Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PETER ŠKODA

HETEROGENI RAČUNALNI SUSTAV S PROGRAMIRLJIVIM POLJEM LOGIČKIH ELEMENATA ZA UČENJE STABLA ODLUKE

DOKTORSKI RAD

Zagreb, 2014.

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PETER ŠKODA

HETEROGENI RAČUNALNI SUSTAV S PROGRAMIRLJIVIM POLJEM LOGIČKIH ELEMENATA ZA UČENJE STABLA ODLUKE

DOKTORSKI RAD

Mentori:

Prof.dr.sc. Vlado Sruk

Prof.dr.sc. Branka Medved Rogina

Zagreb, 2014.

Sveučilište u Zagrebu

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

PETER ŠKODA

HETEROGENEOUS COMPUTING SYSTEM WITH FIELD PROGRAMMABLE GATE ARRAY COPROCESSOR FOR DECISION TREE LEARNING

DOCTORAL THESIS

Supervisors

Professor Vlado Sruk, PhD

Professor Branka Medved Rogina, PhD

Zagreb, 2014

Doktorski rad izrađen je na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva, na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave i Institutu Ruđer Bošković, na Zavodu za elektroniku.

Mentori: prof.dr.sc. Vlado Sruk

prof.dr.sc. Branka Medved Rogina

Doktorski rad ima: 138 stranica

Doktorski rad br.:_____

O mentorima

Vlado Sruk rođen je u Zagrebu 1965. godine. Diplomirao je, magistrirao i doktorirao u polju elektrotehnike na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1988., 1991. odnosno 1998. godine.

Od siječnja 1989. godine radi na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave FER-a. U svibnju 2007. godine izabran je u zvanje izvanrednog profesora. Sudjelovao je na pet znanstvenih projekata Ministarstva znanosti, obrazovanja i sporta Republike Hrvatske i dva EU FP7 projekta. Trenutno sudjeluje u radu znanstvenih projekata Ministarstva znanosti i tehnologije republike „Oblikovanje okolina za ugrađene sustave“ i „Računalne okoline za sveprisutne raspodijeljene sustave“, i voditelj je radnog paketa FP7-ICT-2011-8 projekta „Embedded Computer Engineering Learning Platform“. Objavio je više od 30 radova u časopisima i zbornicima konferencija u području oblikovanja ugrađenih sustava i razvoja programske potpore.

Prof. Sruk član je stručnih udruga IEEE i ACM. Sudjeluje u radu međunarodnih programskih odbora znanstvenih konferencija, te sudjeluje kao recenzent u većem broju inozemnih časopisa i međunarodnih konferencija.

Branka Medved Rogina rođena je u Zagrebu 1958. godine. Diplomirala je, magistrirala i doktorirala u polju elektrotehnike na Sveučilištu u Zagrebu, Fakultetu elektrotehnike i računarstva, 1981., 1992. odnosno 1997. godine.

Od prosinca 1982. godine radi na Institutu Ruđer Bošković u Zagrebu, danas u Zavodu za elektroniku. U studenom 2011. godine izabrana je u znanstveno zvanje znanstvenog savjetnika, a od prosinca 2007. godine je u znanstveno-nastavnom naslovnom zvanju izvanrednog profesora. Sudjelovala je na devet znanstvenih projekata Ministarstva znanosti, obrazovanja i sporta Republike Hrvatske i dva EU FP projekta. Trenutno je istraživač na HRZZ projektu „Algoritmi strojnog učenja za otkrivanje znanja u složenim strukturama podataka“ i voditeljica radnog paketa na EU FP7 projektu „Embedded computer engineering

learning platform“. Autor je više od 50 znanstvenih radova objavljenih u časopisima i zbornicima konferencija u području visokorazlučivih mjerenja u vremenskom i amplitudnom području, te naprednih programirljivih arhitektura i značajki zasnovanih na FPGA ugradbenim sustavima.

Prof. Medved Rogina članica je IEEE, ELMAR i Akademije tehničkih znanosti Hrvatske (HATZ). Sudjelovala je u više od deset međunarodnih programskih odbora znanstvenih konferencija, te sudjeluje kao recenzent u nekoliko inozemnih časopisa. Za izum i prototip „Kvantni generator slučajnih brojeva“ nagrađena je s dvije međunarodne zlatne medalje i nagradom za inovacije u 2005. godini. Dobitnica je nagrade „Rikard Podhorsky“ za 2008. godinu, za istaknuti znanstveni doprinos u području razvoja i primjene visokorazlučivih metoda mjerenja vremenskih razmaka, s naglaskom na korištenje u procesima primjene novih elektroničkih tehnologija u Republici Hrvatskoj i šire.

About the Supervisors

Vlado Sruk was born in Zagreb in 1965. He received B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1988, 1991 and 1998, respectively.

From January 1989 he is working at the Department of Electronics, Microelectronics, Computer and Intelligent Systems at FER. In May 2007 he was promoted to Associate Professor. He participated in 5 scientific projects financed by the Ministry of Science, Education and Sports of the Republic of Croatia and 2 EU FP7 projects. Currently he is a involved in research project: "Embedded Systems Design Environment" and "Computing Environments for Ubiquitous Distributed Systems" financed by the Ministry of Science, Education and Sports of the Republic of Croatia and a work package leader of FP7-ICT-2011-8 project: "Embedded Computer Engineering Learning Platform". He published more than 30 papers in journals and conference proceedings in the area of embedded system design and software design.

Prof. Sruk is a member of IEEE and ACM. He participated in work of conference international programs committees and he serves as a technical reviewer for various international journals and conferences.

Branka Medved Rogina was born in Zagreb in 1958. He received B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1981, 1992 and 1997, respectively.

From December 1982 she is working at the Ruđer Bošković Institute, Zagreb, the Division of electronics. In December 2007 she became an Associate Professor and in November 2011 a Senior Scientist. She participated in 9 scientific projects financed by the Ministry of Science, Education and Sports of the Republic of Croatia and 2 EU FP projects. Currently she is researcher on the project "Machine learning algorithms for insightful analysis of complex data structures" financed by the HRZZ and a WP leader of FP7 project "Embedded computer engineering learning platform". She published more than 50 papers in journals and conference

proceedings in the area of high resolution signal dynamic measurement in the time and amplitude domain and advanced programmable architectures and features based on FPGA embedded systems design.

Prof. Medved Rogina is a member of IEEE, ELMAR and collaborating member of Croatian Academy of Engineering (HATZ). She participated in more than 10 conference international programs committees and she serves as a technical reviewer for various international journals. For invention and prototype *Quantum random number generator* she was awarded by two international gold medals and the award for innovations in 2005. In 2008 she received Annual award *Rikard Podhorsky* (HATZ) for distinguished scientific contribution in the development and application high-resolution time interval measurement methods, and test features of micro-and optoelectronic hardware, with an emphasis on the use of new electronic technologies in the Republic of Croatia and beyond.

Zahvala

Prof. Vladi Sruku i prof. Branki Medved Rogina na vodstvu, savjetima i podršci prilikom izrade ovog doktorskog rada.

Prof. Karolju Skali na razvojnom sklopovlju.

Radomiru i Tomislavu na mnogobrojnim šalicama razgovara za očuvanje mentalnog zdravlja.

Mojoj mami na potpori kroz cijeli život.

Mojoj dragoj ženi Adrijani na ljubavi, potpori i strpljenju kroz ove mjesece ludila.

Najdražoj kćeri Aniki na danima neprovedenim zajedno. Tata će to sve nadoknaditi.

Sažetak

U ovom radu prikazan je heterogeni računalni sustav i novi hibridni algoritam za učenje stabla odluke *Dataflow decision tree construction* – DF-DTC. Algoritam DF-DTC zasnovan je na algoritmu C4.5. Heterogeni sustav sadrži koprocesor izveden programirljivim poljem logičkih elemenata (FPGA, engl. *field programmable gate array*). Razrada arhitekture koprocesora i hibridnog algoritma DF-DTC provedena je metodologijom programsko-sklopovskog suobliokovanja. U koprocesoru je izvedena obrada nominalnih atributa skupa za učenje, a u algoritam su uvedene prilagodbe podatkovnih struktura, te podrška za višedtretveno izvođenje.

Vrednovanje performansi provedeno je mjerenjem ukupnog vremena izvršavanja rada programa, te mjerenjem vremena izvršavanja ključnih dijelova algoritma. Pri vrednovanju su korišteni sintetički skupovi za učenje, te skupovi za učenje javno dostupni na UCI repozitoriju. Performanse DF-DTC-a uspoređene su s performansama postojeće programske implementacije algoritma EC4.5. Ubrzanje obrade nominalnih atributa na DF-DTC-u iznosi u prosjeku 3,00 puta u usporedbi s programskom implementacijom EC4.5. Za cjelokupno izvršavanje programa najbolje ubrzanje iznosi 1,18 puta. Izvedba DF-DTC-a za pokazala je potencijal FPGA-a kao platforme za ubrzanje učenja stabla odluke.

Ključne riječi: *heterogeni računalni sustav, programirljivo polje logičkih elemenata, FPGA koprocesor, dubinska analiza podataka, stabla odluke, C4.5*

Abstract

Heterogeneous computing system with field programmable gate array coprocessor for decision tree learning

Heterogeneous computer systems comprise of a general purpose CPU and a special purpose processor, most commonly a GPU. Another platform that has recently gain traction in computing is field programmable gate arrays (FPGA). FPGAs provide a platform for implementing custom coprocessors tailored to the specific applications. The applications of interest in this thesis are data mining applications – more specifically, decision tree learning. Goal of this research is development of a novel FPGA coprocessor architecture suitable for implementation of decision tree learning algorithms, and definition and implementation of a hybrid algorithm adapted for the FPGA coprocessor.

Chapter 1 (*Introduction*) presents motivation of the work and introduces fundamental concepts. Also, the objective of the research, and scientific contributions are presented. *Chapter 2* (*Field programmable gate arrays – FPGA*) gives a short introduction to FPGA devices, their internal structure, and their use in computing systems. It also provides an outline of main features of Xilinx Virtex-6 FPGA, which is used for implementation of the coprocessor. In *Chapter 3* (*FPGA implementations of data mining algorithms*) a survey of current state of the art of using FPGAs as hardware accelerators for data mining is presented. Three algorithms were surveyed: decision tree learning, support vector machines, and k-means clustering. Only FPGA implementations of learning phase of the algorithms were studied. The reviewed implementations for all algorithms were analyzed, and several common characteristics of implementations were identified.

Chapter 4 (*Software implementations of C4.5 decision tree learning algorithm*) presents decision trees and decision tree learning algorithms in general terms, and in detail the C4.5 decision tree learning algorithm. Two implementations of the C4.5 algorithm were profiled: the original C4.5 Release 8, and its modified version EfficientC4.5 (EC4.5). Profiling revealed that the bulk of execution time for larger sets ($\geq$ 20,000 items) is spent in three

processes: frequency matrix computation for nominal attributes, finding the best split of the set for each nominal attribute, and splitting the input dataset into subsets. EC4.5 performs better than C4.5 due to optimized handling of numerical attributes, and is the better basis for development of the hybrid algorithm.

Chapter 5 (*Heterogeneous computer system for decision tree learning*) presents design and implementation of the FPGA coprocessor. A description of the targeted heterogeneous computer system is given, with emphasis on the FPGA platform used for implementation of the coprocessor. The platform and its development tools are designed to utilize the dataflow model of computation. Using the results of algorithm profiling, two kernels for the coprocessors were defined and implemented – *ComputeFreq*, and *GoupItems* kernels. The *ComputeFreq* kernel implements computation of frequency matrix for nominal attributes, and three variants were developed with varying degree of parallelism. Third variant (*ComputFreq MS2*), which is used in the coprocessor, calculates frequency matrices for six attributes in parallel, and uses parallelism to compensate for internal latencies which would otherwise limit performance. The *GroupItems* kernel implements splitting input set into subsets. The two kernels form the core of the coprocessor. Along with the kernels, the coprocessor contains communication links to the CPU's main memory over the PCIe bus, and a memory controller which provides access to FPGA attached DDR3 SDRAM.

Chapter 6 (*Hybrid algorithm for decision tree learning*) presents the hybrid algorithm for tree learning DF-DTC – Dataflow Decision Tree Construction. DF-DTC is based on a multithreaded implementation EC4.5, which was developed to test different approaches in parallelization of the algorithm, as well as to asses possible gains. Two parallelization strategies were implemented – distributed items, and distributed attributes – and performance for both strategies was evaluated. The distributed attributes strategy was shown to be the superior one, and was selected as basis for development of the hybrid algorithm. DF-DTC implements a subset of EC4.5's features. The most notable difference is that DF-DTC can work only with datasets which don't contain unknown values. The algorithm flow is somewhat modified to accommodate the initial data transfer from CPU to coprocessor, as well as utilization of coprocessor to compute frequency matrices. Computation of frequency matrices can be executed on coprocessor or on CPU, depending on size of the input subset. Also, the hybrid algorithm implements a procedure for dataset synchronization, which ensures that contents of subsets in main memory and in coprocessor memory match.

Chapter 7 (*Performance evaluation of the heterogeneous computer system and the hybrid algorithm*) presents the procedure and results of performance evaluation of the developed computer system and algorithm. The performance evaluation consists of two parts: evaluation of *ComputeFreq* kernel in isolation, and evaluation of DF-DTC executed on heterogeneous system. Evaluation of *ComputeFreq* kernel in isolation was carried out an all variants, and the results were compared to the multithreaded software implementation. Performance of the kernel is dependent on size of the dataset, with best performance achieved on largest datasets. Best speedup achieved is 14.7, while average speedup is 3.00. Evaluation of the DF-DTC is carried out in three parts: subset processing, number of items scaling, and number of attributes scaling. Performance was compared to the multihreaded EC4.5. Best total speedup is 1.18, achieved for datasets with 200 attributes and 2,000,000 items. Analysis of the evaluation results brought to light previously unaccounted for bottlenecks. Possible avenues for improving performance of the DF-DTC are discussed.

Chapter 8 (*Conclusion*) restates the significant results of the research, lays out scientific contributions, and proposes future work.

This thesis presents a novel heterogeneous computer system and hybrid algorithm for decision tree learning DF-DTC. The key process which is accelerated using the FPGA coprocessor is frequency matrix computation. The kernel of the coprocessor provides an average speedup 3.00 times compared to multithreaded software implementation. Total speedup of the DF-DTC is up to 1.18, compared to multithreaded EC4.5. Analysis of performance evaluation results shed light to causes of the lower than expected speedups, and guiding principles for improving the algorithm and coprocessor were derived from them. Development, implementation and performance evaluation of the coprocessor for decision tree learning has shown that FPGA is overall a suitable platform for acceleration of decision tree learning algorithms.

Keywords: *heterogeneous computer system, field programmable gate array, FPGA coprocessor, data mining, decision trees, C4.5*

Sadržaj

1 Uvod

Heterogeni računalni sustavi uz standardne procesore opće namjene sadrže i namjenski procesor za ubrzanje računski najintenzivnijih dijelova algoritma. Već je ustaljena primjena grafičkog procesora (GPU, engl. *graphics processing unit*) kao namjenskog procesora, a u posljednjih desetak godina javljaju se i komercijalno dostupni sustavi temeljeni na FPGA sklopovima. FPGA (engl. *field programmable gate array*, hrv. programirljivo polje logičkih elemenata) programirljivi je digitalni sklop koji služi za izvedbu korisnički definiranih digitalnih sustava [1]. Pomoću njih moguće je razviti procesorske jedinice optimalne za određeni zadatak i to svojstvo čini ih pogodnim za korištenje u računalnim sustavima, gdje dobivaju ulogu koprocesorske jedinice.

Područje primjene od interesa u istraživanju prikazanom u ovoj doktorskoj disertaciji jest dubinska analiza podataka [2]. Učenje klasifikatora jedna je od često korištenih metoda u dubinskoj analizi podataka. Algoritmi za učenje klasifikatora kao ulaz uzimaju skup primjera. Svaki primjer pripada jednoj od klasa, a opisan je određenim brojem atributa. Atributi mogu biti numerički ili nominalni, pri čemu numerički mogu poprimiti bilo koju numeričku vrijednost, dok nominalni imaju određeni konačni broj mogućih vrijednosti. Algoritam kao izlaz daje klasifikator koji predviđa kojoj klasi pripada pojedini primjer, a koristi se pri klasificiranju nepoznatih primjera.

Stabla odluke [3] vrsta su klasifikatora koja koristi stablo kao strukturu za opis modela klasifikacije. Čvorovi stabla određuju test atributa, grane odgovaraju mogućim ishodima testa, a listovi predstavljaju ishod klasifikacije. Nepoznatim primjerom se obilazi stablo, počevši od korijena, i slijede grane prema ishodima testa određenim čvorovima dok se ne dođe do lista koji daje klasifikaciju primjera.

Učenje stabla odluke jedan je od temeljnih postupaka strojnog učenja, kao i jedan od najvažnijih algoritama u dubinskoj analizi podataka [4]. Učenje stabla odluke je rekurzivni proces koji slijedi načelo „podijeli pa vladaj“. Nakon gradnje stabla, ono se obično obrezuje da bi se izbjeglo preučenje, a i smanjila veličina naučenog stabla. Poznatiji algoritmi za učenje stabla odluke su ID3 [5], CART [6], C4.5 [7]. Svi navedeni algoritmi u svojoj srži imaju isti rekurzivni proces učenja, a razlikuju se prvenstveno po nekoliko značajki: mjeri

kvalitete atributa, podržanim vrstama atributa, tolerantnosti na nedostajuće vrijednosti, algoritmu obrezivanja stabla itd.

Algoritam C4.5 identificiran je kao jedan od najvažnijih algoritama u dubinskoj analizi podataka [4]. Uz Quinlanovu originalnu implementaciju algoritma, dostupne su i poboljšana inačica Efficient C4.5 [8], C++ implementacija YaDT [9], [10], te Java implementacija J48, dostupna kao dio sustava za dubinsku analizu podataka Weka [11].

Popularnost učenja stabla odluke, kao i potreba za obradom sve većih skupova podataka dovela je i do razvoja paralelnih algoritama. Paralelni algoritmi namijenjeni su poglavito za rad s velikim skupovima podataka na distribuiranim računarskim sustavima, a ciljani su skupovi koji su preveliki za glavnu memoriju jednog računala. Značajniji paralelni algoritmi su SPRINT, ScalParC i PLANET [12]–[14]. SPRINT je paralelni algoritam za učenje stabla odluke temeljen na CART algoritmu. Pri izvršavanju na višeprocesorskom sustavu SPRINT skup za učenje dijeli na procesore tako da svaki procesor preuzme obradu pojedinih atributa svakog primjera. Takva podjela zahtijeva znatnu komunikaciju među procesorima budući da je potrebno svakom procesoru dostaviti tablicu koja pridjeljuje primjere u čvorove stabla. ScalParC koristi iste podatkovne strukture kao SPRINT, ali uz poboljšanja u komunikaciji. Svaki procesor drži dio tablice pridjeljivanja primjera u čvorove stabla i svaki procesor prosljeđuje svakom drugom procesoru samo dio tablice koji je relevantan. Smanjenjem komunikacije među procesorima postiže se znatno bolja skalabilnost nego kod SPRINT algoritma. PLANET je programski okvir temeljen na MapReduce programskoj arhitekturi za obradu velikih količina podataka, a pokazao je izuzetno dobru skalabilnost.

Prema dosad dostupnim podacima iz literature, izvedbe učenja stabla odluke pomoću FPGA koprocesora su malobrojne [15]. U [16] pomoću FPGA je izvedena jedinica za izračun Gini indeksa, dok se ostatak algoritma izvodi na CPU (engl. *central processing unit*). Izvedba je ograničena na binarne skupove podataka (samo dvije klase) i ograničene veličine. U [17] je izvedena prva cjelovita implementacija algoritma za učenje stabla na FPGA. Implementacija je pokazala dobre performanse, ali je ograničena na skupove koji stanu u unutarnju memoriju FPGA: HC-CART [18] noviji je cjeloviti pristup problemu učenja stabla odluke izvedenog s FPGA. HC-CART implementira CART algoritam za učenje stabla. U ovoj implementaciji rad s nominalnim atributima odvija se na FPGA koprocesoru, dok se numerički atributi obrađuju na CPU. Sustav je pokazao dobru skalabilnost, te veoma dobra ubrzanja za skupove podataka s većim brojem nominalnih atributa.

Algoritmi za učenje stabla odluke općenito su memorijski intenzivni, što znači da su njihove performanse u principu ograničene mogućnostima pristupa podacima, bilo u glavnoj memoriji računala, bilo iz vanjske pohrane podataka. Proces učenja stabla odluke ima mali omjer broja računskih operacija i broja pristupa memoriji. FPGA pokazuje visoku učinkovitost za algoritme koji se mogu prikazati kao tok podataka kroz duboku protočnu strukturu, tj. za algoritme koji imaju velik omjer broja računskih operacija i broja pristupa memoriji. Glavni izazov u izvedbi algoritma na FPGA je upravo u rješavanju problema pristupa podacima, pri čemu se nastoji minimizirati učestalost pristupanja memoriji, te pristup memoriji učiniti što efikasnijim.

U ovoj disertaciji je prikazan heterogeni računalni sustav s FPGA koprocesorom za učenje stabla odluke, zajedno s hibridnim algoritmom za učenje stabla odluke koji se izvršava na njemu. Provedena je dinamička analiza radnih značajki algoritma C4.5 za učenje stabla odluke, na kojem se temelji hibridni algoritam. Opisana je arhitektura i implementacija FPGA koprocesora, kao i hibridni algoritam za učenje stabla odluke koji je prilagođen arhitekturi koprocesora. Posebna pozornost posvećena je postupku vrednovanja performansi razvijenog heterogenog sustava sa slijednim i višedretvenim implementacijama.

Disertacija je podijeljena na osam poglavlja. U drugom poglavlju opisani su FPGA sklopovi, njihove značajke, temeljni sastavni dijelovi, način programiranja i općenito se govori o razvoju sustava temeljenog na FPGA-u. U trećem poglavlju dan je pregled FPGA implementacija algoritama za dubinsku analizu podataka, s naglaskom na algoritme za učenje stabla odluke. Također, dan je osvrt na najčešće značajke FPGA implementacija algoritama za dubinsku analizu podataka. U četvrtom poglavlju opisane su programske implementacije algoritma C4.5, te je opisan postupak i rezultati dinamičke analize dvije implementacije na kojoj se temelji hibridni algoritam.

U petom poglavlju opisana je arhitektura heterogenog računalnog sustava s programirljivim poljem logičkih elemenata za učenje stabla odluke. Dan je kratki uvod u Maxeler platformu pomoću koje je izveden FPGA koprocesor i opisan način na koji komunicira s CPU-om i programskim dijelom algoritma koji se izvodi na njemu. U šestom poglavlju opisan je hibridni algoritam za učenje stabla odluke i njegova realizacija, te način interakcije algoritma s FPGA koprocesorom.

U sedmom poglavlju opisan je postupak vrednovanja performansi heterogenog sustava i hibridnog algoritma. Dani su rezultati provedenog vrednovanja i uspoređeni su s

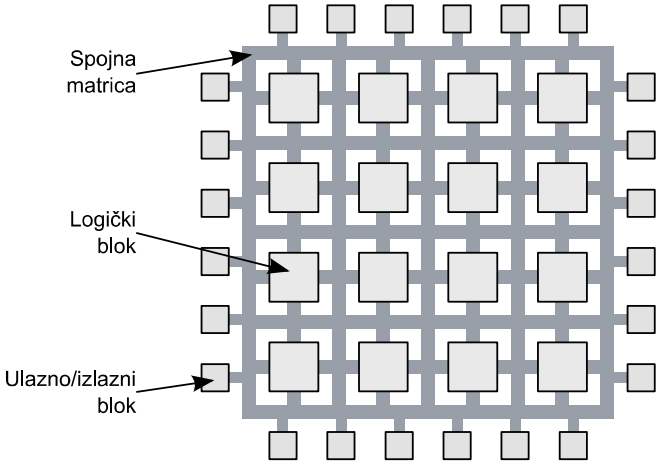
performansama postojećih programskih implementacija algoritma C4.5. U osmom poglavlju zaključen je rad, sažeto su prikazani znanstveni doprinosi i smjernice za budući rad.

2 Programirljiva polja logičkih elemenata – FPGA

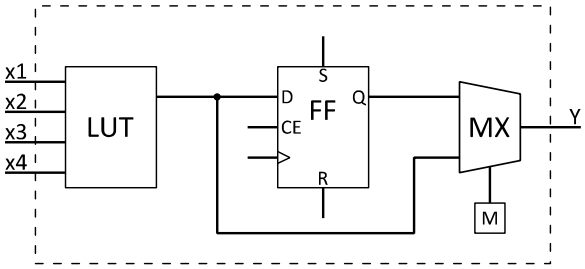
Programirljivo polje logičkih elemenata (FPGA, engl. *field programmable gate array*) vrsta je digitalnih integriranih sklopova namijenjenih konfiguraciji metodom programiranja od strane korisnika [1], [19]. Osnovne značajke FPGA-a su programirljivost i rekonfigurabilnost. Programirljivost označava mogućnost da korisnik definira željenu funkciju. Rekonfigurabilnost znači da se programirani FPGA može izbrisati i ponovno programirati. Obje značajke su vrlo bitne za primjenu u računarstvu. Programirljivost omogućuje realizaciju velikog broja različitih arhitektura, tako da je moguće napraviti specijaliziran sustav koji najefikasnije rješava zadani problem. Rekonfigurabilnost omogućuje promjenu funkcije FPGA-a kad god je to potrebno prema zahtjevima algoritma koji se želi izvršavati.

2.1 Arhitektura FPGA sklopova

Osnovni FPGA integrirani elektronički sklop, shematski prikazan na slici 2.1, sastavljen je od matrice memorijskih tablica, bistabila i multipleksora. Moderni FPGA sklopovi sadrže i dodatne specijalizirane elemente kao što su memorijski blokovi, množila i periferijske jedinice za brzu komunikaciju. Navedeni elementi grupirani su u blokove i obavijeni programirljivom mrežom spojnih vodova.



