.

Dauguma sistemos našumo analizavimo sprendimų koncentruojasi į FLOPS dydį, spartinamosios atminties kreipimosi skaičių ir kitus su procesoriaus matrica susijusius parametrus. Dėl to, kad FLOPS atpigo, duomenų perdavimas tampa vienu brangiausių ir daugiausiai energijos reikalaujančių dalykų. Tačiau šiame tyrime svarbiausias buvo ne tiek FLOPS dydis, kiek siekis išsiaiškinti, kaip greitai duomenys gali keliauti tarp skirtingų atminčių. Būtent teisingas atminčių naudojimas yra viena

sudėtingiausių bet kurios didelio našumo sistemos vietų (Unat D. et al., 2015).

5.2 Didelio dažnio duomenys, naudoti tyrime

Kaip jau minėta, *NANOTICK* kompanija suteikė didelio dažnio duomenis apie 130 skirtingų ateities sandorų, kuriuos sudarė 18 agrokultūros, 15 energijos, 37 valiutų rinkos, 29 palūkanų normų ir 10 metalų ateities sandorių. Kiekvieną ateities sandorį sudaro nanosekundinės tikslumo *tick-by-tick* informacija.

11 lentelė. Ateities sandorių sąrašas, naudotas tyrime

Agro - kultūra	**Energija**	**Akcijos**	**Akcijos**	**FX**	**FX**	**FX**	**Palūk. normos**	**Palūk. normos**	**Metalai**
ZS	CL	ES	RSV	6E	SEK	ACD	GE	TUL	HG
ZM	RB	NQ	TPY	6J	EAD	ENK	ZQ	TAF	GC
KE	HO	YM	SDA	6A	EPZ	_	ZN	FIT	SI
ZL	BZ	RTY	_	RP	ZAR	_	ZB	FIX	MGC
LE	NG	NKD	_	6C	NOK	_	UB	NON	QO
ZW	QM	NIY	_	RY	PSF	_	NBY	NCB	PL
ZC	HH	BTC	_	ECD	J7	_	ZF	TUF	SIL
SOM	MB	EMD	_	M6E	ANE	_	FYN	TFY	ZNC
HE	NOB	IBV	_	6N	6Z	_	TUB	NUB	QC
XK	FOB	FT1	_	6S	MJY	_	FYT	ZT	PA
GF	BOB	XAY	_	AJY	6L	_	TN	N1U	_
XC	HP	XAK	_	E7	M6A	_	TUT	_	_
GD	NBP	XAE	_	6M	PLN	_	TFY	_	_
XW	TTE	FTU	_	PJY	SIR	_	BUB	_	_
DC	QG	XAV	_	RF	MCD	_	NOL	_	_
CSC	_	RS1	_	M6B	MSF	_	FOL	_	_
CJ	_	RSG	_	CNH	ESK	_	TEX	_	_
HET	_	XAI	_	SEK	MIR	_	TUX	_	_

Pagrindinė didelio dažnio duomenų problema yra ta, kad skirtingi ateities sandoriai pateikia informaciją skirtingu dažnumu ir per tą patį laiką gali būti skirtingas kiekis *tick-by-tick* duomenų. Tarkime, gali būti, kad ateities sandoris A pateikė naują informaciją 16:45:00.024827526, kitas ateities

kontraktas B –16:45:00.027226312, o laikotarpiu 16:45:00.024827526 nebuvo jokio judėjimo. Dėl to reikia sulyginti visas didelio dažnio ateities sandorių duomenų eilutes su laiko žymomis. Taip kontraktuose yra užpildomos trūkstamos laiko žymos su jų prieš tai buvusia kaina; jei nebuvo gauta nauja informacija, vadinasi, kaina nesikeitė. Taip ne tik suvienodinamos laiko žymos, bet ir išlaikoma tiksli informacija apie kainas. Tada yra paruošiami duomenys, kurie gali būti pasitelkiami norint simuliuoti prekybą naudojamos statistinio arbitražo didelio dažnio prekybos sistemos MATLAB aplinkoje.

Visos šiam tyrimui reikalingos informacijos iš *NANOTICK* kompanijos pavyzdys pateiktas 12 lentelėje.

12 lentelė. Naudotų duomenų pavyzdys

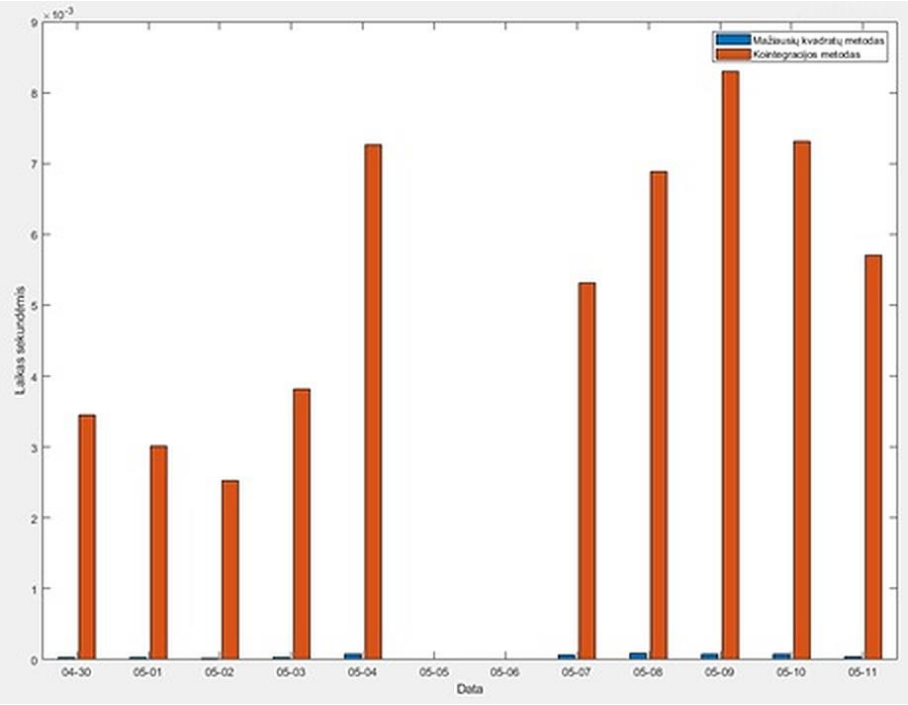
Data	**Laikas**	**ID**	**Simbolis**	**Fin. instrumentas**	**Grupė**	**Tipas**	**Kaina**
20180604	16:45:00.024776853	15144	GE:PS M1-M2	GE	GE	A	0,75
20180604	16:45:00.024794969	15144	GE:PS M1-M2	GE	GE	A	0,5
20180604	16:45:00.024810037	15144	GE:PS M1-M2	GE	GE	A	0,25
20180604	16:45:00.024827526	15144	GE:PS M1-M2	GE	GE	A	0,25
20180604	16:45:00.024827526	15144	GE:PS M1-M2	GE	GE	B	0,25
20180604	16:45:00.024932383	15144	GE:PS M1-M2	GE	GE	A	0,25
20180604	16:45:00.024932383	15144	GE:PS M1-M2	GE	GE	B	0,25
20180604	16:45:00.024949463	15144	GE:PS M1-M2	GE	GE	A	0,25
20180604	16:45:00.024949463	15144	GE:PS M1-M2	GE	GE	B	0,25
20180604	16:45:00.067992741	882493	GE:PK 01Y M0	GE	GE	A	1
20180604	16:45:00.068062075	882493	GE:PK 01Y M0	GE	GE	A	1
20180604	16:45:00.068062075	882493	GE:PK 01Y M0	GE	GE	B	1,25
20180604	16:45:00.068415901	40757	GE:PK 01Y M1	GE	GE	B	0,75
20180604	16:45:00.068584110	40757	GE:PK 01Y M1	GE	GE	A	0,5
20180604	16:45:00.068584110	40757	GE:PK 01Y M1	GE	GE	B	0,75
20180604	16:45:00.070814563	121147	GE:PK 01Y M2	GE	GE	B	0,5
20180604	16:45:00.070885473	121147	GE:PK 01Y M2	GE	GE	A	0,25
20180604	16:45:00.070885473	121147	GE:PK 01Y M2	GE	GE	B	0,5

