VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Aukšto dažnio prekybos sistemų modeliavimas finansų biržose naudojant GPU lygiagrečiųjų skaičiavimų architektūrą bei genetinius algoritmus.

Modeling of a High Frequency Trading Systems Using GPU Parallel Architecture and Genetic Algorithms

Magistro baigiamasis darbas

Atliko:
Justinas Lipnickas (parašas)
Darbo vadovas:
dr. Aistis Raudys (parašas)
Recenzentas:
doc. dr. Rimantas Vaicekauskas (parašas)

Vilnius – 2012

SANTRAUKA

Šiuolaikiniame finansų pasaulyje duomenų analizė bei sugebėjimas greitai prisitaikyti prie jų pokyčio yra ypatingai svarbus, o kadangi duomenų kiekis yra itin didelis, reikalingi būdai kaip greitai ir tiksliai juos apdoroti. Negana to, informacija, naudojama prekybai finansų rinkose, labai greitai kinta, dėl to būtina pastovi ir pakartotina duomenų analizė, norint jog priimami prekybos sprendimai būtų kaip įmanoma teisingesni. Magistro darbe nagrinėjamos galimybės šiuos skaičiavimus pagreitinti naudojant NVIDIA CUDA lygiagrečiųjų skaičiavimų architektūrą bei genetinius paieškos algoritmus. Darbo metu sukurta aukšto dažnio prekybos modeliavimo sistema, kurios pagalba įvertinamas skaičiavimų trukmės sumažėjimas, naudojant GPU lygiagrečiuosius skaičiavimus, bei lyginant juos su skaičiavimų trukme naudojant įprastinius kompiuterio CPU. Atliekama keleto skirtingų GPU lustų skaičiavimų trukmės analizė, apžvelgiami esminiai skaičiavimų trukmę įtakojantys veiksniai, jų optimizavimo galimybės. Pritaikius visus skaičiavimų trukmę mažinančius veiksnius, buvo pasiektas skaičiavimų trukmės sumažinimas daugiau nei 27 kartus negu naudojantis įprastiniu kompiuterio procesoriumi.

Raktiniai žodžiai: lygiagretieji skaičiavimai, GPU, CUDA, genetiniai algoritmai, finansų rinka.

SUMMARY

Data analysis and the ability to quickly adapt to rapidly changing market conditions is the key if you want to have success in the current financial markets. Additionally, the amount of data you have to analyze is huge and fast, but precise, data analysis methods are required. In this Master thesis, I am analyzing the possibilities to use NVIDIA CUDA parallel computing architecture to increase the data analysis speed. Additionally, I am using genetic algorithms as a search technique to further increase the computational performance. During the course of this thesis, a high frequency trading modeling system was created. It is used to compare the time it takes to generate trading results using a GPU parallel architecture and using a standard computer CPU. Analysis of a several different GPUs is done, comparing the time needed for computations in comparison to the CUDA cores and other card specifications. A detailed research of possible optimization techniques is done, providing detailed data of the calculation performance increase for each of them. At the end, after all described optimization methods are applied, a total speed-up of the computations using GPU, while compared to the regular CPU, is more than 27 times.

Keywords: parallel computing, GPU, CUDA, genetic algorithms, financial markets.

TURINYS

ĮVADAS

Temos aktualumas

Šiuolaikiniame finansų pasaulyje duomenų analizė bei sugebėjimas greitai prisitaikyti prie jų pokyčio yra ypatingai svarbus, o kadangi duomenų kiekis yra itin didelis, reikalingi būdai kaip greitai ir tiksliai juos apdoroti. Kadangi įprastinių vieno branduolio kompiuterių procesorių taktinis dažnis nebedidėja jau nuo 2006 metų, skaičiavimų spartos padidėjimas gali būti pasiektas tik pasitelkus daugiabranduolines procesorių sistemas bei paskirstytuosius skaičiavimus. Išlieka esminis klausimas, kokią paskirstytųjų / lygiagrečiųjų skaičiavimų architektūrą pasirinkti, kuri tenkintų esminius problemos reikalavimus bei teiktų geriausią kainos / kokybės santykį. Savo magistro darbe pasirinkau išanalizuoti CUDA lygiagrečiųjų skaičiavimų architektūrą.

Finansinių duomenų analizės uždaviniui puikiai tinka skaičiavimo galimybės, kurias suteikia CUDA architektūra bei GPU (*angl. GPU – Graphics Processing Unit*) lustai. Kadangi pagrindinė prekybos sistemų generavimo bei įvertinimo skaičiavimų dalis yra atitinkamų parametrų rinkinių tikrinimas naudojantis istoriniais duomenimis, kurių kiekis paprastai yra labai didelis, ir atskiri parametrų rinkiniai nėra niekaip priklausomi vieni nuo kitų, šis uždavinys yra puikiai pritaikomas paskirstytiesiems skaičiavimams.

Galimybės atlikti įprastinius skaičiavimus pasinaudojant vaizdo plokštėse esančiais lustais (GPU) sukėlė revoliuciją lygiagrečiųjų skaičiavimų rinkoje, kadangi kiekvienam žmogui ar organizacijai tapo įmanoma su palyginti nedidelėmis pradinėmis investicijomis turėti nedidelį „superkompiuterį“ tiesiog savo namų kompiuteryje [MW12]. Tiesa sakant, daugelis galingesnių namų kompiuterių savininkų jau turi šias galimybes savo kompiuteriuose net nežinodami apie tai. Tai tapo įmanoma pristačius CUDA lygiagrečiųjų skaičiavimų architektūrą.

CUDA – NVIDIA® korporacijos pristatyta, įprastinių lygiagrečiųjų skaičiavimų architektūra, kurios pagalba tapo įmanoma išspręsti daugelį sudėtingų skaičiavimo uždavinių per daug trumpesnį laiką, negu tai užtruktų skaičiuojant įprastiniu kompiuterio procesoriumi (*angl. CPU – Central Prosessing Unit*) [Nvid12]. Palyginimui, įprastiniai namų kompiuterių procesoriai šiuo metu būna su 2-8 šerdimis (*angl. – Cores*), kai tuo tarpu NVIDIA grafiniai lustai turi iki 1024 CUDA šerdžių, todėl gali paraleliai atlikti daug daugiau skaičiavimų vienu metu.

Su daugiau nei 250 milijonų parduotų NVIDIA GPU lustų, kuriuose įdiegtas CUDA palaikymas, ši technologija tampa vis plačiau naudojama šiuolaikiniame pasaulyje. Tiek įprastiniuose grafinių vaizdų apdorojimo uždaviniuose, tokiuose kaip vaizdo žaidimai ar 3D modelių generavimas, tiek vis plačiau naudojama mokslinių tyrimų sferoje, sprendžiant

skaičiuojamosios biologijos, chemijos, skysčių termodinamikos ir kitus uždavinius. CUDA lygiagretieji skaičiavimai taip pat vis plačiau naudojami ir finansinių institucijų, kurioms taip pat yra aktuali skaičiavimų trukmė bei kokybė [Nvid09]. Ypač tai aktualu vykdant finansinę analizę/prekybą aukšto dažnio sistemose, kuomet nėra galimybės skaičiavimų rezultatų laukti ilgai, kadangi tuo metu jie jau būtų beverčiai. Būtent šią probleminę sferą ir planuoju tyrinėti savo magistro darbe.

Kita darbe nagrinėja probleminė sritis yra automatinės prekybos sistemos tinkamiausių parametrų parinkimas iš galimų parametrų aibės. Dažniausiai, siekiant prognozuoti ateities kainų pokyčius, yra naudojami atitinkami techninės ar fundamentaliosios analizės parametrai, kurie galėtų kaip įmanoma tiksliau nuspėti ateities kainų pokyčius. Šių galimų parametrų aibė dažniausiai būna ypač didelė ir tinkamiausių iš jų parinkimas, naudojantis istoriniais prekybos duomenimis, užtrunka labai daug laiko. Siekdamas sumažinti šį laiką bei išvengti būtinybės perskaičiuoti visus parametrų rinkinius, siekiant išrinkti geriausius iš jų, magistro darbe panaudosiu genetinius algoritmus. Genetiniai algoritmai - euristiniai paieškos algoritmai, kuriuose yra stengiamasi atkartoti natūralios evoliucijos procesus ir kurie yra paremti biologijos žiniomis apie gyvybės evoliuciją (Darvino evoliucijos teoriją).

Genetinių algoritmų skaičiavimuose taip pat didžiausią skaičiavimų laiko dalį užima atskirų individų vertinimas ir tik nedidelė dalis laiko yra skiriama pačių individų generavimui. Dėl šios priežasties, jie taip pat puikiai tinka paskirstytiesiems skaičiavimams [Wik12f].

Darbo tikslas ir uždaviniai

Darbo tikslas - atlikti esamų prekybos sistemų modeliavimo analizę, lygiagrečiųjų skaičiavimų naudojimo tyrimą. Panaudojant GPU lygiagrečiuosius skaičiavimus sukurti ypač aukšto dažnio prekybos sistemų modeliavimo įrankius, įvertinti sistemos suteikiamą pagreitėjimą, lyginant su tokios pačios sistemos realizavimu naudojant standartinę CPU architektūrą, naudojant 1 procesorių.

Išsikeltam darbo tikslui įgyvendinti, darbe bus įvykdyti šie **uždaviniai**:

1. Išanalizuoti automatinių prekybos sistemų kūrimo metodus, jų vertinimo kriterijus, apžvelgti fundamentaliosios bei techninės analizės naudojimo galimybes, jų skirtumus, pranašumus bei trūkumus.
2. Išnagrinėti NVIDIA CUDA architektūros specifiką, jos naudojimo būdus, privalumus bei trūkumus. Paruošti įprastinio namų kompiuterio su NVIDIA grafine plokšte paruošimo naudoti lygiagrečiuosius skaičiavimus instrukciją.

3. Išanalizuoti esamas genetinių algoritmų pritaikymo bibliotekas, išanalizuoti jų funkcijas bei taikymo galimybes konkrečiam prekybos sistemų modeliavimo uždaviniui.
4. Sukurti aukšto dažnio prekybos sistemos modeliavimo įrankius, naudojant juos atlikti praktinius tyrimus:
    4.1 Palyginti skaičiavimų greitį naudojant CPU bei GPU skaičiavimo metodikas, įvertinti kokį pagreitėjimą gauname naudojant lygiagrečiuosius skaičiavimus su GPU.
    4.2 Įvertinti, kaip skaičiavimų greitį bei tikslumą įtakoja skaičiavimo duomenų struktūra, vienu metu naudojamų gijų skaičius, išanalizuoti kitus skaičiavimo greitį su GPU įtakojančius veiksnius.
5. Išanalizuoti atliktų testų rezultatus, pateikti darbo išvadas.
6. Apžvelgti tolimesnių tyrimų galimybes, pasinaudojant šio darbo rezultatais.

Darbo struktūra

Šis magistrinis darbas susideda iš santraukos (lietuvių ir anglų kalbomis), įvado, 5 skyrių, suskirstytų į dvi dalis: literatūros apžvalgą bei analitinę dalį, išvadų, šaltinių sąrašo bei priedų.

Literatūros apžvalgoje (1-3 skyriai) yra apžvelgiami literatūros šaltiniai, susiję su darbo tema, problemine sritimi. Išsamiai nagrinėjama automatinės prekybos sistemos sudarymo technologija, CUDA lygiagrečiųjų skaičiavimų architektūra bei genetiniai algoritmai.

Analitinėje darbo dalyje yra aprašyti automatinės prekybos sistemos modeliavimo darbai, analizei bei skaičiavimams naudojami techniniai parametrai, prekybos duomenys (4 skyrius) bei pateikiama skaičiavimų spartos naudojant CPU bei GPU palyginamoji analizė (5 skyrius).

Paskutinėje darbo dalyje, išvadose, pateikiamos bendros darbo išvados, apžvelgiama kaip pasiekti rezultatai galėtų pasitarnauti tolimesnėse šios srities studijose.

LITERATŪROS APŽVALGA

Šioje darbo dalyje yra pristatoma magistro darbo temos literatūros apžvalga. Darbo tikslas yra įsigilinti į įvairiausius literatūros išteklius ir juose aptariamus sprendimus, kurie padėtų pasiekti magistro darbe keliamus tikslus. Darbe bus naudojamos tik plačiai žinomos ir naudojamos technologijos bei įrankiai.

Pirmajame šios dalies skyriuje apžvelgiama prekybos finansų rinkose problematika, automatinių prekybos sistemų kūrimo metodikos, apžvelgiamos fundamentaliosios bei techninės analizės panaudojimo galimybės, įvertinamos jų teigiamos bei neigiamos savybės.

Antrajame skyriuje apžvelgiami lygiagretieji skaičiavimai naudojantis vaizdo plokštėse esančiais lustais (GPU), pristatoma NVIDIA CUDA architektūra, įrankiai reikalingi darbui su ja. Įvertinama, kurie iš esamų sprendimų būtų tinkamiausi panaudoti tolimesniame darbe, siekiant įvykdyti darbe užsibrėžtus tikslus.

Trečiajame literatūros apžvalgos skyriuje apžvelgiami genetiniai algoritmai bei jų pritaikymas finansiniuose uždaviniuose, pristatoma *GALib* genetinio programavimo biblioteka, apžvelgiami jos panaudojimo būdai, pavyzdžiai.

1. Prekyba finansų rinkose

1.1. Investavimo problemos (uždavinio) apibrėžimas

Investavimo tikslas yra kaip įmanoma didesnis finansinis pelnas per apibrėžtą laiko tarpą, kai investuojama į atitinkamas finansines vertybes, atliekant tiek pirkimo, tiek pardavimo operacijas rinkoje. Atitinkamų finansinių išteklių vertė yra nuolat kintanti, priklausomai nuo įvairiausių vidinių bei išorinių faktorių, tokių kaip politiniai sprendimai, įvairus makroekonominiai rodikliai tiek atitinkamos ekonomikos kurioje yra investuojama, tiek viso aplinkinio pasaulio ir pan. Dėl to nėra vienos ir teisingos sistemos, su atitinkamais indikatoriais, kurie visada leistų pasiekti norimą maksimalų pelną. Vietoj to, turime daugybę įvairiausių galimų taisyklių, kurias apjungę į vieną sistemą galime tikėtis teigiamo rezultato. Tai ir leidžia apibrėžti investavimo uždavinį:

1. Surasti tinkamiausią indikatorių derinį, kuris labiausiai tiktų toje rinkoje ir tam turtui kuriuo planuojame prekiauti;
2. Parinkti atitinkamus parametrus, kuriuos pritaikę pasirinktuose indikatoriuose gautume tiksliausias investavimo prognozes, ko pasėkoje ir didžiausią grąžą investavimo pabaigoje.

Norėdamas išsirinkti tinkamus indikatorius, kuriuos naudočiau šiame darbe, apžvelgsiu dvi skirtingas metodologijas – Fundamentinę bei Techninę analizę. Investuojant galima naudotis tiek viena, tiek kita metodologija, dažnai abi šios metodologijos yra naudojamos kartu, viena kitą papildydamos. Toliau šiame skyriuje išsamiau apžvelgsiu šias abi metodologijas bei galimybes jas panaudoti savo darbe.

1.2. Fundamentalioji analizė

Fundamentalioji analizė remiasi mąstymu, jog rinka gali nustatyti neteisingą akcijos ar kito turto kainą tik trumpuoju laikotarpiu, tačiau ilguoju laikotarpiu „teisinga“ kaina, paremta fundamentaliais faktoriais, visada bus pasiekta. Pelnas, naudojantis šia metodologija, ir yra gaunamas siekiant surasti tokias dar rinkoje neįvertintas (pervertintas) akcijas, kurių fundamentalieji faktoriai rodo aiškiai didesnę vertę.

Vykdant fundamentaliąją analizę, yra analizuojami tokie fundamentalieji įmonių rodikliai kaip bendrovės finansiniai duomenys, veiklos rodikliai, finansinės ataskaitos, konkurentų kiekis bei padėtis rinkoje, vadyba, pardavimų vertė, apžvelgiamos bendros sektoriaus makroekonominės prognozės. Analizuojant valiutų kursų pokyčius Forex rinkoje yra gilinamasi į bendrą pasaulio bei atitinkamų valstybių/regionų ekonomikos būklę, palūkanų normas, vartotojų pasitikėjimo indeksus, infliaciją, valstybės skolas, pajamas, vadovybę ir pan. [Wik12a].

Kadangi yra begalė galimų indikatorių, kuriant prekybos sistemą pirmasis žingsnis ir yra atsirinkti tuos, kuriuos naudojant prognozės būtų tiksliausios. Šio darbo pagrindinė tema nėra teisingiausių parametrų parinkimas, jame remsiuosi indikatoriais, parinktais Yiyi Jang bei Laura Nunez [YN09] kaip labiausiai tinkamais genetiniu programavimu paremtose akcijų prekybos sistemose: kainos ir įmonės grynojo pelno santykis (P/E), kainos ir akcijos buhalterinės vertės santykis (P/BV), kainos ir apyvartos santykis (P/CF), įmonės skolų ir vertės santykis (D/BV), pardavimų vertės prieaugis (SG), pelno prieaugis (NIG), apyvartos prieaugis (CFG), turto pelningumas (ROA), pajamų ir turto santykio prieaugis (TOG), pelno maržos prieaugis (PMG) [Ban12], [Inv12]. Visi šie indikatoriai taip pat yra vertinami kaip vertingi sprendžiant dėl akcijų vertės pokyčio ir daugelyje kitų publikacijų ([CY06], [BDT08] ir kiti), bei yra dažniausiai naudojami priimant investavimo sprendimus. Pilną indikatorių aprašą bei jų skaičiavimo metodiką rasite lentelėje (1 Lentelė). Šiems indikatoriams skaičiuoti yra naudojama atitinkamo laikotarpio akcijos kaina rinkoje (KQ) bei devyni skirtingi duomenys iš įmonių ketvirtinių finansinių ataskaitų:

- Bendras įmonės turtas (BTQ)
- Įmonės įprastinių akcijų buhalterinė vertė (IABVQ)
- Trumpalaikis turtas (TTQ)

- Trumpalaikiai įsipareigojimai (iki 1 metų) (TIQ)
- Ilgalaikiai įsipareigojimai, paskolos (virš 1 metų) (IIQ)
- Bendras nusidėvėjimas ir amortizacija (BNAQ)
- Pelnas (nuostoliai) (PNQ)
- Bendrosios pajamos (BPQ)
- Paprastųjų akcijų skaičius (ASQ).