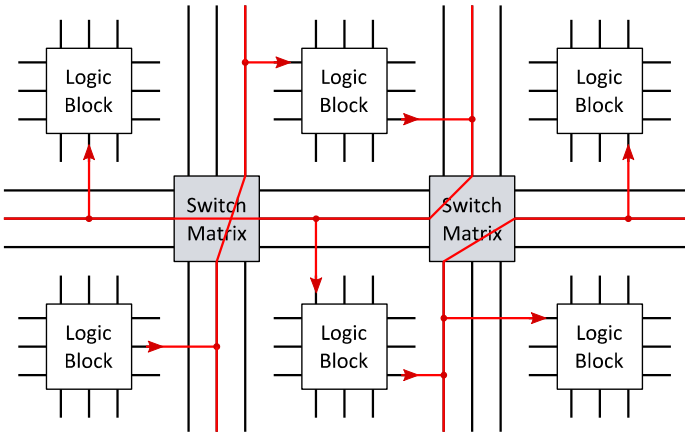
Slika 2.1. Osnovna arhitektura FPGA sklopa



Slika 2.2. Osnovna struktura jednostavnog logičkog bloka

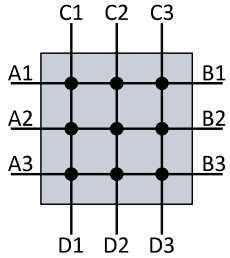
Temeljni element za izvedbu logičkih funkcija u FPGA-u je **logički blok**. Osnovna struktura jednostavnog logičkog bloka prikazana je na 2.2. Sastoji se od pregledne tablice, bistabila i multipleksora. **Pregledna tablica** (LUT, engl. *lookup table*) služi za realizaciju kombinacijskih logičkih funkcija. Ona je u osnovi memorija s brojem lokacija jednakim broju mogućih kombinacija ulaza, koja pohranjuje jedan bit po lokaciji. Kombinacijska funkcija realizira se tako da se za svaku moguću kombinaciju ulaza u tablicu upiše pripadajuća izlazna vrijednost. Moderni FPGA-ovi imaju naprednije izvedbe preglednih tablica koje omogućuju da se one koriste kao RAM, te kao posmačni registri. **Bistabil** (FF, engl. *flip-flop*) obično je D-tip bistabila okidan na signal takta, a namijenjen je izvedbi sekvencijalnih funkcija. Primjenjuje se uglavnom za izvedbu registara, te za pamćenje trenutnog stanja konačnog automata stanja. **Multipleksor** (MX, engl. *multiplexor*) u ovom logičkom bloku služi za izbor izlaza logičkog bloka. Postoji mogućnost odabira izlaza iz pregledne tablice ili izlaza iz bistabila. Odabir izlaza sudjeluje u definiciji funkcije bloka, a u ovom jednostavnom bloku definira izvodi li kombinatornu ili sekvencijalnu funkciju.

Mreža spojnih vodova, pobliže prikazana na slici 2.3, sastoji se od hijerarhije vodova

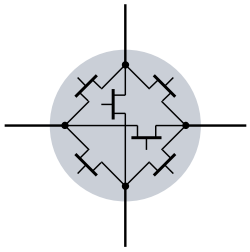


Slika 2.3. Mreža spojnih vodova u FPGA

različitih duljina kojima se međusobno spajaju pojedini blokovi i kroz koju su raspoređene spojne matrice. Različite duljine vodova omogućuju njihovo bolje iskorištenje na način da su bliži blokovi u pravilu spojeni kraćim vodovima, a dulji vodovi se koriste za međusobno udaljenije blokove. Posebna vrsta vodova su vodovi za provođenje signala takta. Posebnost signala takta jest da ga je potrebno razvesti na velik dio integriranog sklopa, što postavlja izazov s obzirom na energetske mogućnosti sklopova i zadovoljavanja zadanih vremenskih uvjeta rada sklopova [20] (vremena postavljanja i zadržavanja bistabila i sl.).



a) **spojna matrica**

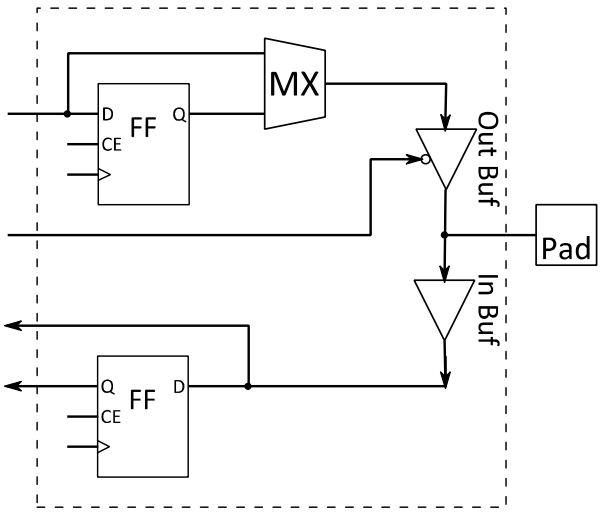


b) čvor spojne matrice

Slika 2.4. Spojna matrica

Uz mrežu spojnih vodova raspoređene su **spojne matrice**. Spojna matrica, prikazana na slici 2.4 a, služi za međusobno povezivanje različitih spojnih vodova. Pomoću nje se svaki spojni vod koji završava ili počinje u njoj može spojiti sa svakim drugim, i na taj način se programira usmjeravanje signala po integriranom sklopu. Spajanje se izvršava čvorovima matrice, prikazanim na slici 2.4 b, koja se sastoji od skupa tranzistora. Tranzistori rade u režimu sklopke i mogu četiri ulazne linije spojiti u različitim kombinacijama, ovisno o potrebama razvođenja signala po integriranom sklopu.

Ulazno-izlazni blokovi (IOB, engl. *input/output block*), u pravilu smješteni na periferiji FPGA integriranog sklopa, povezuju unutarnje sklopovlje FPGA-a sa signalima vanjskih digitalnih sustava. Osnovna struktura ulazno-izlaznog bloka, prikazana na slici 2.5, sastoji se od ulaznih i izlaznih pojačala, po jednog bistabila za svaki smjer signala, te multipleksora kojim se bira izravni izlaz ili izlaz preko bistabila. Izlazno pojačalo redovito je pojačalo s tri stanja i ima kontrolni ulaz koji ga postavlja u stanje visoke impendacije, te



Slika 2.5. Osnovna struktura ulazno-izlaznog bloka

omogućuje dvosmjerni režim signala. Bistabili u ulazno-izlaznom bloku najčešće služe kao registri za postavljanje izlaznih, odnosno prihvat ulaznih signala. Ulazno-izlazni sklopovi programirljivi su u smislu izbora smjera signala (ulazni, izlazni ili dvosmjerni), te u smislu naponskih razina signala. Moderni FPGA-ovi podržavaju i diferencijalnu signalizaciju na način da su određeni ulazno-izlazni blokovi upareni u diferencijalne parove.

Jedan od ključnih dijelova FPGA-a jest njegova **konfiguracijska memorija**. Konfiguracijska memorija pamti uprogramiranu konfiguraciju FPGA, tj. upravlja brojnim tranzistorskim sklopkama koje spajaju pojedine signalne linije, postavlja upravljačke signale za multipleksore koji usmjeravaju signale unutar programirljivih logičkih blokova itd. Postoje dvije glavne metode čuvanja konfiguracije:

- Statička memorija – SRAM. Ova metoda omogućuje brzo programiranje FPGA-a, što je velika prednost kod sustava za koje se predviđa češća promjena konfiguracija tijekom rada (npr. ispitni i razvoji sustavi). Nakon uključivanja FPGA je u neutralnom zadanom stanju u kojem ne izvršava nikakvu određenu funkciju, a svi programirljivi ulazi/izlazi u stanju su visoke impendancije. FPGA se nakon uključivanja mora programirati da bi počeo izvršavati neku funkciju, a to podrazumijeva korištenje dodatnog vanjskog sklopovlja i trajne memorije za pohranu konfiguracije. Nakon gubitka napajanja FPGA gubi uprogramiranu konfiguraciju.
- Električki programirljiva memorija – *Flash* memorija. Ova metoda omogućuje trajnu pohranu konfiguracije unutar FPGA-a, a time izvršavanje zadane funkcije odmah nakon uključenja napajanja. Korištenje *flash* memorije je prednost pri ugrađenim (engl. *embedded*) sustavima koji zahtijevaju kratak ciklus uključivanja i kod kojih se ne očekuje promjena konfiguracije za vrijeme rada. Dodatna prednost je i manji broj vanjskih komponenti nužnih za rad.

2.2 Razvoj sustava temeljenog na FPGA-u

Pri razvoju sustava temeljenog na FPGA-u, pri čemu se podrazumijeva definicija funkcije FPGA-a, uobičajeno se počinje s tekstualnim opisom sustava u nekom od jezika za opis sklopovlja (HDL, engl. *hardware description language*). Najzastupljeniji jezici su VHDL [21] i Verilog [22]. Osim kodiranja u nekom od jezika, postoje i programski paketi koji omogućuju definiciju sustava korištenjem ranije razvijenih blokova. Blokovi sadrže implementacije često korištenih funkcija, kao što su memorijski kontroleri, procesori, sabirnice, video kontroleri

itd. Tekstualni opis sustava nizom se koraka prevodi u oblik pogodan za programiranje FPGA-a:

- Logička sinteza (engl. *logic synthesis*) – prevodi tekstualni opis visoke razine (logički konstrukti i ponašajni model) u mrežu primitiva (engl. *netlist*). Mreža opisuje koji se FPGA resursi (LUT, FF, DSP, BRAM itd.) koriste i kako su međusobno povezani.
- Tehnološko mapiranje (engl. *technology mapping*) – grupira pojedine elemente (primitive) u FPGA logičke blokove, npr. određuje koji će LUT-ovi i FF-ovi biti zajedno u istom CLB-u. Proces proizvodi fizički opis sustava u smislu FPGA blokova i njihove konfiguracije.
- Raspoređivanje i trasiranje (engl. *place and route*) – raspoređuje grupe elemenata u točno određene logičke blokove u FPGA-u i određuje trase za signale koji ih povezuju, tj. konfiguraciju mreže spojnih vodova.
- Generiranje konfiguracije za programiranje (engl. *bitsream generation*) – stvara konfiguraciju FPGA-a u binarnom obliku spremnom za učitavanje u FPGA.

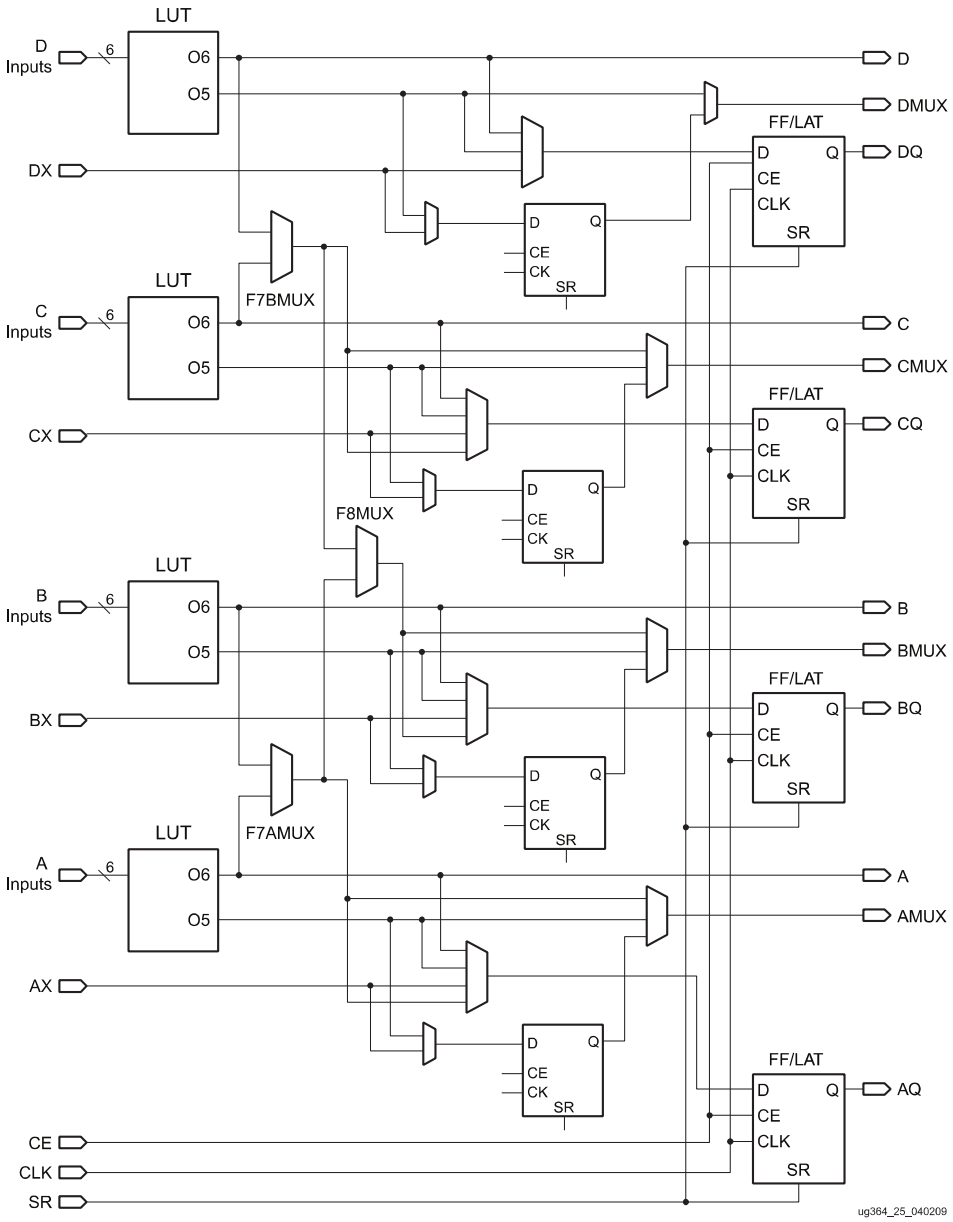
Navedeni koraci izvode se namjenskim programskim alatima, koje u pravilu osiguravaju proizvođači FPGA sklopova. Intervencije u te procese moguće su kroz podesive parametre programskih alata, te dodatnih zahtjeva i ograničenja (engl. *constraints*) koji se mogu postaviti u obliku posebno formatiranih tekstualnih opisa.

2.3 Xilinx Virtex-6 FPGA

Xilinx Virtex-6 FPGA [23] primjer je modernog FPGA sklopa. Virtex-6 obitelj obuhvaća niz modela koji se prvenstveno razlikuju po broju osnovnih blokova različitih funkcija. Virtex-6 mnogo je složeniji od osnovnog FPGA-a opisanog u poglavlju 2.1, a s obzirom da je FPGA koprocesor izveden pomoću FPGA-a iz te obitelji, ovdje će ukratko biti prikazane njegove glavne značajke.

Sa stanovišta korisnika, glavni funkcijski dijelovi su *Configurable Logic Block* (CLB), DSP48E1 blokovi, *Block RAM* i *mixed-mode clock manager* (MMCM). Uz njih FPGA sadrži i primopredajnike za brze ulazne-izlazne signale, te specijalizirane komunikacijske blokove za PCIe sučelje i za Ethernet sučelje.

Configurable Logic Block (CLB) [24] generički je programirljivi logički element i glavni gradivni element za realizaciju digitalnih logičkih funkcija u FPGA-u. CLB je podijeljen u dvije lamele (engl. *slice*). Pojednostavljena blok shema jedne lamele prikazana je na slici 2.6. Sadrži četiri šesteroulazne pregledne tablice, osam bistabila, skup multiplekosora, i pomoćnu logiku za realizaciju aritmetičkih funkcija (engl. *carry logic*). Dio lamela (30% do 40% od ukupnog broja, ovisno o modelu FPGA-a) podržava dva dodatna načina korištenja preglednih tablica: RAM (256 bitova) i posmačni registar (128 bitova).



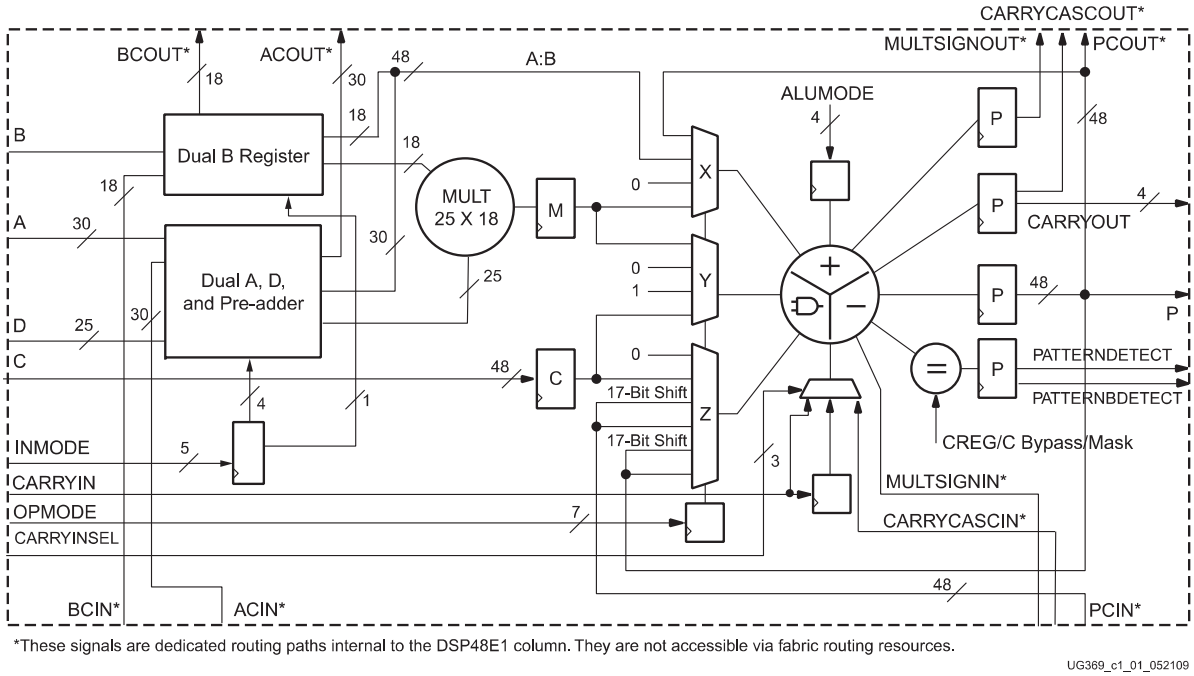
Slika 2.6. Pojednostavljena blok shema Xilinx Virtex-6 lamele
(prilagođeno prema [24])

DSP48E1 blok [25] specijalizirani je blok namijenjen realizaciji funkcija za digitalnu obradu signala. Arhitektura bloka prikazana je na slici 2.7. Kao glavni dijelovi mogu se istaknuti:

- množilo 25×18 bitova
- aritmetičko-logička jedinica (ALU, engl. *arithmetic logic unit*) koja može izvršavati sljedeće operacije: zbrajanje/oduzimanje s tri operanda, logičke operacije s dva operanda (AND, OR, NOT, NAND, NOR, XOR, XNOR)
- ulazni registri za množilo A (30 bitova, nižih 25 je ulaz u množilo) i B (18 bitova), ulazni registar za ALU (48 bitova), te izlazni registar P (48 bitova).

DSP48E1 blok omogućuje efikasnu izvedbu operacije množenje-akumulacija, pomoću koje se izvode temeljne operacije iz digitalne obrade signala kao što su konvolucija i skalarni produkt.

Block RAM [26] namjenski je memorijski blok koji može pohraniti 36 ki bitova. Jedan *block RAM* može biti korišten kao jedan blok od 36 kib, ili dva od 18 kib. Svaki blok od 36 kib može biti konfiguriran kao 64 kib×1 (u sprezi sa susjednim 36 kib blokom), 32 kib×1, 16 kib×2, 8 kib×4, 4 kib×9, 2 kib×18, 1 kib×36 ili 512 b×72 memorija. Svaki blok od 18 kib može biti konfiguriran kao 16 kib×2, 8 kib×4, 4 kib×9, 2 kib×18, 1 kib×36 ili 512 b×72

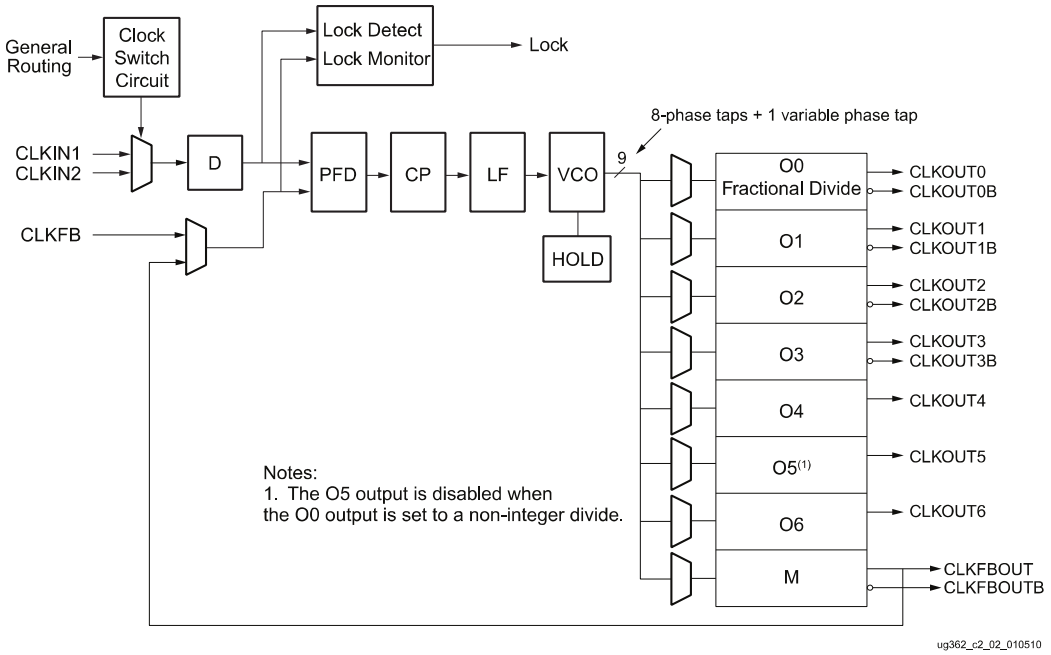


Slika 2.7. Blok shema DSP48E1 bloka
(prilagođeno prema [25])

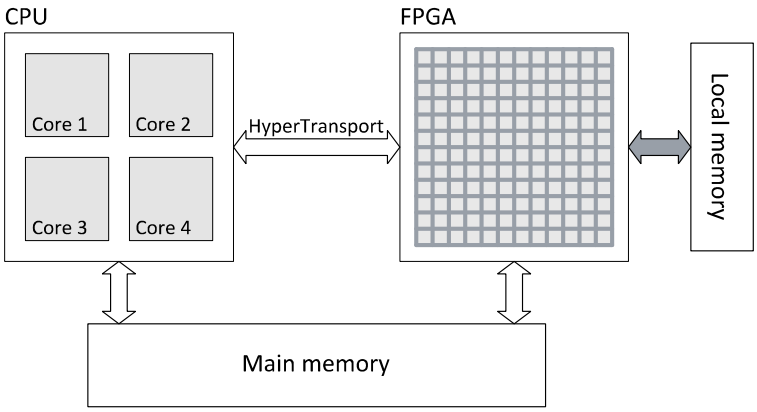
memorija. Blok podržava korištenje pariteta, konfiguriranje u memoriju s različitim širinama podatkovne sabirnice na ulazu i izlazu memorije, te rad u asinkronim domenama takta gdje sučelje za pisanje i sučelje za čitanje memorije rade na različitim frekvencijama signala takta.

Mixed-mode clock manager (MMCM) [27], prikazan na slici 2.8, je PLL (engl. *phase locked loop*) sklop koji omogućuje generiranje unutarnjeg signala takta željene frekvencije i faznog pomaka uz korištenje vanjskog referentnog signala takta. Koristi se za sintezu signala željene frekvencije, filtriranje slučajnih promjena faze signala takta (engl. *clock jitter*) i korekciju razlike u fazama signala takta u različitim dijelova sklopa (engl. *deskew*).

Virtex-6 FPGA sadrži hijerarhiju resursa namijenjenu upravljanju i razvođenju signala takta. Virtex-6 sadrži 32 globalna voda za razvođenje signala takta, koji mogu dovesti signal takta do svakog pojedinog resursa u FPGA-u. FPGA je podijeljen u nekoliko regija takta, gdje broj regija ovisi o modelu i veličini FPGA-a. Svaka regija ima šest stabala za razvođenje signala takta i osam diferencijalnih ulaza (pojačala) za signal takta. Navedeni resursi omogućuju dobru raspodjelu signala takta po FPGA sklopu, sa što manjim vremenskim razlikama među pojedinim dijelovima sklopa, te podjelu sklopa na više dijelova koji svaki radi na svom signalu takta.



Slika 2.8. Blok shema MMCM bloka
(prilagođeno prema [27])

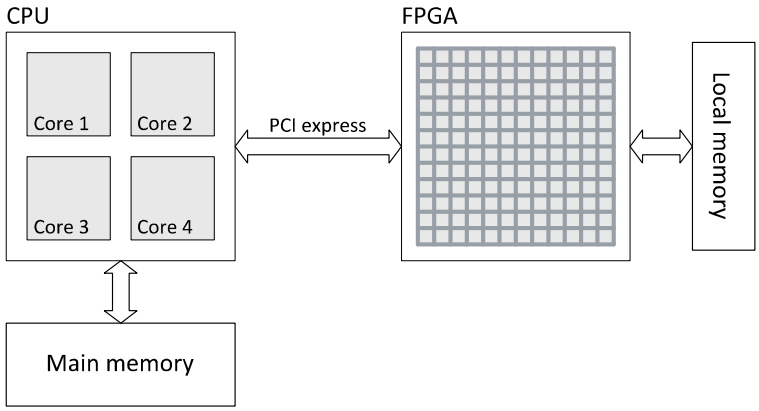


Slika 2.9. FPGA kao koprocesor

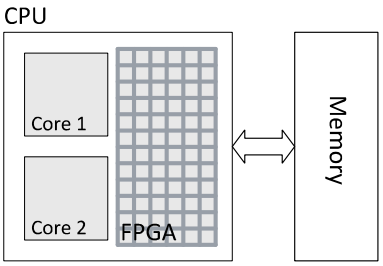
2.4 Mogućnosti primjene FPGA u heterogenim računalnim sustavima

Heterogeni računalni sustav s FPGA sklopom može se realizirati na četiri osnovna načina:

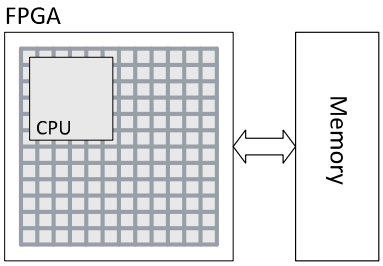
- FPGA kao **koprocesor** (slika 2.9) – spojen je na lokalnu sabirnicu procesora (*HyperTransport*, *Front Side Bus* ili *QuickPath Interconnect*) i ima izravni pristup glavnoj memoriji računala. Ovaj pristup omogućuje izuzetno brzu komunikaciju FPGA-a s procesorom i glavnom memorijom.
- FPGA kao **periferna procesna jedinica** (slika 2.10) – na sustav se spaja preko PCIe sabirnice i sva komunikacija s procesorom i glavnom memorijom odvija se preko nje. Ovo je najlakši način da se u standardno PC računalo ugradi FPGA i stoga najčešće korišten.
- FPGA kao **dio procesora** (slika 2.11) – služi kao proširenje procesora, ima izravnu vezu s jezgrama procesora putem unutarnje sabirnice i s njima dijeli pristup glavnoj



Slika 2.10. FPGA kao periferna procesna jedinica



Slika 2.11. FPGA kao dio procesora



Slika 2.12. FPGA kao temelj sustava

memoriji. FPGA omogućuje realizaciju funkcija prema potrebama korisnika umjesto uobičajenog korištenja nepromjenjivog skupa funkcija.