Pateiktoje lentelėje matyti, kad yra galimi du skirtingi įrašų tipai A ir B. A (*ask*) yra pirkimo informacija, o B (*bid*) yra pardavimo informacija. Prekybos algoritmas, kuris naudojamas šiame tyrime, pasitelkia duomenis atitinkamai, t. y. jis perka ateities sandorius tik už pirkimo kainą, o parduoda tik už pardavimo kainą, jų nemaišydamas. Kadangi šio tyrimo metu statistinio arbitražo didelio dažnio prekyba buvo vykdoma su didelio dažnio duomenimis, tai išvengta galimybės, kad, pateikus pirkimo ar pardavimo kainą, signalui nukeliavus iki biržos, ši kaina bus pasikeitusi.

Tolimesnėje tyrimo eigoje naudotos tos pačios trys statistinio arbitražo prekybos strategijos, kurios buvo aptartos 3 skyriuje. Tačiau šio tyrimo tikslas nėra rasti pačią pelningiausią statistinio arbitražo didelio dažnio algoritmo konfigūraciją, o šį algoritmą panaudoti taip, kad jis prekybos sprendimus priimtų kuo greičiau, aplenkdamas pačią biržą ir kitus jos dalyvius.

5.3 Tyrimo rezultatai

Išsikėlus tyrimo tikslą kuo greičiau apdoroti iš elektroninės biržos gautus duomenis, priimti prekybos sprendimus ir juos perduoti atgal į elektroninę biržą, reikėjo patikrinti, ar abu porų parinkimo metodai yra pakankamai greiti rasti poras naudojama techninė įranga. Siekiant nustatyti jų greičius, buvo pasirinkta jas panagrinėti atskirai su turimais duomenimis laikotarpį nuo 2018-04-30 iki 2018-05-11. Minėta, kad skirtingi finansiniai instrumentai rinkose juda skirtingu dažnumu. Dėl to algoritmai seka ne paprastą laiką, bet GPU procesorių ciklų skaičių, t. y. prekybos langas šiame tyrime yra jau ne 20 sekundžių, bet gautų duomenų kiekis, kuris yra 61 996 duomenų eilučių. Atlikus skaičiavimus šiuo laikotarpiu su mažiausių kvadratų ir kointegracijos metodais, buvo apskaičiuotas vidutinis prekybos sprendimo priėmimo greitis, naudojant visas tris strategijas vienu metu.



34 pav. Laiko palyginimas pagal vidutinį atsakymo laiką kiekvieną prekybos dieną, naudojant mažiausių kvadratų atstumo ir kointegracijos metodus

Baigus šį tyrimą, buvo nustatyta, kad, naudojant mažiausių kvadratų metodą, algoritmas geba greičiau atlikti prekybos sprendimus, negu naudojant kointegracijos metodą prekybos porų parinkimui. Abu porų parinkimo metodai buvo sukonfigūruoti taip, kad jie skaičiavimus vykdytų lygiagrečiai. Nors kointegracijos metodas turi vykdyti daugiau skaičiavimų nei mažiausių kvadratų metodas, tačiau juos galima išskirstyti visiems tyrime naudojamiems CUDA branduoliams. Bet būtent taikant kointegracijos metodą, norint jį vektorizuoti, teko panaudoti keturmečius masyvus ir, kaip minėta anksčiau, GPU geriau dirba su duomenimis iki trijų dimensijų dėl to, kad tokio tipo duomenys telpa į globaliąją atmintį. Atsiradusiai ketvirtai dimensijai reikia daug papildomos vietos ir ne visada šie duomenys telpa GPU atmintyje. Tai lėmė, jog šis metodas lėčiau geba aptikti ateities sandorių poras. Tyrimo metu buvo siekiama aplenkti elektroninę biržą, kuri *tick-by-tick* informaciją pateikdavo vidutiniškai kas 32,27 mikrosekundes.

13 lentelė. Vidutinis atsakymo laikas kiekvieną prekybos dieną, naudojant mažiausių kvadratų atstumo ir kointegracijos metodus prekybos porų parinkimui

Data	Kointegracijos metodas (mikrosekundės)	Mažiausių kvadratų metodas (mikrosekundės)
2018.04.30	345,62	0,9446
2018.05.01	411,8	0,3017
2018.05.02	253,12	0,22312
2018.05.03	381,86	0,3018
2018.05.04	726,42	0,8264
2018.05.05	0	0
2018.05.06	0	0
2018.05.07	531,42	0,67142
2018.05.08	688,87	0,88587
2018.05.09	830,38	0,80938
2018.05.10	731,56	0,80156
2018.05.11	571,07	0,40101

Naudodamas kointegracijos metodą ir priimdamas prekybos signalus, prekybos algoritmas užtrunka ilgiau, negu informacija atkeliauja iš biržos, tad tolimesnis tyrimas buvo sutelktas tik į algoritmą su mažiausių kvadratų porų parinkimo metodu. Tačiau, dar prieš pasirenkant šį metodą, buvo pamatuotas Šarpo rodiklis, skaičiuojant kiekvienos prekybos dienos pelningumą, siekiant įsitikinti, kad pasirinktas metodas yra ne tik greitesnis, bet ir efektyvesnis.

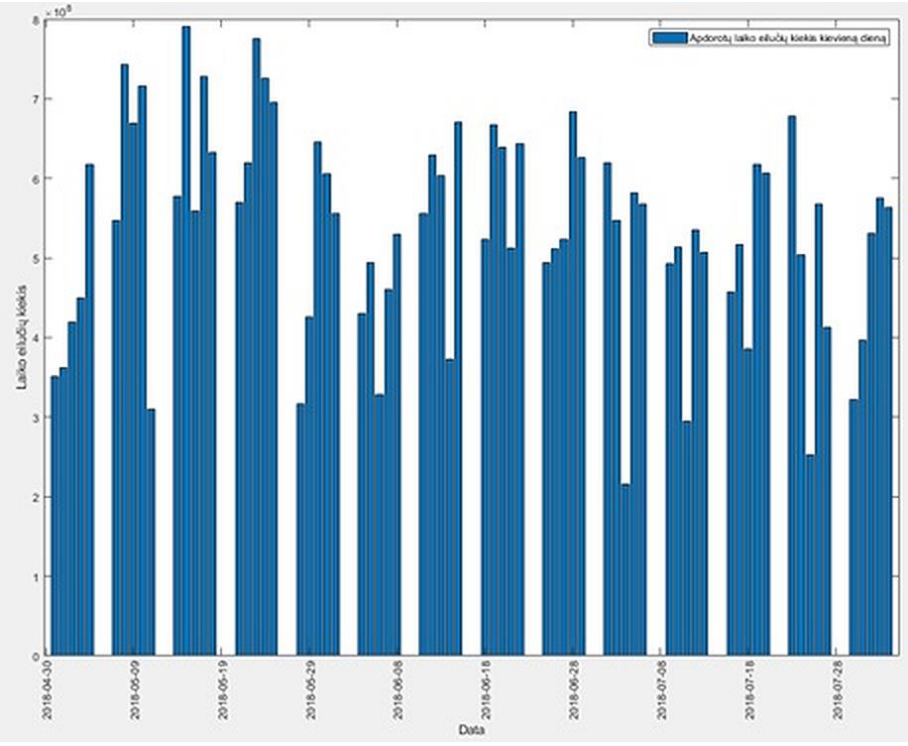
14 lentelė. Abiejų porų parinkimo metodų pelno Šarpo rodiklis