1 Lentelė. Fundamentaliosios analizės indikatoriai akcijų biržoje

Indikatorius	Aprašymas	Skaičiavimas
Indikatoriai skirti apibrėžti įmonės vertę		
P/E_t	Akcijos kainos ir įmonės grynojo pelno santykis	$\frac{KQ_t \cdot ASQ_t}{PNQ_{t-1}}$
P/BV_t	Akcijos kainos ir akcijos buhalterinės vertės santykis	$\frac{KQ_t \cdot ASQ_t}{IABVQ_{t-1}}$
P/CF_t	Akcijos kainos ir apyvartos santykis	$\frac{KQ_t \cdot ASQ_t}{PNQ_{t-1} + BNAQ_{t-1}}$
Indikatoriai parodantys įmonės įsiskolinimo lygį		
D/BV_t	Įmonės skolų ir vertės santykis	$\frac{TIQ_{t-1} + IIQ_{t-1} - TTQ_{t-1}}{IABVQ_{t-1}}$
Indikatoriai skirti įvertinti įmonės augimo perspektyvas		
SG_t	Įmonės pardavimų vertės prieaugis	$\frac{BPQ_{t-1}}{BPQ_{t-2}} - 1$
NIG_t	Įmonės pelno prieaugis	$\frac{PNQ_{t-1}}{PNQ_{t-2}} - 1$
CFG_t	Įmonės apyvartos prieaugis	$\frac{PNQ_{t-1} + BNAQ_{t-1}}{PNQ_{t-2} + BNAQ_{t-2}} - 1$
Indikatoriai skirti įvertinti įmonės valdymo efektyvumą		
ROA_t	Įmonės turto pelningumas	$\frac{PNQ_{t-1}}{BTQ_{t-2}}$
TOG_t	Įmonės pajamų ir turto santykio prieaugis	$\frac{BPQ_{t-1} \div BTQ_{t-1}}{BPQ_{t-2} \div BTQ_{t-2}} - 1$
PGM_t	Įmonės pelno maržos prieaugis	$\frac{PNQ_{t-1} \div BPQ_{t-1}}{PNQ_{t-2} \div BPQ_{t-2}} - 1$

Šie indikatoriai yra pagrinde skirti prekybai akcijomis modeliuoti. Darbų, kuriuose būtų apžvelgiami fundamentaliosios analizės indikatoriai automatiniai prekybai Forex biržoje, nėra

daug, jie gana smarkiai priklauso nuo valiutų poros, kuria yra prekiaujama, bet pagrindiniai faktoriai, turintys įtaką valiutų kursų dinamikai, apžvelgti [ELW+08], būtų šie: bazinių palūkanų norma (BPN), bendrasis vidaus produktas (BVP), užsienio prekybos balansas (UPB) bei vartotojų kainų indeksas (VKI). Kadangi kiekvienai valiutų kainų porai yra po dvi šalis/regionus, todėl viso turėtume 8 skirtingus indikatorius, kuriuos lyginant galima prognozuoti valiutų kursų pokyčius. Indikatoriai bei jų skaičiavimo metodika pateikti *2 Lentelėje*. Lentelėje pateikta tik keletas pavyzdžių, tie patys duomenys gali būti lyginami ir įvairiais kitais būdais bei santykiais.

2 Lentelė. Fundamentaliosios analizės indikatoriai Forex biržoje

Indikatorius	Aprašymas	Skaičiavimas
ΔBPN_t	Bazinių palūkanų pokytis	$BPN_{t-1} - BPN_t$
ΔBVP_t	Bendrojo vidaus produkto augimo santykis	$\frac{BVP_t^1 \div BVP_{t-1}^1}{BVP_t^2 \div BVP_{t-1}^2} - 1$
ΔUPB_t	Užsienio prekybos balanso pokyčio santykis	$\frac{UPB_t^1 \div UPB_{t-1}^1}{UPB_t^2 \div UPB_{t-1}^2} - 1$
ΔVKI_t	Vartotojų kainų indeksų pokyčio santykis	$\frac{VKI_t^1 \div VKI_{t-1}^1}{VKI_t^2 \div VKI_{t-1}^2} - 1$

1.3. Techninė analizė

Priešingai nei fundamentalioji analizė, techninė analizė remiasi mąstymu, jog visa įmanoma informacija apie bendrovę, regiono bei pasaulio ekonomikas jau yra įskaičiuota į akcijos (ar kito turto) kainą. Investuotojų emocinės reakcijos į atitinkamas naujienas formuoja pastebimas bei nuspėjamas kainų tendencijas, kurias ir stengiasi įžvelgti techninės analizės specialistams. „Tikroji“, ekonominiais faktais paremta, akcijos kaina techninės analizės specialistams nerūpi.

Pagrindiniai techninės analizės principai yra šie [Wik12b]:

- Kaina rinkoje įvertina viską – remiasi prielaida, jog kaina rinkoje jau yra įvertinus visą praeities, dabarties ir netgi ateities informaciją apie atitinkamą akciją, bei tai, ką apie tą informaciją mano investuotojai.
- Kaina rinkoje turi savybę kisti pagal nuspėjamas tendencijas – analitikai tiki, jog kainos juda atitinkamomis kryptimis: aukštyn, žemyn, į šoną, arba šių krypčių kombinacijomis. Kainų tendencijos rinkoje pirmą kartą pastebėtos ir aprašytos Šarlio Dau XIX a. antroje pusėje [Wik12c].

- Istorija yra linkusi kartotis – analitikai tiki, kad investuotojai yra linkę atkartoti veiksmus, kuriuos vykdė ankstesni investuotojai iki jų. Kadangi investuotojų elgesys kartojasi dažnai, analitikai tiki, jog grafikuose galima pastebėti (atpažinti) būsimą kainų pokyti naudojantis istoriniais duomenimis.

Naudojantis technine analize, pagrindinis darbo įrankis analitikams yra įvairūs kainų pokyčių grafikai, kuriuose yra stengiamasi surasti kokius nors kainų pokyčių šablonus, tendencijas. Skirtingai nei fundamentaliojoje analizėje, nėra esminių indikatorių skirtumų tarp akcijų ar Forex biržų, kadangi operuojama tais pačiais duomenimis, nepriklausomai nuo turto, kuriuo yra prekiaujama. Vienas paprasčiausių indikatorių, plačiai naudojamų techninėje analizėje, yra slenkantieji vidurkiai (MA), kurį matome *1 paveiksle*. Žvakidės stiliaus grafike matomas realus kainos kitimas, trumpo laikotarpio slenkantis vidurkis (5) pavaizduotas oranžine linija bei ilgesnio laikotarpio slenkantis vidurkis (17) – žalia linija. Pirkimo/pardavimo signalai, naudojant slenkančiuosius vidurkius yra gaunami, kai trumpesnio laikotarpio vidurkis kerta ilgesnio laikotarpio vidurkį bei pagal tai, iš kurios pusės šis perkirtimas įvyksta ir yra nustatoma sandorio operacija. Tipinė supaprastinta taisyklė prekybos sistemai būtų tokia:

- Jei $MA_5 > MA_{17}$ – pirkti
- Jei $MA_5 < MA_{17}$ – parduoti.

1 Paveikslas. MA Techninės analizės indikatoriaus pavyzdys.

Kiti pagrindiniai indikatoriai, taip pat analizuojami šiame darbe, yra: palaikymo bei pasipriešinimo lygiai (SL/RL), lyginamosios jėgos indeksas (angl. Relative Stength Index (RSI)), prekybos apyvartos pokytis (VOL). Indikatorių aprašymai bei skaičiavimo metodika pateikta *3 Lentelė*je. Indikatorių skaičiavimams yra naudojami tik duomenys iš rinkos, tai yra naudojama atitinkamų laikotarpių kaina (K), bei prekybos apyvarta (A).

3 Lentelė. Techninės analizės indikatoriai

Indikatorius	**Aprašymas**	**Skaičiavimas**
MA_N	Slenkantis vidurkis	$\frac{K_t + K_{t-1} + ... + K_{t-N}}{N}$
$MACD_{N\_M}$	Slankiųjų vidurkių kirtimosi indikatorius	$MA_N - MA_M$
SL_N	Palaikymo lygis	$MIN[K_{t-N}; K]$
RL_N	Pasipriešinimo lygis	$MAX[K_{t-N}; K]$
RSI	Lyginamosios jėgos indeksas	$100 - \frac{100}{1+RS}$
VOL_t	Kainos apyvartos pokytis	$\begin{cases} K_t > K_{t-1} \,\&\, V_t > V_{t-1} \rightarrow 1 \\ K_t < K_{t-1} \,\&\, V_t > V_{t-1} \rightarrow -1 \\ 0 \end{cases}$

Kadangi techninė analizė operuoja žymiai didesniais informacijos kiekiais bei ta informacija kinta nepalyginamai greičiau nei fundamentaliojoje analizėje naudojami duomenys, šiame darbe naudosiu techninės analizės duomenis bei indikatorius. Siekiant išsiaiškinti ypač aukšto dažnio bei didelio kiekio duomenų įtaką skaičiavimo spartai taip pat pasirinkau skaičiavimus atlikti su Forex biržos valiutų kursų duomenimis.

1.4. Automatinės prekybos sistemos modeliavimas

Prekybos automatizavimas yra dažnas reiškinys šiuolaikinėse finansų rinkose, kurį yra aktyviai tyrinėję daugybė tyrimų organizacijų bei mokslininkų. Iš esmės, automatinė prekybos sistema, dar vadinama prekybos robotu, yra kompiuterinė programa, kuri sugeba savarankiškai atlikti pirkimo / pardavimo sandorius finansų rinkoje, remdamasi nustatytomis taisyklėmis [LX09]. Taisyklės, kaip jau apžvelgėme ankstesniuose skyriuose, gali būti paremtos fundamentaliąja, technine analize, arba jų deriniu. Šiame darbe automatizuota prekybos sistema bus modeliuojama naudojantis istoriniais Forex biržos duomenimis bei imituojant prekybos sandorius realioje rinkoje. Modeliavimo tikslas - parinkti tokius indikatorių parametrus, kurie suteiktų didžiausią finansinę grąžą per atitinkamą periodą. Kaip jau minėjau, bus naudojami

technine analize paremti indikatoriai, o bendras prekybos taisyklių sąrašas pateiktas *4 Lentelėje*. Modeliuojant sistemą, bus generuojami atitinkami parametrų rinkiniai ir bus vertinama, kiek kuris parametrų rinkinys atneša grąžos (nuostolių) naudojant mokymosi laikotarpio duomenis, bei vėliau bandoma pritaikyti tuos pačius parametrus vėlesniems, įvertinimo duomenims. Parametrai, kurie bus naudojami parametrų rinkiniams generuoti yra šie:

- Trumpo laikotarpio slenkančio vidurkio bazė;
- Ilgo laikotarpio slenkančio vidurkio bazė;
- RSI viršutinė bei apatinė ribos;
- Siekiamas pelnas (TP, *angl. Take Profit*), ar naudojamas;
- Nuostolių apsauga (SL, *angl. Stop Loss*), ar naudojamas;
- Kiekvieno iš indikatorių svoris bendrame sprendimo priėmimo algoritme;
- Sandorio įvykdymo riba;

4 Lentelė. Techninė analize paremtos prekybos taisyklės

Taisyklė	Sąlygos	Rezultatas
Slenkantys vidurkiai (MA)	Jei Trumpo laikotarpio MA > Ilgo laikotarpio MA Jei Trumpo laikotarpio MA > Ilgo laikotarpio MA Kitais atvejais	Pirkti (+1) Parduoti (-1) Jokio veiksmo (0)
MACD	Jei MACD > 0 Jei MACD < 0 Kitais atvejais	Pirkti (+1) Parduoti (-1) Jokio veiksmo (0)
Pasipriešinimo bei palaikymo lygiai (RS)	Jei Kaina laikotarpio pabaigoje > Pasipriešinimo lygis Jei Kaina laikotarpio pabaigoje < Palaikymo lygis Kitais atvejais	Pirkti (+1) Parduoti (-1) Jokio veiksmo (0)
Lyginamosios jėgos indekas (RSI)	Jei RSI < Apatinė riba Jei RSI > Viršutinė riba Kitais atvejais	Pirkti (+1) Parduoti (-1) Jokio veiksmo (0)
RSI nuokrypis	Jei Kaina kyla ir RSI leidžiasi Jei Kaina leidžiasi ir RSI kyla Kitais atvejais	Pirkti (+1) Parduoti (-1) Jokio veiksmo (0)
Apyvarta (VOL)	Jei Kaina kyla ir Apyvarta Kyla Jei Kaina leidžiasi ir Apyvarta Kyla Kitais atvejais	Pirkti (+1) Parduoti (-1) Jokio veiksmo (0)

Kadangi galimų parametrų variantų yra labai daug, ypač jeigu yra parenkamos gana plačios pradinės jų ribos, reikia daug laiko, norint patikrinti visus galimus variantus. Šioms operacijoms paspartinti bus naudojami lygiagretieji skaičiavimai. Taip pat, siekiant sumažinti galimų parametrų variantų kiekį, vietoj intervalų gali būtų naudojamos baigtinės reikšmių sekos. Pavyzdžiui, jeigu vietoj Trumpo laikotarpio MA galimų reikšmių intervale [10;100] būtų naudojamas tik kas 5 elementas iš šio intervalo [10,15,20,...,100], visa galimų parametrų rinkinių

aibė sumažėtų 5 kartus, o rezultatų kokybei žymesnės įtakos neturėtų turėti. Šie bei kiti skaičiavimų paspartinimo variantai taip pat yra apžvelgiami šiame darbe.

2. Lygiagretieji skaičiavimai naudojantis vaizdo plokštėse esančiais lustais (GPU)

2.1. Lygiagretieji skaičiavimai

Lygiagretieji skaičiavimai yra skaičiavimų forma, kuomet daugybė skaičiavimų atliekama vienu metu, vadovaujantis principu, jog sudėtingiausios problemos gali būtų išskaidytos į daugybę mažesnių uždavinių, kurie gali būti apskaičiuojami nepriklausomai vieni nuo kitų. Šis skaičiavimų tipas yra naudojamas jau daug metų, pagrinde superkompiuterių panaudojime, tačiau pastaraisiais metais šis skaičiavimų tipas yra ir vis labiau naudojamas namų kompiuteriuose. Šis išpopuliarėjimas yra siejamas su fiziniais procesorių taktinio dažnio apribojimais dėl didelių energijos sąnaudų bei naujų kompiuterių procesorių, turinčių po keletą šerdžių (*angl. core*) atsiradimu maždaug 2004 metais. Yra keletas skirtingų paskirstytųjų skaičiavimų architektūros klasių, skirstomų pagal tai, kokiame technikos lygmenyje yra realizuotas skaičiavimų lygiagretumas. Galima išskirti tokias grupes kaip daugiaprocesorinės sistemos, daugiabranduolinės sistemos, kompiuterių klasteriai, keleto kompiuterių sistemos ir kitos [Wik12d], [Wik12e]. Šiame darbe nagrinėsime ganėtinai naują lygiagrečiųjų skaičiavimų tipą - skaičiavimus naudojantis grafinių vaizdo kortų procesoriais (GPU).

2.2. GPU paskirtis – vaizdo žaidimai

Norint išgauti sklandžius bei tikroviškus kompiuterinių žaidimų vaizdus yra būtina kaip įmanoma greičiau apskaičiuoti tą patį vaizdo generavimo algoritmą keletui milijonų ekrano taškų (pikselių). Būtent ši kompiuterijos sritis, kuri yra pasaulyje vertinama 74 milijardais JAV dolerių (2011metų duomenimis), ir padiktavo specifinių, didelę skaičiavimų galią turinčių, tačiau nelabai brangių, lustų atsiradimą. Savaime suprantama, kad šie lustai turi apribojimus, t.y. vienu metu visi grafinės kortos branduoliai gali atlikti tik vieną ir tą pačia operaciją, tačiau su skirtingais duomenimis – SIMD paradigma (*angl. SIMD – Single Instruction Multiple Data*) [KMJ+10]. Taigi, palyginus nebrangi, NVidia GTX550 Ti grafinė korta, kuri bus naudojama kaip pagrindinė atliekant skaičiavimus šiame darbe, kainuojanti maždaug 400 litų turi 192 CUDA branduolius, bei galinti atlikti 691,2 milijardų slankiojo kablelio operacijų per sekundę (GFlops). Palyginimui, tai yra maždaug 10 kartų daugiau operacijų nei 3,0 GHz dažnio Intel i5 procesoriaus, kainuojantis šiek tiek brangiau (500 litų) ir turintis 4 procesoriaus branduolius.

2.3. GPU pritaikymas įprastiniams lygiagretiesiems skaičiavimams

Pagrindinė GPU paskirtis - milijonus kartų atlikti tą patį algoritmą, naudojant skirtingus duomenis, puikiai dera su lygiagrečiųjų skaičiavimų uždaviniais, ypač naudojančiais genetiniais algoritmais paremtą programavimą. Genetinių algoritmų „brangiausia“ (daugiausiai skaičiavimo laiko užimanti) dalis, individų tinkamumo įvertinimas, yra labai panaši į grafinio vaizdo

apdorojimo uždavinį. Taip pat tai pastebima ir finansiniuose, prekybos sistemų generavimo, uždaviniuose, kurie taipogi galiausiai susiveda į daugybę kartų atliekamas tas pačias vertinimo operacijas, naudojant skirtingus pradinius duomenis.

Pagrindinis fizinis skirtumas tarp CPU bei GPU lustų architektūros yra tai, jog CPU luste didelė dalis tranzistorių yra skirta atminčiai, kai GPU beveik visa luste esanti erdvė yra užpildyta vadinamaisiais Aritmetiniais loginiais vienetais (*angl. ALU – Arithmetic Logic unit*), kurie yra skirti aritmetinėms bei loginėms operacijoms atlikti. Kadangi vidinė lusto erdvė yra ribota ir GPU lustuose erdvė nėra išnaudota dideliam lokalios atminties kiekiui, tai savaime suprantama, jog jo skaičiavimo galimybės yra žymiai didesnės nei analogiško CPU lusto [LKC+10]. Prie viso to dar prisideda tai, kad GPU lustai yra projektuoti taip, jog galėtų palaikyti didelį skaičių aktyvių skaičiavimo gijų vienu metu. Žinoma, GPU turi ir savo apribojimus. Vienas rimtesnių buvo tai, jog programuoti GPU procesoriams yra ganėtinai sudėtingą, dauguma programavimo kalbų skirtų programuoti GPU yra žemo abstrakcijos lygio bei yra būtina išmanyti bei naudoti atitinkamas grafines aplikacijų programavimo sąsajas (API). Visa tai iš esmės keičiasi NVIDIA pristačius CUDA architektūra bei programavimo priemonės. [Fer11].