- FPGA kao **temelj sustava** (slika 2.12) – procesor i ostali potrebni dijelovi izvedeni su unutar FPGA-a. Procesor može biti prisutan kao posebni blok (engl. *hard core*) ili izveden programirljivim logičkim blokovima (engl. *soft core*). Ovaj pristup koristi se za izvedbu ugrađenih računalnih sustava.

3 FPGA implementacije algoritama za dubinsku analizu podataka

Opći proces dubinske analize podataka može se podijeliti na dvije faze: učenje modela i primjena modela. Učenje modela uključuje obradu skupa za učenje odabranim algoritmom, validaciju naučenog modela i ispitivanja performansi. Rezultati validacije koriste se za optimiranje parametara učenja modela i proces se može iterativno ponavljati dok se ne postignu zadovoljavajuće performanse modela, npr. postignuta točnost klasifikacije ili regresije. U drugoj fazi – primjeni – naučeni model se koristi. Način korištenja ovisi o vrsti modela, npr. stablo odluke može se koristiti za klasifikaciju novih podataka, a i za stjecanje uvida u značajke sustava iz kojeg su podaci dobiveni.

FPGA implementacije algoritama za dubinsku analizu podataka uglavnom su usredotočene na drugu fazu – na primjenu naučenog modela. Npr. razrađene su razne arhitekture za implementaciju stabla odluke na FPGA [28]–[32], strojeva s potpornim vektorima [33]–[36], a neuronske mreže imaju posebno bogatu povijest implementacija u FPGA [37]–[42]

U ovom poglavlju dan je pregled trenutnog stanja polja upotrebe FPGA-a kao koprocesora za ubrzanje izvršavanja algoritama za dubinsku obradu podataka [15]. U dubinskoj analizi podataka postoji i koristi se velik broj različitih algoritama. Wu i sur. su u [4] izdvojili deset najvažnijih i najviše korištenih. S obzirom na širinu područja i brojnost algoritama za dubinsku analizu podataka, napravljen je pregled za tri odabrana algoritma: stabla odluke (engl. *decision tree*), stroj s potpornim vektorima (engl. *support vector machine*) i metoda k-srednjih vrijednosti (engl. *k-means clustering*). Pregled je usredotočen isključivo na FPGA izvedbe faze učenja modela.

3.1 Stabla odluke

Stabla odluke su klasifikatori koji koriste stablo kao strukturu za opis modela klasifikacije. Učenje stabla odluke je rekurzivni proces u kojem se bira atribut prema kojem se dobije najbolja podjela skupa na podskupove. Učenje stabla odluke detaljnije je opisano u 4. poglavlju.

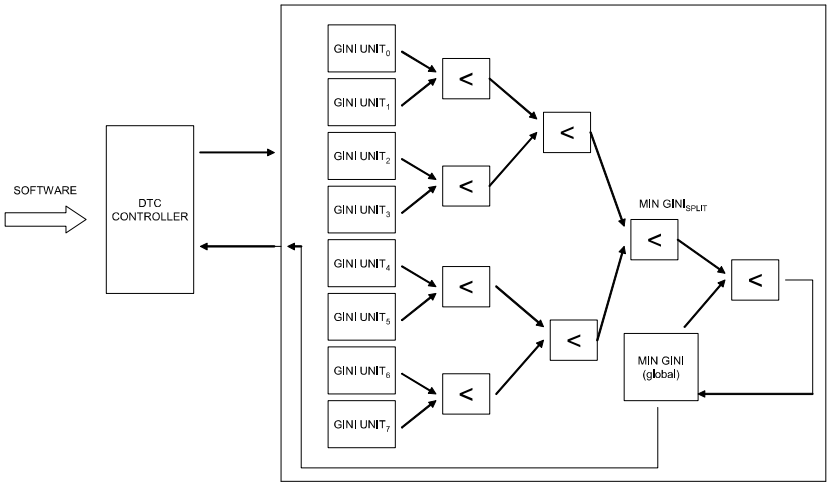
3.1.1 ScalParC algoritam

Narayanan i sur. (2007) [16] predstavljaju prvu implementaciju učenja stabla odluke izvedenu pomoću FPGA sklopa. Implementirali su ScalParC [13] algoritam, a u FPGA-u su izveli izračun Gini indeksa kao računski najintenzivniji dio algoritma, te nalaženje njegovog minimuma.

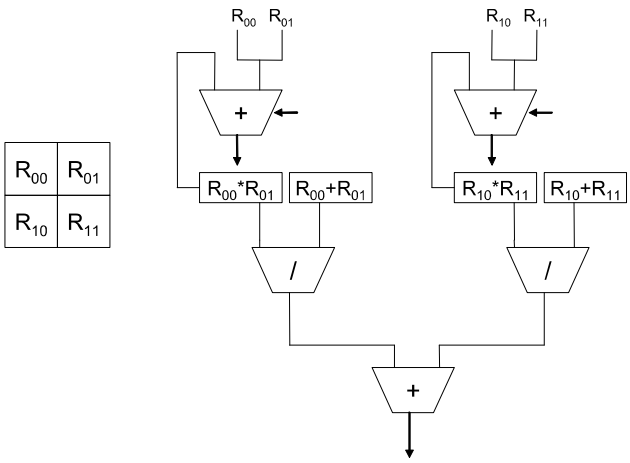
U sklopovlju je izveden izračun Gini indeksa samo za numeričke atribute. Procjena autora je da za nominalne atribute, zbog ograničenog broja različitih vrijednosti atributa, nema značajne dobiti od ubrzavanja izračuna Gini indeksa. Arhitektura izvedena u FPGA-u, prikazana na slici 3.1, omogućuje izračun Gini indeksa za više atributa istovremeno. Ona se sastoji od više „Gini jedinica", na slici označene kao *Gini unit*$_{0\text{-}7}$, od kojih svaka izračunava Gini indeks za po jedan atribut. Izračunati Gini indeksi međusobno se uspoređuju pomoću stabla komparatora, koje na izlazu daje najmanji Gini indeks. Izlaz iz stabla komparatora uspoređuje se s pohranjenim globalno najmanjim Gini indeksom, i manji od njih se pohranjuje kao novi globalno najmanji. Upravljačka jedinica, na slici označena kao *DTC controller*, ima ulogu sučelja između CPU-a i Gini jedinica. Ona podatke dobivene od CPU-a prosljeđuje Gini jedinicama, te CPU-u prosljeđuje vrijednost globalnog minimuma Gini indeksa, te indeks mjesta podjele skupa.

Arhitektura u FPGA-u razvijena je prema sljedećim pretpostavkama:

- skup sadrži dvije klase (binarni skup),



Slika 3.1. Arhitektura za paralelni izračun Gini indeksa za više atributa
(prilagođeno prema [16])



Slika 3.2. Arhitektura jedinice za izračun Gini indeksa
(prilagođeno prema [16])

- podaci se dijele na dva podskupa (gradi se binarno stablo).

Prema tim pretpostavkama izračun Gini indeksa je pojednostavljen s ciljem da se smanji broj aritmetičkih operacija koje se izvode u FPGA-u. Krajnja svrha je minimizacija pojedine Gini jedinice kako bi se moglo što više njih smjestiti u slobodne resurse u FPGA-u. S obzirom na navedene pretpostavke, te činjenicu da za ispravan rad algoritma nisu nužne točne vrijednosti Gini indeksa već međusobni odnos njihovih vrijednosti za pojedine atribute i podjele skupa, izračun Gini indeksa sveden je s:

$$G_i = 1 - \sum_{j=0}^{1} \frac{R_{ij}}{R_i} \tag{3.1}$$

$$G_T = \sum_{i=0}^{1} \frac{R_i}{R} G_i \tag{3.2}$$

na:

$$G'_T = \frac{R_{00}R_{01}}{R_{00} + R_{01}} + \frac{R_{10}R_{11}}{R_{10} + R_{11}} \tag{3.3}$$

gdje je R ukupni broj primjera u skupu, R_i broj primjera u podskupu i, a R_{ij} broj primjera klase j u podskupu i. Konačna arhitektura Gini jedinice, prikazana na slici 3.2, koristi aritmetičke jedinice samo za zbrajanje i dijeljenje. Množenje nije potrebno jer se broj primjera određen u podskupu R_{ij} povećava ili smanjuje samo za jedan element po iteraciji, pa se točni iznos umnoška dobije jednom operacijom zbrajanja, odnosno oduzimanja.

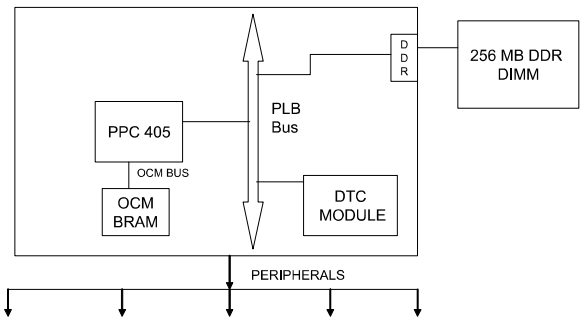
S ciljem povećanja efikasnosti komunikacije između CPU-a i DTC modula, u DTC se ne šalju vrijednosti atributa, već samo oznake klasa kojoj pripada primjer. Oznaka klase može poprimiti samo dvije vrijednosti, pa se kodira samo jednim bitom. Na taj se način istovremeno može prenijeti oznaka klase za više atributa. Kako bi vrijednosti položaja podjele skupa bile konzistentne, oznake klasa za svaki atribut sortiraju se prema vrijednosti atributa. U programu je implementiran dodatni korak koji na taj način priprema „bit-mape“ koje se šalju DTC modulu, a iz kojih on izračunava vrijednosti Gini indeksa.

Arhitektura je implementirana na Xilinx Virtex-II platformi. Ispitni sustav, prikazan na slici 3.3, sastoji se od FPGA-a i DDR RAM-a. FPGA sadrži ugrađeni PowerPC 405 CPU koji izvršava program, te DTC modul koji je izveden kao periferna jedinica koja s CPU-om komunicira putem PLB sabirnice. Skup za učenje pohranjen je u vanjskom DDR RAM-u.

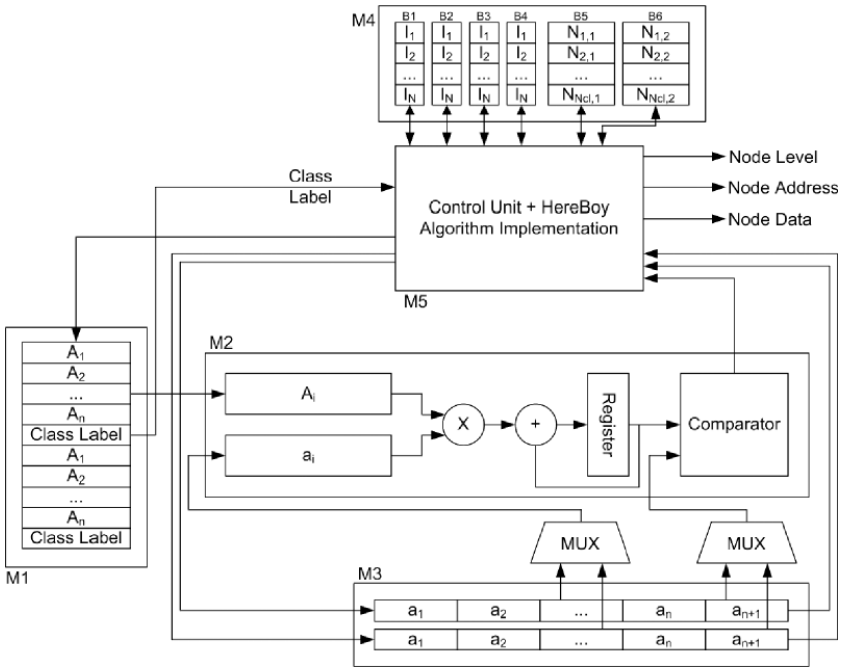
Pri ispitivanju je uspoređen broj ciklusa takta CPU-a potreban za provođenje procesa izračuna Gini indeksa pomoću DTC modula i bez njega. U oba slučaja program se izvodi na istom sustavu, s jedinom razlikom da je bez DTC modula cijeli proces izračuna Gini indeksa proveden na CPU-u. Implementirana arhitektura postiže ubrzanje izračuna Gini indeksa do 5,58 puta sa 16 Gini jedinica. Ubrzanje cijelog procesa učenja stabla pod istim uvjetima procijenjeno je na 1,5 puta.

3.1.2 Učenje linearnih stabla odluke

Struharik i Novak (2009) [17] prvi objavljuju cjelovitu FPGA implementaciju učenja stabla odluke. U svom radu opisuju arhitekturu za učenje linearnih (engl. *oblique*) stabala odluke [43], i to pojedinačnog stabla, te ansambla stabla. Kod linearnog stabla odluke test atributa uključuje linearnu kombinaciju više atributa. Test atributa je oblika:



Slika 3.3. Shematski prikaz ispitnog sustava
(prilagođeno prema [16])

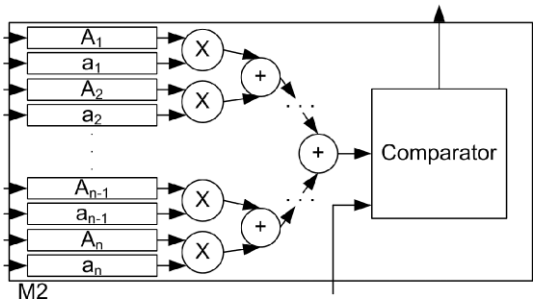


Slika 3.4. H_DTS: Arhitektura FPGA implementacije algoritma za učenje pojedinačnog stabla sa sekvencijalnom izvedbom modula M2
(prilagođeno prema [17])

$$f(A_1, A_2, \ldots, A_n) = \sum_{i=1}^{n} a_i A_i + a_{n+1} \tag{3.4}$$

gdje su a_i koeficijenti, a A_i vrijednosti atributa i. Koeficijenti testa određuju hiperravninu u prostoru koji razapinju atributi. U FPGA-u je implementiran algoritam za učenje stabla odluke koji za nalaženje optimalne hiperravnine u svakom čvoru koristi evolucijski algoritam HereBoy [44], koji je kombinacija elemenata genetskih algoritama i simuliranog kaljenja.

Arhitektura FPGA implementacije učenja pojedinačnog stabla – H_DTS – prikazana je na slici 3.4. Sastoji se od pet glavnih modula, na slici označenih s M1…M5.

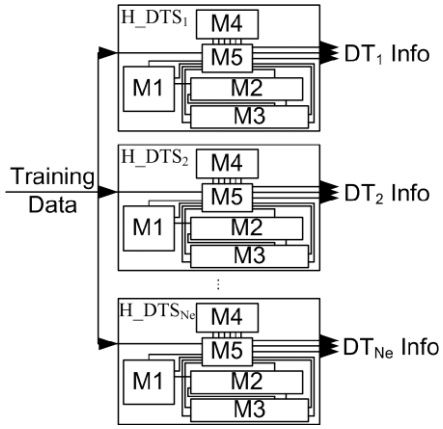


Slika 3.5. Arhitektura paralelne izvedbe modula M2
(prilagođeno prema [17])

- Modul M1 je pohrana skupa za učenje. Svaki primjer u skupu je pohranjen kao vektor od *n* atributa i jedne oznake klase.
- Modul M2 implementira izračun položaja primjera u odnosu na kandidatnu hiperravninu, a izvodi operaciju skalarnog produkta. U radu se koriste dvije izvedbe: slijedna i paralelna. Slijedna izvedba, prikazana na slici 3.4, izvodi množenja slijedno i istovremeno rezultat množenja akumulira u međuzbroj, pomoću klasične operacije množenje-akumulacija (MAC – *multiply-accumulate*). Paralelna izvedba, prikazana na slici 3.5, istovremeno izvodi sva množenja, a zbrajanje se izvodi pomoću stabla zbrajala. Izračunati položaj koristi se pri izračunu mjere kvalitete podjele skupa.
- Modul M3 pohranjuje koeficijente najbolje i trenutne kandidatne hiperravnine.
- Modul M4 sastoji se od šest memorija, od kojih četiri pohranjuju indekse primjera u čvorovima na trenutnoj i sljedećoj razini stabla, sa svrhom praćenja koji primjer pripada kojem čvoru, odnosno listu. Dvije preostale memorije pohranjuju broj primjera svake klase koji se nalaze s jedne, odnosno druge strane kandidatne hiperravnine, a koriste se za izračun mjere kvalitete podjele skupa.
- Modul M5 je upravljačka jedinica koja sadrži automat stanja koji implementira HereBoy algoritam.

FPGA implementacija učenja ansambla stabala prikazana je na slici 3.6. Sastoji se od više H_DTS jedinica koje istovremeno izvršavaju algoritam učenja stabla. Svaki pojedina H_DTS jedinica prima vlastiti skup za učenje dobiven uzorkovanjem s ponavljanjem iz cijelog skupa i na njoj izvršava algoritam za učenje pojedinačnog stabla.

Performanse FPGA izvedbe algoritma ocijenjene su na 29 skupova iz UCI repozitorija [45]



Slika 3.6. H_DTE: Arhitektura FPGA implementacije algoritma za učenje ansambla stabala
(prilagođeno prema [17])

Skupovi podataka imali su do 5000 primjera i do 40 atributa. Performanse su uspoređene s programskom implementacijom, napisanom u programskom jeziku C, a koja se izvodila na osobnom računalu s Intel Pentium D 820 CPU-om, frekvencije takta 2,8 GHz, i 3 GiB radne memorije. Sklopovske realizacije algoritma izvedene su na Xilinx Virtex-5 FPGA-u. Na svakom skupu podataka izvedena je desetostruka unakrsna validacija (provjera). Pri svakom izvršavanju validacije naučeno je jedno pojedinačno linearno stablo, te ansambl od 30 linearnih stabala. Za svaku implementaciju mjereno je prosječno vrijeme učenja, koje ne uključuje dodatne vremenske troškove, npr. vrijeme utrošeno na učitavanje podataka. Iz izmjerenih vremena izračunata su procijenjena ubrzanja sklopovskih implementacija u usporedbi s programskom implementacijom.

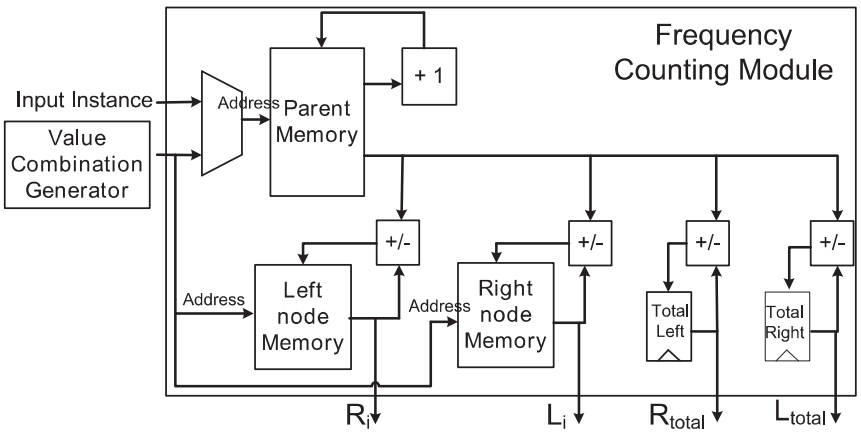
Sklopovska implementacija učenja pojedinačnog stabla sa slijednom izvedbom skalarnog produkta H_DTS1 u prosjeku je za faktor 2,3 puta brža od programske implementacije, uz frekvenciju takta FPGA-a 122,06 MHz. Implementacija s paralelnom izvedbom skalarnog produkta H_DTS2 brža je za faktor 25,53, uz frekvenciju takta FPGA-a 113,41 MHz. Sklopovska implementacija učenja ansambla stabala H_DTE1, sa slijednim skalarnim produktom, brža je za faktor 73,28, s frekvencijom takta 122,06 MHz. Implementacija učenja ansambla stabala H_DTE2, s paralelnim skalarnim produktom, brža je za faktor 692,63, s frekvencijom takta 113,4 MHz.

3.1.3 CART algoritam

Chrysos i sur. (2013) [18] objavljuju implementaciju CART algoritma za učenje stabla odluke na heterogenom računalnom sustavu, koju nazivaju HC-CART.

Autori za izvedbu arhitekture u FPGA-u koriste komercijalnu platformu Convey HC-1 [46]. Platforma se sastoji od računala s četverojezgrenim Intel Xeon CPU-om u koju je ugrađen Convey HC-1 koprocesor. Koprocesor sadrži 4 Xilinx Virtex-5 LX330 FPGA-ova, koji su preko prespojnika spojeni na ukupno osam memorijskih kontrolera, tako da svaki FPGA može pristupiti svakom memorijskom kontroleru.

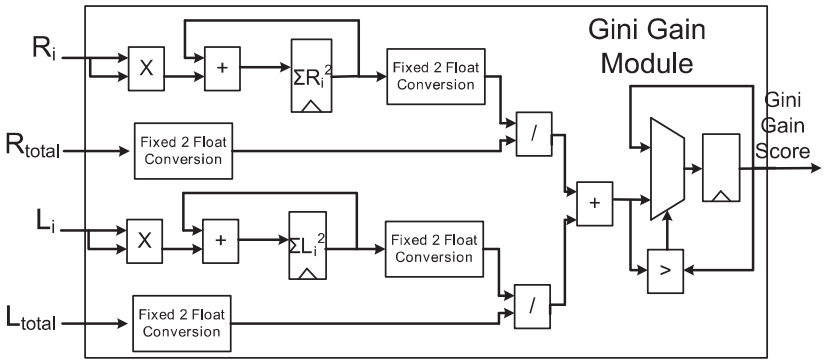
Pomoću FPGA-a izveden je izračun Gini indeksa za nominalne atribute. Autori kao ključnu jedinicu definiraju „CART modul“, kojim se izvodi postupak za izračun Gini indeksa za svaku moguću podjelu ulaznog skupa po određenom atributu. Ulazni skup je podijeljen u parove atribut-klasa, koji su ulaz u CART modul. CART modul se sastoji od *Frequency Counting* modula, *Gini Gain* modula, adresnog generatora i upravljačke logike.



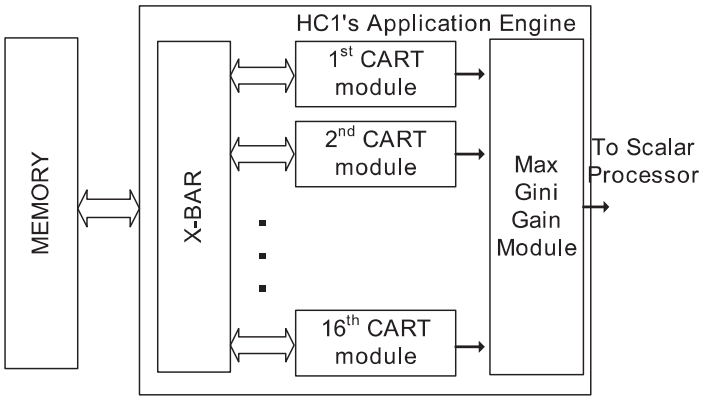
Slika 3.7. „Frequency Counting“ modul
(prilagođeno prema [18])

Frequency Counting modul, prikazan na slici 3.7, sadrži memorije u kojima se pohranjuju tablice frekvencija od čvora roditelja, te lijevog i desnog čvora. Veličina memorije određena je pod pretpostavkom da ulazni skup podataka ima do 64 jedinstvene vrijednosti atributa, i da ima do 64 različitih klasa, što daje da svaka memorija ima 4096 lokacija. Modul na ulaz prima parove atribut-klasa i inicijalno se popuni tablica frekvencija za čvor roditelj (engl. *parent memory*) i ta tablica kopira se u memoriju lijevog čvora (nema kriterija podjele skupa). Potom se generiraju sve moguće podjele skupa i koordiniranim čitanjem/pisanjem i zbrajanjem/oduzimanjem vrijednosti iz memorije čvora roditelja, te memorija lijevog i desnog čvora, dobivaju vrijednosti R_i, L_i, R_{total}, L_{total} koje se koriste za izračun Gini indeksa.

Gini Gain modul, prikazan na slici 3.8, prima vrijednosti koje daje *Frequency Counting* modul, i iz njih izračunava iznos Gini indeksa. Modul ostvaruje protočnu strukturu koja izračunava Gini indeks prema formuli (3.5), te logiku za pamćenje najvećeg Gini indeksa.



Slika 3.8. Arhitektura „Gini Gain“ modula
(prilagođeno prema [18])



Slika 3.9. Arhitektura HC-CART implementirana na FPGA
(prilagođeno prema [18])

$$GiniGain = \frac{\sum_{i=0}^{i<num\_value} R_i^2}{R_{total}} + \frac{\sum_{i=0}^{i<num\_value} L_i^2}{L_{total}} \tag{3.5}$$

Modul koristi aritmetiku fiksnog zareza, do trenutka kad treba izvršiti dijeljenje, koje je izvedeno u aritmetici pomičnog zareza. Upravljačka logika koordinira rad modula.

U svaki FPGA instancirano je 16 CART modula (v. sliku 3.9), izlaz iz svakog od njih spojen je na *Max Gini Gain* modul koji prati najveći Gini indeks među svim CART modulima. Unutarnja struktura *Max Gin Gain* modula jest stablo komparatora, slično onom iz [16]. HC-1 platforma sadrži četiri FPGA-ova, što daje ukupno 64 CART modula. To sustavu omogućuje da obrađuje do 64 nominalna atributa istovremeno. Najveći Gini indeks iz svakog od četiri FPGA-a šalje se u skalarni procesor (engl. *scalar processor*), koji CPU-u vraća konačnu najveću vrijednost Gini indeksa za 64 obrađena atributa.