Porų parinkimo metodas	Vidurkis	Standartinis nuokrypis	Šarpo rodiklis
Mažiausių kvadratų metodas	2060,55	2039,217237	1,010461819
Kointegracijos metodas	951,514	1332,581485	0,714038351

Kuo Šarpo rodiklis yra didesnis, tuo naudojamo algoritmo efektyvumas yra geresnis. Pateiktoje lentelėje matyti, kad, naudojant mažiausių kvadratų metodą, Šarpo rodiklis yra didesnis. Atliekant eksperimentą, visi parametrai,

išskyrus porų parinkimo metodą, buvo išlaikomi vienodi. Todėl šis metodas ne tik greičiau randa ateities sandorių poras, kurias galima naudoti ieškant prekybos signalų, bet ir rastos poros yra efektyvesnės, nes, taikant šį metodą, gaunamas didesnis efektyvumas. Todėl tolimesniame tyrime su likusiais *tick-by-tick* trijų mėnesių duomenimis ir buvo pasirinkta naudoti tik mažiausių kvadratų poros parinkimo metodą; nes tik su juo naudojama techninė įranga galėjo aplenkti iš rinkos gaunamą informaciją.

Tyrimo metu taikant pasiūlytą metodą, kuris sujungia kodo vektorizaciją, daugiamates matricas ir lygiagrečius branduolius, kiekviena prekybos diena vidutiniškai turėdavo 536 909 612 duomenų eilučių visiems 130 ateities sandorių. Viso tyrimo metu buvo apdorota vidutiniškai 69 798 249 560 įrašų statistinio arbitražo didelio dažnio prekybos algoritmu, naudojant dvi GPU (*NVIDIA GeForce GTX* 1060 6GB ir *NVIDIA GeForce GTX* 1070 Ti 8GB.). Priklausomai nuo prekybos dienos likvidumo, duomenų skaičius skyrėsi, tikslesnė informacija apie kiekvienos dienos duomenų eilučių kiekį pateikiama grafike 35 paveikslėlyje.



35 pav. Duomenų eilučių kiekis kiekvieną prekybos dieną

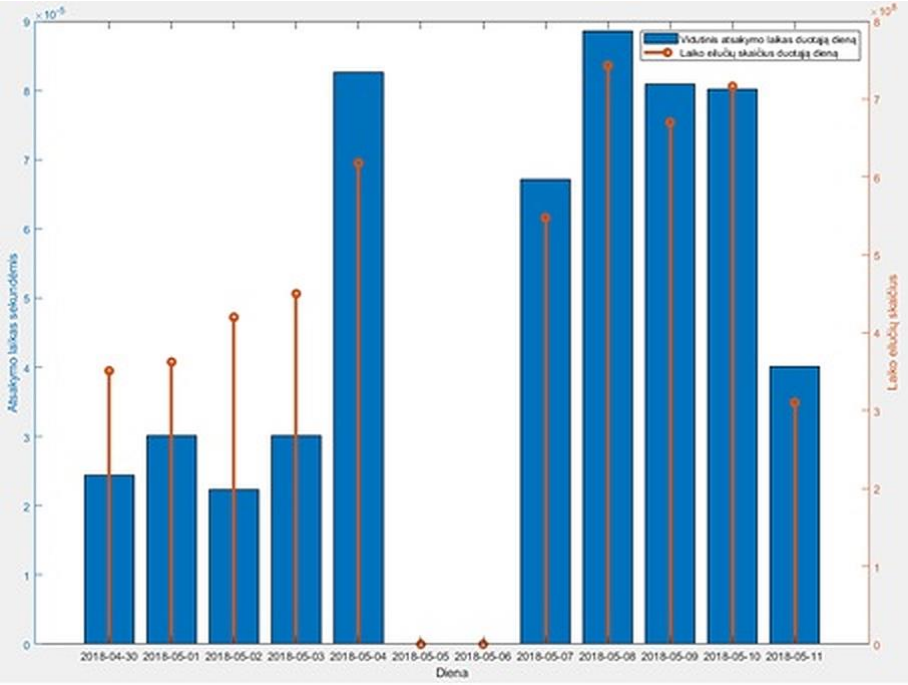
Atlikus tyrimą, taikant darbe siūlomą DDP statistinio arbitražo testavimo metodą, su visomis 69 prekybos dienomis, buvo nustatyta, kad pasitelktas DD statistinio arbitražo prekybos testavimo metodas galėjo apdoroti gautus duomenis ir priimti prekybos sprendimą vidutiniškai per 711,11 nanosekundžių, kai naudojamas prekybos duomenų eilučių langas, kurio dydis 61 996. Žemiau esančiame grafike (36 pav.) matyti vidutinis duomenų apdorojimo ir atsakymo laikas kiekvieną prekybos dieną atskirai. Iš grafiko paaiškėjo, kad apdorojimo laikai šiek tiek skyrėsi. Taip nutiko todėl, kad dienomis, kai atsakymo laikas buvo šiek tiek ilgesnis, rinkos buvo likvidesnės, todėl būdavo didesnis duomenų judėjimas. Tai reiškė, kad ir algoritmas turėdavo dažniau keisti poras, galimų porų skaičiai skyrėsi, reikėjo dažniau sukurti prekybos signalus ir uždarinėti bei atidarinėti pozicijas. Tačiau net ir pats ilgiausias apdorojimo ir atsakymo laikas buvo daug greitesnis už tą, kokiu dažnumu birža galėdavo siųsti informaciją.