2.4. CUDA architektūra

CUDA (*angl. - Compute Unified Device Architecture*) yra techninės bei programinės įrangos architektūra, kuri įgalina NVIDIA sukurtuose GPU vykdyti įprastinėmis programavimo kalbomis, tokiomis kaip C/C++, Fortranas, OpenCL ir kitos, parašytas programas [Nvid09].

Loginė architektūra

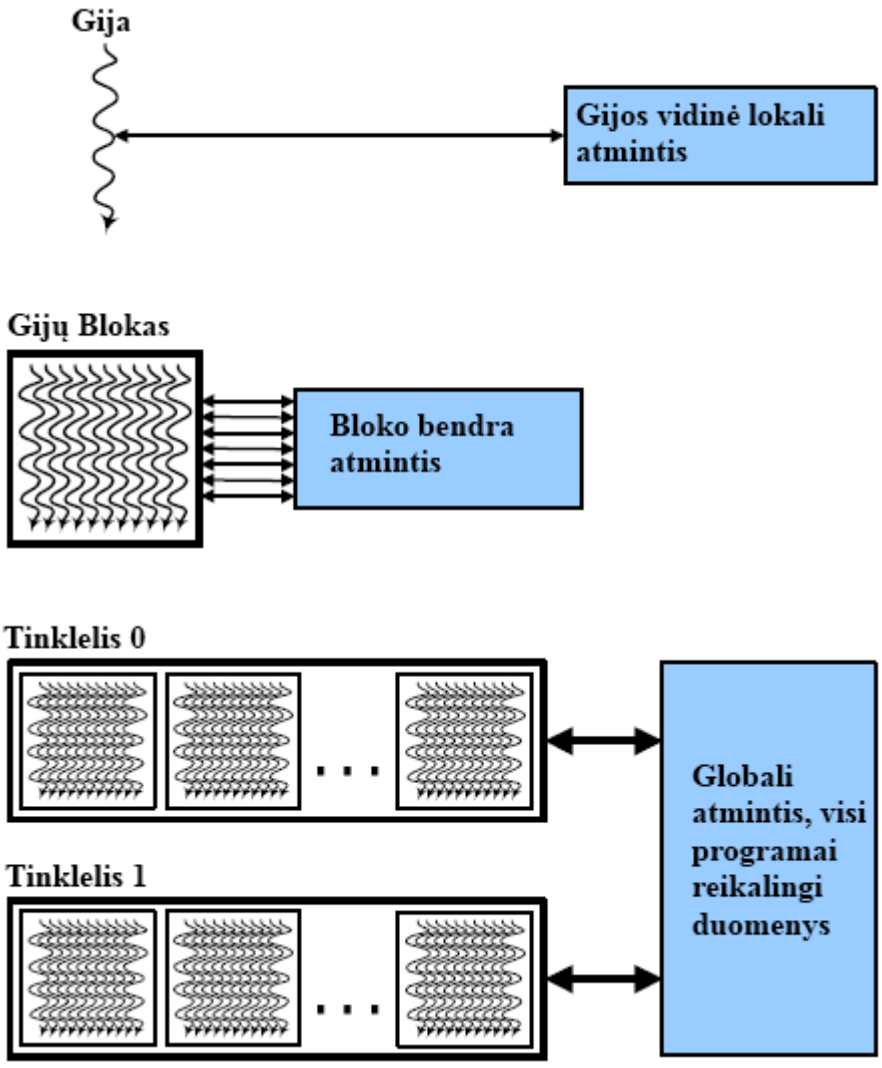
CUDA programa skaičiavimus su GPU inicijuoja lygiagrečiai, kviesdama *kernelius* (procedūros ar funkcijos analogas GPU). Vienas kernelis, pasileisdamas pasiskirsto į rinkinį lygiagrečių gijų (*threads*), kurias programuotojas arba kompiliatorius surūšiuoja į gijų blokus (*blocks*) ir gijų blokų tinklelius (*grids*). CUDA gijų, blokų bei tinklelių hierarchija pavaizduota *2 paveiksle*.

Kiekviena gija iš gijų bloko vykdo vieną kernelio skaičiavimą bei turi unikalų gijos numerį (ID) bloko viduje, vidinius registrus, lokalią atmintį bei gražina atitinkamą rezultatą.

Gijų blokas yra rinkinys tuo pačiu metu vykdomų gijų, kurios gali bendradarbiauti tarpusavyje naudodamosis bendra atmintimi bei sinchronizacijos funkcijomis. Gijų blokas turi savo unikalų numerį (ID) tinklelyje.

Tinklelis yra gijų blokų, kurie vienu metu vykdo to paties kernelio operacijas, rinkinys (masyvas). Tinklelis gauna duomenis iš bendros įrenginio atminties, įrašo skaičiavimų rezultatus į globaliąją atmintį bei atlieka sinchronizaciją tarp priklausomų kernelių iškvietimų.

Ši loginė CUDA gijų architektūra yra tiesiogiai atkartota fiziškai GPU procesorių architektūroje: visas GPU įrenginys vykdo vienu metu vieną ar keletą kernelio tinklelių, kiekvienas GPU daugia-branduolinis procesorius vykdo gijų blokų funkcijas, o kiekvienas to procesoriaus branduolys vykdo vienos gijos skaičiavimus. Procesorius, gijas vykdo sujungdamas jas blokais po 32. Programuotojai, norėdami pasiekti ypač gerą rezultatą skaičiavimų pagreitėjime turėtų atkreipti dėmesį į šį grupavimą bei pasirūpinti, kad visos gijos iš šios grupės vykdytų skaičiavimus tokiu pat keliu bei duomenis reikalingus skaičiavimams gautų iš netolimų atminties vietų.



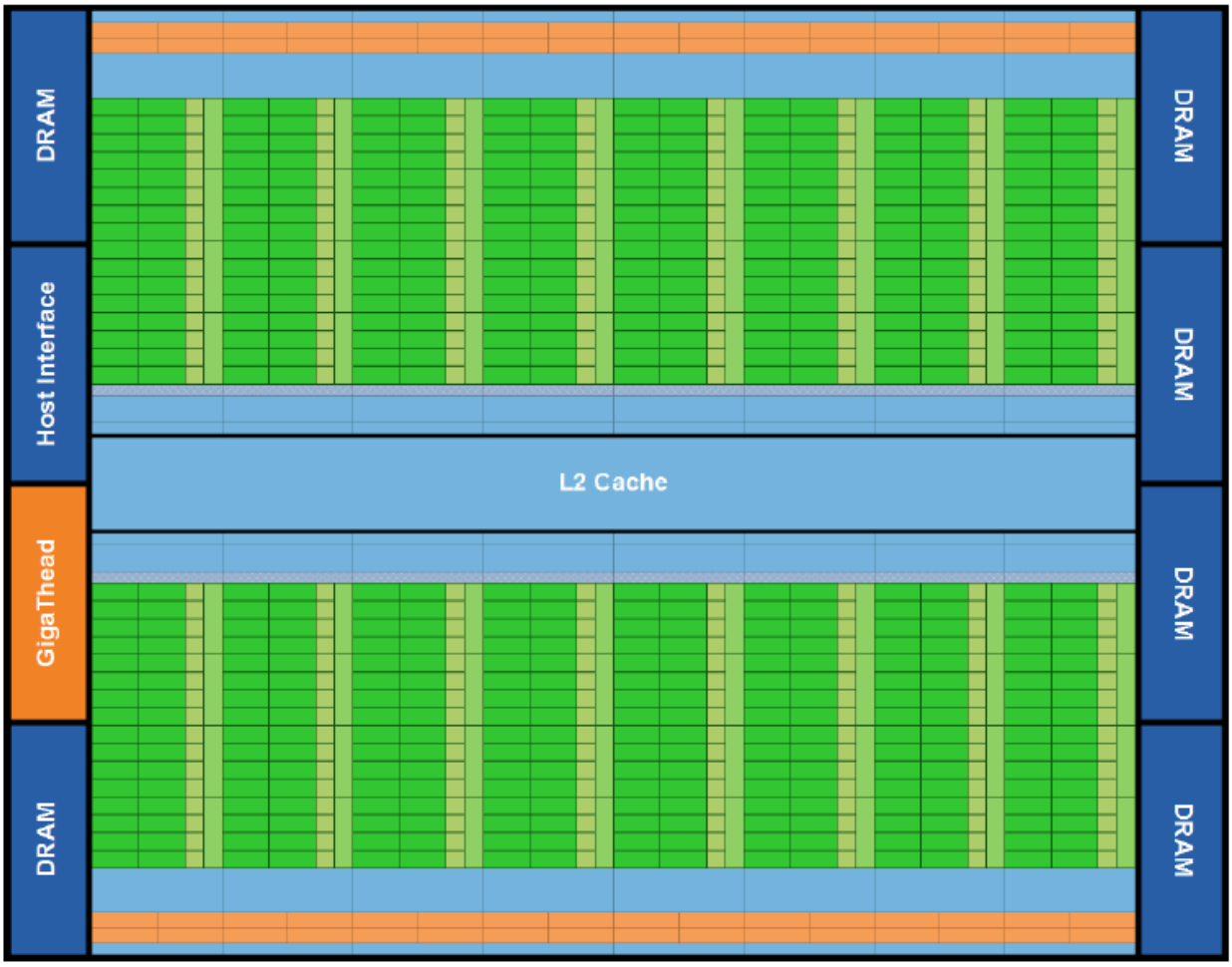
2 Paveikslas. CUDA gijų, blokų bei tinklelių loginė hierarchija.

Fizinė architektūra

Naujausia šiuo metu naudojama NVIDIA GPU architektūra, dar vadinama Fermi architektūra buvo pristatyta 2009 metais ir žymiai pagerino skaičiavimų kokybę GPU įrenginiuose. Bendra Fermi architektūros GPU struktūra pavaizduota *3 paveiksle*. Šiame

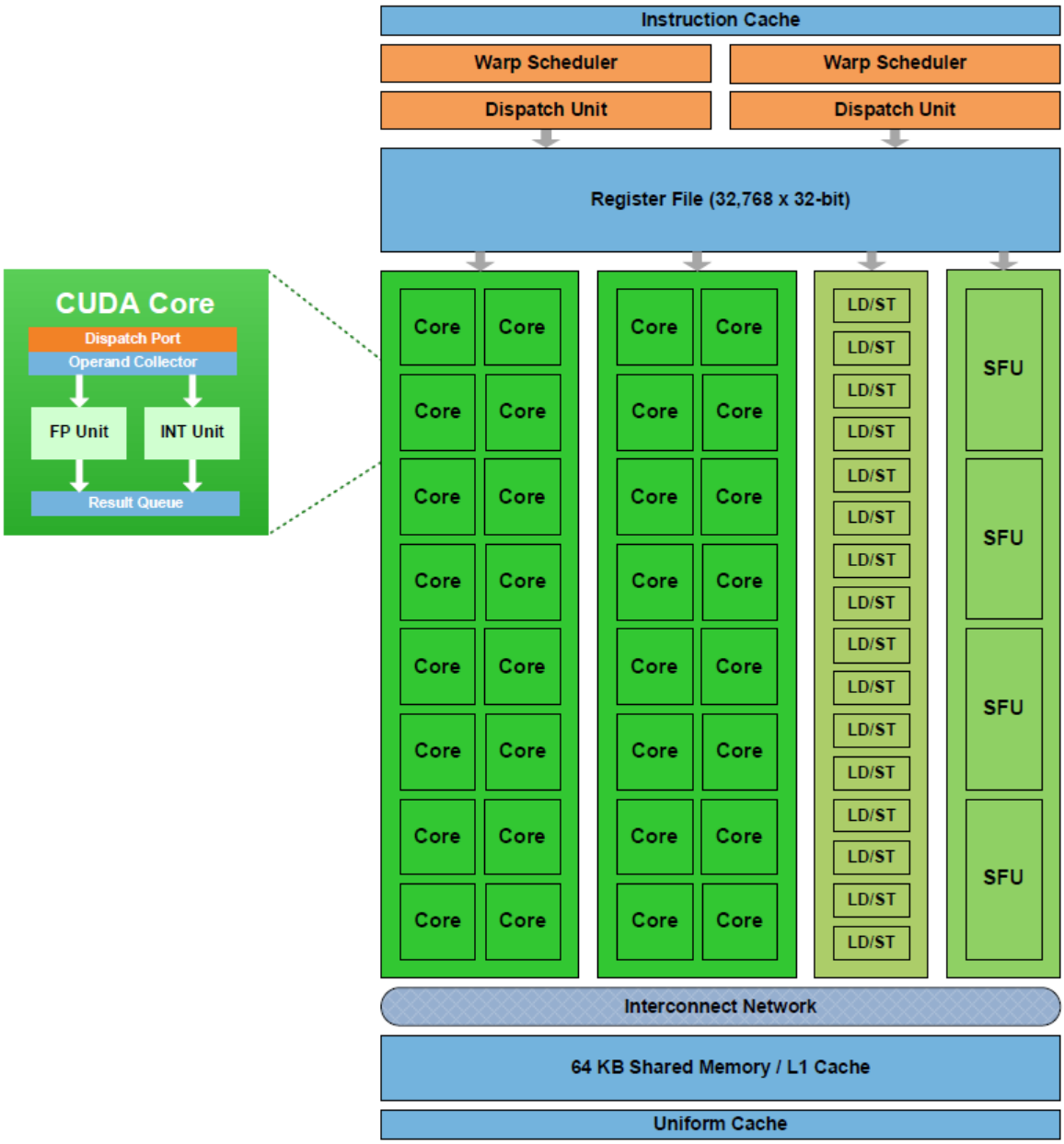
abstrakcijos lygmenyje įrenginys atrodo kaip skaičiavimo vienetų jūra (žalia dalis) tik su keletu pagalbinių elementų salelių (mėlyna). Ši iliustracija puikiai atskleidžia pagrindinį lusto projektuotojų tikslą – slankiojo kablelio operacijų skaičiaus per sekundę (FLOPS) maksimizavimas [Gla09].

Šis GPU modelis susideda iš 16 daugia-branduolinių procesorių, kurių kiekvienas turi po 32 branduolius, kas iš viso sudaro 512 CUDA šerdžių. Įrenginys turi šešias 64 bitų adresacijos atminties particijas, kas leidžia turėti iki 6 GB lokalios DRAM atminties.



3 Paveikslas. GPU lusto architektūra. Šaltinis: NVIDIA

Kiekvienas daugia-branduolinis CUDA procesorius, pavaizduotas *4 paveiksle*, turi po 32 branduolius, iš kurių kiekvienas gali atlikti slankiojo kablelio, sveikų skaičių bei logines operacijas. Prie to, kiekviename procesoriuje dar yra 16 įeities-išeities vienetų, skirtų atminties operacijoms atlikti, bei 64 Kb atminties sritis, kuri yra dalinama tarp vietinės bei spartinančiosios atminties. Fermi architektūra palaiko naująjį IEEE 754-2008 slankiojo kablelio operacijų standartą, kas leidžia pagerinti slankiojo kablelio operacijų skaičiavimų tikslumą palyginus su ankstesne GPU architektūra.



4 Paveikslas. Vieno CUDA daugia-branduolinio procesoriaus architektūra. Šaltinis: NVIDIA

2.5. CUDA naudojimas

CUDA programinė įranga veikia ant daugelio šiuolaikinių operacinių sistemų, tokių kaip Windows XP/Vista/7, Linux ar Mac OS bei palaiko 32 bei 64 bitų architektūrą. Oficialiai CUDA palaiko C, C++ bei Fortran programavimo kalbas, tačiau yra galimybės integruoti CUDA ir su kitoms programavimo kalbomis (Java, Python ir kitomis). Pagrindiniai įrankiai, kurių reikia norint pradėti naudoti CUDA skaičiavimus yra speciali vaizdo kortos tvarkyklės versija bei CUDA įrankių rinkinys (*angl. CUDA toolkit*). Visus šiuos įrankius, bei išsamią informaciją apie jų naudojimą galima rasti oficialioje interneto svetainėje [Nvid12].

Tipinis apibendrintas CUDA C++ programos veikimas, pavaizduotas *5 paveiksle*, yra toks:

- Pradiniai veiksmai kompiuteryje, tokie kaip pradinių reikšmių kintamiesiems suteikimas, parametrų, reikalingų skaičiavimams, priskyrimas;
- Atminties rezervavimas GPU bei kintamųjų nukopijavimas iš kompiuterio (host) į vaizdo plokštę (device);
- skaičiavimų ant GPU inicijavimas, kernelio paleidimas;
- rezultatų kopijavimas iš GPU į kompiuterį;
- tolimesni skaičiavimai naudojant gautus rezultatus iš GPU.

```
__global__ void add_matrix( float* a, float *b, float *c, int N ) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int index = i + j*N;
    if ( i < N && j < N )
      c[index] = a[index] + b[index];
}
int main() {
    //kintamųjų priskyrimas kompiuteryje, reikšmių jiems suteikimas.
    const int N = 1024, blocksize = 16;
    float *a = new float[N*N], *b = new float[N*N], *c = new float[N*N];
    for ( int i = 0; i < N*N; ++i ) {
      a[i] = 1.0f; b[i] = 3.5f;
    }
    float *ad, *bd, *cd;
    const int size = N*N*sizeof(float);
    //atminties alokavimas GPU
    cudaMalloc( (void**)&ad, size );
    cudaMalloc( (void**)&bd, size );
    cudaMalloc( (void**)&cd, size );
    //kintamųjų kopijavimas iš kompiuterio į GPU
    cudaMemcpy( ad, a, size, cudaMemcpyHostToDevice );
    cudaMemcpy( bd, b, size, cudaMemcpyHostToDevice );
    //
    dim3 dimBlock( blocksize, blocksize );
    dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
    //kernelio paleidimas, skaičiavimai ant GPU
    add_matrix<<<dimGrid, dimBlock>>>( ad, bd, cd, N );
    //rezutaltų kopijavimas atgal į kompiuterį bei atminties atlaisvinimas
GPU
    cudaMemcpy( c, cd, size, cudaMemcpyDeviceToHost );
    cudaFree( ad ); cudaFree( bd ); cudaFree( cd );
    //Vykdyti veiksmus su rezultatais, kurie yra c masyve
    ...
    return 1;
}
```

5 Paveikslas. Apibendrinta CUDA programa, skirta matricų sudėčiai.

3. Genetiniai algoritmai, jų taikymas finansiniuose uždaviniuose

3.1. Genetiniai algoritmai

Genetiniai algoritmai (GA) yra euristiniai paieškos algoritmai, kuriuose yra stengiamasi atkartoti natūralios evoliucijos procesus ir yra paremti biologijos žiniomis apie gyvybės evoliuciją (Darvino evoliucijos teoriją). GA yra platesnės Evoliucinių algoritmų (EA) klasės dalis, kurios problemų analizės bei sprendimų generavimo technika yra įkvėpta neutralių gamtoje egzistuojančių evoliucijos mechanizmų: paveldėjimo, mutacijos, natūraliosios atrankos bei rekombinacijos [Gol89]. Genetiniai algoritmai yra metodas analizuoti duomenis, jų dėka galima rasti apytikslį užduoties sprendimą, kuris yra randamas naudojant evoliucinį ciklą, veikiantį gamtoje. Genetiniai algoritmai tinka ne visur, tačiau jais galima rasti palyginti neblogus sprendinius užduočių, kurių tikslaus sprendimo algoritmai nežinomi [Wik12f].