HC-CART sustav integriran je u R programski paket za statistiku [47], tj. *rpart* programsku biblioteku koja je njezin dio, a implementira CART algoritam. Kada je *rpart* pokrenut, pozivom funkcija u njemu pokreće se izvršavanje HC-CART sustava. Izvršavanje na HC-CART pokreće se u svakom čvoru i iz njega se dobije najveći Gini indeks i pripadajućem atributu, te se te informacije koriste pri gradnji stabla. HC-CART u sklopovlju izvršava obradu jedino nominalnih atributa. Obrada numeričkih atributa odvija se u programu izvršavanom na CPU-u istovremeno s obradom numeričkih atributa na FPGA-u.

Performanse sustava ispitivane su na više skupova podataka, koji su sadržavali i numeričke i nominalne atribute. Ispitni skupovi podataka imali su od 8 to 128 atributa, sa 4 to 20 jedinstvenih vrijednosti po nominalnom atributu i veličine od nekoliko kiB do nekoliko GiB.

Ispitni Convey HC-1 sustav sadrži 4-jezgreni Intel Xeon CPU, frekvencije takta 2,13 GHz i 32 GiB RAM. Programske implementacije CART algoritma – WEKA SimpleCART i izvorna *rpart* biblioteka – izvršavane su na CPU-u, kao i programski dio HC-CART implementacije algoritma. Provedena su ispitivanja performansi s obzirom na broj primjera u ulaznom skupu, na broj atributa u ulaznom skupu, te na broj jedinstvenih vrijednosti atributa.

Pri ispitivanju skaliranja veličine skupa (od 500 do 1.000.000 primjera), HC-CART je u usporedbi s WEKA SimpleCART brži 7,2 do 130 puta, s većim ubrzanjem pri većim skupovima podataka. U usporedbi s *rpart* dobiveno je ubrzanje od 2,6 puta do 32 puta, ali sa smanjenjem ubrzanja s povećanjem skupa podataka. Autori uzrok takvom rezultatu nalaze u trošku prijenosa podataka iz memorije računala u memoriju HC-1 sustava. Pri ispitivanju skaliranja broja atributa (od 8 do 128 atributa), HC-CART je u usporedbi sa SimpleCART brži 52 puta do 270 puta, a u usporedbi s *rpart* 2,4 puta do 9,1 puta. U oba slučaja ubrzanje raste s brojem atributa. Pri ispitivanju skaliranja broja mogućih vrijednosti atributa (od 4 do 20), HC-CART je u usporedbi sa SimpleCART brži 52 puta do 210 puta, a u usporedbi sa *rpart* 1,4 puta do 53 puta. U oba slučaja ubrzanje raste s brojem mogućih vrijednosti.

Uz navedene jednodretvene programske implementacije, provedena je i usporedba sa višedretvenom implementacijom – MATALAB *classregtree*. U ovom slučaju provedeno je ispitivanje skaliranja broja primjera i skaliranja broja atributa. Dobiveno je ubrzanje od oko dva puta.

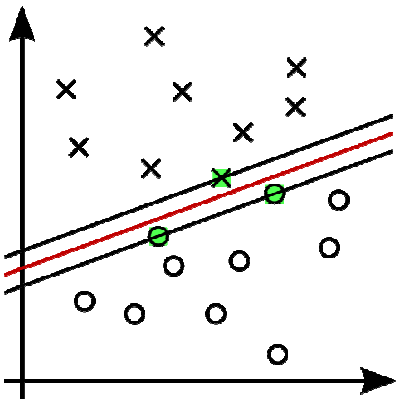
3.2 Ostali algoritmi za dubinsku analizu podataka

3.2.1 Strojevi s potpornim vektorima

Stroj s potpornim vektorima (SVM, engl. *support vector machine*) [48], [49] jedna je od najviše korištenih i najuspješnijih metoda za klasifikaciju. Ideja metode je određivanje klasifikacijske funkcije koja razlikuje točke dvije klase iz skupa za učenje $\{\boldsymbol{x}_i, y_i\}$, gdje je $\boldsymbol{x}_i \in \mathbb{R}^n$ i $y_i \in \{-1,1\}$, nalaženjem hiperravnine koja ih najbolje razdvaja. Klasifikacijska funkcija je oblika:

$$f(\boldsymbol{t}) = \text{sign}\left(\sum_i \alpha_i y_i K(\boldsymbol{t}, \boldsymbol{x}_i) + b\right) \tag{3.6}$$

gdje su: $\boldsymbol{t}$ točka koji želimo klasificirati, α_i pozitivne realne konstante, b realna konstanta, a $K(\boldsymbol{t}, \boldsymbol{x_i})$ je jezgrena funkcija. Jezgrena funkcija služi za preslikavanje točaka u prostor u



Slika 3.10. SVM – primjer linearno separabilnog problema u dvodimenzionalnom prostoru

kojem su linearno separabilne. Često korištene jezgrene funkcije su linearna (3.7), polinomna (3.8) i Gaussova (3.9).

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \tag{3.7}$$

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i \cdot \boldsymbol{x}_j)^p \tag{3.8}$$

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0 \tag{3.9}$$

Hiperravnina koja najbolje razdvaja skupove je ona koja maksimira marginu, tj. ima najveću udaljenost od hiperravnine do najbližih točaka iz oba skupa. Za skupove koji se ne mogu potpuno razdvojiti uvedena je meka margina (engl. *soft margin*) kojom se dopušta da neke točke budu s „krive“ strane hiperravnine. Točke koje su najbliže hiperravnini s maksimalnom marginom nazivaju se potporni vektori (engl. *support vectors*) i one definiraju dobiveni klasifikator. Na slici 3.10 ilustriran je primjer primjene SVM na skupu u dvodimenzionalnom prostoru. Hiperravnina je označena crvenom crtom, granice margine označene crnim crtama, a potporni vektori označeni su zelenom bojom.

Nalaženje parametara α_i koji opisuju SVM klasifikator je kvadratni optimizacijski problem:

$$\min_{\alpha} \ \frac{1}{2}\sum_{i,j} \alpha_i q_{ij} \alpha_j - \sum_i \alpha_i \tag{3.10}$$

pri čemu mora vrijediti:

$$0 \leq \alpha_i \leq C, \qquad \sum_i y_i \alpha_i = 0 \tag{3.11}$$

gdje su $q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, a C je proizvoljna konstanta. Postoji nekoliko algoritama za učenje SVM, npr. sekvencijalna optimizacija (engl. *sequential minimal optimization, SMO*) [50] i gradijentna metoda (engl. *gradient projection*) [51].

Najveći dio vremena izvršavanja algoritma za učenje odnosi se na postupak množenja matrice i vektora, pogotovo jer matrica može biti jako velikih dimenzija.

Anguita i sur. (2003) [52] predložili su vjerojatno prvu arhitekturu za učenje SVM klasifikatora. Prikazali su sažetu analizu pogreške kvantizacije na radna svojstva SVM, s naglaskom na utjecaj korištenja aritmetike s fiksnim zarezom. U radu prezentiraju novi algoritam za učenje SVM, koji je pogodan za sklopovsku implementaciju u aritmetici s fiksnim zarezom. Algoritam se sastoji od dva dijela koji se naizmjenično izvršavaju. Prvi dio uz konstantni parametar b rješava problem kvadratne optimizacije za dobivanje parametara α_i. Drugi dio uz dobivene α_i metodom bisekcije računa b za sljedeću iteraciju algoritma. Proces učenja izvodi se u potpunosti u sklopovlju. Izračun sume $\sum_i q_{ij}\alpha_j$ paraleliziran je s do 32 procesne jedinice. Eksperimentalno je implementacijom s malom širinom riječi (fiksni zarez formata 8.8 do 16.13) dobiven klasifikator usporediv s onim koji daje implementacija s pomičnim zarezom. Sustav je izveden na Xilinx Virtex-II platformi, s radom u području 19 do 35 MHz, ovisno o širini riječi.

Pedersen i sur. (2006) [53] eksperimentirali su s ubrzanjem učenja i klasifikacije sa SVM-om, s fokusom na ubrzanje izračuna skalarnog produkta uvođenjem sklopovske jedinice za množenje i akumulaciju (MAC). Za pokretanje programa koristili su sklopovski izveden Java virtualni stroj [54] na koji je spojena MAC jedinica. Sustav je implementiran na Altera Cyclone platformi s radom na 100 MHz. Korištenjem MAC jedinice dobiveno je ubrzanje do 59,8% pri učenju i do 6,5% pri klasifikaciji.

Cadambi i sur. (2009) u [55] prikazuju naprednu arhitekturu koprocesora za ubrzanje učenja SVM. Koriste aritmetiku s fiksnim zarezom formata 16.16, a na dva skupa podataka koristili su i format 16.4. Sustav se sastoji od računala koje izvršava algoritam za učenje i FPGA koprocesora za izračun skalarnih produkata. Koprocesor se sastoji od 128 procesnih elemenata koji izvršavaju množenje i akumulaciju, organiziranih u četiri grupe od kojih svaki ima po četiri polja sa osam elemenata. Koprocesor koristi vlastitu namjensku memoriju u kojoj su pohranjeni vektori iz skupa za učenje i priručne memorije kojima omogućuje istovremen izračun i prijenos podataka iz računala i prema njemu. Priručne memorije se koriste za kompenzaciju kašnjenja pri prijenosu podataka i osiguranje stalne aktivnosti

koprocesora. Sustav je izveden na Xilinx Virtex-5 platformi. Brzina učenja SVM uspoređena je sa skalarnom i vektorskom programskom implementacijom na dvojezgrenom 2,2 GHz Opteron procesoru. Dobivena su ubrzanja od 18,2 puta u usporedbi sa skalarnom, i 6,5 puta u usporedbi s vektorskom implementacijom.

Cao i sur. (2010) u [56] prikazuju implementaciju prilagođenu Keerthijevom algoritmu [57]. Keerthi uvodi matricu pogreške:

$$e_i = \sum_j \alpha_j y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - y_j \tag{3.12}$$

koju koristi za korekciju α_j i b parametara. Za određivanje gornje i donje granice parametra b radi se usporedba preko svih e_i. Ta dva dijela algoritma, izračun i usporedba e_i, paraleliziraju se. Autori paralelizaciju izvode preko jedinica za ažuriranje pogreške (engl. *error cache update unit*, ECU) koje implementiraju izračun elementa matrice pogreške i prate lokalno gornju i donju granicu b za vektore iz skupa za učenje. ECU sadrži tablice Gaussove jezgrene funkcije i MAC jedinice za izračun e_i. Dodatno, svaki ECU ima vlastitu lokalnu memoriju u kojoj pohranjuje vektore za učenje i trenutne vrijednosti parametara. Ispitano je ponašanje klasifikatora dobivenog uz razne kombinacije širine riječi i veličine tablice Gaussove jezgrene funkcije. Sa 16-bitnom aritmetikom i tablicom 1024 vrijednosti dobivene su točnosti klasifikacije jednake onima dobivenim programski s aritmetikom pomičnog zareza. Sustav je izveden na Xilinx Virtex-4 platformi i radom na 75 MHz.

Papadonikolakis i Bouganis (2010) u [58] su predstavili arhitekturu koja uzima u obzir raspone vrijednosti atributa u skupu za učenje. Skupove razvrstavaju u dvije vrste: homogene i heterogene. U homogenim skupovima rasponi vrijednosti različitih atributa približno su jednaki, dok se u heterogenim skupovima rasponi vrijednosti različitih atributa izrazito razlikuju. Kao vremenski najintenzivniji dio algoritma prepoznata je evaluacija jezgrenih funkcija. Evaluacija je izvedena u modulima koji sadrže jedinicu za skalarni produkt i procesor jezgrene funkcije. Skalarni produkt implementiran je paralelnim množilima i stablom zbrajala koji su izvedeni aritmetikom fiksnog zareza. Rezultat skalarnog produkta ulaz je u procesor jezgrene funkcije koji je izveden aritmetikom pomičnog zareza. Skup za učenje je pohranjen u unutrašnjoj memoriji FPGA. Da bi se postiglo što manje zauzeće resursa u FPGA, parametri korištene aritmetike prilagođeni su značajkama skupa za učenje. Kod heterogenih skupova parametri aritmetike prilagođeni su za svaki modul zasebno, tako da odgovaraju rasponu vrijednosti atributa koji će pojedini modul obrađivati. Ovaj pristup

zahtijeva analizu značajki skupa za učenje prije implementacija modula. Sustav je implementiran na Altera Stratix III FPGA platformi.

Wang i sur. (2011) [59] predstavili su implementaciju LS-SVM (engl. *least-squares support vector machines*) algoritma [60] namijenjenu za primjenu u ugrađenim računalnim sustavima. Sustav su podijelili na statički i dinamički dio. Dinamički dio sustava može se preprogramirati u tijeku rada, ovisno o funkciji koja se u danom trenutnu treba izvršavati. Statički dio sastoji se od CPU, memorijskog kontrolera i perifernih jedinica za komunikaciju s glavnim računalom. Dva dijela LS-SVM algoritma prepoznata su kao vremenski najzahtjevnija: stvaranje jezgrene matrice (engl. *kernel matrix*) i rješavanje sustava metodom najmanjih kvadrata. Ta dva dijela čine dinamički dio sustava. Stvaranje jezgrene matrice implementirano je procesnim elementima koji rade paralelno. Metoda najmanjih kvadrata izvedena je faktorizacijom Choleskog, koja je implementirana pomoću paralelnih jedinica za izračun skalarnog produkta. Cijeli sustav izveden je aritmetikom pomičnog zareza jednostruke preciznosti. Sustav je implementiran na Xilinx Virtex-5 platformi, s frekvencijom takta iznosa 150 MHz. U usporedbi s Xeon CPU-om frekvencije takta 2,93 GHz, postignuto je ubrzanje od 6 puta do 218 puta, za skupove od 512 do 8192 primjera.

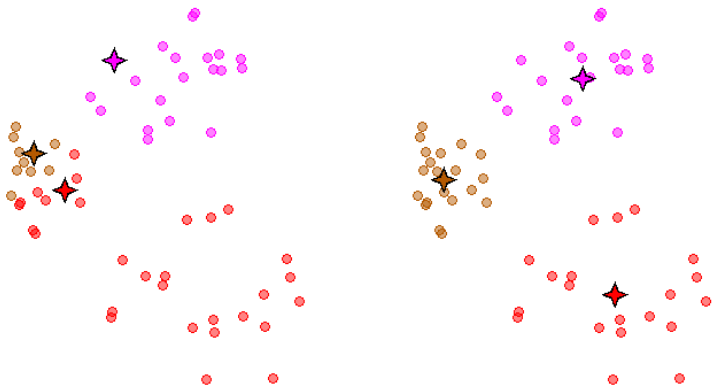
3.2.2 Algoritam k-srednjih vrijednosti

Algoritam k-srednjih vrijednosti (engl. *k-means*) [61] jednostavan je iterativni postupak kojim se skup podataka podijeli u određeni broj grupa. Broj grupa, k, unaprijed određuje korisnik. Algoritam se izvršava na skupu d-dimenzionalnih vektora $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$, gdje je $\boldsymbol{x}_i \in \mathbb{R}^d$. Grupe $\{S_1, S_2, \dots, S_k\}$ predstavljaju se vektorima $\boldsymbol{\mu}_j$ koji se nazivaju centri. Algoritam k-srednjih vrijednosti minimizira ukupnu varijancu unutar grupa

$$V = \sum_{j} \sum_{x_i \in S_j} \left\| \boldsymbol{x}_i - \boldsymbol{\mu}_j \right\|^2 \tag{3.13}$$

Pri izvršavanju algoritma prvo se odabere k točaka iz $\mathbb{R}^d$ koje će biti početne vrijednosti centara $\boldsymbol{\mu}_j$. Mogu se odabrati na više načina, npr. slučajnim odabirom ili nekom heurističkim metodom. Zatim se iterativno izvršavaju sljedeća dva koraka dok algoritam ne konvergira:

- formiraju se grupe tako da se točke iz skupa dodjeljuju najbližem centru
- izračunaju se nove vrijednosti centara tako da se izračuna srednja vrijednost svih točaka u grupi.



Slika 3.11. Primjer grupiranja u dvodimenzionalnom prostoru
lijevo: početno stanje, desno: završno stanje

Algoritam konvergira kada se više ne mijenja raspodjela točaka po grupama. Kao mjera udaljenosti najčešće se koristi euklidska udaljenost. Primjer početnog i završnog stanja grupiranja algoritmom k-srednjih vrijednosti prikazan je na slici 3.11. Točkama su označeni ulazni vektori, a zvjezdicama centri grupa.

FPGA implementacije algoritma k-srednjih vrijednosti primarno su usmjerene na primjene u obradi slike i videa. Istraživanja su najvećim dijelom usredotočena na problem izračuna udaljenosti točaka od centara, na koji se pri izvođenju algoritma troši najveći dio vremena.

Estlick i sur. (2001) [62] razmotrili su dvije promjene u samom algoritmu koje bi mogle povećati raspoloživi paralelizam: upotreba alternativne mjere udaljenosti i smanjenje širine riječi ulaznih vektora. Alternativne mjere su eksperimentalno ocijenjene i dobivena je Manhattan mjera, $\sum_i |x_i - \mu_i|$, kao najpogodnija s obzirom na kvalitetu grupiranja i složenost implementacije. Eksperimentalno su pokazali da se širina riječi ulaznih podataka može znatno smanjiti bez pogoršanja kvalitete grupiranja. FPGA koristi dvije vlastite vanjske memorije. U prvoj pohranjuje vrijednosti piksela, dok u drugu sprema rezultat grupiranja. Sustav je realiziran kao protočna struktura koja paralelno izračunava udaljenost piksela od svakog centra i pronalazi najmanju udaljenost i pripadnu grupu. Program koji se izvršava na računalu učitava piksele u FPGA memoriju, i za svaku iteraciju postavlja nove vrijednosti centara. Algoritam je implementiran na Xilinx Virtex platformi, s radom na 50 MHz. Dobiveno je ubrzanje za 200 puta u usporedbi s potpuno programskom implementacijom izvršavanoj na 500 MHz Pentium III računalu.

Gokhale i sur. (2003) u [63] su implementirali algoritam k-srednjih vrijednosti za segmentaciju hiperspektralnih slika na Altera Excalibur platformi, koja uz Altera Apex FPGA uključuje i ARM procesor. Koristili su dvije vrste izvedbe: s ARM procesorom i s NIOS *soft core* procesorom. U radu daju ilustrativan prikaz razvoja sustava iz kojeg su vidljivi utjecaji komunikacije između procesora i FPGA-a, granularnosti procesne jedinice u FPGA-u i korištenja izravnog pristupa memoriji na vrijeme izvršavanja algoritma. Algoritam je implementiran u nekoliko iteracija, počevši od potpuno programske, zamjenom dijela koda koji troši najviše vremena sklopovljem koje vrši istu funkciju. Završna implementacija u sklopovlju izvršava izračun udaljenosti točaka od centara i za čitanje piksela koristi izravni pristup memoriji, dok rezultate segmentacije vraća u posebni međuspremnik iz koje ih čita procesor. Jedinica za izračun udaljenosti sadrži 32 procesna elementa koji istovremeno računaju udaljenosti i radi na 33 MHz. Postignuto je ubrzanje za 11,8 puta u usporedbi s 1 GHz Pentium III računalom.

Wang i Leeser (2007) [64] eksperimentiraju s dodavanjem jedinice za dijeljenje s pomičnim zarezom u sklopovsku implementaciju algoritma. Struktura za pridjeljivanje piksela grupi nadogradnja je na protočnu strukturu koja paralelno izračunava udaljenost piksela od svakog centra i pronalazi najmanju udaljenost i pripadnu grupu iz [62]. U sustav su dodani akumulatori koji računaju sumu piksela koji pripadaju grupi, te brojila koja prate broj piksela u grupi. Nova vrijednost centra izračunava se dijeljenjem vrijednosti iz akumulatora s vrijednosti iz brojila, čime se određuje srednja vrijednost piksela iz grupe. Vrijednosti centara i akumulatori rade u aritmetici s fiksnim zarezom, stoga su na ulazu i izlazu djelila postavljene jedinice za pretvorbu brojeva između prikaza s fiksnim i pomičnim zarezom. Sustav je implementiran na Xilinx Virtex-II platformi. Uspoređene su tri izvedbe: potpuno programska implementacija, FPGA implementacija s dijeljenjem na procesoru i FPGA implementacija s dijeljenjem na FPGA-u. Program se izvršavao na 3,2 GHz Pentium 4 računalu. U usporedbi s potpuno programskom implementacijom, za 50 iteracija algoritma dobiveno je ubrzanje za 11 puta, dok je za 1000 iteracija postignuto ubrzanje za 174 puta. Razlike u vremenu izvođenja implementacija s djelilom u FPGA-u i s dijeljenjem izvršenim na procesoru su neznatne. Razlog tome je da dijeljenje uzima relativno mali dio vremena izvršavanja algoritma u odnosu na izračun udaljenosti.

Saegusa i Maruyama (2006, 2007) [65], [66] implementiraju algoritam k-srednjih vrijednosti za grupiranje na slikama u boji u stvarnom vremenu. Standardni postupak za grupiranje s euklidskom mjerom udaljenosti nadograđuju filtrom temeljenom na kd-stablu [67], čija je

zadaća da za pojedini piksel pronađe kandidate za centar. Izračun udaljenosti od piksela provodi se samo za taj mali broj kandidata. Kd-stablo izvedeno je korištenjem memorijskih banki i omogućuje istovremeno traženje centara za 4 piksela i daje 24 kandidata za svaki piksel. Sustav je realiziran kao protočna struktura. U prvom stupnju se nalazi kd-stablo, u drugom se izračunavaju kvadrati udaljenosti, a u trećem nalazi najbliži centar za svaki piksel. Jedinice za izračun kvadrata udaljenosti koriste se i za konstrukciju kd-stabla koje se provodi za svaku iteraciju algoritma. Sustav je izveden na Xilinx Virtex-II platformi i radi na 66 MHz. Pri segmentaciji u 256 grupa postiže najmanje 30 fps za slike dimenzija 512×512 i 640×480, te 20 do 30 fps za slike dimenzija 768×512.

Covington i sur. (2006) [68] koriste algoritam k-srednjih vrijednosti za grupiranje srodnih dokumenata. Dokumenti su predstavljeni vektorima, gdje svaka dimenzija vektora sadrži broj pojava riječi u dokumentu. Riječi se prevode u prostor značajki s 4000 dimenzija primjenom metode temeljene na raspršenom adresiranju (engl. *hash*). Dokument vektori i centri pohranjuju se u vanjskim memorijama spojenim na FPGA. Sustav ima tri ključna modula. Prvi modul služi za izračun udaljenosti, a sastoji se od jedinica za izračun kvadratnog korijena, skalarnog produkta i dijeljenja. Modul je složen i daje jedan rezultat svakih 294 ciklusa takta. Za svaki centar ugrađen je po jedan modul za izračun udaljenosti. Drugi modul uspoređuje udaljenosti vektora od centara i odlučuje kojem će ga centru pridružiti. Treći modul izračunava nove vrijednosti centara. Brzina sustava uspoređena je s 3,6 GHz Xeon procesorom. Na Virtex-E platformi sustav radi na 80 MHz i dobiveno je ubrzanje za 26 puta. Na Virtex-4 platformi sustav radi na 250 MHz i dobiveno je ubrzanje za 328 puta.

Nagarajan i sur. (2009) [69] uzimaju općenitiji pristup s namjerom određivanja obrazaca i modela koji su zajednički za više algoritama (engl. *design patterns*). Ideja je da poznavanje i korištenje tih obrazaca znatno pomaže u razvoju sustava. U radu daju pregled čestih komunikacijskih i računskih obrazaca. Razvoj sklopovlja počinju s problemom procjene funkcije gustoće vjerojatnosti i koristeći strukturalne sličnosti algoritama dobiveno rješenje prilagođuju za implementaciju izračuna korelacije i algoritma k-srednjih vrijednosti, umjesto da ih razvijaju ispočetka. Implementacija algoritma k-srednjih vrijednosti ima tri stupnja povezana u protočnu strukturu: (a) izračun kvadrata udaljenosti, (b) nalaženje minimalne udaljenosti i (c) ažuriranje centara. Dobivena prva dva stupnja slična su implementaciji Seagusa i Maruyama [66]. Sustav su implementirali i testirali na Xilinx Virtex-4 platformi. Dobiveno je ubrzanje za 3,2 puta u usporedbi s 3,2 GHz Xeon računalom. Uzrok malog faktora ubrzanja objašnjava se niskom propusnosti komunikacije između FPGA i procesora.

Hussain i sur. (2011) [70] predstavljaju FPGA implementaciju metode k-srednjih vrijednosti za primjenu u bioinformatici. Sustav je namijenjen grupiranju podataka dobivenih iz eksperimenata provedenih tehnologijom mikročipova. Svi dijelovi algoritma implementirani su u sklopovlju, s paraleliziranim izračunom mjere udaljenosti. Korištena je aritmetika fiksnog zareza koja je prilagođena rasponima vrijednosti tipičnih za podatke iz mikročipova. Broj grupa je ograničen na osam, i sustav sadrži isti broj jedinica za izračun udaljenosti koje rade paralelno. Ulazni podaci pohranjeni su u unutrašnjoj memoriji FPGA-a. U jednom FPGA-u može se izvesti više jezgara za grupiranje, gdje svaka obrađuje svoj skup podataka. Sustav sa pet jezgara implementiran je na Xilix Virtex-4 platformi uz radnu frekvenciju od 124 MHz. Dobivena su ubrzanja od 10,3 puta za jednojezgrenu do 51,7 puta za petojezgrenu implementaciju, u odnosu na računalo s 3 GHz Core 2 Duo CPU-om.

Singaraju i Chandy (2011) [71] implementirali su algoritam k-srednjih vrijednosti na mrežnim usmjerivačima temeljnim na FPGA-u. Njihova arhitektura omogućuje obradu podataka koji se prenose u Ethernet paketima. Na FPGA-u implementiran je cijeli algoritam i za sve izračune korištena je aritmetika s pomičnim zarezom. Paralelizirani su izračun mjere udaljenosti i traženje minimalne udaljenosti. Budući da su aritmetičke operacije izvedene dubokim protočnim strukturama, za kompenzaciju kašnjenja korišteno je vremensko ispreplitanje podataka. Sustav s jednim FPGA-om podržava paralelizam na razini grupa, čime omogućuje istovremeni izračun udaljenosti podatka od centra svake grupe. Sustav s više FPGA-ova podržava i podatkovni paralelizam, gdje svaki FPGA obrađuje dio podataka. Sustav je implementiran na Xilinx Virtex-II platformi. Sustav s jednim FPGA-om postigao je ubrzanje od deset puta u usporedbi s Opteron procesorom na 1,8 GHz. Sustav s više FPGA-ova uspoređen je s paralelnom programskom implementacijom koja se izvršava na ekvivalentnom broju čvorova. Postignuto je ubrzanje od devet puta.