15 lentelė. Didelio dažnio statistinio arbitražo prekybos strategijos vidutinis atsakymo laikas kiekvieną prekybos dieną

Data	Vid. Atsakymo laikas (ns)	Data	Vid. Atsakymo laikas (ns)	Data	Vid. Atsakymo laikas (ns)	Data	Vid. Atsakymo laikas (ns)
2018-04-30	244,62	2018-05-24	386,78	2018-06-17	0	2018-07-11	341,47
2018-05-01	301,7	2018-05-25	354,23	2018-06-18	324,76	2018-07-12	614,15
2018-05-02	223,12	2018-05-26	0	2018-06-19	368,3	2018-07-13	489,85
2018-05-03	301,86	2018-05-27	0	2018-06-20	363,87	2018-07-14	0
2018-05-04	826,42	2018-05-28	253,9	2018-06-21	373,68	2018-07-15	0
2018-05-05	0	2018-05-29	394,4	2018-06-22	307,42	2018-07-16	274,59
2018-05-06	0	2018-05-30	392,25	2018-06-23	0	2018-07-17	577,54
2018-05-07	671,42	2018-05-31	298,06	2018-06-24	0	2018-07-18	314,32
2018-05-08	885,87	2018-06-01	666,66	2018-06-25	338,35	2018-07-19	875,09
2018-05-09	809,38	2018-06-02	0	2018-06-26	513,61	2018-07-20	911,02
2018-05-10	801,56	2018-06-03	0	2018-06-27	542,07	2018-07-21	0
2018-05-11	401,07	2018-06-04	285,35	2018-06-28	935,94	2018-07-22	0

2018-05-12	0	2018-06-05	344,73	2018-06-29	842,56	2018-07-23	929,27
2018-05-13	0	2018-06-06	325,77	2018-06-30	0	2018-07-24	465,58
2018-05-14	401,07	2018-06-07	333,56	2018-07-01	0	2018-07-25	306,27
2018-05-15	428,17	2018-06-08	361,91	2018-07-02	822,69	2018-07-26	719,94
2018-05-16	443,23	2018-06-09	0	2018-07-03	694,62	2018-07-27	300,83
2018-05-17	475,52	2018-06-10	0	2018-07-04	256,65	2018-07-28	0
2018-05-18	409,72	2018-06-11	339,09	2018-07-05	823,5	2018-07-29	0
2018-05-19	0	2018-06-12	327,37	2018-07-06	778,66	2018-07-30	321,89
2018-05-20	0	2018-06-13	344,87	2018-07-07	0	2018-07-31	301,02
2018-05-21	416,18	2018-06-14	338,61	2018-07-08	0	2018-08-01	612,55
2018-05-22	421,61	2018-06-15	364,34	2018-07-09	433,09	2018-08-02	752,91
2018-05-23	388,58	2018-06-16	0	2018-07-10	570,39	2018-08-03	741,65

Naudojami lygiagrečių skaičiavimų metodai, kodo vektorizavimas jį perkėlus į GPU, daugiamatės matricos ir lygiagretūs branduoliai padeda paspartinti duomenų normalizavimą, ateities sandorių prekybos porų parinkimą ir prekybos signalų bei pozicijų uždarymo signalų išsiuntimą biržai. Kadangi visa informacija atkeliauja iš biržos vidutiniškai kas 32,27 mikrosekundes, tai pasiūlytas sprendimas turėjo apdoroti gautą informaciją bei rasti prekybos signalus greičiau už šį laiką; tą ir pavyko padaryti.



36 pav. Didelio dažnio statistinio arbitražo strategijos vidutinis atsakymo laikas ir duomenų eilučių skaičius nurodytą prekybos dieną

Atidžiau panagrinėjus prekybos periodą nuo 2018-04-30 iki 2018-05-11, matyti, kaip yra susijęs duomenų eilučių kiekis ir jų apdorojimo laikas kiekvienai nurodyto periodo prekybos dienai. Išanalizavus duomenis, paaiškėjo, kad yra koreliacija tarp duomenų kiekio ir atsakymo greičio, nes, esant didesniam duomenų kiekiui, atsakymų laikai tampa ilgesni – tai nutinka todėl, kad atsiranda daugiau galimų ateities sandorių porų, dažniau keičiasi atidarymo ir uždarymo pozicijos bei turimų resursų nepakanka. Todėl kai kurie skaičiavimai, deja, turi laukti eilėje, nes nepakanka vietos GPU atmintyje, kad būtų galima juos perkelti į lygiagrečius skaičiavimus.

Tačiau pasiūlytas DD statistinio arbitražo didelio dažnio prekybos testavimo metodas, kurį taikant algoritmas yra perkeltas į GPU su turimais resursais, paspartino duomenų apdorojimo laiką ir prekybos sprendimų priėmimą 83,7 %, palyginti su prieš tai atliktais tyrimais, kurių metu į GPU buvo perkeltas ne visas algoritmas, o tik jo dalis, nenaudojant daugiamačių matricų, lygiagrečių branduolių ir pasitelkiant tik penkis ateities sandorius su tokio pat dažnumo duomenimis. Šiuo atveju jų atsakymo laikas buvo 849,5 mikrosekundės. Pasiūlytas metodas ne tik greičiau apdoroja duomenis, bet

gali apdoroti ir 26 kartus daugiau duomenų bei tą atlikti 119,46 kartų greičiau, t. y. priimti prekybos sprendimus per 711,11 nanosekundžių.

5.4 Skyriaus išvados

Tyrimo pabaigoje pavyko įrodyti, kad pasiūlytas didelio dažnio statistinio arbitražo prekybos testavimo metodas leidžia vykdyti prekybą tokiu greičiu, kuriuo prekybos sprendimai priimami greičiau, negu gaunama nauja informacija iš elektroninės biržos. Pasitelkus šį metodą, GPU CUDA branduoliai panaudojami lygiagretiesiems skaičiavimams, GPU globalioji atmintis – duomenų laikymui ir spartesniam jų perdavimui CUDA skaičiavimo branduoliams, daugiamatės matricos – vektorizuoti kodui, ir lygiagretinami branduoliai leidžia vykdyti skirtingus skaičiavimus vienu metu. Pasitelkus visus išvardytus metodus, buvo pasiektas užsibrėžtas tikslas ir paspartinti prekybos algoritmo skaičiavimai. Bendrai naudojamos visos trys didelio dažnio statistinio arbitražo prekybos strategijos su mažiausių kvadratų metodu, skirtu ieškoti prekybos porų, pagerino prekybos sprendimų priėmimo laiką 83 %, palyginti su anksčiau tyrimuose taikytais metodais ir technine įranga. Šio tyrimo metu, vietoje anksčiau pasiekto 84,95 mikrosekundžių prekybos sprendimo priėmimo greičio, pavyko pasiekti 711,11 nanosekundžių prekybos sprendimo priėmimo greitį.

6 BENDROS IŠVADOS

1. Darbe nustatytos didžiausią rinkos dalį užimančios DDP strategijos. Formalizuota rinkoje lyderio pozicijas užimanti, statistinio arbitražo strategija, užtikrinanti koreliuojančių finansinių instrumentų susietumą, finansinių rinkų tarpusavio integralumą. Atlikus eksperimentinius tyrimus, nustatyta, kad dienos uždarymo kainų statistinio arbitražo strategijos gali būti efektyviai pritaikomos didelio dažnio prekyboje. Kompiuterizavus statistinio arbitražo DDP strategijas, nustatyta lygiagrečių skaičiavimų būtinybė siūlomame metode, siekiant atlikti DDP patikimumo testavimus su istoriniais duomenimis, vykdyti didelio dažnio prekybą realiu laiku. Atlikus palyginamąją FPGA ir GPU taikymo analizę, skirtą spręsti keliamus uždavinius, nustatytas GPU pranašumas dėl slankiojo kablelio skaičiaus operatoriaus, skaičiavimų resursų pakankamumo ir ekonomiškumo.
2. Sudarant metodą, išskirtos DDP algoritmo dalys, kurioms taikytinas lygiagretinimas, t. y. DDP duomenų normalizavimo, prekybinių instrumentų porų parinkimo ir anuliavimo, DDP prekybos signalų identifikavimo, prekybinių pozicijų atidarymo / uždarymo algoritmo dalys perkeltos į GPU skaičiavimus. Algoritmo kodo vektorizavimas parodė, kad vidutinis sprendimų priėmimo greitis lygus 2,77 milisekundėms, tačiau jis yra nepakankamas didelio dažnio duomenų aplinkos prekyboje.
3. Siekiant paspartinti vidutinį DDP sprendimų priėmimo greitį, kartu su kodo vektorizacija pritaikytos daugiamatės matricos GPU skaičiavimams. Atlikus eksperimentą, nustatyta, kad taikant šiuos skaičiavimų lygiagretinimo metodus, prekybos sprendimų greitis pagerėjo iki 76,88 mikrosekundžių, tačiau jis buvo nepakankamas, palyginti su gaunamų duomenų greičiu iš elektroninių biržų, kuris lygus 32,27 mikrosekundėms.
4. Norint geriau išnaudoti GPU branduolius ir atmintis, buvo panaudotas branduolių lygiagretinimas. Sujungus jį su kodo vektorizavimu ir daugiamatėmis matricomis, darbe siūlomas metodas pasiekė DDP greitį, kuris yra didesnis už gaunamų didelio dažnio duomenų greitį iš elektroninių biržų. Atliktas eksperimentas atskleidė, kad, taikant pasiūlytą metodą, buvo sumažinta skaičiavimų išsišakojimų, todėl sprendimų greitis padidėjo iki 389,63 kartų, palyginti su pradiniu metodu, ir tapo lygus 711,11 nanosekundžių. Tai yra 45,37 kartų greičiau nei gaunami duomenys iš biržų.