Bendras užduoties sprendimo algoritmas, naudojant genetinius algoritmus, pavaizduotas *6 paveiksle* ir susideda iš šių etapų:

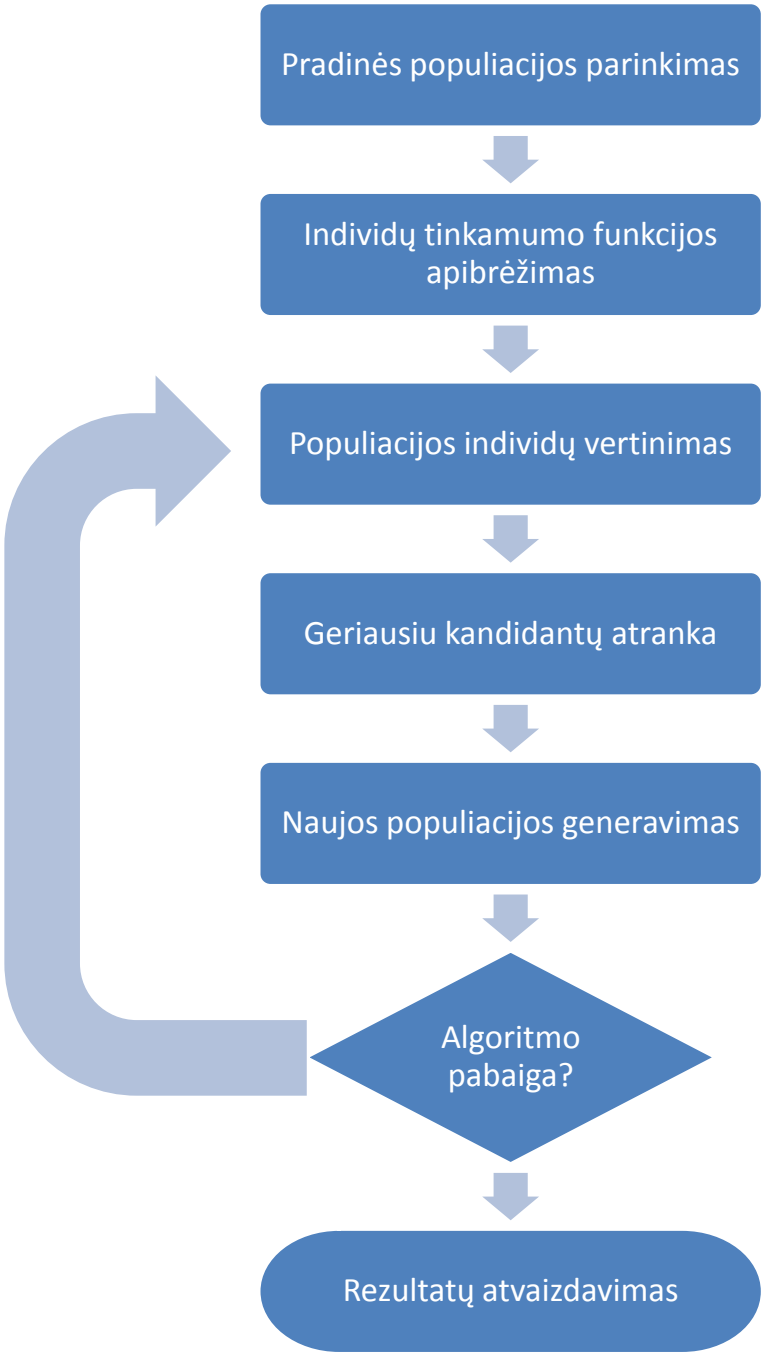
a) pradinės populiacijos parinkimas;
b) individų tinkamumo įvertinimo funkcijos apibrėžimas;
c) visų esamos populiacijos individų tinkamumo vertinimas;
d) geriausių kandidatų, iš kurių bus sudaryta naujoji populiacija, atrinkimas;
e) naujos populiacijos generavimas naudojantis atrinktaisiais kandidatais;
f) jeigu netenkinama algoritmo nutraukimo sąlyga ciklas kartojamas nuo (c) etapo.

Pradinės populiacijos parinkimas

Uždavinio sprendimo pradžioje yra generuojama pradinė uždavinio sprendinių populiacija. Populiacijos dydis yra parenkamas pagal užduoties sudėtingumą, dažniausiai tai būna nuo kelių šimtų iki kelių tūkstančių individų. Dažniausiai, bet nebūtinai visada, ši pradinė populiacija būna sugeneruojama atsitiktiniu būdu, apimant didelį galimų sprendinių paieškos diapazoną.

Individų tinkamumo įvertinimo funkcija, populiacijos individų tinkamumo vertinimas

Tinkamumo vertinimo (atrankos) funkcija (angl. – *fitness function*) yra esminė genetinio algoritmo dalis, pagal kurią yra vertinama kiekvieno atitinkamos populiacijos individo (uždavinio sprendimo) tinkamumas. Šio proceso tikslas - apibrėžti kaip įmanoma geresnę tinkamumo vertinimo funkciją, kurios pagalba būtų lengva atskirti tinkamiausius populiacijos atstovus, iš kurių būtų generuojama sekanti populiacijos karta. Finansiniuose uždaviniuose ši funkcija dažniausiai būtų atitinkamų parametrų vertinimas su istoriniais prekybos duomenimis ir geriausi individai būtų tie, kurie pasiektų didžiausią finansinę grąžą per atitinkamą laikotarpį.



6 Paveikslas. Genetinio algoritmo uždavinio sprendimo schema

Atranka

Atlikus visų populiacijos individų vertinimą yra atliekama atranka - iš populiacijos atrenkama dalis tinkamiausių sprendinių (sėkmingiausių „palikuonių“), iš kurių bus kuriama naujoji populiacija. Taip pat, prie šios geriausiųjų grupės yra parenkama ir dalis atsitiktinių populiacijos individų (nebūtinai geriausiųjų), siekiant išlaikyti populiacijos įvairovę. Iš šios atrinktosios populiacijos biologinių procesų būdų bus generuojama visiškai nauja populiacijos karta. Norint labai efektyviai pagerinti sprendimo paiešką yra taikomas metodas, ignoruojantis

įprastą evoliucijos eigą, kai konstruojant naują populiaciją, leidžiama keliems patiems sėkmingiausiems organizmams (sprendiniams) pereiti į naują populiaciją visai nepakitus. Ši strategija vadinama elito atranka (angl. - *elitist selection*).

Naujos populiacijos generavimas

Antrosios ir kitų populiacijų kūrimą sudaro genetinių mechanizmų: rekombinacijos ir mutacijos – pritaikymas. Rekombinaciją ir mutacija yra biologinių mechanizmų analogai.

Kiekvieno naujo sprendinio sukūrimui iš atrinktosios populiacijos imama pora sprendinių „tėvų“, kurie sukryžminami ir mutacijos ar rekombinacijos būdu gaunamas sprendinys - „vaikas“. Toliau imama kita „tėvų“ pora ir vėl sukuriamas „vaikas“ ir taip tęsiama kol pasiekiamas tam tikras naujos kartos individų skaičius, sukuriama pilna nauja sprendinių populiacija [Wik12g]:

Mutacija - yra būdas genetiniuose algoritmuose išlaikyti genetinę įvairovę. Mutacija naudojama GA yra biologinės mutacijos analogas, kadangi jos mechanizmas buvo kuriamas pagal biologinę mutaciją. Mutacijos padeda išvengti populiacijos chromosomų supanašėjimo (lokalaus konvergavimo), o tai vestų į evoliucijos sulėtėjimą ar net visišką stagnaciją. Tai taip pat paaiškina, kodėl GA sistemos vengia naudoti tokį atrinkimo būdą, kai atrenkami vien tik geriausi.

Rekombinacija - yra būdas perteikti vienos chromosomos ar kelių chromosomų sandarą į kitą kartą. Ji yra biologinės rekombinacijos analogas, kadangi būtent biologiniu procesu ir paremta. Yra nemažai rekombinacijos būdų, kurie skirtingai perteikia biologinės rekombinacijos esmę:

- Su vienu rekombinacijos tašku. Pasirenkamas rekombinacijos taškas tėvų duomenyse. Vėliau visi „tėvų“ duomenys, buvę už taško, apkeičiami vietomis. Rezultatas – „vaikai“, kurie turi dalį vieno ir kito tėvo (genetinių) duomenų.
- Su dviem rekombinacijos taškais. Pasirenkami du rekombinacijos taškai tėvų duomenyse. Vėliau visi „tėvų“ duomenys, esantys tarp šių taškų sukeičiami vietomis. Rezultatas – „vaikai“, kurie turi dalį vieno ir kito tėvo duomenų.
- „Pjaustymas“ – rekombinavimo technika, kai keičiamas „vaikų“ duomenų ilgis. Skirtumas nuo ankščiau minėtų technikų yra tas, kad čia tėvų duomenyse pasirenkami taškai skirtingose vietose.
- Abiejų technikų atvejais „tėvų“ duomenų pagrindu sukuriami du nauji „vaikai“. Pirmuoju atveju „tėvų“ duomenys sulyginami tarpusavyje ir sukeičiami vietomis su 50 proc. tikimybe. Pusiau vienodos rekombinacijos atveju sukeičiama vietomis pusė nesutampančių tėvų duomenų.

Galiausiai po šių procesų gaunama naujoji karta, kurią sudaro individai, kurių chromosomos (sandara) yra visiškai skirtingos nei pradinės populiacijos. Bendras populiacijos tinkamumo vidurkis taip pat pakyla, kadangi išlieka pagrinde tik geriausiųjų individų „genus“ turintys palikuonys (sprendiniai).

Algoritmo pabaiga

Bendras sprendinių evoliucijos procesas yra ciklišKai kartojamas kol pasiekiama nutraukimo sąlyga. Dažniausios nutraukimo sąlygos yra šios:

- Sprendinys tenkina minimalų kriterijų;
- Pasiektas nustatytas generacijų skaičius;
- Tam skirtas biudžetas (laikas/pinigai) - išnaudotas;
- Pasiekta stabili būsena, kai nėra gaunama geresnių sprendinių;
- Rankinis sustabdymas, atliekamas rezultatų patikrinimas;
- Aukščiau minėtų priežasčių kombinacijos.

Verta atkreipti dėmesį

Naudojant genetinius algoritmus vertėtų atkreipti dėmesį į keletą dažniausiai pastebimų galimų problemų, bei stengtis jas išspręsti:

Lokalus konvergavimas - GA gali turėti tendenciją konverguoti link lokalaus (riboto) sprendimo, vietoje globalaus tinkamiausio sprendimo. Problemą sumažinti ar net visai išspręsti gali metodai naudojami išlaikyti kuo įvairiapusiškesnę sprendinių populiaciją.

Ankstyvas konvergavimas - Sunkumų iškyla dirbant su dinaminiais duomenų rinkiniais, kai genomai pradeda anksti konverguoti, tokiu būdu nelieka reikalingų duomenų, iš jų sekančių sprendinių kūrimui [AK99].

3.2. GA taikymas finansiniuose uždaviniuose

Finansiniai uždaviniai, kurių dauguma yra sudėtingi bei neturintys vieno tikslaus sprendimo, puikiai tinka genetiniams algoritmams ir yra vis plačiau naudojami finansinėse institucijose bei pavienių investuotojų. Kadangi norint parinkti teisingiausias automatines prekybos taisykles reikia atlikti paiešką tarp labai didelės aibės galimų sprendimų, genetiniai algoritmai suteikia pranašumą prieš kitas euristines paieškos metodologijas.

3.3. GALib biblioteka

Kadangi šio darbo pagrindinis tikslas yra ne genetinių algoritmų išsami analizė, o jie yra tik papildomas būdas rezultatams pasiekti darbe, nusprendžiau pasinaudoti esama GALib genetinių algoritmų programavimo biblioteka C/C++ programavimo kalbai, kurios pagalba atliksiu visus su genetika susijusius veiksmus, tokius kaip populiacijos generavimą, kodavimą,

naujų kartų kūrimą. GALib biblioteka, parašyta 1996 metais, Matthew Wall Masačiusetso Technologijos institute (MIT). Jos naudojimas nepelno tikslais yra atviras, bei nemokamas. Platesnė informaciją apie šią biblioteką bei jos naudojimą galite rasti jos dokumentacijoje [Wall96].

Norint teisingai panaudoti lygiagrečiuosius skaičiavimus, darbo eigoje teks perrašyti keletą bibliotekos funkcijų, skirtų atskirų individų tinkamumo vertinimui.

ANALITINĖ DALIS

Šioje darbo dalyje yra aprašomi realizuoti Magistro baigiamojo darbo principiniai sprendimai, atlikta probleminės sferos analizė bei gauti rezultatai. Įgyvendinant Magistro darbo iškeltus uždavinius buvo atlikti darbai, kuriuos išskyriau į du skyrius, pateikiamus šioje darbo dalyje.

Pirmajame (4) skyriuje aprašomi automatinės prekybos sistemos modeliavimo darbai, analizei bei skaičiavimams naudojami techniniai parametrai, prekybos duomenys. Pateikiami tyrimo atrastų geriausių prekybos strategijų pelningumas mokymosi bei testiniams duomenims.

Antrajame (5) skyriuje pateikiami skaičiavimų spartos naudojant CPU bei GPU palyginamoji analizė, pateikiant atrastus programos bei duomenų optimizavimo sprendimus bei jų įtaką bendrai skaičiavimų spartai.

4. Automatinė prekybos sistema

Magistro darbo uždaviniams įgyvendinti buvo reikalinga realius prekybos sandorius imituojanti sistema, kurios pagalba būtų galima įvertinti kokius rezultatus (pelną/nuostolį) gautume prekybai naudodami konkrečius parametrų rinkinius. Šiam tikslui naudojant C++ programavimo kalbą buvo parašytas kodas, kuris buvo naudojamas skaičiavimams tiek naudojant pagrindinį kompiuterio procesorių (CPU), tiek grafinės vaizdo kortos procesorių (GPU).

4.1. Automatinės prekybos sistemos modelis

Automatinei prekybai rinkoje realizuoti naudojamas techninės analizės indikatoriais paremta sistema, kurios pagalba yra automatiškai priimami sprendimai pirkti ar parduoti atitinkamą finansinį vienetą. Abstraktus automatinės prekybos sistemos pagrindinės funkcijos veikimo algoritmas pavaizduotas *7 paveiksle*. Pagrindiniai sistemos veikimo žingsniai yra šie:

- Parametrų rinkinio, naudojamo prekybos sprendimų priėmime, sudarymas, perdavimas pagrindinei algoritmo funkcijai;
- Pradinių indikatorių reikšmių skaičiavimas, reikalingų kintamųjų priskyrimas;
- Pažingsniui einama per visą tiriamų duomenų aibę, kiekvienam laiko elementui skaičiuojant prekybos sistemos parametrų reikšmes bei tikrinant ar nėra tenkinamos pirkimo / pardavimo sąlygos. Jei atitinkamos sąlygos yra tenkinamos, inicijuojamas pirkimo / pardavimo sandoris, atnaujinami prekybos sistemos rezultatai. Kiekvienam prekybos sandoriui yra taikomas nustatytas komisinis mokestis;
- Grąžinami prekybos sandorių rezultatai.

```
tr_results simulate_trading (/*... parametrai prekybai ...*/) {
    /*
    Pradiniai indikatorių įvertinimai, kintamųjų priskyrimai
    EMA, MACD, RSI ir kiti naudojami
    */
    in_trade = 0;
    tr_results rezultatai;
    trade_eval = 0;
    ...
    for (i = start_at; i < historical_data; i++) {
      //keliaujame per visą testinių duomenų istoriją
      //įvertindami sandorių galymybes
      if (in_trade) {
        if (exit_criteria_met) {
          close_trade();
        }
      }
      /*
      Įvertinti visų indikatorių generuojamų pirkimo / pardavimo
      signalų visumą.
      */
      trade_eval = p1_signal + p2_signal + ...;
      if (trade_eval < sell_margin) {
        open_short();
      }
      else if (trade_eval > buy_margin) {
        open_long();
      }
      else {
        //no action
      }
    }
    if (in_trade) {
      close_trade();
    }

    //grąžiname prekybos rezultatų finansinę išraišką
    return rezultatai;
}
```

7 Paveikslas. Automatinės prekybos sistemos pagrindinės funkcijos pseudo-kodas

Sėkmingai įvykdžius prekybos simuliaciją yra gaunami du rezultatų rinkiniai:

1. Bendri prekybos simuliacijos rezultatai (C duomenų struktūra), kurie susideda iš šių dedamųjų:
    a. Pinigų suma turėta prieš pradedant prekybą (*funds_start*);
    b. Pinigų suma po prekybos simuliacijos *(funds_end*);
    c. Maksimali pinigų suma, kurią turėjome prekybos simuliacijos metu (f*unds_max*);
    d. Sandorių, įvykdytų prekybos metu skaičius (*trades*). Skaičiuojami tik įėjimo į poziciją sandoriai, t.y. ,pavyzdžiui, pradinis pirkimo bei vėlesnis tos pozicijos pardavimo sandoriai yra traktuojami kaip vienas sandoris;

e. Kiek iš sandorių buvo sėkmingi (*successfull_trades*). Sėkmingu sandoriu yra laikomas toks, po kurio pilno įvykdymo turima grynųjų pinigų suma tampa didesnė negu buvo prieš sandorį.