3.3 Analiza zajedničkih značajki implementacija

U tablici 3.1 sažeto su prikazani rezultati pregleda stanja polja. Iz ovog pregleda rezultata u području razvoja i izvedbe ubrzavanja algoritama pomoću sklopova programirljive logike mogu se istaknuti neke značajke koje su zajedničke većini implementacija algoritama na FPGA platformi.

Sve implementacije u početnoj fazi provode analitičku obradu algoritama. Određuju se dijelovi u algoritamskom postupku koji najduže traju, te se provodi procjena dobitka u brzini

izvršavanjem tih dijelova u sklopovlju. Najčešće se radi o kratkom nizu operacija koje se mnogo puta ponavljaju u petlji.

FPGA je po radnom taktu sporiji od procesora, razlike su obično jedan do dva reda veličine, što se kompenzira paralelizmom. FPGA, ovisno o konkretnoj implementaciji, može istovremeno izvesti više stotina, pa i tisuća operacija.

Implementacije na FPGA-u redovito koriste protočne strukture. Osim dodatnog „dubinskog" paralelizma koji daje protočna struktura, razbijanje procesa na više manjih koraka omogućuje i povećanje radne frekvencije, a time i brže izvođenje algoritma. Provodi se optimizacija algoritma radi povećanja učinkovitosti izvođenja temeljne funkcionalnosti.

Ranije implementacije izbjegavaju uporabu aritmetike s pomičnim zarezom. Razlog tome je da su aritmetičke jedinice s pomičnim zarezom složene i zauzimaju mnogo veći broj osnovnih elemenata FPGA-a nego jedinice s fiksnim zarezom. Veće iskorištenje osnovnih elemenata FPGA-a znači manje mogućnosti za paralelizaciju, jer smanjuje broj jedinica koje se mogu istovremeno implementirati. Iz tog su se razloga algoritmi uglavnom izvodili u aritmetici s fiksnim zarezom. Zbog navedenog ograničenja potrebno je provesti numeričku analizu kako bi se odredio format brojeva s fiksnim zarezom koji osigurava ispravnost dobivenih rezultata. Novije implementacije sve češće koriste aritmetiku s pomičnim zarezom, budući da razvojem tehnologije FPGA-ova raspoloživi resursi postaju manje izraženo ograničenje.

Vrlo su rijetke implementacije koje algoritam u potpunosti izvode u sklopovlju. Gotovo uvijek se radi o preciznoj podjeli izvedbe algoritma u sklopovski i programski dio i pri tome se koriste metode programsko-sklopovskog suoblikovanja [72]. Čak i potpuno FPGA izvedbe sadrže *soft core* procesor koji izvodi glavni program. Razlog tome je da je dijelove koji upravljaju tijekom algoritma mnogo jednostavnije implementirati kao program koji se izvršava na procesoru, nego u sklopovlju razvijati poseban automat koji izvršava istu funkciju.

Čest problem u sklopovskim implementacijama je propusnost komunikacijskog kanala između FPGA-a i procesora, te FPGA-a i vanjske memorije. Vremenski trošak komunikacije ponekad je veći od dobitka uslijed bržeg izvršavanja u sklopovlju, što rezultira sporijim izvršavanjem algoritma [63]. Ovaj problem moguće je riješiti izravnim pristupom memoriji, gdje procesor (samo) dostavlja instrukcije o memorijskoj adresi i broju podataka. Dodatno poboljšanje postiže se uz FPGA koji ima značajnu količinu vlastite radne memorije, kao i

korištenjem međuspremnika koji omogućuju da se jedan blok podataka obrađuje dok se drugi prenosi.

Ranije implementacije rijetko su koristile operacije množenja, kao relativno skupe operacije u FPGA-u. Novije generacije FPGA-ova u sebi uključuju određen broj množila, a sve češće i cijele jedinice za množenje i akumulaciju, što rješava ovaj problem.

Pojavom modernih FPGA-ova koji sadrže i do tisuću MAC jedinica, te značajne količine unutarnje memorije, otvorile su se mogućnosti za implementaciju složenijih računskih postupaka. Također, pojavom ulazno-izlaznih blokova za izvedbu sučelja visoke brzine otvorila se mogućnost korištenja brzih vanjskih memorija kao što je DDR-3 SDRAM. Brza vanjska memorija omogućuje implementaciju memorijski intenzivnih algoritama kao što je učenje stabla odluke.

Tablica 3.1 Pregled FPGA implementacija algoritama za dubinsku obradu podataka

Autori	Algoritam	Opis	Platforma	Radni takt	Rezultati
Narayanan i sur. [16] 2007.	Stabla odluke	Paralelizirani i izvedeni u sklopovlju: izračun Gini indeksa i traženje minimalnog Gini indeksa. Komunikacija putem procesorske lokalne sabirnice.	Xilinx Virtex-II Pro	100 MHz	Ubrzanje 5,58×
Anguita i sur. [52] 2003.	Stroj s potpornim vektorima	Svi dijelovi algoritma izvedeni u sklopovlju. Paraleliziran izračun skalarnog produkta. Koristi se vlastita memorija.	Xilin Virtex-II	19 35 MHz	–
Pedersen i sur. [53] 2006.	Stroj s potpornim vektorima	Ubrzanje skalarnog produkta korištenjem sklopovske jedinice za množenje i akumulaciju. Program se izvršava na sklopovskoj izvedbi Java virtualnog stroja.	Altera Cyclone	100 MHz	Ubrzanje do 59,8%.
Cadambi i sur. [55] 2009.	Stroj s potpornim vektorima	Paraleliziran i u sklopovlju izveden izračun skalarnih produkata. Koristi se vlastita memorija, izravno pristupanje memoriji računala i međuspremnici za povećanje učinkovitosti komunikacije. Komunikacija putem PCI sabirnice.	Xilinx Virtex-5	133 MHz	Ubrzanje 18,2× (2,2 GHz Opteron)
Cao i sur. [56] 2010.	Stroj s potpornim vektorima	Svi dijelovi algoritma izvedeni u sklopovlju. Parelelizirani izračun i usporedbe elemenata matrice pogreške. Koristi se vlastita memorija.	Xilinx Virtex-4	75 MHz	–
Papadoni-kolakis i Bouganis [58] 2010.	Stroj s potpornim vektorima	U sklopovlju izveden skalarni produkt i evaluacija jezgrene funkcije. Paralelna obrada više atributa. Parametri aritmetike prilagođeni rasponu vrijednosti u skupu za učenje.	Altera Stratix III	–	–
Wang i sur. [59] 2011.	Stroj s potpornim vektorima	Paralelizirani i u sklopovlju izvedeno stvaranje jezgrene matrice i faktorizacija Choleskog. Koristi se dinamičko preprogramiranje FPGA.	Xilinx Virtex-5	150 MHz	Ubrzanje do 218× (2,93 GHz Xeon)
Estlick i sur. [62] 2001.	K-srednjih vrijednosti	Paralelizirani i u sklopovlju izvedeni: izračun udaljenosti i traženje najmanje udaljenosti. Koristi se vlastita memoriju. Komunikacija putem PCI sabirnice.	Xilinx Virtex	50 MHz	Ubrzanje 200× (500 MHz Pentuim III)
Gokhale i sur. [63] 2003.	K-srednjih vrijednosti	Paraleliziran i u sklopovlju izveden izračun udaljenosti. Koristi se izravno pristupanje memoriji računala, i međuspremnik za povećanje učinkovitosti komunikacije. Komunikacija putem AHB sabirnice.	Altera Apex	33 MHz	Ubrzanje 11,8× (1 GHz Pentium III)
Wang i Leeser [64] 2007.	K-srednjih vrijednosti	Paralelizirani i izvedeni u sklopovlju: izračun udaljenosti, traženje najmanje udaljenosti i izračun nove vrijednosti centara. Sadrži sklopovsko djelilo. Koristi se vlastita memorija. Komunikacija putem PCI sabirnice.	Xilinx Virtex-II	–	Ubrzanje do174× (3,2 GHz Pentium 4)
Saegusa i Maruyama [65], [66] 2006, 2007.	K-srednjih vrijednosti	Paralelizirani i u izvedeni sklopovlju: filtriranje kd-stablom, izračun kvadrata udaljenosti, usporedba udaljenosti i gradnja kd-stabla.	Xilinx Virtex-II	66 MHz	> 30 fps (512×512, 640×480), 20-30 fps (768×512)
Convington i sur. [68] 2006.	K-srednjih vrijednosti	Paralelizirani i izvedeni u sklopovlju: izračun udaljenosti, usporedbe udaljenosti i izračun novih vrijednosti centara. Korištenje vlastite memorije. Komunikacija putem IP protokola.	Xilinx Virtex-E Xilinx Virtex-4	80 MHz 250 MHz	Ubrzanje 26× (3,6 GHz Xeon) Ubrzanje 328× (3,6 GHz Xeon)
Nagarjan i sur. [69] 2009.	K-srednjih vrijednosti	Paralelizirani i izvedeni u v: izračun kvadrata udaljenosti, traženje minimalne udaljenosti i ažuriranje centara. Komunikacija putem PCI sabirnice.	Xilinx Virtex-4	–	Ubrzanje 3,2× (3,2 GHz Xeon)
Hussain i sur. [70]	K-srednjih vrijednosti	Svi dijelovi algoritma izvedeni u sklopovlju, paraleliziran izračun mjere udaljenosti.	Xilinx Virtex-4	124 MHz	Ubrzanje 51,7× (3 GHZ Core2 Duo)
Singaraju i Chandy [71]	K-srednjih vrijednosti	Svi dijelovi algoritma izvedeni su u sklpovlju. Pareleliziran izračun udaljenosti i traženje minimalne udaljenosti.	Xilinx Virtex-II	–	Ubrzanje 10× (1,8 GHz Opteron)

4 Programske implementacije algoritma C4.5

4.1 Stabla odluke

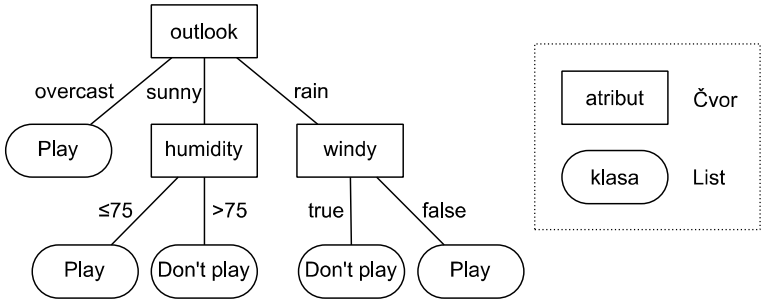
Stabla odluke su vrsta klasifikatora koja koristi stablo kao strukturu za opis modela klasifikacije [3]. Klasifikator kao ulaz prima primjer koji je opisan nizom atributa. Primjer strukture stabla odluke prikazan je na slici 4.1. Čvorovi stabla određuju upit atributa, grane odgovaraju mogućim ishodima upita, a listovi predstavljaju ishod klasifikacije. Nepoznatim primjerom se, počevši od korijena, obilazi stablo i slijede grane prema ishodima upita određenim čvorovima dok se ne dođe do lista koji daje klasifikaciju primjera.

Učenje stabla odluke je rekurzivni proces koji slijedi načelo „podijeli pa vladaj“. Tijek procesa je sljedeći:

- Na danom skupu za učenje S:
- ako svi primjeri pripadaju istoj klasi, napravi se list
- u protivnom, odabere se ispitni atribut i napravi čvor
- skup S se prema ishodima testa atributa podijeli na podskupove $S_1, S_2, \ldots$ i proces se rekurzivno ponavlja na svakom podskupu.

Proces gradnje završava kada stablo ispravno klasificira sve primjere za učenje, tj. kada se više ne stvaraju novi čvorovi. Nakon gradnje stabla, ono se obično obrezuje da bi se izbjegla prenaučenost, a i smanjila veličina naučenog stabla.

Važan dio procesa je odabir ispitnog atributa. Atribut se odabire prema nekoj mjeri njegove



Slika 4.1. Primjer stabla odluke

kvalitete, tj. kvaliteti podjele skupa S na podskupove prema vrijednostima tog atributa. Neke od mjera kvalitete su: Gini indeks, entropija, informacijski dobitak i faktor informacijskog dobitka.

Poznatiji algoritmi za učenje stabla odluke su ID3 [5], CART [6] i C4.5 [7]. Svi navedeni algoritmi u svojoj srži imaju opisani rekurzivni proces učenja, a razlikuju se prvenstveno po nekoliko značajki, npr. mjeri kvalitete atributa, podržanim vrstama atributa, tolerantnosti na nedostajuće vrijednosti, algoritmu obrezivanja stabla itd.

4.2 Algoritam C4.5

Algoritam C4.5 prepoznat je kao jedan od najvažnijih algoritama u dubinskoj analizi podataka [4]. C4.5 kao ulaz prihvaća skup primjera S i gradi stablo sljedećim postupkom:

- Ako su svi primjeri u S iste klase, ili je broj primjera u S malen, stablo se pretvara u list označen klasom najveće frekvencije u S.
- Inače, odabere se test na jednom atributu koji ima dva ili više ishoda. Taj test postaje korijen stabla koje ima po jednu granu za svaki ishod testa. S se podijeli na podskupove $S_1, S_2, \ldots$ prema ishodu testa za svaki primjer i postupak se ponovi rekurzivno na svakom podskupu.

Pri izboru testa atributa, C4.5 koristi dvije mjere kvalitete: informacijski dobitak (engl. *information gain*), te faktor informacijskog dobitka (engl. *information gain ratio*).

Atributi mogu biti numerički ili nominalni, te o tome ovisi način postupanja s njima. Numerički atributi predstavljaju brojčane vrijednosti, realne ili cijele, dok nominalni atributi poprimaju određen konačan broj diskretnih vrijednosti. Bitna razlika je da numerički atributi mogu poprimiti vrijednosti iz beskonačnog skupa, dok je kod nominalnih skup mogućih vrijednosti konačan i unaprijed poznat. O vrsti atributa ovisi oblik testa. Ako je atribut numerički, test ima dva ishoda: $\{A \leq t, A > t\}$, gdje je t prag koji maksimizira informacijski dobitak. Ako je atribut nominalni, test ima onoliko ishoda koliko ima mogućih vrijednosti: $\{A = a_1, A = a_2, \ldots, A = a_n\}$, gdje su $a_1, a_2, \ldots, a_n$ moguće vrijednosti atributa.

Kad je proces izgradnje početnog stabla završen, ono se obrezuje. Obrezivanje se izvršava od listova prema korijenu. Za podstablo, C4.5 zbraja procijenjene pogreške grana i uspoređuje zbroj s procijenjenom pogreškom lista koji bi nadomjestio to podstablo. Podstablo se podrezuje samo kada procijenjena pogreška lista nije veća od pogreške podstabla. Također,

C4.5 provjerava procjenu pogreške nadomještavanja podstabla jednom od njenih grana, te to nadomještavanje provodi kada ono smanjuje pogrešku.

Izvorni tekst programa C4.5 Release 8 dostupan je javno na web stranici autora algoritma [73]. Uz izvornu implementaciju autora algoritma, javno su dostupne i slijedeće implementacije algoritma C4.5: EC4.5, J48 i YaDT.

EC4.5 (Efficient C4.5) [8] preinačena je izvedba C4.5, s poboljšanjima vezanim uz rad s numeričkim atributima. EC4.5 uvodi sljedeće promjene u originalni algoritam:

- dodane su podatkovne strukture kojima se prate značajke numeričkih atributa kao što su jedinstvene vrijednosti atributa i broj pojava pojedinih jedinstvenih vrijednosti
- umjesto slijednog uvedeno je binarno pretraživanje praga za podjelu skupa prema numeričkom atributu
- uz *quicksort*, ovisno o rasponu vrijednosti atributa koristi se i *counting-sort*, a dodana je i strategija koja ne koristi sortiranje.

Izvorni tekst programa EC4.5 dostupan je javno na web stranici autora [74] kao zakrpa na C4.5 Release 8.

J48 je Java implementacija algoritma C4.5 dostupna kao dio Weka sustava za dubinsku analizu podataka [11], [75].

YaDT [9], [10] reimplementacija je C4.5 u jeziku C++. Ova implementacija uvodi neke optimizacije u pogledu podatkovnih struktura koje koristi za pohranu skupa za učenje, a u izvedbi algoritma koristi iste optimizacije kao i EC4.5, te dodatnu optimizaciju prema metodi Fayyada i Iranija [76], koja ubrzava nalaženje lokalnog praga za podjelu skupa pri obradi numeričkog atributa. YaDT na operacijskom sustavu Linux podržava višedretveno izvođenje programa. YaDT javno je dostupan u obliku programske biblioteke na web stranicama autora [77].

4.3 Dinamička analiza algoritma C4.5 i EC4.5 za učenje stabla odluke

Dinamička analiza programa vrsta je ispitivanja koja se provodi s ciljem postizanja boljeg razumijevanja radnih značajki programa. Najčešći povod za provođenje dinamičke analize jest optimizacija programa sa svrhom postizanja kraćeg vremena izvođenja. U tu svrhu dinamičkom analizom izradi se profil programa koji daje informaciju na kojim dijelovima

programa se provodi najviše vremena. Dvije osnovne vrste alata za dinamičku analizu su: instrumentacijski i statistički. Kod instrumentacijskih alata prilikom prevođenja i povezivanja programa dodaju se funkcije koje mjere vrijeme izvršavanja pojedinih dijelova programa. Kod statističkih alata nema značajne modifikacije programa prilikom prevođenja/povezivanja, a tijekom pokretanja programa u redovitim vremenskim razmacima identificira se dio programa koji se u tom trenutku izvršava.

U ovom radu cilj dinamičke analize je utvrditi koji dijelovi algoritma za učenje stabla odluke imaju najveći udio u vremenu izvršavanja programa. Prikazat će se rezultati dinamičke analize implementacije algoritma za učenje stabla odluke C4.5 [73] i njegove preinačene izvedbe EfficientC4.5 (EC4.5) [74], a koji su osnova za izvedbu hibridnog algoritma.

4.3.1 Metoda provedbe dinamičke analize

Programske implementacije analizirane su pomoću gprof [78] alata za dinamičku analizu programa. Dinamička analiza provedena je na osobnom računalu s Intel Core 2 Q8400 CPU-om i 8 GiB RAM-a, pod operacijskim sustavom Linux, verzija 2.6.32. Korišten je prevodilac gcc, verzija 4.4.2. Parametri prevodioca su: „-O2 -pg -Wall -c -fmessage-length=0“. Parametri povezivanja su: „-pg -lm“.

Kao ulazni podaci korišteni su skupovi podataka iz javno dostupnih izvora: „UCI machine learning repository“ [45] i arhive s izvornim tekstom programa C4.5 [73]. Iz navedenih izvora odabrano je šest skupova različitih veličina. Popis korištenih skupova podataka zajedno s njihovim osnovnim značajkama prikazan je u tablici 4.1.

Provedene su tri vrste ispitivanja:

- dinamička analiza programa na potpunim skupovima
- dinamička analiza programa na uzorcima skupova različitih veličina

Tablica 4.1. Korišteni skupovi podataka i njihove osnovne značajke

Naziv	Broj primjera	Broj atributa			Broj klasa	Izvor
		Ukupno	Numeričkih	Nominalnih		
Adult	32.561	14	6	8	2	UCI
Census-Income	199.523	40	7	33	2	UCI
Covertype	581.012	54	10	44	7	UCI
Golf	14	4	2	2	2	C4.5
Soybean	683	35	0	35	19	C4.5
Vote	300	16	0	16	2	C4.5

UCI = UCI machine learning repository [45]
C4.5 = Arhiva s izvornim tekstom programa C4.5 [73]

- dinamička analiza programa na uzorcima skupova s različitim brojem atributa.

Prvo ispitivanje, na potpunim skupovima, provedeno je sa svih šest odabranih skupova. S obzirom na način rada alata za dinamičku analizu, potrebno je određeno vrijeme izvršavanja programa da bi se dobili pouzdani rezultati. Stoga je učenje stabla za svaki skup provedeno više puta, tako da ukupno vrijeme izvršavanja programa bude najmanje deset minuta.

Drugo ispitivanje, na uzorcima skupova različitih veličina, provedeno je s dva najveća skupa (od šest odabranih), a to su skupovi Covertype i Census-Income. Iz navedenih skupova uzeti su uzroci različitih veličina. Veličine uzoraka određene su logaritamski da bi se razumnim brojem uzoraka ispita što veći raspon veličina. Uzorci su dobiveni iz skupa slučajnim odabirom bez ponavljanja. Za svaku veličinu skupa, napravljeno je više uzoraka, čiji broj je određen pomoću geometrijske razdiobe, tako da svaki primjer iz punog skupa ima određenu vjerojatnost da bar jedanput bude odabran u neki od uzoraka. Broj uzoraka izračunat je formulom:

$$S = \left\lceil \frac{\ln(1-T)}{\ln\left(1-\frac{n}{N}\right)} \right\rceil \tag{4.1}$$

gdje je S broj uzoraka, T minimalna vjerojatnost izbora primjera u barem jedan uzorak, n broj primjera u uzorku i N broj primjera u skupu iz kojeg se uzimaju uzorci. Parametar T za skup Covertype iznosi 0,80, a za skup Census-Income iznosi 0,85. Veličine uzoraka i njihov broj

Tablica 4.2. Veličine i broj uzoraka za skaliranje veličine skupa za učenje

	Broj uzoraka	
Broj primjera	Covertype (T = 0,80)	Census-Income (T = 0,85)
500	1.870	757
792	1.180	477
1.256	744	301
1.991	469	190
3.155	296	120
5.000	187	75
7.924	118	47
12.559	74	30
19.905	47	19
31.548	29	12
50.000	18	7
79.245	11	4
125.594	7	2
199.054	4	1
315.479	3	–
500.000	1	–

Tablica 4.3 Veličine i broj uzoraka za skaliranje broja atributa

Broj atributa	Broj uzoraka Covertype (T = 0,995)	Census-Income (T = 0,999)
5	55	52
7	39	36
10	26	25
13	20	18
17	15	13
23	10	9
30	7	5
40	4	1
54	1	–

prikazani su u tablici 4.2. Da bi se statistička obrada provela što kvalitetnije, dinamička analiza je provedena 10 puta sa svakim uzorkom.

Treće ispitivanje, na uzorcima skupova s različitim brojem atributa, također je provedeno s dva najveća skupa: Covertype i Census-Income. Iz navedenih skupova uzeti su uzorci različitog broja atributa. Broj atributa određen je logaritamski u rasponu od 5 do 54 (najveći broj atributa u oba skupa). Za svaki broj atributa napravljeno je više uzoraka, čiji broj je određen geometrijskom razdiobom tako da svaki atribut ima određenu vjerojatnost da barem jedanput bude odabran u neki od uzoraka. Broj uzoraka izračunat je formulom (4.1). Parametar T za skup Covertype iznosi 0,995, a za skup Census-Income iznosi 0,999. Broj atributa u uzorku i broj uzoraka prikazan je u tablici 4.3. Broj primjera je na oba skupa izjednačen i iznosi 199.054. Ovaj iznos je izabran jer on predstavlja vrijednost veličine skupa koja je najbliža broju primjera u punom skupu Census-Income. Primjeri za ispitni podskup odabrani su slučajno, a ostaju isti za sva ispitivanja skaliranja broja atributa.

4.3.2 Rezultati dinamičke analize

4.3.2.1 Dinamička analiza na cjelovitim skupovima

Rezultati dinamičke analize na cjelovitim skupovima dani su tablicama 4.4 i 4.5. Uz udjele pojedinih funkcija u vremenu učenja stabla, prikazani su i prosječno vrijeme učenja stabla, te težinski prosjek udjela funkcije u vremenu učenja stabla. Prosječno vrijeme učenja stabla izračunato je kao aritmetička sredina izmjerenih vremena. Težinski prosjek udjela funkcije u vremenu izvršavanja izračunat je otežavanjem pojedinih izmjerenih udjela za pojedini skup s prosječnim vremenom učenja na tom skupu prema formuli:

Tablica 4.4 Rezultati dinamičke analize programa C4.5

Naziv funkcije	Skup podataka / Udio u vremenu izvršavanja						Težinski prosjek
	Golf	Vote	Soybean	Adult	Census-Income	Covertype	
GreatestValueBelow	0,81%	0,00%	0,00%	61,37%	50,78%	70,10%	69,06%
ComputeFrequencies	3,92%	46,81%	41,01%	3,41%	18,89%	5,70%	6,39%
Group	12,77%	11,66%	8,14%	7,04%	4,83%	5,37%	5,35%
EvalContinuousAtt	13,69%	0,00%	0,00%	8,80%	11,02%	4,75%	5,09%
EstimateErrors	7,79%	4,82%	4,19%	1,71%	1,54%	1,72%	1,71%
Quicksort	10,59%	0,00%	0,00%	7,65%	8,54%	9,49%	9,43%
QuicksortCP	–	–	–	–	–	–	–
TotalInfo	22,09%	11,33%	9,53%	3,16%	1,22%	0,55%	0,59%
Swap	5,65%	4,58%	3,39%	4,06%	1,53%	1,83%	1,82%
FormTreeInt	–	–	–	–	–	–	–
SetValuesInteger	–	–	–	–	–	–	–
SetValues	–	–	–	–	–	–	–
Countsort	–	–	–	–	–	–	–
CountItems	1,73%	1,01%	9,30%	1,12%	0,36%	0,17%	0,18%
FormTree	6,62%	5,87%	2,09%	0,67%	0,69%	0,18%	0,21%
ComputeGain	4,73%	2,76%	1,24%	0,67%	0,34%	0,09%	0,10%
DiscrKnownBaseInfo	1,58%	4,56%	12,04%	0,06%	0,13%	0,01%	0,02%
ResetFreq	2,24%	3,00%	7,79%	0,20%	0,10%	0,01%	0,02%
AddErrs	1,73%	0,44%	0,26%	0,03%	0,00%	0,00%	0,00%
EvalDiscreteAtt	0,51%	1,49%	0,74%	0,01%	0,01%	0,00%	0,00%
CopyTree	1,12%	0,12%	0,05%	0,01%	0,00%	0,00%	0,00%
InitialiseTreeData	1,12%	0,15%	0,02%	0,00%	0,00%	0,00%	0,00%
Vrijeme učenja stabla	19,6 µs	368,47 µs	8,58 ms	2,00 s	34,20 s	614,38 s	

$$P_i = \frac{\sum_j t_j p_{ij}}{\sum_j t_j} \tag{4.2}$$

gdje je P_i težinski prosjek za funkciju i, t_j vrijeme izvršavanja programa na skupu j, a p_{ij} udio funkcije i u vremenu izvršavanja programa na skupu j.