LITERATŪRA

[1] Adil S.H. and Qamar S., (2009). Implementation of association rule mining using CUDA Emerging Technologies, *ICET 2009, International Conference*, pp. 332 - 336.

[2] Aldridge I. (2013), High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems. John Wiley & Sons, 368 p. ISBN: 978-0-470-57977-0.

[3] Aldrige, I. (2010), High-Frequency Trading - A Practical Guide to Algorithmic Strategies and Trading Systems, John Wiley and Sons, Inc, Hoboken, New Jersey.

[4] Anane M. and Abergel F., (2014). Optimal high frequency strategy in an omniscient order book.

[5] Asaduzzaman A., Gummadi D. and Yip C. M., (2014). A talented CPU-to-GPU memory mapping technique, *SOUTHEASTCON 2014*, Lexington, KY, pp. 1-6.

[6] Avellaneda, M. and Stoikov, S. (2008), High-frequency trading in a limit order book, *Quantitative Finance,* Vol. 8, No. 3.

[7] Baporikar N., (2009). Understanding Financial Engineering, *MBA Journal*, Vol. 10, No. 1.

[8] Beckhardt B., Frankl D.E., Lu C. and Wang, M.I. (2016). A Survey of High-Frequency Trading Strategies.

[9] Bershova, N. ir Rakhlin, D., (2013), High-frequency trading and long-term investors: a view from the buy-side, *Journal of Investment Strategies*, Vol. 2, pp. 25-69.

[10] Binh D., Faff R. and Hamza K., (2006). A new approach to modeling and estimation for statistical arbitrage. *In Proceedings of 2006 Financial Management Association European Conference*, pp. 87–99.

[11] Bogdan O., Tudorel A. and Dragoescu R. M., (2012). Improving the performance of the linear systems solvers using CUDA, *in Proceedings of the Challenges of the Knowledge Society International Conference*.

[12] Bogoev D. and Karam A. (2016). An Empirical detection of High Frequency Trading Strategies, *6th International Conference of the Financial Engineering and Banking Society*, June 10-12, Melaga.

[13] Brogaard J., Hendershott J. T., Riordan R. (2013), High frequency trading and price discovery, *ECB Lamfalussy fellowship programme/ Working paper series*, No 1602, European central bank Press.

[14] Busch D. (2016) MiFID II: regulating high frequency trading, other forms of algorithmic trading and direct electronic market access, *Law and Financial Markets Review*, Vol. 10, No. 2, pp.72-82.

[15] Caldeira J., Moura G. V. (2013). Selection of a portfolio of pairs based on cointegration: A statistical arbitrage strategy, *Revista Brasileira de Financas*, Vol. 11, No. 1, pp. 49–80.

[16] Capgemini (2012), High Frequency Trading: Evolution and the Future , [žiūrėta 2019-07-21]. Prieiga per Internetą: https:// www.capgemini.com/resource-file-access/resource/pdf/High_Frequency_Trading__Evolution _and_the_Future.pdf.

[17] Chelva M. S. and Sharanappa V. H., (2016). A Performance Study of GPU, FPGA, DSP and Multicore Processors For Embedded Vision Systems.

[18] Clark, C., (2011). Improving speed and transparency of market data. [žiūrėta 2018-03-21]. Prieiga per Internetą: https://exchanges.nyx.com/cclark/improving-speed-andtransparencymarket-data.

[19] Coon B. W. and Lindholm J. E. (2008). System and method for managing divergent threads in a SIMD architecture. US Patent 7,353,369.

[20] Coon B. W. and Lindholm J. E. (2009). System and method for managing divergent threads using synchronization tokens and program instructions that include setsynchronization bits. US Patent 7,543,136.

[21] Coon B. W., Lindholm J. E., Mills P. C. and Nickolls J. R. (2010). Processing an indirect branch instruction in a SIMD architecture. US Patent 7,761,697.

[22] Coon B. W., Nickolls J. R., Lindholm J. E. and S. Tzvetkov D. (2011). Structured programming control flow in a SIMD architecture. US Patent 7,877,585.

[23] Dacorogna M. M., Gencay R., Muller U., Olsen R. B., and Olsen O. V., (2001). An introduction to high frequency finance. *International Review of Economics & Finance*, Elsevier, Vol. 12, No. 4, pp. 525-529.

[24] Desai J.,Trivedi R., Nisarg A. Joshi.(2012) The case of Gold and Silver: A New Algorithm for Pairs Trading, Shri Chimanbhai Patel business and reserch institute.

[25] Dickey D., Fuller W. (1979). Distribution of the Estimator for Autoregressive Time series with a Unit Root, *Journal of the American Statistical Association*, Vol. 74, pp. 427-431.

[26] Driaunys K., Masteika S., Sakalauskas V., Vaitonis M. (2014). An algorithm-based statistical arbitrage high frequency trading system to forcast prices of natural gas futures. *Transformations in business and economics*, Vol.13, No. 3, pp. 96–109.

[27] Dunis C., Giorgioni G., Laws J., Rudy J. (2010). Statistical arbitrage and high frequency data with an application to Eurostoxx 50 equities. *Alternative investment analyst review, in series: Trading strategies*, p.35-57.

[28] Durbin M., (2010). All About High-Frequency Trading. McGraw-Hill, New York.

[29] Easley, D., de Prado, M. L. and OHara, M. (2013), High-Frequency Trading - New Realities for Traders, Markets and Regulators, isk Books, a Division of Incisive Media Investments Ltd.

[30] Elliot R. J., Hoek J. V., Malcolm W. P. (2005). Pairs Trading. Quantitative Finance, 5(3), p.271-276.

[31] Engle, R. F. and Granger C. W. J., (1987). Co-integration and error correction: Representation, estimation, and testing, *Econometrica,* Vol. 55, No. 2, pp. 251–276.

[32] Fabozzi, F. J., Focardi, S. M. and Jonas, C. (2011), `High-frequency trading: Methodologies and market impact', *Review of Futures Markets* Vol. 19, 738.

[33] Farber R. (2011) CUDA application design and development. Elsevier.