2. Sandorių sąrašas (masyvas), kuriame yra detali informacija apie visus prekybos simuliacijos metu įvykdytus sandorius. Ši informacija susideda iš duomenų apie tai, kada tiksliai įvykdytas kiekvienas sandoris, sandorio įvykdymo metu galiojusį valiutos kursą, sandorio kiekį, kokia tai buvo operacija (*long/short*) bei kokia pinigų suma lieka atlikus sandorį.

Naudojantis šiais duomenimis yra galima gana nesunkiai įvertinti, kuris parametrų rinkinys yra geras, kuris prastesnis ar kuris visai niekam tikęs. Šie duomenys taip pat bus naudojami vertinant atskirų populiacijos individų tinkamumą.

4.1.1. Naudojami indikatoriai

Atlikęs keletą pradinių testų, šioje prekybos modeliavimo sistemoje nusprendžiau naudoti techninės analizės indikatorius, pateiktus 5 Lentelėje. Būtent šiuos indikatorius iš apžvelgtų šio darbo literatūros apžvalgos dalyje pasirinkau dėl to, jog išankstiniuose testuose jie pateikdavo geriausius prognozių rezultatus, lyginant su tuo, kiek skaičiavimo laiko (*angl. computational cost*) jie užtrunka.

5 Lentelė. Techninės analizės indikatoriai naudojami prekybos modeliavime

Indikatorius	**Aprašymas**	**Parametrai indikatoriui skaičiuoti**
MACD	Slankiųjų vidurkių kirtimosi indikatorius – vaizduojamas kaip dviejų eksponentinių slankiųjų vidurkių skirtumas (MA1 - MA2). Pirkimo / pardavimo signalas generuojamas kai MACD linija kertasi su MACD signaline linija, kuri yra apskaičiuojama kaip MACD reikšmių slankusis vidurkis.	EMA1 EMA2 MACD
RSI	Lyginamosios jėgos indeksas - tai osciliatorius, sekantis kainą ir svyruojantis diapazone nuo 0 iki 100. Skaičiuojamas naudojantis parametrais nurodytu paskutinių laikotarpių kainas.	RSI RSI_bound
RSI nuokrypis	Lyginamosios jėgos indekso nuokrypis skaičiuojamas pagal tai, kaip keitėsi RSI indekso reikšmė tarp dviejų paskutinių iteracijų, tai yra lyginame naujausią RSI indeksą su buvusiu prieš vieną skaičiavimų laikotarpį.	RSI RSI_bound

4.1.2. Naudojami parametrai

Vykdant prekybos modeliavimą, buvo generuojami unikalūs parametrų rinkiniai, kurie buvo vertinami su istoriniais prekybos duomenimis, siekiant išrinkti geriausius, kurie vėliau bus naudojami vertinant ateities prekybos duomenis bei stebint, kiek teisingai jie veiktų su naujais duomenimis. Be parametrų, skirtų grynai indikatorių skaičiavimams (pateikti ankstesniame poskyryje) kiekvienas parametrų rinkinys turėjo papildomus parametrus, kurie turėjo padėti priimti tikslesnius prekybos sprendimus. Pilnas kiekvienas generuotas parametrų rinkinys aprašytas 6 Lentelėje.

6 Lentelė. Vieno parametrų rinkinio struktūra

Nr.	Kintamasis	Aprašymas / naudojimas	Galimos reikšmės
1	MA1	Pirmojo slankiojo vidurkio, naudojamo MACD indikatoriuje, bazė.	[1;1000]
2	MA2	Kiek antrojo (didesniojo) slankiojo vidurkio bazė didesnė už pirmojo.	[1;1000]
3	MACD	Iš kelių MACD indikatoriaus reikšmių skaičiuojama MACD signalinė linija.	[1;25]
4	use_SL_TS	Ar skaičiuojant naudojamos siekiamo pelno / nuostolių apsaugos (SL/TP) funkcijos. Ar naudojama slenkančioji nuostolių apsauga (trailing stop). Galimos reikšmės: 0 – nenaudojama SL/TP; 1 – naudojami įprastiniai SL/TP; 2 – naudojamas slenkančioji nuostolių apsauga, nenaudojamas TP.	{0,1,2}
5	SL	Kokio dydžio SL, jei naudojamas.	[1;100]
6	TP	Kokio dydžio TP, jei naudojamas.	[1;100]
7	nRSI	Iš kelių paskutinių laikotarpių kainų skaičiuojamas RSI indeksas.	[1;100]
8	RSI_bound	RSI indekso riba, generuojanti pirkimo/pardavimo signalus.	[11;40]
9	MACD_diff	Minimalus skirtumas tarp MACD ir MACD signalinės linijų, reikalingas jog pirkimo / pardavimo signalas būtų sugeneruotas.	[0;1000]
10	action	Riba, kurią pasiekus priimamas bendras pirkimo / pardavimo sprendimas.	[0;300]
11	wMACD	MACD indikatoriaus svoris bendrame sprendimo priėmime.	[0;100]
12	wRSI1	RSI indikatoriaus svoris bendrame sprendimo priėmime.	[0;100]
13	wRSI2	RSI nuokrypio indikatoriaus svoris bendrame sprendimo priėmime.	[0;100]

Kaip matote, kiekvienas parametrų rinkinys susideda iš 13 skirtingų komponentų, kurių kiekvienas gali įgauti atitinkamą reikšmę iš jiems priskirto intervalo. Intervalai parinkti remiantis literatūros apžvalgoje peržiūrėta informacija bei pradiniais skaičiavimų testais. Galimų parametrų rinkinių yra labai daug ($6{,}75 * 10^{26}$ jei tiksliai), ir suskaičiuoti visus galimus variantus užtruktų be galo daug laiko. Siekiant paspartinti tinkamų parametrų paieškas darbe yra naudojami genetiniai algoritmai, kurių panaudojimas plačiau aprašytas 2 šio darbo skyriuje.

Kitas sprendimas, kuris buvo panaudotas siekiant sumažinti galimų parametrų rinkinių skaičių, yra parametrų intervalų skaidymas į diskrečias reikšmes (t.y. didinant žingsnius, kuriais gali kisti parametrų reikšmės). Tokiu būdu, pavyzdžiui, MA1 parametras vietoj galimų turėti 1000 skirtingų reikšmių, įvertinus kitimo žingsnį (5), gali turėti reikšmes [5,10,15,...,995,1000], kas sumažina visą galimų parametrų rinkinių aibę 5 kartus. Tą pačią metodiką pritaikius kai kuriems kitiems parametrams, tokiems kaip MA2, MACD_diff, action, wMACD, wRSI1 ar wRSI2 bendra parametrų rinkinių skaičių pavyktų sumažinti gana ženkliai nedarant labai didelės įtakos skaičiavimų kokybei.

4.1.3. Prekybos duomenys

Siekdamas tinkamai įvertinti skaičiavimų spartą modeliuojant labai aukšto dažnio prekybos sistemą šiame darbe pasirinkau naudoti Forex rinkos *tikinius* duomenis. Tikiniai duomenys yra patys dažniausi, atspindintys kiekvieną biržos sandorį, kainos pasikeitimą. Nors naudojantis kai kurių populiariausių bendrovių akcijų prekybos duomenimis taip pat galima būtų gauti tokius pat ar net didesnius sandorių skaičius prekybos metu kaip ir Forex rinkoje, kadangi Forex rinkoje prekyba vyksta darbo dienomis 24 val. per parą, kai tuo tarpu vertybinių popierių biržos dirba 6-7 val. per parą.

Duomenys testavimui buvo paimti iš viešai prieinamo Šveicarijos Forex prekybos sistemos Dukascopy šaltinio [Duka12]. Prekybos duomenų failai buvo naudojami csv (*angl. comma separated values*) formate. Tipinio duomenų failo struktūra pavaizduota 8 paveiksle. Faile matomas tikslus (milisekundžių tikslumu) sandorio laikas, tuo metu galiojanti kaina bei įvykdyta apyvarta. Šie trys stulpeliai iš šio duomenų failo (Time, Open, Volume) ir yra naudojami prekybos modeliavime.

Konkrečiai šiam darbui buvo atliekami tyrimai su trimis skirtingomis valiutų poromis: EUR/USD, GBP/USD ir USD/CHF. Parametrų atrinkimui (mokymuisi) buvo naudojami 4 savaičių bei 8 savaičių rinkos duomenys, rezultatų vertinimui atitinkamai 1, 2 savaičių einančių po mokymosi laikotarpių duomenys. Išsamiau apie atliktus prekybos modeliavimo testus bei jų rezultatus rasite šio darbo 4.3 poskyryje.

```
1  Time,Open,High,Low,Close,Volume
2  12.03.2012 10:34:27.150,1.31182,1.31182,1.31182,1.31182,1.50
3  12.03.2012 10:34:27.694,1.31180,1.31180,1.31180,1.31180,3.82
4  12.03.2012 10:34:28.220,1.31180,1.31180,1.31180,1.31180,1.50
5  12.03.2012 10:34:28.578,1.31178,1.31178,1.31178,1.31178,2.25
6  12.03.2012 10:34:28.936,1.31175,1.31175,1.31175,1.31175,1.94
7  12.03.2012 10:34:29.483,1.31174,1.31174,1.31174,1.31174,1.57
8  12.03.2012 10:34:30.061,1.31171,1.31171,1.31171,1.31171,2.63
9  12.03.2012 10:34:30.356,1.31165,1.31165,1.31165,1.31165,1.57
10 12.03.2012 10:34:30.574,1.31164,1.31164,1.31164,1.31164,1.76
11 12.03.2012 10:34:30.928,1.31164,1.31164,1.31164,1.31164,1.01
12 12.03.2012 10:34:31.328,1.31163,1.31163,1.31163,1.31163,3.26
13 12.03.2012 10:34:31.468,1.31161,1.31161,1.31161,1.31161,6.07
14 12.03.2012 10:34:31.890,1.31161,1.31161,1.31161,1.31161,3.07
15 12.03.2012 10:34:31.966,1.31161,1.31161,1.31161,1.31161,4.57
16 12.03.2012 10:34:33.179,1.31164,1.31164,1.31164,1.31164,1.57
17 12.03.2012 10:34:38.744,1.31164,1.31164,1.31164,1.31164,1.57
18 12.03.2012 10:34:39.278,1.31166,1.31166,1.31166,1.31166,1.50
```

8 Paveikslas. Tipinio biržos duomenų failo struktūra

4.2. Genetiniai algoritmai

Kadangi galimų parametrų rinkinių yra labai daug (10^{26} eilės skaičius), siekdamas ženkliai paspartinti tinkamiausių parametrų rinkinių paiešką darbe naudoju genetinius algoritmus. Genetinių algoritmų skaičiavimų įgyvendinimui darbe buvo pasitelkta *GALib* genetinio programavimo biblioteka, kurios pagalba buvo atliekamos visos su genetika susijusios operacijos: pradinės bei tolimesnių populiacijų generavimas pasitelkiant natūraliosios atrankos bei kitus biologinius procesus.

Kadangi šio darbo pagrindinis tikslas nebuvo išsamus genetinių algoritmų taikymo tokiems uždaviniams spręsti tyrimas, apsiribojau tik genetinio algoritmo individų vertinimo funkcijos pritaikymu lygiagretiesiems skaičiavimams. Likę parametrai, išskyrus populiacijos kartų kiekį bei populiacijos dydį, buvo naudojami standartiniai ir daug su jais neeksperimentuota. Skaičiavimams pasirinkau Kenneth De Jong aprašytą persidengiančios populiacijos tipą [Jong75], t.y. dalis geriausiųjų senosios populiacijos atstovu išlieka ir naujoje populiacijoje.

Norėdamas, jog visi populiacijos individai būtų vertinami naudojant lygiagrečiuosius algoritmus, pagrindinę vertinimo funkciją (*GAPopulation::DefaultEvaluator(GAPopulation & p)*) pakoregavau taip, jog pirmiausia visų individų genai būtų transformuojami į bendrą parametrų masyvą (*ParamArray*), kuris yra perduodamas į vaizdo plokštės atmintį bei atliekami skaičiavimai su GPU procesoriais. Vėliau, grįžus rezultatams iš GPU, yra vykdomas rezultato priskyrimas kiekvienam populiacijos nariui atskirai iš rezultatų masyvo. Šią C++ kodo ištrauką rasite 9 paveiksle.

```
void GAPopulation::DefaultEvaluator(GAPopulation & p) {
  ParamArray = (TradeParams*) malloc(p.size() * sizeof(TradeParams));
  ResultArray = (tr_results*) malloc(p.size() * sizeof(tr_results));
  for(int i=0; i<p.size(); i++) {
 ParamArray[i] = genomeToParams(p.individual(i),i);
  }

  //CUDA lygiagrečiųjų skaičiavimų dalis
  cudaMalloc( (void **) &ParamArray_GPU, sizeof(TradeParams)*p.size());
  cudaMalloc( (void **) &ResultArray_GPU, sizeof(tr_results)*p.size());

  cudaMemcpy (ParamArray_GPU, ParamArray, sizeof(TradeParams)*p.size(),
cudaMemcpyHostToDevice);
  kernel <<<dimBlock,dimGrid>>>
(TickArray_GPU,ParamArray_GPU,ResultArray_GPU,tick_elem);
  cudaMemcpy(ResultArray, ResultArray_GPU, sizeof(tr_results)*p.size(),
cudaMemcpyDeviceToHost);

  cudaFree(ParamArray_GPU);
  cudaFree(ResultArray_GPU);

  //lygiagrečiųjų skaičiavimu pabaiga, rezultatai apdorojami bei priskiriami
  individams

  float score;
  for(int i=0; i<p.size(); i++) {
 score = ResultArray[i].funds_end - 80000;
 if ( (ResultArray[i].trades < 5) && score > 0 )  {
    score -= 20000 / (ResultArray[i].trades+1);
 }
 if ( score < 0 ) score = 0;

 p.individual(ResultArray[i].params_used.nGenome).evaluate(score);
  }

 free (ParamArray);
 free (ResultArray);
}
```

9 Paveikslas. Genetinio algoritmo vertinimo funkcija pritaikyta lygiagretiesiems skaičiavimams

4.3. Prekybos modeliavimo tyrimas

Prekybos modeliavimo tyrimas buvo atliekamas su trimis skirtingomis valiutų poromis: EUR/USD, GBP/USD ir USD/CHF. Kiekvienai iš šių valiutų porų buvo parinkti du 4 savaičių trukmės prekybos duomenų rinkiniai, skirti geriausių parametrų nustatymui, bei trys 1 savaitės trukmės duomenų rinkiniai, skirti patikrinti gautiems sprendiniams. Bendri skaičiavimų su mokymosi duomenimis rezultatai pateikti 7 lentelėje. Lentelėje pateikiami rezultatai po 20, 40 bei 60 evoliucijos ciklų.

Skaičiavimai buvo atliekami naudojant šiuos pagrindinius genetinio algoritmo nustatymus:

- Individų skaičius populiacijoje: **6144**
- Kartų skaičius populiacijoje iki algoritmo pabaigos: **60**
- Individų skaičius, pereinantis iš senosios populiacijos į naująją: **500**

- Kiekvieno individo mutacijos tikimybė pereinant į naują populiaciją: **5%**
- Tikimybė, kad naujos kartos individai bus kuriami rekombinacijos būdu: **80%**

Visi skaičiavimai buvo atliekami kiekvienam parametrų rinkiniui suteikiant 100000 pradinį biudžetą bei modeliuojant prekybą. Vėliau, individams buvo suteikiamas atitinkamas rezultatas, įvertinant, kokia buvo galutinė finansinė grąža, kiek buvo įvykdytą prekybos sandorių. Individams, kurie per visą mokymosi laikotarpį nesugeneruodavo nei vieno, ar sugeneruodavo tik iki penkių sandorių, buvo priskiriamas žemas įvertinimas, kadangi tokie individai nėra tinkami prekybai modeliuoti. Mūsų tikslas buvo surasti tokius individus, kurie inicijuotų nemažai prekybos sandorių, bei kurių sandorių rezultatas būtų teigiamas bei kuo didesnis.

7 lentelė. Bendri skaičiavimų rezultatai su mokymosi duomenimis

nGen	**Duomenų laikotarpis**	**Geriausias individas populiacijoje** (parametrų rinkinys, struktūra aprašyta 2 lentelėje)	**Geriausio individo įvertinimas**
EUR/USD			
20	2012.03.04 – 2012.03.31	445 830 4 0 35 12 66 16 23 10 10 5 6	23786.3
40		595 860 17 0 20 25 100 17 29 2 10 1 2	25080
60		505 450 20 1 54 98 94 15 35 0 10 4 5	25750.8
20	2012.03.11 – 2012.04.07	45 305 18 0 30 61 77 11 655 3 6 5 2	23476.6
40		15 70 21 2 85 98 100 24 177 11 7 6 8	24073.4
60		15 70 21 2 76 51 29 24 91 13 5 9 6	24485.1
GBP/USD			
20	2012.03.04 – 2012.03.31	225 745 13 0 80 46 82 11 153 6 8 8 4	24770.2
40		45 940 10 0 24 28 87 12 562 0 1 3 1	25341.1
60		45 985 10 0 34 28 87 12 562 0 1 6 5	26382.4
20	2012.03.11 – 2012.04.07	5 510 20 0 85 56 58 32 882 14 9 8 4	24462.1
40		5 510 20 0 85 56 58 32 882 14 9 8 4	24462.1
60		5 510 20 0 85 93 58 32 882 14 9 8 4	24462.1
USD/CHF			
20	2012.03.04 – 2012.03.31	35 955 16 0 78 71 83 27 813 6 7 4 9	24655
40		65 55 19 1 82 67 37 21 197 3 5 2 10	25133.6
60		65 55 19 1 88 55 38 21 197 4 5 2 7	25368.6
20	2012.03.11 – 2012.04.07	150 185 18 1 57 96 96 14 104 8 10 2 2	24987.3
40		880 485 8 1 59 57 50 40 2 5 3 4 2	25536.9
60		880 485 8 1 59 57 50 40 2 5 3 4 2	25536.9

Kaip matome iš 7 lentelės, beveik visi skaičiavimai atrado individus, kurių galutinis įvertinimas būtų bent 24000 ar daugiau. Toks įvertinimas reiškia, jog per tiriamą laikotarpį (4 savaites) individas sugebėjo prekyboje pasiekti 4% ar didesnį pelningumą nuo pradinės investicijų sumos. Šie 5% per 4 savaites atitinka daugiau nei 50% metinį investicinį pelningumą, ką galima vertinti kaip tikrai gerą rezultatą.