Iz rezultata navedenih u tablicama 4.4 i 4.5 mogu se vidjeti sljedeće značajke. Kod programa C4.5 najviše vremena provodi se u izvršavanju funkcija *GreatestValueBelow*, *Quicksort*, *ComputeFrequencies*, *Group*, *EvalContinuousAtt*. Kod EC4.5 najviše vremena provodi se u izvršavanju funkcija *ComputeFrequencies*, *Group*, *EvalContinuousAtt* i *EstimateErrors*.

Funkcija ***ComputeFrequencies*** izračunava matricu frekvencija za nominalni atribut, koja služi za izračun informacijskog dobitka. Matrica frekvencija sadrži rezultate prebrojavanja primjera određene klase koji imaju određenu vrijednost nominalnog atributa.

Tablica 4.5. Rezultati dinamičke analize programa EC4.5

Naziv funkcije	Skup podataka / Udio u vremenu izvršavanja						Težinski prosjek
	Golf	Vote	Soybean	Adult	Census-Income	Covertype	
GreatestValueBelow	0,00%	0,00%	0,00%	0,00%	0,00%	0,01%	0,01%
ComputeFrequencies	2,72%	48,43%	41,08%	7,25%	48,04%	34,23%	35,46%
Group	5,38%	10,21%	7,82%	14,26%	9,87%	27,44%	25,69%
EvalContinuousAtt	26,79%	0,00%	0,00%	34,81%	22,37%	19,11%	19,48%
EstimateErrors	3,55%	4,58%	4,26%	2,48%	3,07%	9,55%	8,90%
Quicksort	0,00%	0,00%	0,00%	2,38%	3,90%	0,040%	0,42%
QuicksortCP	26,73%	0,00%	0,00%	15,09%	2,08%	2,14%	2,18%
TotalInfo	3,09%	11,83%	9,56%	0,69%	3,07%	3,11%	3,10%
Swap	3,85%	5,19%	3,79%	4,07%	1,33%	1,64%	1,62%
FormTreeInt	5,01%	5,69%	2,24%	1,79%	1,95%	1,00%	1,09%
SetValuesInteger	0,00%	0,00%	0,00%	10,03%	1,02%	0,46%	0,55%
SetValues	12,75%	0,00%	0,00%	3,24%	0,83%	0,34%	0,40%
Countsort	2,09%	0,00%	0,00%	1,52%	0,17%	0,36%	0,35%
CountItems	0,76%	0,67%	9,33%	1,59%	1,00%	0,17%	0,26%
FormTree	0,07%	0,03%	0,00%	0,00%	0,00%	0,00%	0,00%
ComputeGain	0,50%	2,73%	1,21%	0,20%	0,15%	0,02%	0,03%
DiscrKnownBaseInfo	1,39%	3,77%	11,79%	0,17%	0,25%	0,04%	0,06%
ResetFreq	1,20%	3,30%	7,71%	0,13%	0,21%	0,03%	0,05%
AddErrs	1,23%	0,43%	0,29%	0,13%	0,02%	0,04%	0,04%
EvalDiscreteAtt	0,37%	1,41%	0,64%	0,03%	0,06%	0,01%	0,01%
CopyTree	0,27%	0,26%	0,06%	0,00%	0,00%	0,00%	0,00%
InitialiseTreeData	0,90%	0,24%	0,01%	0,00%	0,00%	0,00%	0,00%
Vrijeme učenja stabla	30,06 µs	359,05 µs	8,75 ms	457,58 ms	11,80 s	109,84 s	

Funkcija ***Group*** grupira primjere iz skupa koji imaju istu vrijednost testnog atributa. Funkcija se koristi pri gradnji i pri obrezivanju stabla. Pri učenju stabla, namjena joj je podijeliti skup na podskupove prema vrijednosti odabranog atributa. Pri obrezivanju stabla, namjena joj je grupirati zajedno primjere skupa koji pripadaju istoj grani stabla, a u svrhu procjene pogreške čvora.

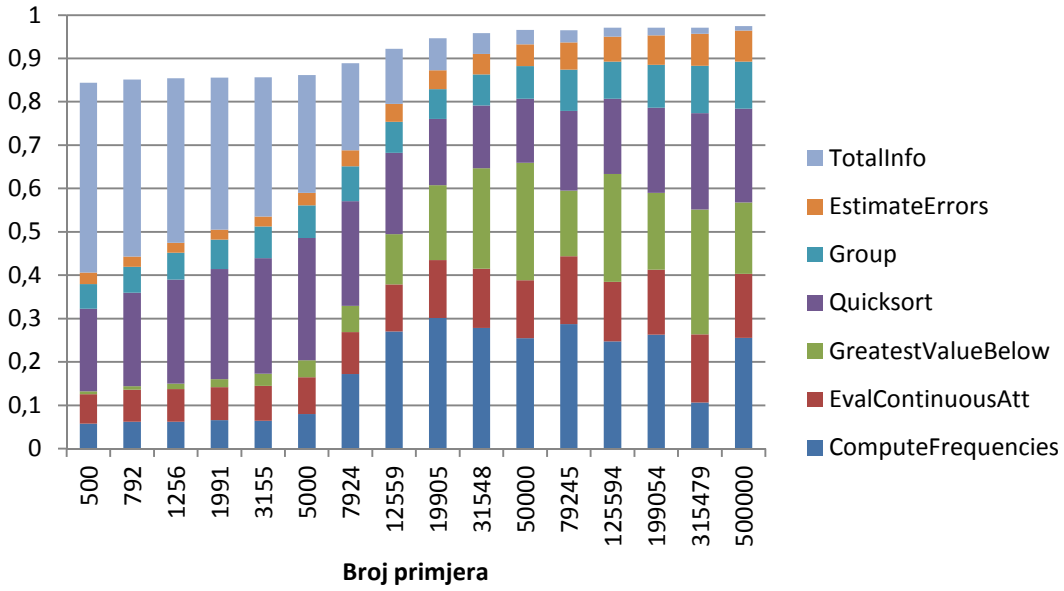
Funkcija ***EvalContinuousAtt*** izračunava prag koji maksimizira informacijski dobitak za obrađivani numerički atribut, koji se potom koristi za izračun informacijskog dobitka.

Funkcija ***Quicksort*** sortira skup prema vrijednosti određenog atributa.

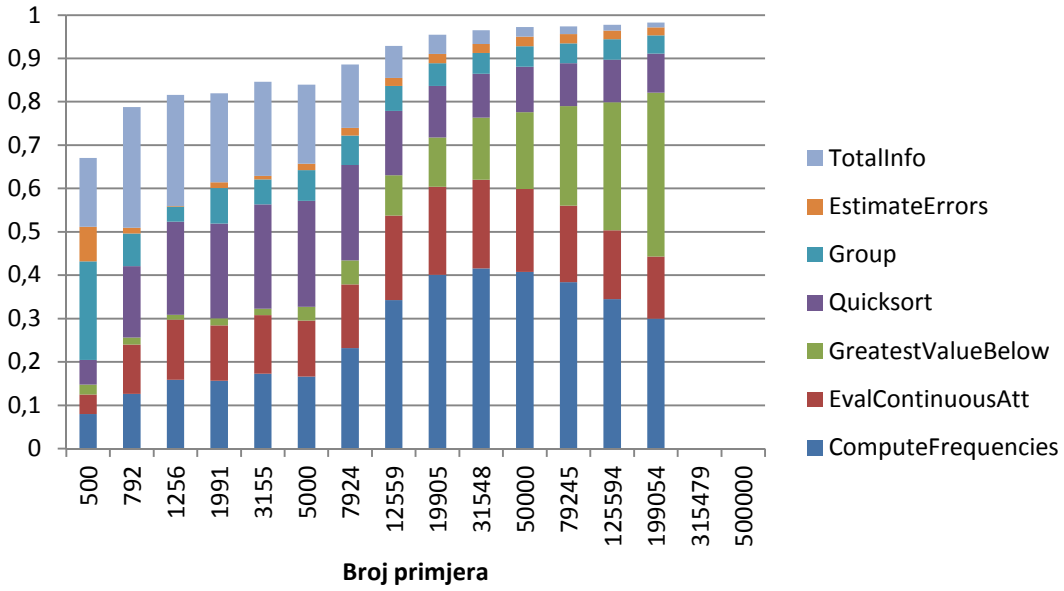
Funkcije ***GreatestValueBelow*** traži najveću vrijednost atributa u skupu za učenje, a koja je manja ili jednaka zadanom pragu. Koristi se pri odabiru vrijednosti praga za podjelu skupa na dva podskupa prema vrijednosti numeričkog atributa.

4.3.2.2 Analiza utjecaja veličine skupa

Rezultati skaliranja veličine skupa dani su na slikama 4.2, 4.3, 4.4 i 4.5. Prikazano je sedam funkcija koje u prosjeku imaju najveći udio u vremenu izvršavanja. Na navedenim slikama prikazano je kako se mijenjaju udjeli pojedinih funkcija u vremenu izvršavanja učenja stabla pri različitim veličinama skupova za učenje.

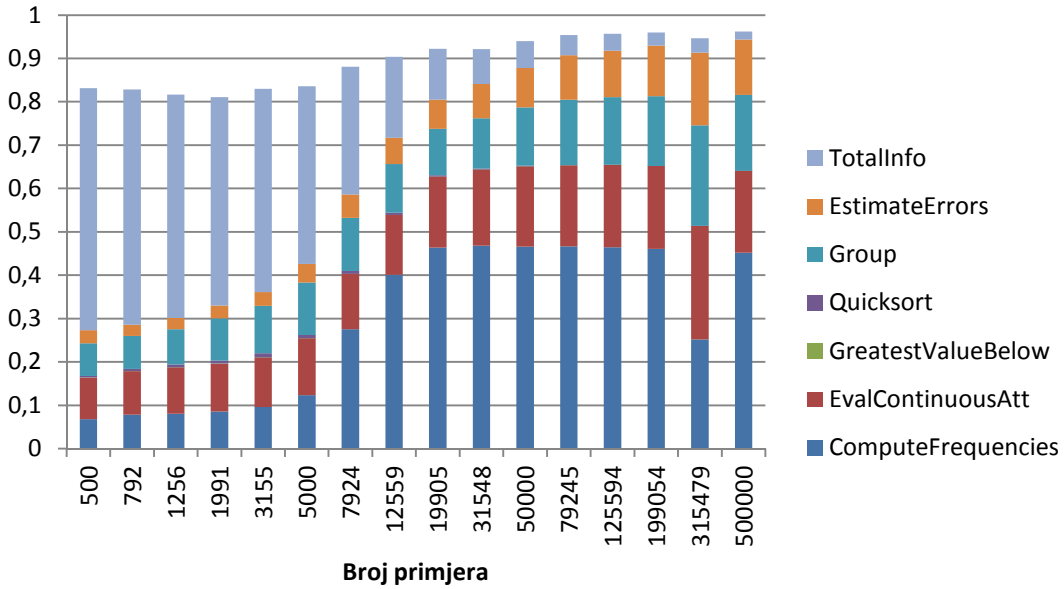


Slika 4.2. Udjeli funkcija u vremenu izvršavanja u odnosu na broj primjera – C4.5, Covertype

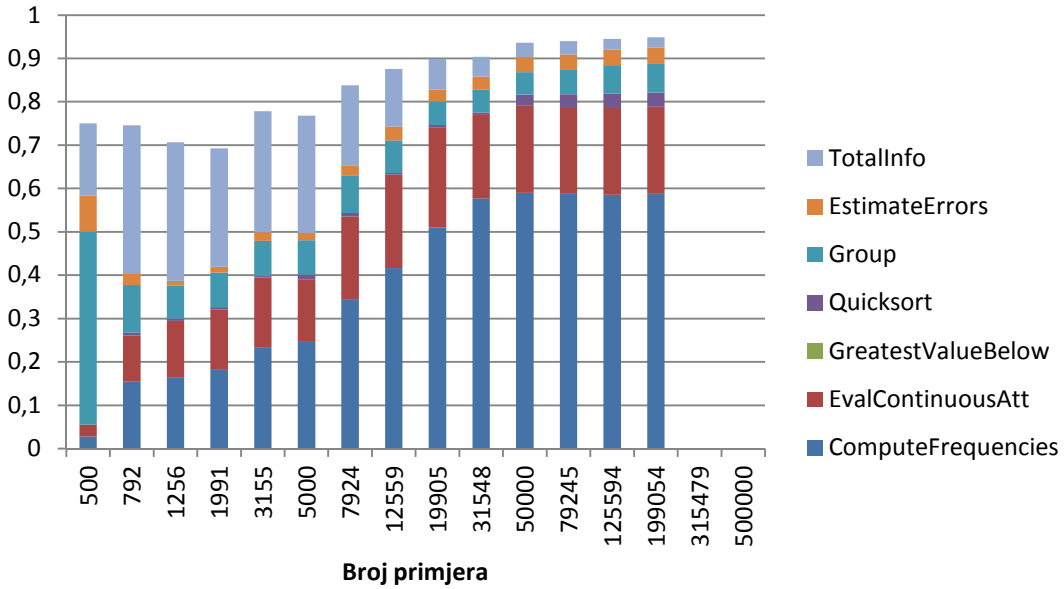


Slika 4.3. Udjeli funkcija u vremenu izvršavanja u odnosu na broj primjera – C4.5, Census-Income

Na slikama 4.2 i 4.3 prikazani su rezultati dinamičke analize za program C4.5. Vidljivo je da na manjim skupovima podataka u vremenu izvršavanja dominiraju funkcije *EvalContinuousAtt*, *QuickSort* i *TotalInfo*. Razlike u udjelima tih funkcija između skupa Covertype i Census-Income posljedica su različitih značajki skupova podataka, tj. omjera broja numeričkih i nominalnih atributa. Povećanjem veličine skupa udjeli *EvalContinuousAtt* i



Slika 4.4. funkcija u vremenu izvršavanja u odnosu na broj primjera – EC4.5, Covertype



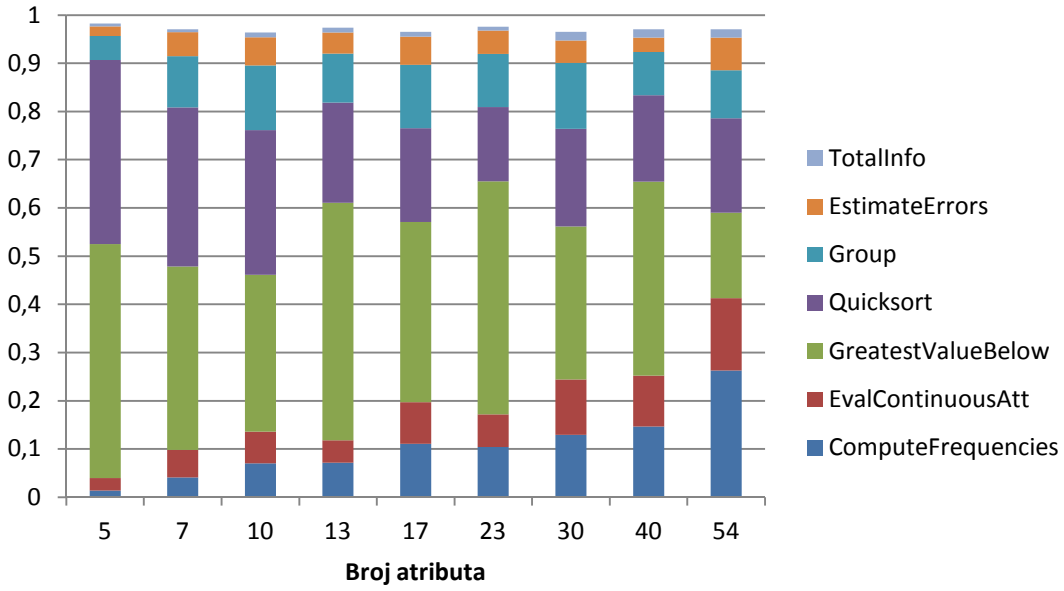
Slika 4.5. funkcija u vremenu izvršavanja u odnosu na broj primjera – EC4.5, Census-Income

QuickSort ostaju približno jednaki, a povećava se udio funkcije *GreatestValueBelow*, koja pri najvećim skupovima dominira u vremenu izvršavanja programa. S obzirom na namjenu te funkcije – pretraživanje u jednodimenzionalnom polju – to ukazuje na neefikasnost samog algoritma za skupove veličine iznad 30.000 elemenata.

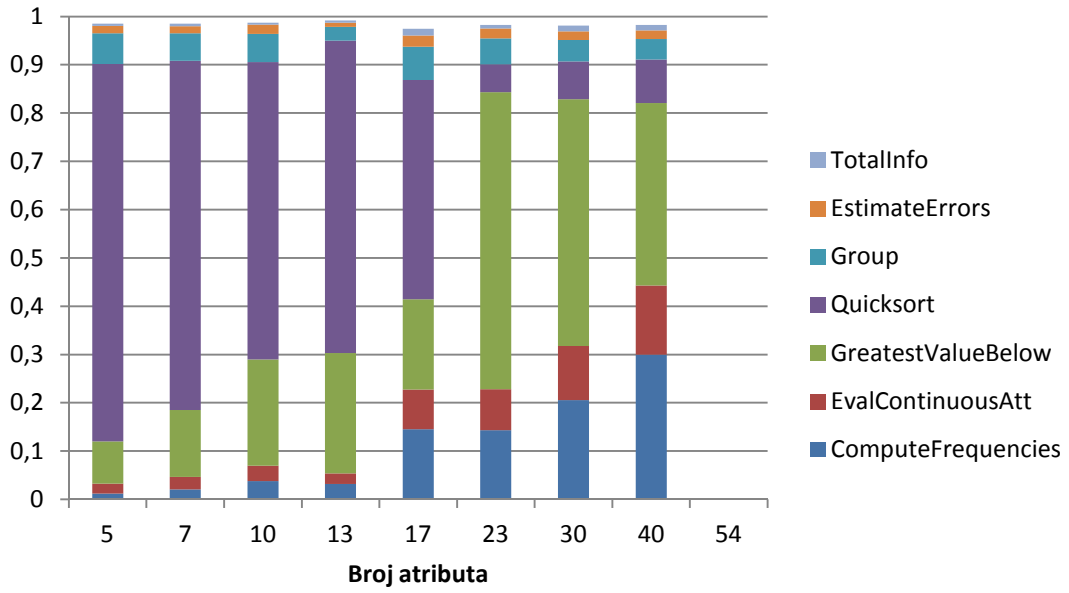
Na slikama 4.4 i 4.5 prikazani su rezultati dinamičke analize za program EC4.5. Pri manjim skupovima dominira funkcija *TotalInfo*, dok se povećanjem skupa povećava udio funkcije *ComputeFrequncies*, koja na najvećim skupovima preuzima dominaciju. Zbog drugačije izvedbe funkcije *GreatestValueBelow* nego što je u C4.5, ona nema značajan utjecaj na vrijeme učenja stabla, bez obzira na veličinu skupa. Funkcije *EvalContinuousAtt* i *Group* imaju približno konstantni udio u vremenu izvršavanja za sve veličine skupova podataka.

4.3.2.3 Analiza utjecaja broja atributa

Rezultati skaliranja veličine skupa dani su na slikama 4.6, 4.7, 4.8 i 4.9. Prikazano je sedam funkcija koje u prosjeku imaju najveći udio u vremenu izvršavanja. Na navedenim slikama prikazano je kako se mijenjaju udjeli pojedinih funkcija u vremenu izvršavanja učenja stabla pri različitom broju atributa u skupovima za učenje.

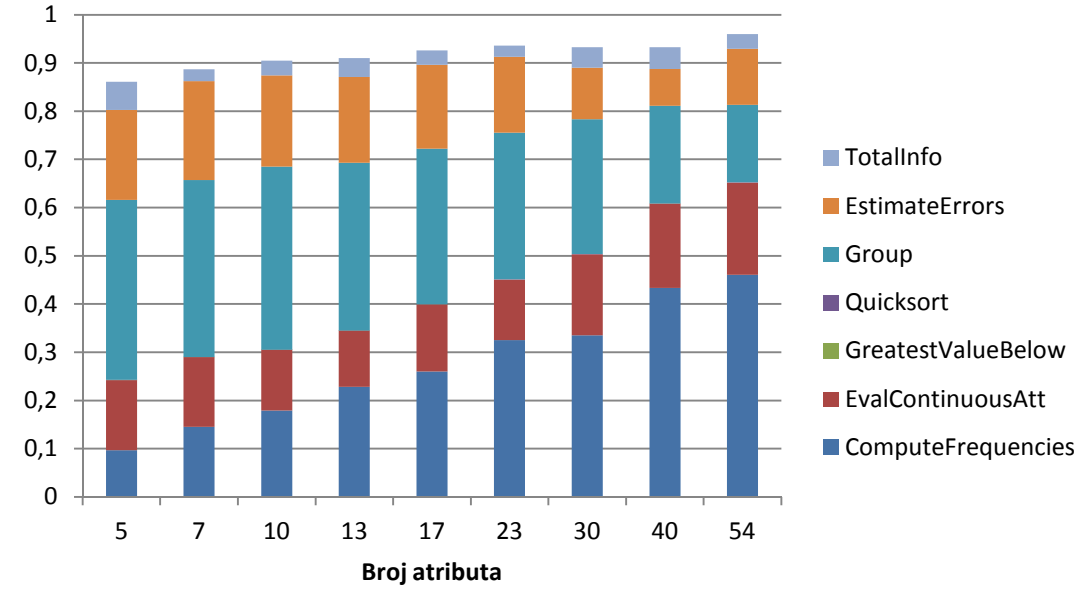


Slika 4.6. Udjeli funkcija u vremenu izvršavanja u odnosu na broj atributa – C4.5, Covertype

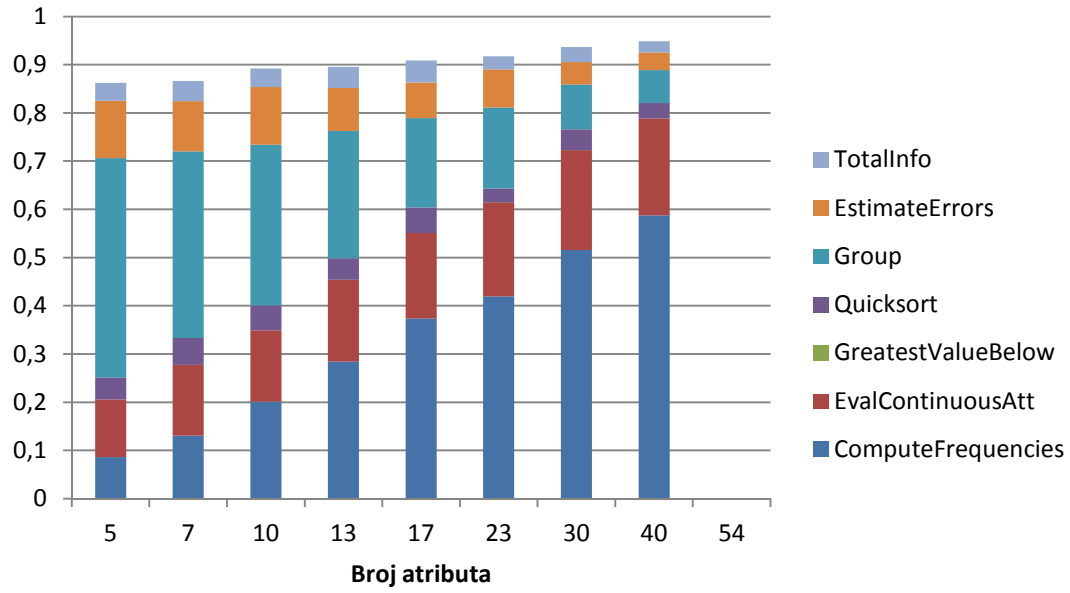


Slika 4.7. Udjeli funkcija u vremenu izvršavanja u odnosu na broj atributa – C4.5, Census-Income

Na slikama 4.6 i 4.7 prikazani su rezultati za program C4.5. U vremenu izvršavanja dominiraju funkcije *GreatestValueBelow*, *QuickSort* i *ComputeFrequencies*, bez obzira na broj atributa. Povećanjem broja atributa najosjetnije raste udio funkcije *ComputeFrequencies*, dok udio funkcije *QuickSort* pada.



Slika 4.8. Udjeli funkcija u vremenu izvršavanja u odnosu na broj atributa – EC4.5, Covertype



Slika 4.9. Udjeli funkcija u vremenu izvršavanja u odnosu na broj atributa – EC4.5, Census-Income

Na slikama 4.8 i 4.9 prikazani su rezultati sza program EC4.5. U vremenu izvršavanja dominiraju funkcije *ComputeFrequencies*, *EvalContinuousAtt* i *Group*, bez obzira na broj atributa. Povećanjem broja atributa najosjetnije raste udio funkcije *ComputeFrequencies*, dok udio funkcije *Group* pada. Funkcija *EvalContinuousAtt* ima približno jednak udio u vremenu izvršavanja za svaki broj atributa.

5 Heterogeni računalni sustav za učenje stabla odluke

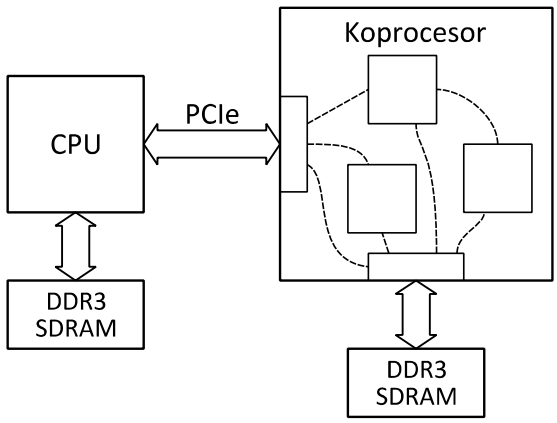
5.1 Heterogeni sustav

Heterogeni računalni sustav sadrži CPU opće namjene i specijalizirani koprocesor. Postoji više načina izvedbe koprocesora, pri čemu je najraširenije korištenje grafičkog procesora. U ovom radu koprocesor je temeljen na FPGA sklopovima.

Na slici 5.1 prikazana je blok shema heterogenog računalnog sustava. Sustav se sastoji od:

- CPU-a opće namjene, koji u sebi ima integriran memorijski kontroler i PCIe sučelje
- koprocesora izvedenog pomoću FPGA-ova, koji s CPU-om komunicira putem PCIe sabirnice i ima sučelje prema DDR3 SDRAM.