[34] Ferguson R., Laster D. (2007). Hedge funds and systemic risk. Financial Stability Review – Special issue on hedge funds. Banque de France.

[35] Flynn M., (2011). Flynn's taxonomy. In: *Encyclopedia of parallel computing*. Springer, 2011, pp. 689–697.

[36] Foley D. and Danskin J., (2017). Ultra-performance Pascal GPU and NVLink interconnect. In: *IEEE Micro,* Vol. 37, No. 2, pp. 7–17.

[37] Fox M. B., Glosten L. R., Rauterberg G. V. (2015), The New Stock Market: Sense and Nonsense, 65 Duke L.J. 191.

[38] Garland M., et al. (2008). Parallel Computing Experiences with CUDA, *IEEE Micro*, Vol. 28, No. 4, pp. 13-27.

[39] Gatev E., Goetzmann W. N., Rouwenhorst K. G. (2006). Pairs Trading: Performance of a Relative-Value Arbitrage Rule. *The Review of Financial Studies*, Vol. 19, No.3, pp. 797-827.

[40] GPU applications catalog. [žiūrėta 2018-03-21]. Prieiga per Internetą: https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/catalog/.
[41] Gregoriou, G. N. (2015), The Handbook of High Frequency Trading, Academic Press, Elsevier Inc.
[42] Gresse C., (2011), Effects of the competition between multiple trading platforms on market liquidity: evidence from the MiFID experience, [žiūrėta 2019-09-11]. Prieiga per Internetą: http://www.cencor.com/Media/Docs/E§ects_of_competition_between_multiple_trading_platforms_on_mkt_liquidity_- Greese.pdf.
[43] Grozea C., Bankovic Z. and Laskov P., (2010). FPGA vs. Multi-core CPUs vs. GPUs: Hands-On Experience with a Sorting Application, Facing the Multicore-Challenge.
[44] Harris, L. (2003), Trading and Exchanges, Oxford University Press.
[45] Hasbrouck, J., Sofianis, G., ir Sosebee, D., (1993) New York Stock Exchange Systems and Trading Procedures. Working Paper No. 93-01, New York Stock Exchange, New York, NY.
[46] He F., Ren Y., Chen Y. ir Cao Q. (2015). GPU Accelerate RD Algorithm Based on MATLAB, *International Journal of Advanced Research in Computer Science & Technology*, Vol. 3, No. 4, pp. 84-86.
[47] Herlemont D. (2013), Pairs Trading, Convergence Trading, Cointegration, *Quantitative Finance*, Vol. 12, No. 9.
[48] High-frequency trading – a discussion of relevant issues (2013), [žiūrėta 2018-09-15]. Prieiga per Internetą: http://www.eurexchange.com/blob/exchange-en/455384/490346/6/data/presentation_hft_media_workshop_lon_en.pdf
[49] Hoffmann H., Agarwal A., and Devadas S., (2009). Partitioning strategies for concurrent programming, Massachusetts Institute of Technology (MIT), CSAL Lab.
[50] Hogan S., Jarrow R., Teo M., Warachka M. (2004). Testing market efficiency using statistical arbitrage with applications to momentum and value strategies. *Journal of Financial Economics*, Vol. 73, No. 3, p.525-565.
[51] Horrigue, L., Ghodhbane, R., Saidani, T. ir Atri M. (2018). GPU acceleration of image processing algorithm based on Matlab CUDA. *International Journal of Computer and Network Security*., Vol. 18, pp. 91–99.

[52] Huck N. and Afawubo K., (2015). Statistical arbitrage and selection methods: is cointegration superior?, *Applied Economics*, Vol. 47, No. 6, pp. 599-613

[53] Jacquier A., (2017). Some Notes On Python For Finance, Department of Mathematics, Imperial College Londo, [žiūrėta 2018-09-21]. Prieiga per internetą: https://wwwf.imperial.ac.uk/~ajacquie/IC_IntroPython/IC_IntroPython_Docs/PythonNotes.pdf.

[54] Jaramillo C. (2016), The Revolt against High-Frequency Trading: From Flash Boys, to Class Actions, to IEX, *Review of banking & financial law*, Vol. 35, pp. 483 – 499.

[55] Jones D. H., Powell A., Bouganis C. H. and Cheung P. Y. K., (2010). GPU Versus FPGA for High Productivity Computing, *In Proceedings of the 2010 International Conference on Field Programmable Logic and Applications (FPL '10). IEEE Computer Society*, Washington, DC, USA.

[56] Josephine A. and Fransson L., (2016). Algorithmic Trading Based on Hidden Markov Models, , [žiūrėta 2018-09-29]. Prieiga per internetą: https://gupea.ub.gu.se/bitstream/2077/44767/1/gupea_2077_44767_1.pdf.

[57] K. Group (1992). OpenGL - The Industry's Foundation for High Performance Graphics. [žiūrėta 2018-02-10]. Prieiga per Internetą: https://www.opengl.org/.

[58] K. Group (2009). The open standard for parallel programming of heterogeneous systems. [žiūrėta 2018-02-10]. Prieiga per Internetą: https://www.khronos.org/opencl/.

[59] Kaya O. (2016), High – frequency trading. Reaching the limits, *Automated trader magazine*. Vol. 41, pp. 23 – 27.

[60] Kantz H. and Schreiber T., (2004). Nonlinear Time Series Analysis. Cambridge University Press.

[61] Kauffman R. J., Liu J. ir Ma D., (2015) Innovations in Financial IS and Technology Ecosystems: High-Frequency Trading Systems in the Equity Market,. *Technological Forecasting and Social Change*, Vol. 99, pp. 339-354.

[62] Kearns M., Kulesza A. and Nevmyvaka Y., (2010). Empirical Limitations on High Frequency Trading Profitability, [žiūrėta 2018-11-28]. Prieiga per Internetą: https://ssrn.com/abstract=1678758.

[63] Kirk D.B., Hwu W.M., (2010). Programming Massively Parallel Processors: A Hands-On Approach, Morgan Kaufmann Publishers.

[64] Klöckner A., Pinto N., Lee Y., Catanzaro B., Ivanov P. ir Fasih A.(2012). PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation, *Parallel Computing*, Vol. 38, No. 3, pp. 157–174.

[65] Krauss C. (2015), Statistical arbitrage pairs trading strategies: Review and outlook, *IWQW Discussion Paper Series*, No. 09/2015.

[66] Labaki J., Ferreira L. O. S. and Mesquita E., (2011). Constant Boundary Elements on graphics hardware: a GPU-CPU complementary implementation. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, Vol. 33, No. 4, pp. 475-482.

[67] Lai T. L. and Xing H., (2008). Statistical Models and Methods for Financial Markets, Springer Texts in Statistics, Springer.

[68] Li P., Zhang Q., Zhao R. and Yu H., (2015). Data layout transformation for structure vectorization on SIMD architectures, *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Takamatsu,, pp. 1-7.

[69] Limaye S. S., (2014). Electronically aided High frequency trading, *International Journal of Engineering Research and Applications*, pp. 14 – 18.

[70] Linton O. ir Mahmoodzadeh S., (2018), Implications of high-frequency trading for security markets, *Annual Review of Economics*, Vol. 10, pp. 237–259.