Iš 3 lentelės taip pat galime pastebėti, jog kažkuriuose skaičiavimuose po 20 ar 40 kartos nebebūdavo surandami nauji geresni parametrų rinkiniai. Iš to galima spėti, jog genetinis algoritmas konvergavo į lokalų galimų sprendinių maksimumą bei geresnių sprendimų rasti nebepavyko. Ši problema galėtų būti vieno iš tolimesnių tyrimų dalis.

Geriausių individų vertinimas su testiniais duomenimis

Šioje dalyje vertinsime geriausius individus, atrastus mokymosi aibėje su testiniais duomenimis. Bus naudojami vienos savaitės imties duomenys, einantys po mokymosi duomenų.

8 Lentelė. Parametrų rinkinių validavimas su vėlesniais nei mokymosi duomenimis

Duomenų laikotarpis	**Parametrų rinkinys**	**Gautas rezultatas**	**Rezultatas %**	**Sand. skaičius / sėkmingų**
EUR/USD				
2012.04.01 – 2012.04.07	505 450 20 1 54 98 94 15 35 0 10 4 5 15 70 21 2 76 51 29 24 91 13 5 9 6	100974.30 -	+0,974 -	9 / 7 -
2012.04.08 – 2012.04.13	505 450 20 1 54 98 94 15 35 0 10 4 5 15 70 21 2 76 51 29 24 91 13 5 9 6	100152.00 99965.03	+0,152 -0,035	10 / 6 3 / 1
2012.04.04 – 2012.04.21	505 450 20 1 54 98 94 15 35 0 10 4 5 15 70 21 2 76 51 29 24 91 13 5 9 6	98435.30 99184.04	-1,365 -0,816	7 / 1 4 / 1
GBP/USD				
2012.04.01 – 2012.04.07	45 985 10 0 34 28 87 12 562 0 1 6 5 5 510 20 0 85 56 58 32 882 14 9 8 4	98680.68 -	-1,32 -	13 / 5 -
2012.04.08 – 2012.04.13	45 985 10 0 34 28 87 12 562 0 1 6 5 5 510 20 0 85 56 58 32 882 14 9 8 4	99495.91 100476.48	-0,405 +0,476	6 / 3 1 / 1
2012.04.04 – 2012.04.21	45 985 10 0 34 28 87 12 562 0 1 6 5 5 510 20 0 85 56 58 32 882 14 9 8 4	100431.93 99152.08	+0,431 -0,848	8 / 5 2 / 1
USD/CHF				
2012.04.01 – 2012.04.07	65 55 19 1 88 55 38 21 197 4 5 2 7 880 485 8 1 59 57 50 40 2 5 3 4 2	98291.65 -	-1,709 -	6 / 1 -
2012.04.08 – 2012.04.13	65 55 19 1 88 55 38 21 197 4 5 2 7 880 485 8 1 59 57 50 40 2 5 3 4 2	99545.81 101172.06	-0,455 +1,172	3 / 1 3 / 2
2012.04.04 – 2012.04.21	65 55 19 1 88 55 38 21 197 4 5 2 7 880 485 8 1 59 57 50 40 2 5 3 4 2	100584.73 99351.63	+0,584 -0,0649	1 / 1 3 / 0

Naudojami trijų savaičių duomenys:

1. iš karto po pirmojo 4 savaičių mokymosi laikotarpio;
2. iš karto po antrojo 4 savaičių mokymosi laikotarpio, praėjus savaitei nuo pirmojo mokymosi laikotarpio pabaigos;
3. praėjus savaitei nuo antrojo mokymosi laikotarpio pabaigos.

Kaip matome iš skaičiavimų rezultatų 8 Lentelėje, ne visi parametrų rinkiniai, kurie generavo tikslius prekybos sandorius mokymosi duomenų aibėje, taip pat sėkmingai veikia ir vėlesniuose duomenyse. Galima įžvelgti šiokią tokią tendenciją, kad kuo mažesnis laiko tarpas yra tarp mokymosi ir validavimo duomenų, tuo geresni rezultatai gaunami validacijoje. Iš to galima teigti, jog dažnas naujų parametrų ieškojimas, naudojantis pačiais naujausiais prekybos duomenimis yra labai svarbus, norint pasiekti gerus rezultatus aukšto dažnio prekybos sistemoje. Dėl to yra labai svarbus ir pats skaičiavimų laikas, kadangi jeigu skaičiavimai užtruks labai ilgai, rezultatai, kuriuos galiausiai gausime, gali būti jau nebetinkami naudojimui, kadangi bus pasenę. Skaičiavimų pagreitinimas, panaudojus GPU lygiagrečiuosius skaičiavimus, kurį man pavyko pasiekti šio darbo metu yra apžvelgtas kitame šio darbo skyriuje.

5. Skaičiavimų spartos analizė, GPU ir CPU palyginimas

Šioje darbo dalyje apžvelgiama skaičiavimų sparta lyginant GPU bei CPU sprendimus. Išnagrinėjus literatūros šaltinius bei atlikus praktinius testus buvo atrinktos šios sritys, kurios galėtų turėti didesnę ar mažesnę įtaką bendrai skaičiavimų spartai:

- Programos kodo optimizavimas lygiagretiesiems CUDA skaičiavimams;
- Naudojamų kintamųjų tipų įtaka;
- Prekybos duomenų struktūros optimizavimas;
- Vienu metu paleidžiamų GPU skaičiavimų gijų, tinklelių skaičius;
- Populiacijos kartų bei individų skaičiaus optimalus parinkimas;
- Skaičiavimų parametrų rūšiavimas;

Visos šios sritys ir yra išsamiai apžvelgiamos toliau šiame skyriuje.

5.1. Skaičiavimų spartos tarp CPU ir GPU palyginimas

Pradinis pagreitėjimo įvertinimas, atliktas tiesiog perkėlus kodą, parašytą atlikti skaičiavimams su CPU. Šis skaičius bus naudojamas kaip atskaitos taškas, tai yra stebėsime kokį papildomą pagreitėjimą pavyks pasiekti panaudojus kiekvieną iš tolimesnių optimizavimo technikų. Pradiniai įverčiai, naudojant keletą skirtingų skaičiavimo bazių, pateikti 9 Lentelėje. Visi skaičiavimai čia pateikiami lyginant NVIDIA GeForce GTX 550 Ti GPU lustą su kompiuteryje esančiu Intel Core i5-2320, 3.00 GHz 4 šerdžių CPU. Vėlesniame 2.2 darbo skyriuje bendras pagreitėjimas bus įvertintas ir su galingesne GeForce GTX 560 vaizdo korta.

Lentelės stulpelyje „Bendra trukmė“ yra pateikiama bendra programos veikimo trukmė, įskaitant ir grynai su CPU atliekamus genetinių algoritmų skaičiavimus, kintamųjų perkėlimą iš CPU į GPU ir atgal. Vienos kartos populiacijos vertinimo vidurkis yra pateikiamas tikrai populiacijos vertinimo algoritmo trukmei. Bendram pagreitėjimo rodikliui naudosiu abiejų algoritmų bendros trukmės reikšmes, kadangi norint tvarkingai įvertinti pagreitėjimą reikia prie GPU skaičiavimų laiko pridėti ir tų veiksmų trukmę, kurie skaičiuojant su CPU būtų nereikalingi. Jeigu lyginti grynai vienos populiacijos vertinimo laikus, pagreitėjimas gaunasi šiek tiek didesnis.

9 Lentelė. Pradiniai GPU ir CPU skaičiavimo spartos palyginimai

Skaičiavimo bazė	GTX 550 Ti		CPU Intel i-5, 3.0 GHz		Pagreitėjimas GPU vs CPU, kartais
	Bendra trukmė (ms)	Vienos kartos populiacijos įvertinimas (ms)	Bendra trukmė (ms)	Vienos kartos populiacijos įvertinimas (ms)	
nPop = 3072 nGen = 30	65583	2096.5	829270	26729	12,6
nPop = 6144 nGen = 30	114302	3602	1657261	53388.5	14,5
nPop = 3072 nGen = 60	122320	1984	1657076	26735	13,54

Kaip matome iš duomenų lentelėje, visiškai neoptimizuotas bei neanalizuotas kodas, tiesiog perkėlus jį iš CPU į GPU, suteikia iki 14,5 kartų pagreitėjimą lyginant su laiku, kurį užtrukome skaičiuodami su CPU. Atsižvelgiant į tai, kad bandomasis GPU yra gana paprastas bei nebrangus, rezultatas yra tikrai neblogas. Pamėginsime jį pagerinti tolimesniuose šio darbo skyriuose.

5.1.1. Programos kodo optimizavimas lygiagretiesiems CUDA skaičiavimams

Lygiagretieji CUDA skaičiavimai skiriasi nuo lygiagrečiųjų skaičiavimų atliekamų įprastinių CPU masyvu ar superkompiuteriu tuo, kad skaičiavimai CUDA yra atliekami visoms gijoms vykdant vieną operaciją, naudojant skirtingus duomenis / parametrus. Tokiu būdu, jeigu dėl duomenų skirtumų skaičiuojančiosioms gijoms reikia keliauti skirtingais keliais programoje, kitų gijų skaičiavimai yra užlaikomi ir bendra skaičiavimų sparta krenta. Taigi, kuo mažiau programoje yra išsišakojimų (*if, else, switch* teiginių) tuo skaičiavimai bus spartesni. Šis parametras, vadinamas užėmimo parametru (*angl. occupancy*), yra vienas iš matmenų, kuriuo yra matuojamas teorinis bei pasiektas CUDA algoritmo efektyvumas. Šį parametrą įtakoja ne tik pati programos algoritmo struktūra, bet ir daugelis kitų veiksmų, kurie specifiniai kiekvienam atskiram algoritmui.

Atlikęs savo kodo peržiūrą, radau keletą nebūtinų ar lengvai pakeičiamų kitais būdais *if/else* sąlygų, kurias pakeičiau. Deja, šie smulkūs pataisymai pastebimos įtakos nei skaičiavimo spartai su GPU nei su CPU neturėjo.

5.1.2. Naudojamų kintamųjų tipų įtaka

Pradiniame algoritme visi kintamųjų tipai buvo parinkti pagal tai, kokios galimos reikšmės buvo numatytos tam kintamajam. Tokiu būdu, daugelis parametrų rinkinio kintamųjų turėjo *char* arba *short* kintamųjų tipus, kas leido sumažinti bendrą reikalingos atminties kiekį. Atliekant šio darbo literatūros apžvalgą, suradau, kad CUDA algoritmuose, jeigu jūsų programa neturi ribojimų dėl panaudojamo atminties kiekio, yra rekomenduojama naudoti įprastinį *int* duomenų tipą sveikiesiems skaičiams, kadangi visi CUDA registrai yra 32-bitų ilgio, ir naudojimas kintamųjų užimančių mažiau atminties neturi jokios įtakos. Iš tiesų, teigiama, kad pvz. *short* tipo naudojimas netgi sulėtina skaičiavimus, kadangi atliekant kiekvieną operaciją kernelis verčia kintamąjį į *int* tipą, kad atliktų skaičiavimus bei gautą rezultatą verčia atgal į prieš tai buvusį tipą. Dėl to be reikalo yra išeikvojamos dvi kernelio operacijos. Pakeitus visus mano parametrus ir kintamuosius į *int* tipą gautas toks pagreitėjimas (naudojant nPop = 6144, nGen = 30 skaičiavimų bazę bei lyginant su 5 lentelėje esančiais duomenimis):

- GPU
  - Bendra trukmė: **109598**. Pagreitėjimas**: 4,1 %**
  - Vienos kartos įvertinimo trukmė: **3452**. Pagreitėjimas**: 4,2%**
- CPU
  - Bendra trukmė: **1621146**. Pagreitėjimas: **2,2 %**
  - Vienos kartos įvertinimo trukmė: **52250**. Pagreitėjimas: **2,2%**
- GPU vs CPU
  - Bendras pagreitėjimas: **14,8 karto.**

5.1.3. Prekybos duomenų struktūros optimizavimas

Norint pasiekti gerą skaičiavimų greitį, reikalinga užtikrinti, jog informacijos paėmimas iš bendros atminties būtų kaip įmanoma retesnis bei nuoseklesnis. Tai yra būtina užtikrinti, jog duomenys reikalingi skirtingoms vienu metu veikiančioms gijoms nebūtų smarkiai išmėtyti bendrojoje atmintyje. Pati šio uždavinio specifika padiktavo ir savo duomenų (informacijos apie valiutų kursus kiekvienu laiko momentu) struktūrą, tai yra duomenys yra išdėstyti pagal laiką. Šiam uždaviniui toks išdėstymas ir yra tinkamiausias, dėl to jokios papildomos duomenų struktūros optimizacijos nereikia.

Verta atsižvelgti į tai, jog esant tokiam atvejui, kai skaičiavimams būtų naudojami keleto valiutų porų kursų duomenys, šių duomenų struktūra turėtų būti tokia, kad to pačio laikotarpio duomenys būtų kuo arčiau vieni prie kitų. Tokiu atveju dvimatis masyvas Data[N][2], kur N yra laiko intervalų skaičius, būtų netinkamas variantas, kadangi bet kurio laikotarpio D duomenys

kompiuterio atmintyje bus atskirti per visą vieno komponento kainų istoriją. Tokiu atveju tiesiog masyvo pakeitimas į Data[2][N] turėtų ženklią įtaką skaičiavimų spartai.

5.1.4. Populiacijos individų / parametrų rinkinių rūšiavimo įtaka skaičiavimams

Siekiant padidinti CUDA procesorių užėmimo (*occupancy*) parametrą, t.y. kaip įmanoma daugiau suvienodinti skaičiavimų kelią vienu metu skaičiuojančiose gijose, atliekamas parametrų rinkinių rūšiavimas. Vadovaujantis tokia logika, jog kuo panašesni parametrai gretimai veikiančiose gijose, tuo labiau tikėtina, jog jų sprendimų kelias bus panašus. Tyrimo, kuriuo nustatoma skaičiavimų sparta surūšiavus populiaciją pagal kiekvieną iš galimų parametrų, bei keletu jų porų, rezultatai pateikti 6 lentelėje. Skaičiavimams buvo naudojama nPop = 6144, nGen = 30 skaičiavimų bazė, lyginama su pradiniu, neoptimizuotu algoritmu (5 lentelė). Parametrų rūšiavimui naudojamas *QuickSort* algoritmas [Wik12h], kuris kainuoja palyginus labai nedaug skaičiavimų laiko (vidutinė algoritmo kaina - O(n log n)) kadangi parametrų (n) yra žymiai mažiau nei valiutų kursų duomenų. Kaip matome iš 10 Lentelės, didžiausias pagreitėjimas buvo pasiektas surūšiavus parametrų rinkinius pagal use_SL_TS, nRSI, MACD parametrus. Įvertinęs tai, pabandžiau surūšiuoti pagal šių parametrų derinius, ko pasekoje pavyko pasiekti 20% pagreitėjimą.

10 Lentelė. Skaičiavimų pagreitėjimas surūšiavus populiaciją pagal parametrus

Parametras	Bendra trukmė (ms)	Vienos kartos populiacijos įvertinimas (ms)	Pagreitėjimas: Bendra trukmė, %	Pagreitėjimas: Vienos kartos populiacijos vertinimo, %
MA1	111587	3514.5	2.4	2.4
MA2	108655	3416.9	4.9	5.2
MACD	101994	3207.6	**10.8**	**11.0**
use_SL_TS	98842	3104.5	**13.5**	**13.8**
SL	107766	3398.4	5.7	5.7
TP	108125	3410.9	5.4	5.3
nRSI	99794	3137.7	**12.7**	**12.9**
RSI_bound	108702	3403.4	4.9	5.5
MACD_diff	107921	3394.6	5.6	5.8
Action	105488	3321.8	7.7	7.8
wMACD	110277	3463.2	3.5	3.9
wRSI1	112508	3544.8	1.6	1.6
wRSI2	110651	3497.0	3.2	2.9
use_SL_TS ir nRSI	94443	2970.7	**17.4**	**17.5**
use_SL_TS, nRSI ir MACD	91479	2861.5	**20.0**	**20.6**

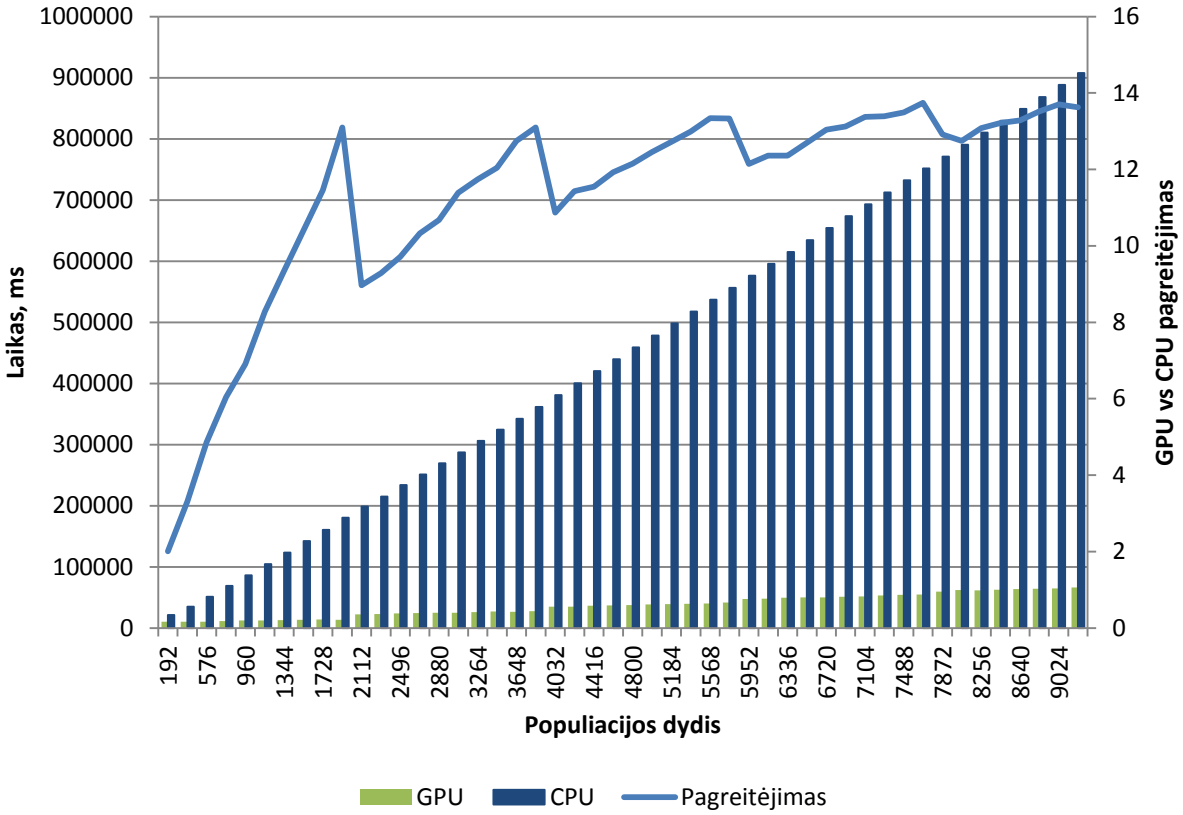
5.1.5. Populiacijos individų bei kartų kiekio įtaka skaičiavimams

Jau 5 lentelėje galima pastebėti, kad priešingai nei CPU, GPU skaičiavimų trukmė, dvigubai padidinus skaičiavimų aibę, nėra tiesiškai priklausoma. Matome, kad padidinus populiacijos dydį dvigubai, bet nepakeitus kartų skaičiaus, skaičiavimai atlikti greičiau, nei padidinus kartų skaičių, bet nepadinius populiacijų skaičiaus.