Koprocesor je fizički izveden kao kartica koja se spaja na PCIe sučelje računala i na sebi sadrži barem jedan FPGA. Na tržištu postoji više takvih kartica koja su pogodna za korištenje kao koprocesor, te nekoliko cjelovitih rješenja koja osim FPGA kartice uključuju i razvojne alate za definiciju koprocesora i integraciju u programsko rješenje. Neka od cjelovitih rješenja su Convey Computer HC serija [79], Maxeler [80] i Nallatech akceleratorske kartice [81]. Takva cjelovita rješenja znatno olakšavaju izgradnju heterogenog sustava pružajući podršku u vidu osiguravanja temeljne infrastrukture za upravljanje i prijenos podataka između CPU-a i koprocesora. Infrastruktura uključuje sklopovlje kao što su memorijski kontroleri za koprocesorsku memoriju i sučelje prema CPU-u (najčešće PCIe), te programsku podršku u



Slika 5.1. Arhitektura heterogenog računalnog sustava

vidu upravljačkih programa i programskih biblioteka za rad s koprocesorom. Na taj je način glavnina aktivnosti razvoja heterogenog računalnog sustava usmjerena na razradu sklopovske implementacije algoritma.

U ovom radu korišten je heterogeni sustav koji se sastoji od radne stanice s Intel Xeon E5-1650 CPU-om i 16 GiB DDR3-1600 RAM-a, u koju je ugrađena Maxeler Vectis-Lite FPGA platforma. Maxeler platforma koristi se za realizaciju koprocesora za učenje stabla odluke. Realizacija koprocesora uključuje definiciju i implementaciju jezgri za realizaciju pojedinih dijelova algoritma u sklopovlju. U koprocesoru su implementirane jezgre za izračun matrice frekvencija, te za grupiranje primjera u podskupove.

5.2 Maxeler Vectis-Lite FPGA koprocesor

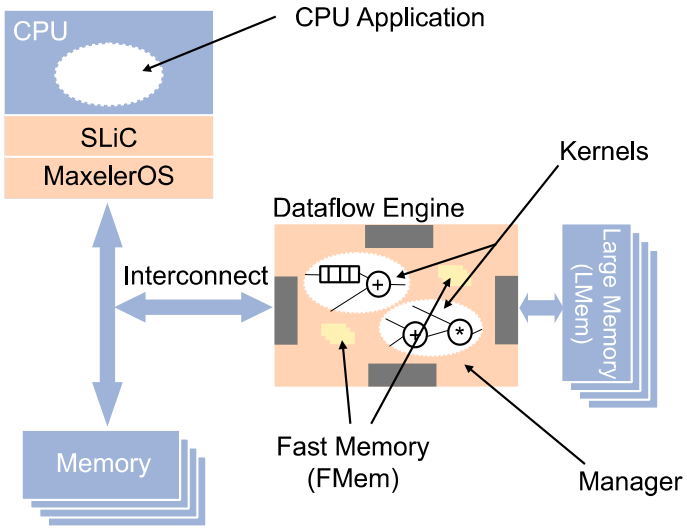
Maxeler platforma oblikovana je za rad prema modelu toka podataka (engl. *data-flow computing model*) [82]. Većina računarskih sustava u praksi oblikovana je prema modelu upravljačkog toka. Upravljački tok označava redoslijed operacija koji se izvršavaju nad ulaznim podacima, i to je način rada procesora opće namjene – niz instrukcija koji se izvršava u zadanom redoslijedu.

Kod modela podatkovnog toka, transformacije nad podacima opisane su u obliku grafa toka podataka. Svaka operacija predstavljena je jednim čvorom u grafu, a njihovi ulazi i izlazi predstavljeni su bridovima. Graf toka podataka može se preslikati u sklopovsku izvedbu gdje je svaka operacija predstavljena jednom jezgrom, a sve jezgre istovremeno izvršavaju svoju operaciju. Podjelom grafa u stupnjeve odvojene registrima on se pretvara u protočnu strukturu koja je prikladna za implementaciju u FPGA-u.

Na slici 5.2 prikazana je opća arhitektura heterogenog računalnog sustava s Maxeler FPGA platformom. Sustav sadrži CPU opće namjene na kojem se izvodi operacijski sustav, a u koji su dodane dvije komponente vezane za rad s Maxeler FPGA platformom: MaxelerOS, i SLiC (engl. *simple live CPU interface*). Maxeler FPGA koprocesor, na slici označen s DFE (engl. *dataflow engine*), spojen je u računalo putem PCIe sabirnice. DFE ima pristup vlastitoj memoriji (*LMem*).

Osnovne značajke Maxeler Vectis-Lite kartice:

- DFE FPGA: Xilinx Virtex-6 XC6VSX475T
- RAM (*LMem*): 6 GiB DDR3-800 SDRAM



Slika 5.2. Općenita arhitektura heterogenog sustava s Maxeler FPGA platformom
(prilagođeno prema [88])

- RAM na FPGA (*FMem*): 4 MiB (Block RAM)
- Propusnost CPU ↔ FPGA: 2 GB/s
- Propusnost *LMem* ↔ FPGA: 38,4 GB/s

MaxelerOS i SLiC osiguravaju programsko sučelje između programa koji se izvršava na CPU-u i sklopovlja na FPGA-u. MaxelerOS upravlja sustavom na nižoj razini, osigurava sistemske pozive kojima sučelja više razine šalju podatke prema kartici i primaju podatke s nje. On upravlja i procesom konfiguracije FPGA-a, te ga pokreće prema potrebi. SLiC je programsko sučelje više razine koje se oslanja na MaxelerOS za izvršavanja funkcija. Programi putem sučelja komuniciraju s Maxeler sustavom u obliku funkcijskih poziva.

5.2.1 MaxCompiler

Arhitektura koprocesora opisuje se u Javi, korištenjem paketa i klasa definiranih za Maxcompiler. Maxcompiler tekstualni opis u Javi prevodi u VHDL. VHDL tekst je ulaz u Xilinx alate koji izvode postupak logičke sinteze, tehnološkog mapiranja, raspoređivanja i trasiranja, te generiranja binarnog oblika konfiguracije za programiranje FPGA-a.

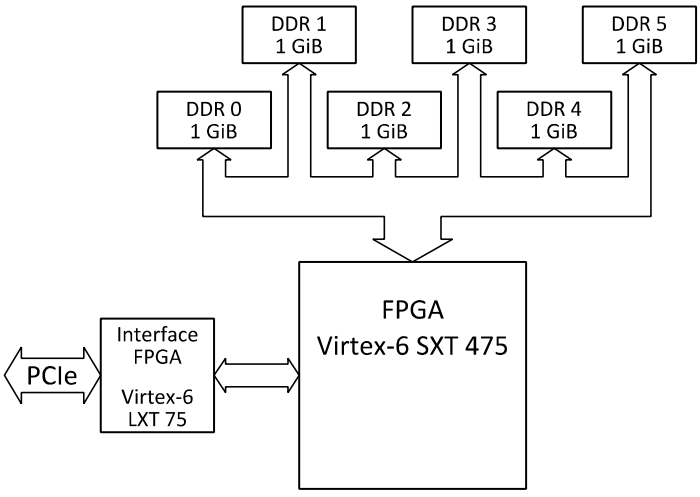
Maxcompiler, uz VHDL tekst, generira programsko sučelje koje se sastoji od zaglavne datoteke i objektnog programa ili dijeljene biblioteke, koja implementira funkcijske pozive potrebne za korištenje koprocesora. Maxcompiler izravno podržava uključivanje generiranog programskog sučelja za programske jezike C i FORTRAN. Putem omotača (engl. *wrapper*, *SLiC skins*) podržava programske sustave R i MATLAB, te programski jezik Python.

5.2.2 Sklopovska arhitektura Maxeler Vectis-Lite kartice

Pojednostavljena sklopovska arhitektura Maxeler Vectis-Lite kartice prikazana je na slici 5.3. Virtex-6 SXT 475 je glavni FPGA kojim se implementira koprocesor. Na FPGA je izravno spojeno šest memorijskih modula, svaki veličine 1 GiB. Glavni FPGA s računalnim sustavom komunicira preko veznog FPGA (engl. *interface FPGA*), koji implementira sučelje prema PCIe x8 sabirnici računala.

Iako PCIe x8 sabirnica omogućuje propusnost od 4 GB/s u svakom smjeru, stvarna propusnost ograničena je sučeljem između veznog i glavnog FPGA-a. Propusnost sučelja između dva FPGA-a iznosi 2 GB/s i to je konačna propusnost prijenosa podataka prema računalu.

Memorijski moduli koji su spojeni na glavni FPGA su tipa DDR3-800 SO-DIMM. Svaki modul ima širinu podatkovne sabirnice iznosa 64 bita, spojen je izravno na FPGA i ne dijeli podatkovne linije s drugim modulima. Takav način spajanja daje ukupnu širinu memorijske sabirnice iznosa 384 bita. Svi moduli adresiraju se zajednički. Minimalnim ciklusom čitanja ili pisanja pristupa se bloku od 16 bajtova u svakom modulu. Sa zajedničkim adresiranjem svih modula, u minimalnom ciklusu pristupa se bloku od 96 bajtova. Memorijski kontroler izveden u FPGA-u prihvaća adrese u bajtovima, ali ne pruža mogućnost izdvajanja pojedinih bajtova iz primljene riječi od 96 bajtova. Zbog navedenog ograničenja adresiranje je ograničeno na memorijske lokacije koje su višekratnik broja 96, pa se sva čitanja i pisanja izvode u blokovima od 96 bajtova.



Slika 5.3. Sklopovska arhitektura Maxeler Vectis-Lite kartice

5.3 Arhitekture jezgara

Iz provedene dinamičke analize programa C4.5 i EC4.5, opisanoj u poglavlju 4.3, vidljivo je da se kod većih skupova za učenje najviše vremena izvršavanja programa troši na:

- izračun tablice frekvencija, kod nominalnih atributa
- izračun informacijskog dobitka za svaku moguću podjelu skupa, kod numeričkih atributa.

Efikasna obrada numeričkih atributa zahtijeva korištenje sortiranja. Ono se izvršava u svakom čvoru stabla za svaki numerički atribut. Budući da bi proces sortiranja na koprocesoru zahtijevao višestruka kopiranja cijelog obrađivanog skupa, obrada numeričkih atributa nije pogodna za implementaciju na FPGA. Nasuprot numeričkim, obrada nominalnih atributa ne zahtijeva sortiranje podskupa po vrijednostima obrađivanog atributa. Glavnina vremena troši se u jednoj kratkoj petlji koja uzima primjere prema zatečenom redoslijedu i pribraja njihove težine na odgovarajuća mjesta u tablici. Ovakva petlja pogodna je za implementaciju na FPGA prema modelu podatkovnog toka i iz tog se razloga na FPGA platformi implementira samo obrada nominalnih atributa.

Sklopovska obrada nominalnih atributa izvedena je uz sljedeća ograničenja:

- za prijenos podataka u i iz koprocesora dostupno je ukupno 15 podatkovnih tokova
- pristup SDRAM-u iz FPGA-a je eksplicitno programiran korištenjem predefiniranih naredbi ili generiranjem korisničkih naredbi za memorijski kontroler
- adresiranje, kao i čitanje i pisanje u SDRAM vrši se u blokovima od 96 bajtova
- u FPGA-u nema sustava priručne memorije (engl. *cache*), pa sva kašnjenja pri pristupu SDRAM-u ostaju vidljiva
- veza CPU ↔ FPGA je niske propusnosti (2 GB/s) u odnosu na vezu FPGA ↔ SDRAM (do 38,4 GB/s).

Navedena ograničenja utječu na konačnu arhitekturu jezgri, kao i cijelog koprocesora. Ona utječu na broj paralelnih podatkovnih tokova koji se mogu koristiti pri obradi, učestalost prijenosa podataka između CPU-a i koprocesora, redoslijed pristupa podacima u SDRAM-u, te na podatkovnu strukturu za pohranu skupa za učenja.

Sklopovska implementacija obrade nominalnih atributa, uz implementaciju izračuna matrice frekvencija, uključuje i implementaciju grupiranja primjera. Grupiranje primjera je korak

algoritma koji se provodi nakon odabira testnog atributa, a izvršava podjelu ulaznog skupa na podskupove. Izvršavanje grupiranja na CPU-u zahtijevalo bi opetovano prenošenje podataka iz CPU-a u koprocesor. Budući da je propusnost veze između CPU-a i koprocesora relativno niska, opetovano prenošenje podataka tim kanalom loše je za ukupne performanse sustava. Stoga se grupiranje primjera mora izvesti i u sklopovlju. U konačnici se grupiranje nominalnih atributa izvršava na koprocesoru, dok se grupiranje numeričkih atributa izvršava na CPU-u.

5.3.1 Jezgra za izračun matrice frekvencija (*ComputeFreq*)

Matrice frekvencija izračunavaju se za svaki atribut posebno. Sam izračun je jednostavna petlja:

```
ForEach(p, Fp, Lp)
{
    Case = Item[p];
    Freq[ DVal(Case,Att) ][ Class(Case) ] += Weight[p];
}
```

Petlja uzima primjere u zatečenom redoslijedu i pribraja težine trenutnoj sumi težina u matrici *Freq* u retku *DVal* i stupcu *Class*, gdje je *DVal* vrijednost atributa, a *Class* je klasa primjera. Ako skup za učenje nema nedostajućih vrijednosti, težine primjera iznose 1,0 i ne mijenjaju se za vrijeme gradnje stabla. Težine su relevantne samo za skupove koji imaju nedostajuće vrijednosti, tj. primjere s nepoznatom vrijednošću nekih atributa. Budući da je definirano da hibridni algoritam radi na skupovima bez nedostajućih vrijednosti, izgradnja matrice frekvencija svodi se na prebrojavanje primjera koji imaju određenu vrijednost atributa i pripadaju određenoj klasi. Ovaj uvid omogućuje sljedeću optimizaciju: vrijednosti u tablici frekvencije su cijeli brojevi, veći ili jednaki nuli, što povlači primjenu cjelobrojne aritmetike bez predznaka (engl. *unsigned integer*).

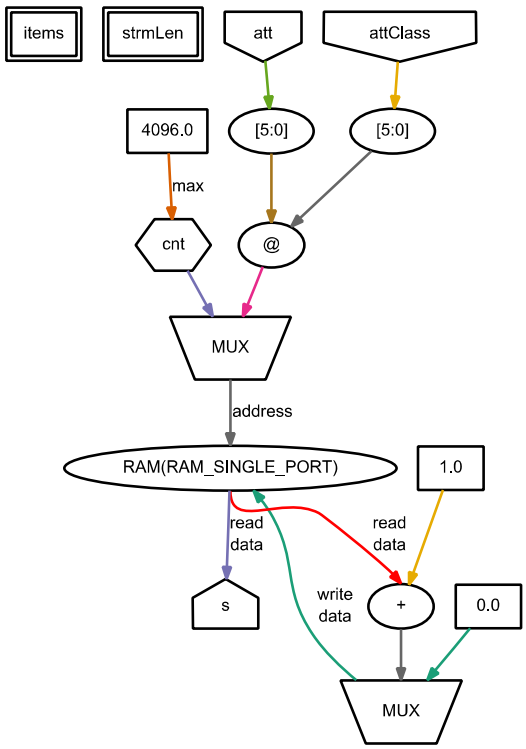
Jedna matrica frekvencije izračunava se za jedan atribut. Nužni ulazi u jezgru su jedan tok podataka za vrijednosti atributa, i jedan tok podataka za klase primjera. Ulazni tokovi su sinkroni, tj. vrijednost atributa i oznaka klase koji su istovremeno postavljeni na ulaz jezgre pripadaju istom primjeru.

Jezgra za izračun matrice frekvencija izvedena je u tri varijante: SS, MS i MS2. U nastavku je opisana svaka od tih izvedbi.

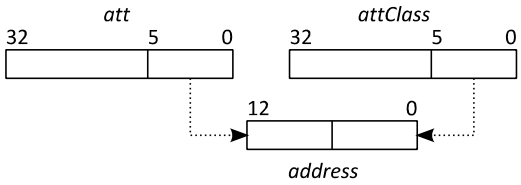
5.3.1.1 Jezgra s jednostrukim tokom atributa – SS

Jezgra SS (engl. *single-stream*) [83] je minimalna jezgra s jednostrukim tokom atributa. Jezgra ima dva ulazna podatkovna toka: za vrijednosti atributa (*att*) i za oznake klase (*attClass*), dva skalarna ulaza: za broj primjera (*items*) i za duljinu podatkovnog toka (*strmLength*), te jedan izlazni podatkovni tok za izračunate vrijednosti frekvencija (*s*). Duljina ulaznog podatkovnog toka može biti veća od broja primjera koji se obrađuju i u tom se slučaju preostali ulazni elementi zanemaruju. Arhitektura jezgre prikazana je na slici 5.4. Radi jasnoće prikaza na slici nisu prikazani upravljački signali.

Iz dva ulazna toka izdvaja se određeni broj najnižih bitova iz kojih se formira tok adresa (*address*) za memoriju (*RAM*) u koju je pohranjena matrica frekvencija. Bitovi izdvojeni iz toka *attClass* postaju niži, a iz toka *att* viši bitovi adrese, kao što je prikazano na slici 5.5. Posljedično, na uzastopnim memorijskim lokacijama pohranjene su vrijednosti za različite vrijednosti atributa, a iste oznake klase, kao što je prikazano na slici 5.6. Zbog načina formiranja adrese, broj redaka i stupaca matrice frekvencije, odnosno najveći broj jedinstvenih vrijednosti atributa i oznaka klasa, uvijek je potencija broja dva.



Slika 5.4. Arhitektura *ComputeFreq* SS jezgre za izračun matrice frekvencija



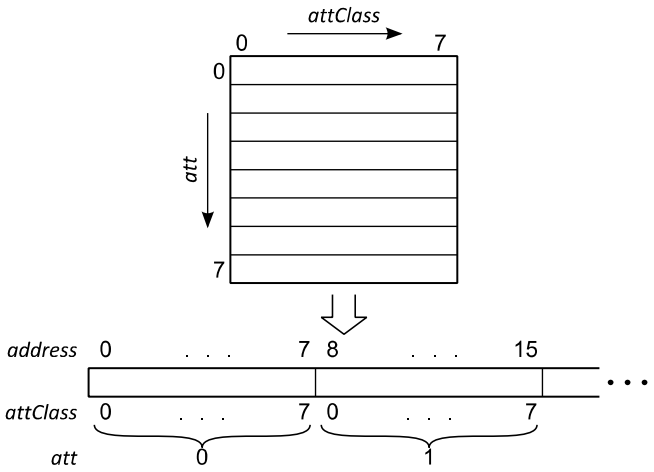
Slika 5.5. Formiranje adrese za memoriju matrice frekvencija

Tok *address* adresira matricu. Pročitana vrijednost na trenutnoj adresi uvećava se za jedan i zapisuje na istu lokaciju, čime se ažurira stanje matrice frekvencija. Nakon što su pročitani svi primjeri, vrijednosti matrice frekvencija čitaju se iz memorije u izlazni podatkovni tok *s*. Adrese za čitanje generiraju se brojilom *cnt* koje slijedno adresira sve memorijske lokacije. Izlaz iz jezgre je matrica frekvencija u linearnom obliku, zapisan po oznakama klasa.

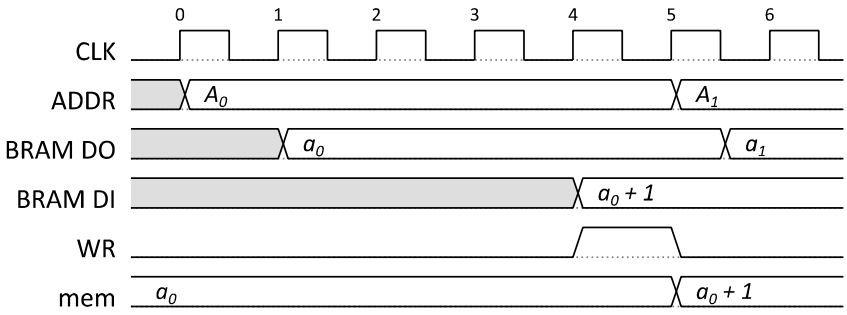
Istovremeno s čitanjem, sadržaj memorije briše se upisivanjem nule. Memorija je tipa *read-first*, što znači da se pri pisanju u memoriju na podatkovni izlaz postavlja stara vrijednost adresirane lokacije. Takav način rada omogućuje jednostavno istovremeno čitane i brisanje postavljanjem nule na podatkovni ulaz i omogućavanjem pisanja. Na adresiranu lokaciju upisuje se nula, dok se na podatkovnom izlazu pojavljuje stari sadržaj lokacije.

Veličina memorije određena je najvećim brojem jedinstvenih vrijednosti nominalnog atributa, te najvećim brojem klasa koji se očekuju u skupovima za učenje. Odabrana vrijednost je 64 za oba parametra, što znači da broj lokacija memorije iznosi 4096. Odabrane vrijednosti su dovoljne za većinu javno dostupnih skupova podataka. Širina riječi memorije iznosi 32 bita.

Pri prevođenju opisa jezgre u VHDL model koji je pogodan za sintezu u FPGA konfiguraciju,



Slika 5.6. Zapis matrice frekvencija u memoriji za osam vrijednosti atributa i osam klasa



Slika 5.7. Vremenski dijagram rada s memorijom prilikom obrade jednog para atribut-klasa

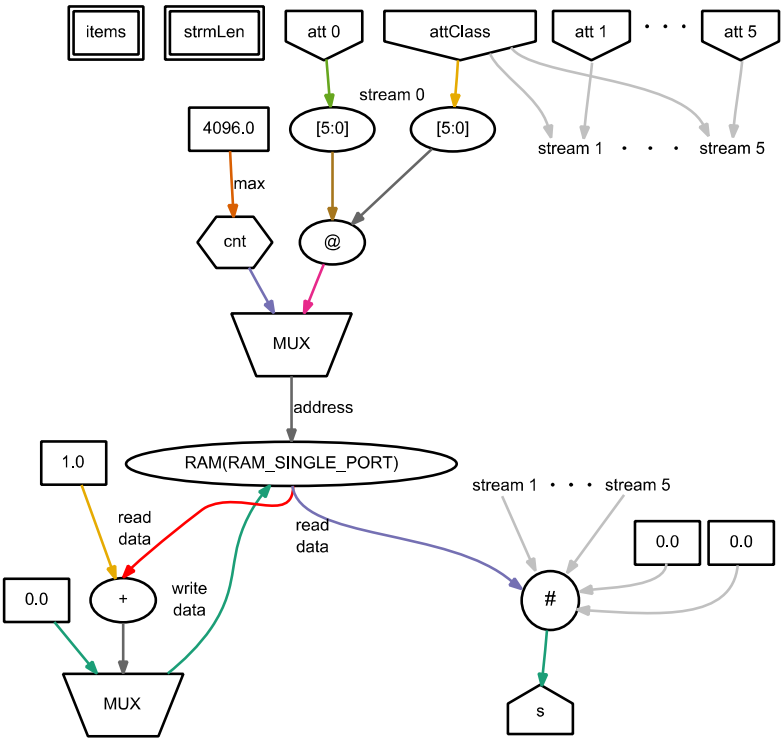
prevodilac automatski stavlja registre tamo gdje je to prikladno. Svrha postavljanja registara je odvajanje pojedinih dijelova sklopovlja kako bi se dobila protočna struktura koja može raditi na višoj frekvenciji takta. Kao posljedica pojavljuje se vremenski pomak (kašnjenje) između čitanja iz memorije i dolaska nove vrijednosti na ulaz za upis u memoriju. U konačnoj implementaciji jezgre taj pomak iznosi četiri ciklusa (perioda takta). Uz dodatni ciklus potreban za upis nove vrijednosti u memoriju ukupno vrijeme potrebno za obradu jednog para atribut-klasa iznosi pet ciklusa. Vremenski pomak ilustriran je dijagramom na slici 5.7. U skladu s potrebnim vremenom za obradu jednog ulaznog para, nova vrijednost iz ulaznog toka uzima se tek na svaki peti ciklus, iako ona može biti dostupna na svaki ciklus.

Ova jezgra uspješno je implementirana za rad na frekvenciji takta 333 MHz. Uz obradu podataka na svaki peti ciklus, najveća teoretska propusnost iznosi $66{,}6 \times 10^6$ parova atribut-klasa u sekundi.

5.3.1.2 Jezgra s višestrukim tokom atributa – MS

S obzirom na raspoloživu propusnost LMem, postoji mogućnost ostvarivanja više podatkovnih tokova prema njoj. Jezgra MS (engl. *multi-stream*) ima više ulaznih podatkovnih tokova za atribute, te jedan ulazni podatkovni tok za oznake klase. Tok oznaka klasa zajednički je i koristi se u svim tokovima za adresiranje memorija koje pohranjuju matrice frekvencija.

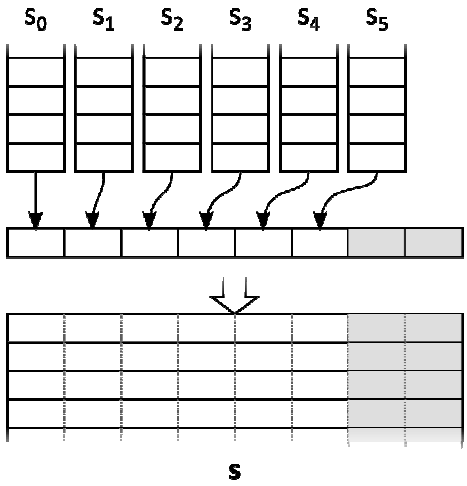
Arhitektura jezgre prikazana je na slici 5.8. Radi jasnoće prikaza na slici nisu prikazani upravljački signali, a prikazana je struktura samo jednog toka atributa. Ostali tokovi atributa su u strukturi identični. Na slici je prikazano i popunjavanje izlaznog vektorskog toka na širinu od osam elemenata. Popunjavanje se provodi zbog efikasnije pretvorbe izlaznog toka na širinu riječi od 128 bitova, koja je određena protokolom komunikacije putem PCIe sabirnice.



Slika 5.8. Arhitektura *ComputeFreq* MS jezgre

Način rada jezgre je isti kao kod jezgre SS. Razlika je isključivo u broju ulaznih podatkovnih tokova, broju memorija, te formatu izlaznog podatkovnog toka. Izračunate vrijednosti frekvencija iz svakog toka spajaju se u jedinstveni vektorski tok *s* u kojem svaki element pripada jednom toku atributa, kao što je prikazano na slici 5.9.