[71] Lyuu, Y. (2001). Financial Engineering and Computation: Principles, Mathematics, Algorithms. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511546839

[72] Lou, X. (2012). Acceleration of Distance-to-Default with GPU (Dissertation). [žiūrėta 2019-08-11]. Prieiga per Internetą: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-102877

[73] Mackintosh P., (2019), Time is Relative: Where Trade Speed Matters, and Where It Doesn't, [žiūrėta 2019-07-21]. Prieiga per Internetą: https://www.nasdaq.com/articles/time-relative%3A-where-trade-speed-matters-and-where-it-doesnt-2019-05-30.

[74] Mariano R. S. and Kuen Tse Y., (2008). Econometric Forecasting And HighFrequency Data Analysis,Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore, World Scientific Publishing Company.

[75] Matlab. (2015). se.mathworks.com. [žiūrėta 2018-11-15]. Prieiga per Internetą: https://se.mathworks.com/discovery/matlab-gpu.html.
[76] Matlab. (2016), se.mathworks.com. [žiūrėta 2018-11-15]. Prieiga per Internetą: https://se.mathworks.com/help/distcomp/gpu-computing.html.
[77] Maureen O., (2015), High frequency market microstructure, *Journal of Financial Economics*, Vol. 116, No. 2, pp. 257-270.
[78] Mesquita, E., Labaki, J., Ferreira, L.O.S., 2009, An Implementation of the Longman's Integration Method on Graphics Hardware, *CMES: Computer Modeling in Engineering & Sciences*, Vol. 51, No. 2, pp. 143-168.
[79] Miao, G. J. (2014). High Frequency and Dynamic Pairs Trading Based on Statistical Arbitrage Using a Two-Stage Correlation and Cointegration Approach. *International Journal of Economics and Finance*, Vol. 6, No. 3, pp. 96 – 110.
[80] Microsoft (1996). Microsoft DirectX / Direct3D. [žiūrėta 2018-02-10]. Prieiga per Internetą: https://msdn.microsoft.com/cscz/library/windows/desktop/hh309466(v=vs.85).aspx.
[81] Miika S. (2013) Algorithmic Pairs Trading: Empirical Investigation of Exchange Trade Funds, Master Thesis, Faculty of finance Aalto University.
[82] Miller R. S., Shorter G., (2016). High Frequency Trading: Overview of Recent Developments, *Congressional Research Service*, April 4; Washington D.C.
[83] Miller S. J. (2006). The method of least squares. Mathematics Department Brown University.
[84] Milutinovic V., Kotlar M., Stojanovic M., Dundic I., Trifunovic N., Babovic Z. (2017) DataFlow Systems: From Their Origins to Future Applications in Data Analytics, Deep Learning, and the Internet of Things. In: *DataFlow Supercomputing Essentials. Computer Communications and Networks*. Springer, Cham
[85] Minhas U. I., Bayliss S. and Contantinides G. A., (2014). GPU vs FPGA: A Comparative Analysis for Non-standard Precision, *In proceedings of Reconfigurable Computing: Architectures, Tools, and Applications*, pp. 298-305
[86] Morelli M., (2017), Implementing High Frequency Trading Regulation: A Critical Analysis of Current Reforms, *The Michigan Business & Entrepreneurial Law Review*, Vol. 6, No. 2, pp. 201-229.

[87] Mushtaq R. (2011). Augmented Dickey Fuller Test. [žiūrėta 2018-09-15]. Prieiga per Internetą: https://ssrn.com/abstract=1911068.

[88] Nambia P.P., Saveetha V., Sophia S. and Sowbarnika A., (2014). GPU Acceleration Using CUDA Framework, *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 2, No. 3, pp. 200-205.

[89] Nvidia (2007). CUDA Toolkit Documentation. [žiūrėta 2019-01-10]. Prieiga per Internetą: http://docs.nvidia.com/cuda/cuda-cprogramming-guide/index.html.7

[90] Nvidia (2007). Nvidia CUDA C/C++. [žiūrėta 2019-01-10]. Prieiga per Internetą: https://developer.nvidia.com/cuda-toolkit.

[91] Nvidia (2013). Parallel Thread Execution ISA. [žiūrėta 2019-01-10]. Prieiga per Internetą: http://docs.nvidia.com/cuda/parallel-threadexecution/index.html.

[92] Nvidia (2014). Nvidia Anaconda Accelerate. [žiūrėta 2019-01-10]. Prieiga per Internetą: https://developer.nvidia.com/anacondaaccelerate.

[93] Nvidia (2014). NVIDIA GeForce GTX 980 Featuring Maxwell, The Most Advanced GPU Ever Made. [žiūrėta 2018-02-11]. Prieiga per Internetą: http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_980_Whitepaper_FINAL.PDF

[94] Nvidia (2018). Nvidia Volta Architecture Whitepaper. [žiūrėta 2019-01-10]. Prieiga per Internetą: http://images.nvidia.com/content/volta-architecture/pdf/volta-architecturewhitepaper.pdf.

[95] Nvidia (2019). CUDA toolkit documentation. [žiūrėta 2019-06-01]. Prieiga per Internetą: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[96] Nvidia. OpenCL Programming Guide for the CUDA Architecture. [žiūrėta 2018-02-11]. Prieiga per Internetą: http://www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingGuide.pdf

[97] OHara M., ir Ye M., (2011) Is market fragmentation harming market quality, *Journal of Financial Economics*, Vol. 100, pp. 459-474.

[98] Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J., (2007). A survey of general-purpose computation on graphics hardware, *Computer Graphics Forum*, Vol. 26, No. 1, pp. 80-113.

[99] Perino F., (2014). Getting Started with MATLAB. [žiūrėta 2018-02-11]. Prieiga per Internetą: https://www.fisicamedica.it/sites/default/files/documenti/Perino.pdf.
[100] Perlin M. S. (2009). Evaluation of Pairs-trading strategy at the Brazilian financial market, *Journal of Derivatives & Hedge Funds*, Vol. 15, No. 2, pp. 122–136.
[101] PGI (2009). PGI CUDA Fortran Compiler. [žiūrėta 2018-02-11]. Prieiga per Internetą: http://www.pgroup.com/resources/cudafortran.htm.
[102] Ploskas N. ir Samaras N. (2016). GPU Programming in MATLAB, Morgan Kaufmann, pp. 37-70, ISBN 9780128051320
[103] Rasmusson, A., Mosegaard, J., Sørensen, T.S., 2008, Exploring parallel algorithms for volumetric mass-spring-damper models in cuda, *In: International Symposium on Computational Models for Biomedical Simulation*, pp. 49-58.
[104] Ryoo, S., Rodrigues, C.I., Baghsorkhi, S.S., Stone, S.S., Kirk, D.B., Hwu, W.M.W., (2008), Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, *In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, February 20-23, Salt Lake City, UT, USA.
[105] Saikia M. J., Kanhirodan R., and Vasu R. M., (2014). High-Speed GPU-Based fully three-dimensional diffuse optical tomographic system. *Journal of Biomedical Imaging*, Vol 3.
[106] Sangman K., Seonggu H., Yige H., Xinya Z. and Witchel E., (2014). GPUnet: Networking Abstractions for GPU Programs, *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, October 6–8, Broomfield, CO.
[107] Spampinato D. G. and Elstery A. C., (2009). Linear optimization on modern GPUs, *IEEE International Symposium on Parallel & Distributed Processing*, Rome, pp. 1-8.doi: 10.1109/IPDPS.2009.5161106
[108] Stantchev, G., Dorland, W., Gumerov, N., (2008), Fast parallel ParticleTo-Grid interpolation for plasma PIC simulations on the GPU, *Journal of Parallel and Distributed Computing*, Vol. 68 No. 10, pp. 1339-1349.
[109] Stantchev, G., Juba, D., Dorland, W., Varshney, A., (2009), Using Graphics Processors for High-Performance Computation and

Visualization of Plasma Turbulence, *Computing in Science and Engineering*, Vol. 11, No. 2, pp. 52-59.