Kita vertus, pastebime, jog skaičiuojant vis tolesnes individų populiacijas, jų nariai panašėja tarpusavyje, dėl ko kelias kuriuo yra vykdoma programa taip pat panašėja.

Nuo populiacijos individų skaičiaus, priklauso kiek tinklelių / blokų galima bus paleisti skaičiuoti vienu metu, viename procesoriuje. Populiacijų skaičius turėtų be liekanos dalintis iš vienu metu leidžiamų gijų skaičiaus, kad skaičiavimai tolygiai pasidalintų į blokus. Palyginimas, kaip priklauso pasiektas pagreitėjimas nuo pradinės populiacijos, pavaizduotas 10 paveiksle. Paveiksle, aiškiai matyti kaip prie tam tikrų populiacijos dydžių įvyksta staigūs pagreitėjimo pokyčiai. Tai įvyksta dėl jau minėto nepakankamo GPU procesorių išnaudojimo. Kadangi šiame

GPU yra 192 CUDA šerdys, o norint pilnai išnaudoti visus , pastebima jog geriausias skaičiavimų pagreitėjimas yra pasiekiamas, kai populiacijos dydis yra 1920 kartotinis.



10 Paveikslas. GPU ir CPU skaičiavimų trukmė pagal populiacijos dydį

5.1.6. Tinkamiausio blokų / tinklelių skaičiaus GPU skaičiavimams parinkimas

Norint parinkti tinkamiausią gijų skaičių, kad pasiekti maksimalų galimą CUDA procesorių užimtumą (*occupancy*), galima naudotis tam skirtu užimtumo skaičiuotuvu [Nvid12b]. Kompiliuojant CUDA kodą, kompiliatorius pateikia informaciją, pagal kurią galima suskaičiuoti tinkamiausią gijų skaičių viename bloke. Kompiliuojant pradinį kodą gauname tokią informaciją, iš kurios matome jog kernelis naudos 63 registrus bei naudojama naujausia, sm_20, CUDA architektūra (11 paveikslas).

```
ptxas info: Compiling entry function '_Z6kernelP9tick_dataP11TradeParamsP10tr_resultsi' for
'sm_20'
ptxas info: Function properties for _Z6kernelP9tick_dataP11TradeParamsP10tr_resultsi
8 bytes stack frame, 4 bytes spill stores, 4 bytes spill loads
      ptxas info: Used 63 registers, 60 bytes cmem[0], 16 bytes cmem[16]
```

11 Paveikslas. Kompiliatoriaus informacija

Naudojantis *CUDA Occupancy* skaičiuotuvu suskaičiavau šiuos, tinkamiausius mano programai bei GPU, tinklelių (*grid)* dydžius:

- Naudojant sm_20 architektūrą: 64
- Naudojant sm_13 architektūrą: 192

5.1.7. Papildomi CUDA kompiliatoriaus nustatymai

Lig šiol visi skaičiavimai buvo atliekami kompiliuojant CUDA programą naudojant numatytuosius CUDA kompiliatoriaus nustatymus. Nors kompiliatorius ir stengiasi parinkti optimaliausius parametrus analizuodamas kodą, rezultatai, kuriuos gauname iš jo, nebūtinai visada yra geriausi.

Iš kompiliatoriaus pateiktos informacijos pastebėjau jog kompiliuojant yra naudojamas naujausias *sm_20* architektūros variantas. Kadangi mano programa nenaudoja jokių specifinių naujausios architektūros sprendimų, o CUDA vartotojų forumuose yra nemažai temų, aptariančių tai, jog *sm_13* architektūros skaičiavimų greitis yra didesnis nei sm_20. Pritaikius šį parametrą pastebėjau ženklų skaičiavimo pagreitėjimą – lyginant su sm_20 architektūros kodu, skaičiavimai paspartėjo **25%.** Šį pagreitėjimą galima paaiškinti tuo, jog įprastai naudojant sm_20 architektūrą yra naudojami sprendimai, labiau orientuoti į skaičiavimų tikslumą, nei į skaičiavimų spartą [Gil10]. Šie sprendimai yra IEEE-tikslioji dalyba, IEEE-tikslioji kvadratinė šaknis ir pan. Norint pasiekti panašų skaičiavimų greitį naudojant sm_20 architektūrą, kompiliuojant galima panaudoti parametrus, išjungiančius atitinkamus tikslaus skaičiavimo reikalavimus:

```
-ftz=true -prec-div=false -prec-sqrt=false
```

Atsižvelgiant į tai, jog šiame darbe naudojamuose skaičiavimuose nėra naudojama kvadratinės šaknies traukimo operacija, dalybos operacijos taip pat nėra labai dažnos, nusprendžiau šiuos parametrus naudoti tolimesniuose skaičiavimuose. Taip pat, verta pažymėti, kad ypatingai aukštas skaičiavimų tikslumas prekybos modeliavimo uždavinyje nėra būtinas.

Kitas kompiliatoriaus parametras, kurį pakoregavus gaunamas papildomas pagreitėjimas yra maksimalus registrų skaičius. Kompiliuojant kodą sm_13 architektūra matome, jog bus naudojama 44 registrai. Pagal *CUDA Occupancy* skaičiuotuvą matyti, jog jeigu būtų naudojami 42 registrai, būtų galima vienu metu skaičiuoti 2 gijų blokus viename procesoriuje vietoje vieno. Pridėjus parametrą *-maxrregcount=42* kompiliatoriuje buvo pastebėtas papildomas **10%** skaičiavimų pagreitėjimas

5.2. Bendras pasiektas pagreitėjimas

Sudėjus visas 2.1.1 – 2.1.7 poskyriuose apžvelgtas optimizavimo technologijas, buvo gautas ženklus skaičiavimų pagreitėjimas, ne tik lyginant su skaičiavimų sparta naudojant CPU skaičiavimus, bet ir su pradiniu, neoptimizuotu GPU kodu. Bendras kiekvieno optimizacijos žingsnio suteiktas pagreitėjimas, bei pasiekto pagreitėjimo įvertinimas apjungus visus optimizacijos metodus pateikiamas 7 lentelėje.

11 Lentelė. Optimizavimo metodų suteikto pagreitejimo apžvalga

Optimizacija	**Optimizuoto kodo skaičiavimų trukmė**	**Lyginant su pradiniu GPU, %**	**Lyginant su pradiniu CPU, kartais**
Programos kodo optimizacija	**-**	**0%**	**-**
Kintamųjų tipų pakeitimas	**109598**	**4,1%**	**14,6**
Prekybos duomenų struktūros optimizavimas	**-**	**0%**	**-**
Parametrų rinkinių surūšiavimas	**91479**	**20%**	**18,1**
Tinkamiausio populiacijos dydžio / skaičiavimo gijų parinkimas	Jau buvo skaičiuota su tinkamiausiu dydžiu	-	-
Optimizavimas naudojant papildomus CUDA kompiliatoriaus nustatymus	**75757**	**34%**	**21,46**
Bendras pagreitėjimas (sujungus visas anksčiau minėtas optimizacijas)	**61262**	**46,41%**	**27**

Kaip matome, pritaikius visas skaičiavimų optimizavimo metodikas, bendras skaičiavimų pagreitėjimas, kurį pavyko pasiekti palyginus su įprastiniu kompiuterio procesoriumi, yra 27 kartai. Kadangi uždavinys nėra labai paprastas ir jo skaičiavimuose yra naudojamas labai didelis duomenų kiekis, manau pasiektas pagreitėjimas yra tikrai geras. Turint tokį pagreitėjimą, rezultatus, kuriais remiantis galima priimti atitinkamus aukšto dažnio prekybos sprendimus realiame laike, galime gauti daug greičiau. Vietoj to, jog skaičiuojant įprastiniu kompiuterio

procesoriumi rezultatų turėtume laukti vieną ar keletą valandų, skaičiuojant su GPU juos gauname per keletą minučių.

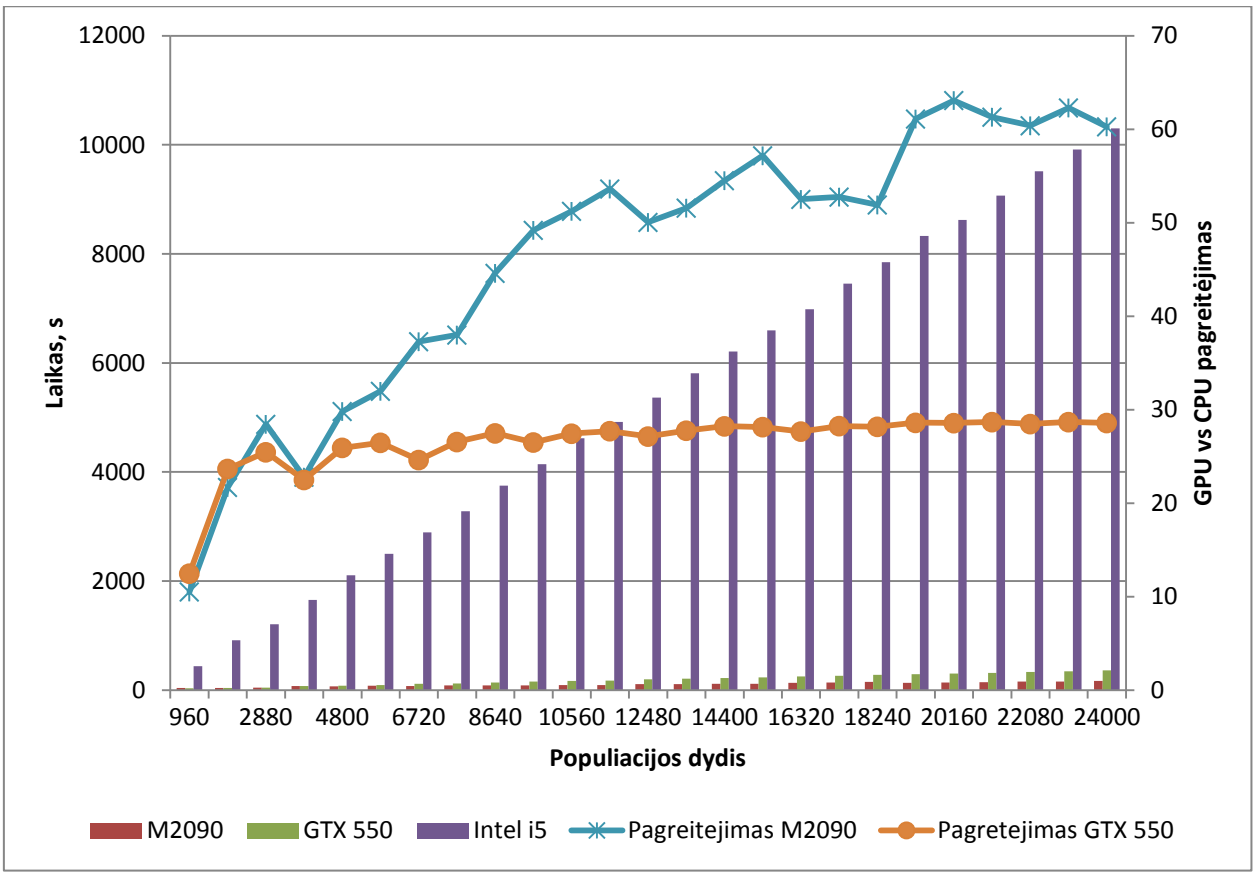
5.3. Skirtingų GPU lustų skaičiavimų spartos palyginimas

Lig šiol, darbe visi skaičiavimai buvo atliekami lyginant naudojant NVIDIA GTX 550Ti GPU bei Intel i5-2320 CPU. Norėdamas įvertinti galimą skaičiavimų pagreitėjimą naudojant galingesnį GPU lustą, gavau prieigą prie vieno iš didžiausių Europos GPU masyvų MinoTauro, Barselonos superkompiuterių centre, bei atlikau skaičiavimus su vienu iš ten esančių NVIDIA M2090 GPU lustų. Lyginamų GPU bei CPU specifikacijos pateiktos 12 Lentelėje.

12 Lentelė. Lyginamų GPU ir CPU specifikacijos

Parametras	**GPU**		**CPU**	
	GTX 550 Ti	**NVIDIA M2090**	**Intel i5-2320 @ 4x3,00 GHz**	**Intel E5649 @ 6x2,53 GHz**
Šerdžių skaičius	192	512	4	6
DRAM Atmintis	1 GB	6 GB	8 GB	24 GB
Procesoriaus taktinis dažnis	1,8 GHz	1,3 GHz	3,00 GHz	2,53 Ghz

Skaičiavimų trukmės palyginimas pavaizduotas 12 paveiksle. Skaičiuojant su GPU buvo naudojami visi ankstesniame skyriuje apžvelgti bei atrasti skaičiavimų optimizacijos būdai, siekiant maksimaliai paspartinti skaičiavimus bei realiai įvertinti gaunamą pagreitėjimą. Kaip matome iš grafiko, esant nedideliam populiacijos skaičiui nėra pilnai išnaudojamos visos NVIDIA M2090 grafinės plokštės galimybės, kadangi nėra pakankamai skirtingų parametrų tam, kad išnaudoti visus jos turimus 512 procesoriaus branduolius. Maksimalus pagreitėjimas yra pasiekiamas maždaug ties 20000 populiacijos dydžiu. Taigi, iš šio paveikslo dar kartą įsitikiname, kaip svarbu yra tinkamai parinkti skaičiavimuose naudojamą populiacijos dydį, kuris tiesiogiai įtakoja, kiek gijų bei gijų tinklų bus naudojama programoje. Taip pat matome, jog šis skaičius yra skirtingas naudojant skirtingų klasių GPU lustus, todėl kuriant kodą būtų pravartu žinoti kokia technine įranga bus naudojama jam skaičiuoti.



12 Paveikslas. GPU ir CPU skaičiavimų trukmės palyginimas

5.4. Išvados

Taigi, apibendrinant, atlikus išsamią turėto kodo bei naudojamų duomenų analizę, darbo eigoje pavyko pasiekti GPU skaičiavimų pagreitėjimą iki 27 kartų naudojantis įprastiniu namuose turimu grafiniu lustu GTX 550Ti, bei daugiau nei 60 kartų pagreitėjimas naudojant aukštesnės klasės, specialiai lygiagretiesiems skaičiavimams skirtą NVIDIA Tesla M2090 grafinį lustą.

Atsižvelgiant į tai, kad pats skaičiavimų uždavinys nėra labai paprastas, ir jo skaičiavimuose yra naudojamas labai didelis duomenų kiekis, manau pasiektas pagreitėjimas yra tikrai geras. Turint tokį pagreitėjimą, rezultatus, kuriais remiantis galima priimti atitinkamus aukšto dažnio prekybos sprendimus realiame laike galime gauti daug greičiau. Vietoj to, jog skaičiuojant įprastiniu kompiuterio procesoriumi rezultatų turėtume laukti vieną ar keletą valandų, skaičiuojant su GPU juos gauname per keletą minučių.

REZULTATAI IR IŠVADOS

Šiame magistro darbe buvo išanalizuoti literatūros šaltiniai, susiję su automatizuotos prekybos sistemos kūrimu, fundamentaliosios bei techninės analizės taikymu prekybos automatizacijoje. Taip pat pristatyta NVIDIA CUDA lygiagrečiųjų skaičiavimų fizinė bei loginė architektūra, jos skirtumai nuo įprastiniuose CPU naudojamos architektūros, atlikta esamų genetinių algoritmų bibliotekų analizė bei pasirinkta *GALib* genetinė biblioteka kaip tinkamiausia naudoti šiame darbe.

Pagrindiniai pasiekti darbo **rezultatai** yra šie:

1. Sukurta aukšto dažnio automatinės prekybos modeliavimo sistema Forex biržoje, atliktas prekybos modeliavimas, naudojant tris skirtingas valiutų poras bei keletą skirtingų laiko intervalų mokymosi bei testavimo rezultatams.
2. Naudojantis sukurta prekybos modeliavimo sistema, atlikta skaičiavimų spartos skirtumo tarp CPU ir GPU palyginamoji analizė. Apžvelgti pagrindiniai skaičiavimų su GPU optimizavimo parametrai, įtakojantys skaičiavimų su GPU spartą. Pritaikius visus atrastus optimizacijos būdus, galiausiai buvo pasiektas daugiau nei 27 kartų pagreitėjimas naudojant GTX 550Ti GPU, bei daugiau nei 60 kartų pagreitėjimas, lyginant su CPU skaičiavimais, naudojant greitesnį NVIDIA M2090 GPU.