[110] Sugerman J., Fatahalian K., Boulos S., Akeley K. and Hanrahan P., (2009). Gramps: A programming model for graphics pipelines, *ACM Trans. Graph*., Vol. 28, pp. 4:1–4:11.

[111] Uhrie R., Chaitali C. and John Brunhaver (2020). Automated Parallel Kernel Extraction from Dynamic Application Traces. ArXiv abs/2001.09995.

[112] Unat D., Chan C., Zhang W., Williams S., Bachan J., Bell J. and Shalf J. (2015). ExaSAT: An exascale co-design tool for performance modeling, *The International Journal of High Perfomance Computing Applications*, Vol. 29, pp. 209-232.

[113] Vaitonis M. (2017). Statistical arbitrage Using HFT in OMX Baltic Market, *Baltic J. Modern Computing*, Vol. 5, No. 1, pp. 37-49.

[114] Vaitonis M., (2018). CPU and GPU Implementations for High Frequency Trading in Algorithmic Finance, *Proceedings of the International Conference on Information Technologies*, pp. 119 – 124

[115] Vaitonis M., Masteika S. (2016). Research in High Frequency Trading and Pairs Selection Algorithm with Baltic Region Stocks, In: Dregvaite G., Damasevicius R. *Information and Software Technologies. ICIST 2016. Communications in Computer and Information Science*, Vol. 639. Springer.

[116] Vaitonis M., Masteika S. (2017). Statistical arbitrage trading strategy applied on future commodity market using nanosecond information, *ICIST 2017, Communications in Computer and Information Science*, Vol. 756. Springer.

[117] Vaitonis M., Masteika S., (2018), Experimental Comparison of HFT Statistical arbitrage Strategies Using the Data of Microsecond and Nanosecond Future Commodity Contracts, *Baltic J. Modern Computing*, Vol. 6, No. 2, pp. 195-216.

[118] Vaughan C.T., Cook J., Benner R.E., Dinge D.C., Lin P.T., Hughes C., Hoekstra R.J., Hammond S.D., (2018), On the Use of Vectorization in Production Engineering Workload, *Proceedings CUG 2018 Stockholm*, Sweden from 20-24 May.

[119] Véstias M. P. and Horácio C. N., (2014). Trends of CPU, GPU and FPGA for high-performance computing. *24th International Conference on Field Programmable Logic and Applications* (FPL).

[120] Vidyamurthy G. (2004). Pairs Trading – Quantitative Methods and Analysis. John Wiley & Sons, p. 224.
[121] Virgilio, G.P.M., (2019), High-frequency trading: a literature review, *Finance Market Profit Management* Vol. 33, pp. 183–208.
[122] Wang, Z. ir Zheng, W. (2014). High-Frequency Trading and Probability Theory. World Scientific Books, World Scientific Publishing Co. Pte. Ltd., No. 9233.
[123] Weston, J.P., (2002) Electronic Communication Networks and Liquidity on the NASDAQ, *Journal of Financial Services Research*, Vol. 22, No. 1, pp. 125-139.
[124] Zubulake, P. and Lee, S. (2011), The High Frequency Game Changer - How Automated Trading Strategies Have Revolutionized the Markets, John Wiley and Sons, Inc, Hoboken, New Jersey.

AUTORIAUS PUBLIKACIJŲ SĄRAŠAS DISERTACIJOS TEMA

1. Masteika S., Vaitonis M., Quantitative Research in High Frequency Trading for Natural Gas Futures Market, Business Information Systems Workshops, Springer International Publishing, Vol. 228, p. 29–35, 2015 m.

2. Vaitonis M., Masteika S. (2016). High frequency statistical arbitrage strategy engineering and algorithm for pairs trading selection. 7th International Workshop on Data Analysis Methods for Software Systems [abstracts book], Druskininkai, Lithuania, December 3-5, 2015. ISBN 978-9986-680-58-1. p. 51.

3. Vaitonis M., Masteika S. (2016). Research in high frequency trading and pairs selection algorithm with Baltic region stocks. Information and Software Technologies. 22nd International Conference, ICIST 2016, Druskininkai, Lithuania, October 13-15, 2016, Proceedings. ISBN 978-3-319-46254-7, p.p. 208 – 217.

4. Vaitonis M., (2017). Pairs Trading Using HFT in OMX Baltic Market. Baltic J. Modern Computing, Vol. 5(2017), No. 1, 37-49.

5. Vaitonis M., Masteika S. (2017) „Statistical Arbitrage Trading Strategy in Commodity Futures Market with the Use of Nanoseconds Historical Data“,Information and Software Technologies: 23rd International Conference, ICIST 2017, Druskininkai, Lithuania, October 12–14, 2017, Proceedings. R. Damaševičius and V. Mikašytė (Eds.): ICIST 2017, CCIS 756, pp. 303–313, ISBN 978-3-319-67642-

6. Vaitonis M., Masteika S. (2018). Experimental Comparison of HFT Pair Trading Strategies using Microsecond and Nanosecond Future Commodity Contracts Data. Baltic J. Modern Computing, Vol. 6(2018), No. 2, 195-216, ISSN 2255-8950.

PADĖKA

Labai dėkoju mokslinio darbo vadovui doc. dr. Sauliui Masteikai už atvedimą į mokslininko kelią, nuoseklų ir nuoširdų vadovimą rengiant disertaciją, kantrybę ir nuolatinį motyvavimą. Dėkoju už visas konsultacijas, pagalbą ir tikėjimą manimi.

Esu dėkingas disertacijos recenzentams prof. habil. dr. Leonidui Sakalauskui ir prof. dr. Audriui Lopatai, pateikusiems vertingų pastabų ir patarimų, padėjusių pagerinti šio darbo kokybę.

Taip pat dėkoju Vilniaus universiteto Kauno fakulteto bei Duomenų mokslo ir skaitmeninių technologijų instituto kolegoms už patarimus, konstruktyvią kritiką ir pagalbą rengiant disertaciją.

Ypač noriu padėkoti savo žmonai Janinai už kantrybę, motyvavimą judėti pirmyn ir supratingumą. Nėra žodžių, kuriais galėčiau išsakyti savo dėkingumą jai už nuolatinį palaikymą. Noriu padėkoti ir savo tėvams bei sesei, kurie visada buvo šalia.

Prie šios disertacijos rengimo vienaip ar kitaip prisidėjo ir daugiau žmonių. Visiems jiems noriu padėkoti už skatinimą ir kelio judėti pirmyn parodymą.

Mantas Vaitonis

UŽRAŠAMS

UŽRAŠAMS

UŽRAŠAMS

Mantas Vaitonis
DIDELIO DAŽNIO KOMPIUTERIZUOTŲ
PREKYBOS STRATEGIJŲ INŽINERIJA FINANSINĖSE RINKOSE
Daktaro disertacijos santrauka
Technologijos mokslai
Informatikos inžinerija (T 007)
Redaktorė Gabija Bankauskaitė

Vilniaus universiteto leidykla
Saulėtekio al. 9, LT-10222 Vilnius
El. p. info@leidykla.vu.lt
www.leidykla.vu.lt
Tiražas 20 egz.