Pasiekus šiuos rezultatus, kurie ir buvo pagrindinis šio darbo tikslas, galima pateikti tam tikras **išvadas** bei pastebėjimus:

1. Naudojant prekybos modeliavimo sistemą, geriausi surastieji parametrų rinkiniai generuoja apie 5% kapitalo prieaugį per 4 savaičių mokymosi laikotarpį. Ne visi surastieji parametrų rinkiniai duodavo tokius pat gerus rezultatus ir vėlesniuose testiniuose duomenyse, tačiau buvo pastebėta gana aiški tendencija, kad kuo mažesnis laiko skirtumas tarp mokymosi ir testinių duomenų, tuo labiau tikėtina, jog parinkti parametrai generuos gerus rezultatus ir su naujais duomenimis. Tai leidžia daryti išvadą, jog **skaičiavimų trukmė yra labai svarbi**, ir kuo ankščiau mes gausime rezultatus bei galėsime juos panaudoti, tuo didesnė tikimybė, kad prekyba bus sėkminga.
2. Siekiant padidinti GPU lygiagrečiųjų skaičiavimų spartą, buvo išskirti pagrindiniai juos paspartinantys veiksniai:
    a. Parametrų rinkinių surūšiavimas, kuris leido panašius kelius algoritme turinčius skaičiavimus atlikti vienu metu;

b. Tinkamiausias skaičiuojamų populiacijos individų skaičius, kuris priklauso nuo grafinės plokštės, kuria bus atliekami skaičiavimai;
c. Atitinkamų CUDA kompiliatoriaus parametrų parinkimas, siekiant kaip įmanoma geriau išnaudoti visas grafinio lusto skaičiavimo galimybes.

3. Pasiektas bendras skaičiavimų pagreitėjimas leidžia rezultatus, kurių, naudodamiesi įprastiniais CPU skaičiavimais, turėtume laukti valandą, gauti bei pradėti naudoti per 1-2 minutes, priklausomai nuo to, kokį grafinį lustą naudojame.

Tikiuosi, kad šio magistro darbo metu surinkta medžiaga galėtų pasitarnauti tiems, kurie planuoja atlikti tyrimus panašiose ar susijusiose srityse. Šio darbo pagrindinis tikslas buvo orientuotas į skaičiavimų spartos analizę, bei palyginimą, tik paviršutiniškai apžvelgiant genetinių algoritmų panaudojimą parametrų rinkinių paieškai. Dėl to, kaip vienas iš galimų tolimesnių šio darbo tyrimo etapų galėtų būti išsami genetinių algoritmų panaudojimo analizė.

ŠALTINIŲ SĄRAŠAS

[MW12] Dave McKenney, Tony White. Stock trading strategy creation using GP on GPU. In: Journal of Soft Computing, Volume 16, Issue 2, 2012, pp 247–259.

[Nvid12] What is CUDA | NVIDIA Developer Zone, [žiūrėta: 2012-03-12]. Prieiga per Internetą: http://developer.nvidia.com/what-cuda

[Wik12a] Fundamental analysis - Wikipedia, the free encyclopedia, [žiūrėta: 2012-03-20]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Fundamental_analysis

[YN09] Yiyi Jiang, Laura Nunez. Efficient market hypothesis or adaptive market hypothesis? A test with the combination of technical and fundamental analysis. In: Proceedings of the 15th International Conference. Computing in Economics and Finance, University of Technology, Sydney, Australia, 2009.

[Ban12] Terminų žodynas | Bankai.lt, [žiūrėta: 2012-03-23]. Prieiga per Internetą: http://www.bankai.lt/ziniu-bankas/terminu-zodynas

[Inv12] Financial Dictionary | Investopedia, [žiūrėta: 2012-03-23]. Prieiga per Internetą: http://www.investopedia.com/dictionary

[CY06] John Y. Campbell, Motohiro Yogo. Efficient tests of stock return predictability. In: Journal of Financial Economics, Volume 81, 2006, pp. 27-60.

[BDT08] Turan G. Bali, K. Ozgur Demirtas, Hassan Tehranian. Aggregate Earnings, Firm-Level Earnings and Expected Stock Returns. In: Journal of Financial and Quantitative Analysis, Volume 43(3), 2006, pp. 657-684.

[ELW+08] Ming Hao Eng, Yang Li, Qing-Guo Wang, Tong Heng Lee. Forecast Forex With ANN Using Fundamental Data. In: International Conference on Information Management, Innovation Management and Industrial Engineering, Volume 1, 2008, pp. 279-282.

[Wik12b] Technical analysis - Wikipedia, the free encyclopedia, [žiūrėta: 2012-03-23]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Technical_analysis

[Wik12c] Dow teorija – Vikipedija, [žiūrėta: 2012-03-23]. Prieiga per Internetą: http://lt.wikipedia.org/wiki/Dow_teorija

[LX09] Zhihong Liu, Deyun Xiao. An Automated Trading System with Multi-indicator Fusion Based on D-S Evidence Theory in Forex Market. In: Sixth International Conference on Fuzzy Systems and Knowledge Discovery, Volume 3, 2009, pp. 239-243.

[Wik12d] Parallel computing - Wikipedia, the free encyclopedia, [žiūrėta: 2012-03-25]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Parallel_computing

[Wik12e] Paskirstytasis skaičiavimas – Vikipedija, [žiūrėta: 2012-03-25]. Prieiga per Internetą: http://lt.wikipedia.org/wiki/Paskirstytasis_skai%C4%8Diavimas

[KMJ+10] Frédéric Krüger, Ogier Maitre, Santiago Jiménez, Laurent Baumes, Pierre Collet. Speedups between ×70 and ×120 for a Generic Local Search (Memetic) Algorithm on a Single GPGPU Chip. In: Applications of Evolutionary Computation, Volume 6024/2010, 2010, pp. 501-511.

[LKC+10] Victor W Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher. Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. In: Proceedings of the 37th annual international symposium on Computer architecture, France, 2010, pp. 451-460.

[Fer11] Iván Contreras Fernández. Parallel Architectures To Improve a GA Based Real-Time System For Trading the Stock Market, Master thesis, Universidad Complutense de Madrid, Madrid 2011, 81 pages.

[Nvid09] NVIDIA's Next Generation CUDA Compute Architecture: Fermi, Whitepaper, 2009, 21 pages. Prieiga per Internetą: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[Gla09] Peter N. Glaskowsky. NVIDIA's Fermi: The First Complete GPU Computing Architecture, Whitepaper, 2009, 26 pages. Prieiga per Internetą: http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA%27s_Fermi-The_First_Complete_GPU_Architecture.pdf

[Gol89] David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989, 432 pages.

[AK99] Franklin Allen, Risto Karjalainen. Using genetic algorithms to find technical trading rules. In: Journal of Financial Economics, Volume 51, 1999, pp. 245-271.

[Wik12f] Genetic algorithm - Wikipedia, the free encyclopedia, [žiūrėta: 2012-03-25]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Genetic_algorithms

[Wik12g] Genetiniai algoritmai – Vikipedija, [žiūrėta: 2012-03-25]. Prieiga per Internetą: http://lt.wikipedia.org/wiki/Genetiniai_algoritmai

[Wall96] Matthew Wall. GAlib: A C++ Library of Genetic Algorithm Components, [žiūrėta: 2012-03-25]. Prieiga per Internetą: http://lancet.mit.edu/ga/dist/galibdoc.pdf

[Duka12] Historical Data Feed :: Dukascopy Bank SA, [žiūrėta: 2012-03-20]. Prieiga per Internetą: http://www.dukascopy.com/swiss/english/data_feed/historical/

[Jong75] Kenneth Alan De Jong. Analysis of the Behaviour of a Class of Genetic Adaptive Systems, PhD thesis, University of Michigan, Michigan, USA, 1975, 271 pages.

[Wik12h] Quicksort - Wikipedia, the free encyclopedia, [žiūrėta: 2012-04-15]. Prieiga per Internetą: http://en.wikipedia.org/wiki/Quicksort

[Nvid12b] CUDA Occupancy Calculator - NVIDIA, [žiūrėta: 2012-04-15]. Prieiga per Internetą: http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls

[Gil10] Mike Giles. Advanced CUDA 01, Presentation, 2010, 81 pages. Prieiga per Internetą: http://people.maths.ox.ac.uk/gilesm/cuda/cudaconf_oxford.pdf

PRIEDAI

1 Priedas. Prekybos modeliavimo sistemos pagrindinės funkcijos išeities kodas

```
__device__ tr_results simulate_trading_GPU (tick_data * TickArray_GPU,
TradeParams p, int tick_elem) {
    float funds = 100000;
    tr_results tr_r;
    tr_r.params_used = p;
    tr_r.funds_start = funds;
    tr_r.trades = 0;
    tr_r.successfull_trades = 0;
    tr_r.funds_max = funds;
    tr_r.funds_end = 0;

    int in_trade = 0; // jei -1 tai sell, jei 0 - no trade, jei 1 - buy
    float currency = 0;
    int i, j;
    float MACD, MACD_signal, MACD_compare, cur_ma1, cur_ma2;
    float uRSI = 0, dRSI = 0, RS, RSI, RSI_prev = 0;

    int trade_eval;
    trade current_trade;
    int start_at = 2001;

    //*** Skaičiuojam pirmąja MACD_signal, paprastumo dėlei MA vietoj EMA ***
    cur_ma1 = EMA1_GPU(TickArray_GPU, p.MA1, start_at-1);
    cur_ma2 = EMA1_GPU(TickArray_GPU, p.MA2, start_at-1);
    MACD = cur_ma1 - cur_ma2;
    MACD_signal = MACD;
    for (j=1; j<p.MACD; j++) { //suskaičiuojame pradine MACD Signaline
linija: EMA nuo MACD
      cur_ma1 = EMA2_GPU(p.MA1, cur_ma1, TickArray_GPU[start_at+j-1].price);
      cur_ma2 = EMA2_GPU(p.MA2, cur_ma2, TickArray_GPU[start_at+j-1].price);
      MACD = cur_ma1 - cur_ma2; //kai > 0 buy, kai < 0 sell, kai = 0 nieko
  MACD_signal += MACD;
 }
    MACD_signal = MACD_signal / p.MACD;
    MACD_compare = MACD - MACD_signal;
    start_at = start_at + p.MACD;
    // ********************

    // ***************** Skaičiuojam pradini RSI U ir D vidurki ********
    if (p.nRSI > p.MA2) p.nRSI = p.MA2;
    for (j=0; j<p.nRSI; j++) {
      if (TickArray_GPU[start_at-j-1].price > TickArray_GPU[start_at-j-
2].price)
        uRSI += TickArray_GPU[start_at+j-1].price - TickArray_GPU[start_at+j-
2].price;
      if (TickArray_GPU[start_at-j-1].price < TickArray_GPU[start_at-j-
2].price)
        dRSI += TickArray_GPU[start_at+j-2].price - TickArray_GPU[start_at+j-
1].price;
 }
    uRSI = uRSI / p.nRSI;
    dRSI = dRSI / p.nRSI;
    RS = uRSI / dRSI;
    RSI = 100 - (100/(1+RS));

    // skaičiuojam prekyba
    for (i=start_at; i<tick_elem; i++) { //keliaujam per visa istorija
```

```
      trade_eval = 0;
      if (in_trade != 0) { //jeigu jau yra nupirkta kasnors, žiūrim ar laikas
parduot
        //ar pasiekta SL arba TP
        if (p.use_SL_TS == 1) { //naudojam SL ir TP
          if (in_trade == -1) {
            if (TickArray_GPU[i].price > current_trade.SL) trade_eval = 999;
            if (TickArray_GPU[i].price < current_trade.TP) trade_eval = 999;
          }
          if (in_trade == 1) {
            if (TickArray_GPU[i].price < current_trade.SL) trade_eval = -999;
            if (TickArray_GPU[i].price > current_trade.TP) trade_eval = -999;
          }
        }
        else if (p.use_SL_TS == 2) { // naudojam trailing stop'a
          if (in_trade == -1) {
            if (TickArray_GPU[i].price > current_trade.SL) trade_eval = 999;
            if ( (TickArray_GPU[i].price + p.SL) < current_trade.SL )
current_trade.SL = TickArray_GPU[i].price + p.SL;
          }
          if (in_trade == 1) {
            if (TickArray_GPU[i].price < current_trade.SL) trade_eval = -999;
            if ( (TickArray_GPU[i].price - p.SL) > current_trade.SL )
current_trade.SL = TickArray_GPU[i].price - p.SL;
          }
        }
      }
        //vertinam MACD taisykle
        cur_ma1 = EMA2_GPU(p.MA1, cur_ma1, TickArray_GPU[i-1].price);
        cur_ma2 = EMA2_GPU(p.MA2, cur_ma2, TickArray_GPU[i-1].price);
        MACD = cur_ma1 - cur_ma2;
        MACD_signal = EMA2_GPU(p.MACD,MACD_signal,MACD);

        // jei mACD x MACD_signal - signalas
        if ( ((MACD_compare >= 0) && (MACD - MACD_signal) < 0 ) ||
((MACD_compare <= 0) && (MACD - MACD_signal) > 0 ) ) { //buvo persikirtimas
          if ( (MACD_compare > 0) && (MACD - MACD_signal) < (-1) *
p.MACD_diff ) {
            trade_eval += -1 * p.wMACD; //sell
          }
          else if ( (MACD_compare < 0 ) && (MACD - MACD_signal) > p.MACD_diff
) {
            trade_eval += 1 * p.wMACD; //buy
          }
        }
        MACD_compare = MACD - MACD_signal;
        //vertinam RSI

        RSI_prev = RSI;
        if (TickArray_GPU[i-1].price > TickArray_GPU[i-2].price) {
          uRSI = EMA2_GPU(p.nRSI, uRSI, TickArray_GPU[i-1].price -
TickArray_GPU[i-2].price);
          dRSI = EMA2_GPU(p.nRSI, dRSI, 0);
        }
        else if (TickArray_GPU[i-1].price < TickArray_GPU[i-2].price) {
          uRSI = EMA2_GPU(p.nRSI, uRSI, 0);
          dRSI = EMA2_GPU(p.nRSI, dRSI, TickArray_GPU[i-2].price -
TickArray_GPU[i-1].price);
        }
        else {
          uRSI = EMA2_GPU(p.nRSI, uRSI, 0);
          dRSI = EMA2_GPU(p.nRSI, dRSI, 0);
        }
```

```
        RS = uRSI / dRSI;
        RSI = 100 - (100/(1+RS));

        if (RSI < p.RSI_bound) {
          trade_eval += 1 * p.wRSI1;
        }
        else if (RSI > (100 - p.RSI_bound) ) {
          trade_eval += (-1) * p.wRSI1;
        }
        //vertinam RSI nuokrypi
        if ( (RSI_prev > RSI) && (TickArray_GPU[i-2].price < TickArray_GPU[i-
1].price ) ) {
          trade_eval += 1 * p.wRSI2;
        }
        else if ( (RSI_prev < RSI) && (TickArray_GPU[i-2].price >
TickArray_GPU[i-1].price ) ) {
          trade_eval += (-1) * p.wRSI2;
        }

      //ar perkam/parduodam?
      if (trade_eval != 0) {
        if (in_trade != 0 ) { //jau turim nupirke
          if ( trade_eval >= p.action ) { //LONG
            if (in_trade != 1) { //update LONG
              currency = leave_short_GPU (&funds, currency,
TickArray_GPU[i].price);
              in_trade = 0;
              if (tr_r.funds_max < funds) tr_r.funds_max = funds;
              current_trade.close_price = TickArray_GPU[i].price;
              if (current_trade.close_price < current_trade.open_price)
tr_r.successfull_trades++;

            }
          }
          else if ( trade_eval <= (-1)*p.action ) { //SHORT
            if (in_trade != -1) {
              currency = leave_long_GPU (&funds, currency,
TickArray_GPU[i].price);
              in_trade = 0;
              if (tr_r.funds_max < funds) tr_r.funds_max = funds;
              current_trade.close_price = TickArray_GPU[i].price;
              if (current_trade.close_price > current_trade.open_price)
tr_r.successfull_trades++;

            }
          }
        }
        else { //neturim nupirke
          if ( trade_eval >= p.action ) { //LONG
            currency = enter_long_GPU (&funds, TickArray_GPU[i].price);
            in_trade = 1; //long
            tr_r.trades++;
            current_trade.open_price = TickArray_GPU[i].price;
            current_trade.operation = 1;
            current_trade.SL = TickArray_GPU[i].price - p.SL*PIP;
            current_trade.TP = TickArray_GPU[i].price + p.TP*PIP;
            current_trade.number_owned = currency;

          }
          else if ( trade_eval <= (-1)*p.action ) { //SHORT
            currency = enter_short_GPU (&funds, TickArray_GPU[i].price);
            in_trade = -1; //short
            tr_r.trades++;
```

```
            current_trade.open_price = TickArray_GPU[i].price;
            current_trade.operation = -1;
            current_trade.SL = TickArray_GPU[i].price + p.SL*PIP;
            current_trade.TP = TickArray_GPU[i].price - p.TP*PIP;
            current_trade.number_owned = currency;
          }
        }
      }
      if ( (funds < 1000) && (in_trade == 0) ) break;
    }
    if (in_trade != 0 ) {
      if (in_trade == 1 ) {
        currency = leave_long_GPU (&funds, currency, TickArray_GPU[i-
1].price);
        in_trade = 0;
        if (tr_r.funds_max < funds) tr_r.funds_max = funds;
        current_trade.close_price = TickArray_GPU[i-1].price;
        if (current_trade.close_price > current_trade.open_price)
tr_r.successfull_trades++;
      }
      else {
        currency = leave_short_GPU (&funds, currency, TickArray_GPU[i-
1].price);
        in_trade = 0;
        if (tr_r.funds_max < funds) tr_r.funds_max = funds;
        current_trade.close_price = TickArray_GPU[i-1].price;
        if (current_trade.close_price < current_trade.open_price)
tr_r.successfull_trades++;
      }
    }
    tr_r.funds_end = funds;
    return tr_r;
     }
```