Za izvedbu jezgre s višestrukim tokom korišteni su isti parametri kao i za jezgru s jednostrukim tokom: do 64 jedinstvenih vrijednosti atributa i klase. S obzirom na ograničenje



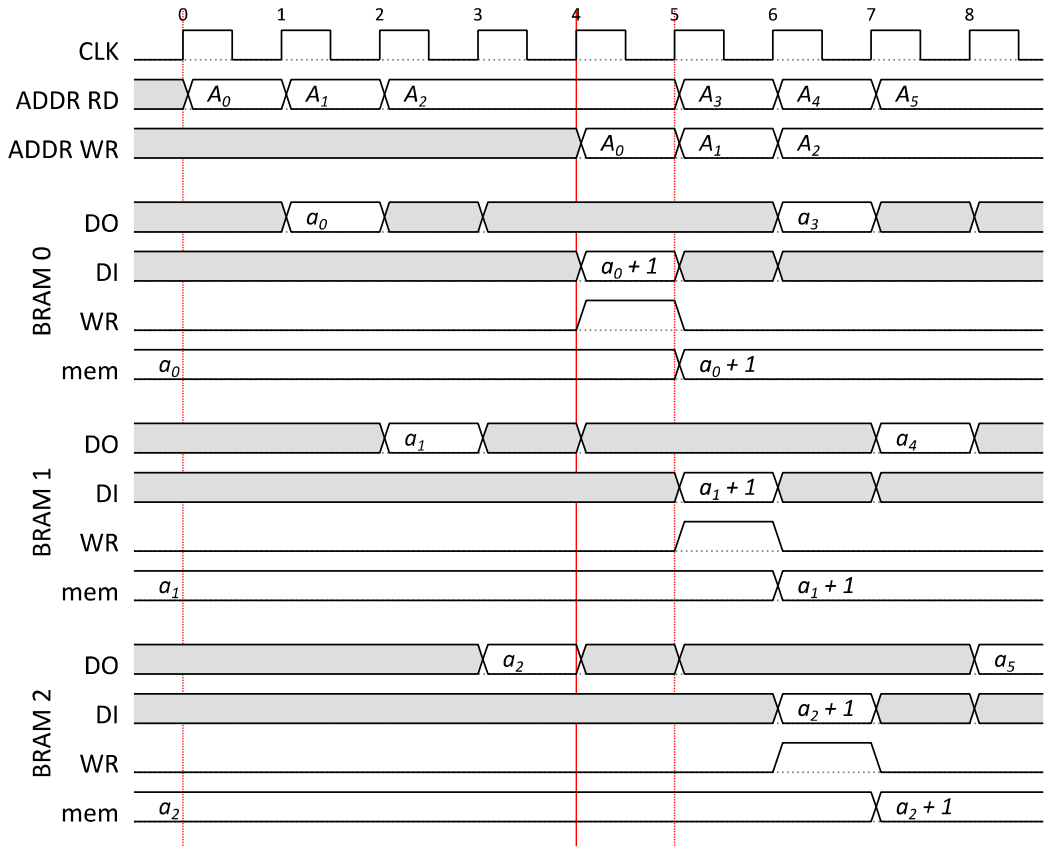
Slika 5.9. Spajanje šest izlaznih tokova matrice frekvencije u jedan vektorski tok

ukupnog broja podatkovnih tokova koji ulaze/izlaze u/iz DFE-a, te na druge jezgre koje su sastavni dio DFE, ukupni broj ulaznih podatkovnih tokova iznosi sedam. Od toga je šest za tokove vrijednosti atributa, a jedan za tok oznaka klasa. Tokovi su sinkroni, što znači da sve vrijednosti atributa i oznaka klase koji su istovremeno prihvaćeni pripadaju istom primjeru. U konačnoj izvedenoj arhitekturi ukupno vrijeme za obradu jednog para atribut-klasa iznosi pet ciklusa, što znači da jezgra u svakom toku obrađuje jedan par svakih pet ciklusa.

Jezgra s višestrukim tokom podataka uspješno je implementirana za rad na frekvenciji takta 300 MHz. Uz obradu jednog podatka na pet ciklusa, i uz šest paralelnih tokova atributa, najveća teoretska propusnost iznosi 360×10^6 parova u sekundi.

5.3.1.3 Jezgra s višestrukim tokom atributa i vremenski pomaknutim parcijalnim matricama – MS2

Uz višestruki tok, koji omogućuje istovremenu obradu više parova atribut-klasa, izveden je i izračun matrice frekvencija pomoću vremenski pomaknutih (engl. *staggered*) memorija. Višestrukim ponavljanjem strukture RAM – zbrajalo – multipleksor izvodi se kompenzacija vremenskog pomaka između čitanja vrijednosti iz memorije i upisa nove vrijednosti.



Slika 5.10. Vremenski dijagram rada memorija s vremenskim pomakom

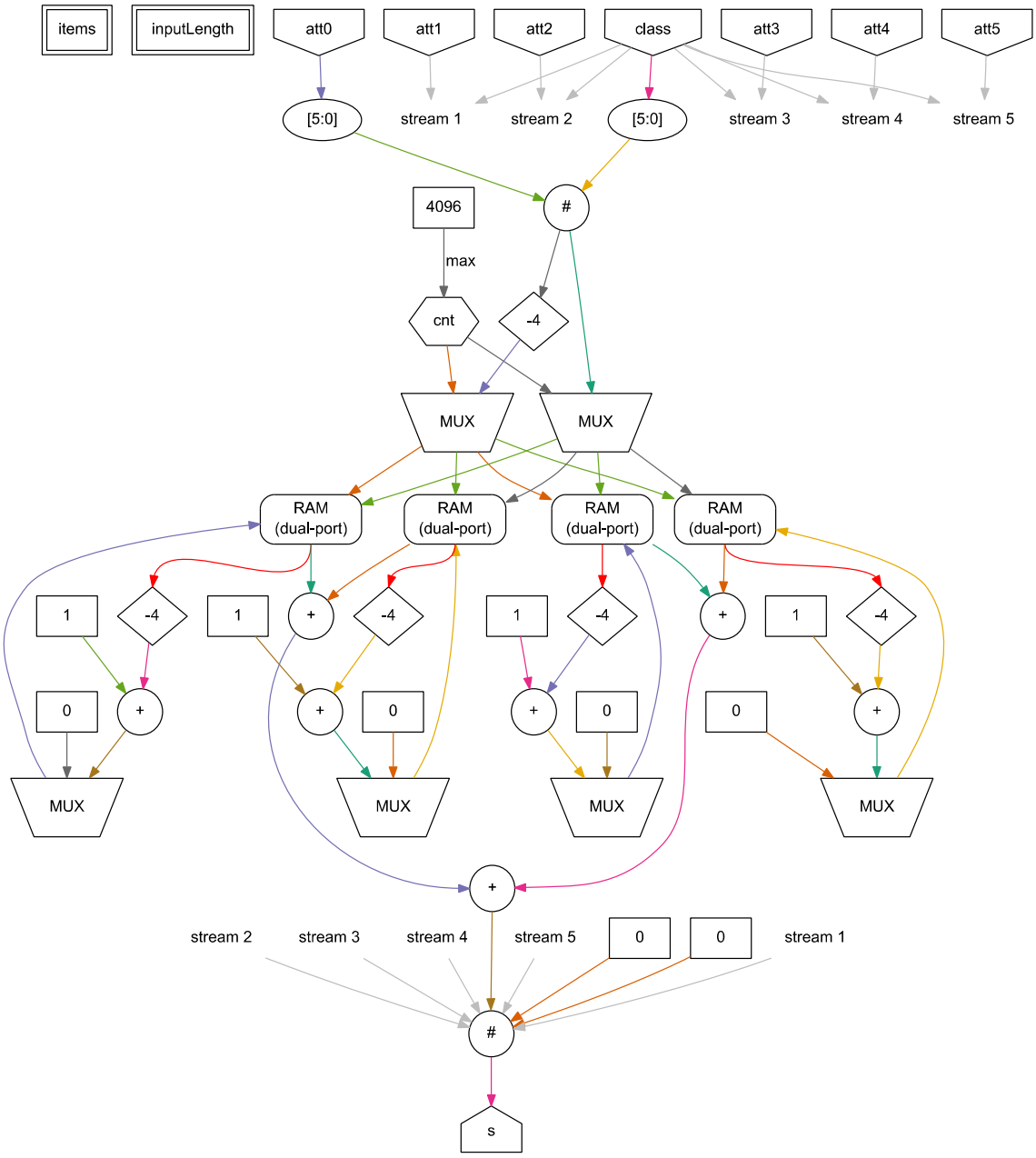
Vremenski pomak unutar svake pojedine strukture ostaje nepromijenjen. Rad pojedinih struktura međusobno je vremenski pomaknut. Svaki ciklus po jedna struktura prihvati po jedan ulazni par atribut-klasa. Broj struktura ograničen je ukupnim brojem ciklusa takta potrebnim za obradu jednog para atribut-klasa. U ovakvom sustavu višestrukih memorija, svaka memorija sadrži parcijalnu matricu frekvencija, a konačna matrica frekvencija dobije se zbrajanjem parcijalnih matrica.

Za pohranu matrice frekvencija koristi se *dual-port* RAM, koji omogućuje istovremeno adresiranje dvije lokacije iste memorije. Jedan adresni ulaz služi za adresiranje lokacije za čitanje, dok drugi služi za adresiranje lokacije za pisanje. Adresni ulazi međusobno su pomaknuti u vremenu za isti broj ciklusa koliko iznosi vremenski pomak između čitanja i pojave nove vrijednosti za pisanje. Rad arhitekture s vremenskim pomaknutim memorijama ilustriran vremenskim dijagramom na slici 5.10, za sustav s vremenskim pomakom od četiri ciklusa i tri memorije.

Način rada jezgre sličan je radu jezgara SS i MS. Razlika je u korištenju vremenski pomaknutih memorija za pohranu parcijalnih matrica frekvencija. Prilikom čitanja izračunatih vrijednosti matrica frekvencija, adrese se generiraju brojilom *cnt* koje slijedno adresira sve memorijske lokacije svih memorija u jezgri. Pri čitanju se parcijalne frekvencije za pojedine atribute zbrajaju u konačni rezultat.

Za izvedbu jezgre s višestrukim tokom korišteni su isti parametri kao i za jezgru MS: do 64 jedinstvenih vrijednosti atributa i klase, te šest tokova vrijednosti atributa. Za kompenzaciju vremenskog pomaka u svakom podatkovnom toku koriste se četiri memorije, što omogućuje obradu četiri para atribut-klasa u četiri uzastopna ciklusa. U konačnoj izvedenoj arhitekturi ukupno vrijeme za obradu jednog para atribut-klasa iznosi pet ciklusa, što znači da jezgra u svakom toku obrađuje četiri para svakih pet ciklusa. Arhitektura jezgre prikazana je na slici 5.11. Radi jasnoće prikaza, na slici nisu prikazani upravljački signali i detaljno je prikazana struktura samo jednog toka atributa. Ostali tokovi atributa u strukturi su identični. Na slici je prikazano i popunjavanje izlaznog vektorskog toka na širinu od osam elemenata.

Jezgra s višestrukim tokom podataka i vremenski pomaknutim parcijalnim matricama uspješno je implementirana za rad na frekvenciji takta 200 MHz. Uz obradu četiri podatka na pet ciklusa, i uz šest paralelnih tokova atributa, najveća teoretska propusnost iznosi 960×10^6 parova u sekundi.



Slika 5.11. Arhitektura *ComputeFreq* MS2 jezgre

5.3.2 Jezgra za grupiranje primjera (*GroupItems*)

Grupiranje primjera je operacija kojom se izvodi jedan od ključnih dijelova algoritma – podjela skupa na podskupove. Grupiranje je korak koji se provodi nakon obrade svih atributa (izračuna informacijskog dobitka) i odabira testnog atributa čvora. Ulazni skup čvora dijeli se prema vrijednosti testnog atributa na dva ili više podskupova, koji postaju ulazi za čvorove sljedeće razine stabla (čvorove djecu).

Izvedba grupiranja primjera na CPU-u uključivala bi stvaranje podskupova u CPU memoriji i njihovo prenošenje u FPGA memoriju. Taj bi se postupak provodio u svakom čvoru stabla. S obzirom na to da je propusnost veze CPU ↔ FPGA relativno niska (2 GB/s), opetovano prenošenje podataka imalo bi negativan utjecaj na ukupne performanse sustava. Stoga se grupiranje primjera izvodi i u sklopovlju, gdje se može iskoristiti znatno veća propusnost FPGA memorije (do 38,4 GB/s).

Na CPU-u grupiranje se izvodi promjenom redoslijeda u polju pokazivača na primjere, a sami primjeri se ne pomiču. Pristup vrijednostima atributa pojedinih primjera izvodi se indirektnim adresiranjem preko polja pokazivača. Taj pristup nije pogodan za model toka podataka. U modelu toka podataka svi primjeri koji pripadaju istoj grupi moraju se u memoriji nalaziti u jednoj kompaktnoj skupini. Posljedica toga je da se grupiranje u FPGA memoriji izvodi kopiranjem primjera iz izvorne skupine u više odredišnih skupina, koja svaka ima rezerviran svoj blok memorije.

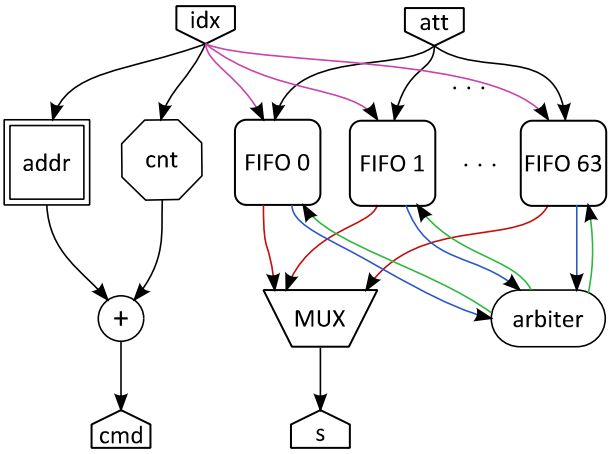
Jezgra ima minimalno dva ulaza. Barem jedan ulaz za podatkovni tok atributa, oznaka klasa i identifikacijskih brojeva primjera, te jedan za oznake grupa. Tok oznaka zapravo je tok vrijednosti testnog atributa kojim se određuje u koju će grupu biti kopiran trenutni primjer. Svi ulazni tokovi su sinkroni, što omogućuje jedan zajednički tok oznaka grupa za usmjeravanje više ulaznih tokova vrijednosti atributa.

Uz ulazne podatkovne tokove, jezgra ima i polje skalarnih ulaza za početne adrese grupa. Početne adrese koriste se za izračun adresa odredišnih lokacija u koje se usmjeravaju vrijednosti koje pripadaju odgovarajućim grupama. Broj početnih adresa određen je brojem ulaznih podatkovnih tokova atributa, te brojem očekivanih jedinstvenih vrijednosti atributa. Ako se neke vrijednosti atributa ne pojavljuju, što je poznato iz značajki skupa za učenje, odgovarajuće početne adrese grupa mogu se postaviti na proizvoljnu vrijednost.

Arhitektura jezgre prikazana je na slici 5.12, a sastoji se od:

- više FIFO među-spremnika
- konačnog automata stanja za upravljanje pisanjem i čitanjem FIFO među-spremnika
- generatora naredbi za memorijski kontroler.

Broj FIFO međuspremnika jednak je očekivanom najvećem broju jedinstvenih vrijednosti atributa. Ulazi FIFO spremnika adresirani su tokom indeksa, tako da vrijednost toka indeksa omogućuje upis primljene vrijednosti atributa u odgovarajući FIFO. Kada se bar jedan FIFO



Slika 5.12. Arhitektura jezgre *GroupItems*

spremnik napuni s dovoljno elemenata, aktivira se zahtjev za pisanje u memoriju i zaustavi ulazni tok podataka. Zaustavljanje ulaznog toka nužno je radi sprečavanja gubitka podataka zbog preljeva spremnika.

Zahtjevi za pisanje ulaze u arbitar, koji generira aktivacijski signal za izlaz FIFO spremnika. Aktivacijski signal u jednom trenutku ima aktivan samo jedan bit, tako da se u jednom trenutku može čitati iz samo iz jednog FIFO spremnika. Arbitarska funkcija pridjeljuje viši prioritet spremnicima koji zadržavaju elemente nižeg iznosa oznake, tj. niže vrijednosti testnog atributa. Aktiviranjem izlaza FIFO spremnika pokreće se pisanje njegova sadržaja u SDRAM. Kada je završeno čitanje bloka podataka iz spremnika, deaktivira se zahtjev za pisanje i arbitar generira novi aktivacijski signal.

Ako je aktivirani FIFO jedini koji ima aktivan zahtjev za pisanje, ulazni tok vrijednosti atributa ponovno se pokrene jer nema opasnosti od preljeva. Ako još barem jedan FIFO ima aktivan zahtjev za pisanje, ulazni tok podataka se zaustavlja i ostaje zaustavljen dok se ponovno ne uspostave uvjeti za njegovo pokretanje: nema aktivnih zahtjeva za pisanje, ili je jedini aktivni zahtjev od FIFO spremnika iz kojeg se trenutno čita.

Izlaz arbitra ujedno je i upravljački ulaz multipleksora kojim se odabire izlaz FIFO spremnika za izlazni podatkovni tok, te multipleksora kojim se odabire adresa lokacije za izlazni tok. Za svaku grupu jezgra sadrži registar s početnom adresom grupe i brojilo zapisanih blokova. Njihov zbroj daje adresu na koju se zapisuje slijedeći blok te grupe. Ta se adresa multipleksorom usmjerava u generator naredbe za memorijski kontroler. Istovremeno s posljednjim elementom koji se zapisuje u izlazni tok, šalje se i naredba za memorijski kontroler, koji tada podatke iz toka proslijedi na određenu memorijsku lokaciju.

Operacija grupiranja izvodi se u dvije inačice: grupiranje prema nominalnom, i grupiranje prema numeričkom atributu. Za grupiranje prema nominalnom atributu, skup u FPGA memoriji grupira se normalnim radom jezgre. Za grupiranje prema numeričkom atributu, identifikacijski brojevi primjera prenesu se iz FPGA memorije u CPU memoriju prije izvršavanja grupiranja. Program na CPU-u na temelju polja identifikacijskih brojeva generira polje indeksa koje se prenese u FPGA memoriju i koristi kao ulazni tok indeksa za grupiranje.

Broj elemenata određen je minimalnom veličinom bloka koji se može pisati u SDRAM, a iznosi 96 bajtova. Širina riječi ulaznih i izlaznih tokova iznosi 32 bita, tj. 4 bajta, iz čega slijedi da je jedan blok za pisanje u SDRAM veličine 24 elemenata. FIFO spremnici aktiviraju zahtjev za pisanje kada prime taj broj elemenata, a moraju moći prihvatiti barem dvostruko više elemenata radi sprečavanja mogućeg preljeva.

Izvedena jezgra ima sedam ulaznih podatkovnih tokova, šest za atribute i jedan za testni atribut, te šest izlaznih podatkovnih tokova. Najveći očekivani broj jedinstvenih vrijednosti atributa je 64, a broj elemenata u polju početnih adresa grupa iznosi 384. Jezgra je uspješno implementirana za rad na frekvenciji takta 166 MHz. Uz navedene parametre najveća teoretska propusnost jezgre je 966×10^6 elemenata u sekundi.

5.3.3 Podatkovne strukture za pohranu primjera u FPGA memoriji

Prethodno navedena ograničenja imaju znatan utjecaj na odabir podatkovne strukture u kojoj je pohranjen skup za učenje. Nedostatak sustava priručne memorije za posljedicu ima da sva kašnjenja prilikom adresiranja i čitanja iz SDRAM-a ostaju izložena i utječu na konačne performanse sustava. Nadalje, svi pristupi SDRAM-u izvršavaju se u blokovima od 96 bajtova, što je iznos minimalne količine podataka koji se mogu prenijeti jednom instrukcijom memorijskom kontroleru. Adresiranje je također poravnato na blokove od 96 bajtova, tj. adresa lokacije kojoj se pristupa mora biti višekratnik broja 96.

Najefikasniji pristup SDRAM postiže se korištenjem predefiniranih naredbi za memorijski kontroler. Njihovo korištenje nudi tri predefinirana obrasca pristupa memoriji: linearni (engl. *linear*), dvodimenzionalni s korakom (engl. *2D strided*) i trodimenzionalni blokovski (engl. *3D blocked*) [84]. Obrazac pristupa koji osigurava najveću brzinu prijenosa je linearni, kojim se od početne lokacije slijedno pišu ili čitaju uzastopne lokacije.

Slika 5.13. Prikaz zapisa matrice u memoriji koprocesora, transponirana matrica duljine stupca *n*, s popunjavanjem stupca na višekratnik od 96 bajtova (*N*).

Za zapis skupa za učenje memoriji koristi se matrica, odnosno dvodimenzionalno polje. Redci matrice predstavljaju primjere, dok stupci matrice predstavljaju atribut, te klasu i identifikacijski broj primjera.

Matrice frekvencija izračunavaju se za pojedine atribute, pri čemu se čitaju vrijednosti atributa, klase i težine za svaki pojedini primjer u obrađivanom podskupu. Za izračun matrice frekvencija za jedan atribut, pristupa se dvama stupcima matrice: atributu i klasi. Budući da linearni obrazac pristupa pruža najveću brzinu prijenosa, najpogodniji zapis matrice jest zapis po stupcima. U tom zapisu vrijednosti istog atributa za različite primjere pohranjene su na uzastopne memorijske lokacije. Zapis matrice prikazan je na slici 5.13.

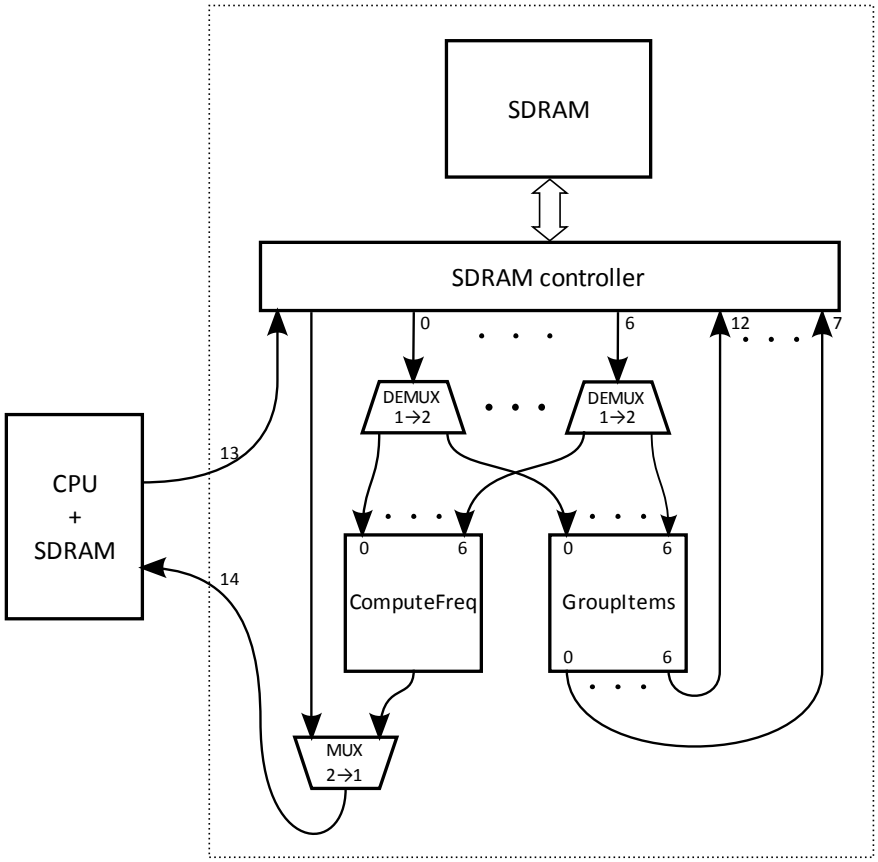
S obzirom na to da se memoriji pristupa u blokovima od 96 bajtova, svaki pojedini stupac-atribut matrice mora biti duljine koja je višekratnik veličine bloka. Ako duljina stupca nije višekratnik veličine bloka, on se dopunjava. Tako svaki stupac-atribut počinje na adresi poravnatoj na 96 bajtova.

Format zapisa vrijednosti atributa, oznaka klasa i identifikacijskih brojeva primjera određen je podatkovnim strukturama programa EC4.5. Izvorni program za zapis vrijednosti atributa koristi *union* strukturu koja za zapis nominalnih atributa koristi tip *short* – 16 bitni cijeli broj s predznakom, a za zapis identifikacijskog broja primjera koristi se tip uint32_t – 32 bitni cijeli broj bez predznaka. Ukupna širina *union* strukture je 32 bita. Svi ulazni i izlazni tokovi jezgri koji su spojeni na SDRAM širine su 32 bita.

5.4 Arhitektura koprocesora

Koprocesor sadrži dvije jezgre: *ComputeFreq* za izračun matrice frekvencija i *GroupItems* za grupiranje primjera. Za izračun matrice frekvencija koristi se MS2 izvedba jezgre *ComputeFreq*. Shema načina na koji su povezani u jedinstveni koprocesor prikazana je na slici 5.14. S obzirom na ograničen broj podatkovnih tokova, najviše 15, jezgre moraju dijeliti neke tokove. To nije problematično jer je priroda algoritma takva da te dvije jezgre nikad ne moraju raditi istovremeno.

Jezgre *ComputeFreq* i *GroupItems* dijele sedam podatkovnih tokova za čitanje iz SDRAM-a. Dijeljenje se izvodi pomoću demultipleksora, kojim se podatkovni tok iz SDRAM-a usmjerava samo na jednu od jezgri. Jezgra *GroupItems* spojena je sa šest izlaznih tokova prema SDRAM-u, koje koristi ekskluzivno. Izlazni tok jezgre *ComputeFreq* je preko multipleksora spojen prema CPU-u, tj, njegovi rezultati prenose se preko PCIe sabirnice u glavnu memoriju računala, gdje su odmah na raspolaganju programu koji se izvršava na CPU-u.



Slika 5.14. Arhitektura koprocesora

Tablica 5.1. Zauzeće resursa FPGA-a

Resurs	Iskorišteno	Ukupno dostupno	Iskorištenje
Logički resursi:	91992	297600	30,91 %
LUT	63418	297600	21,31 %
FF primarni	73491	297600	24,96 %
FF sekundarni	15610	297600	5,25 %
Množila (25×18)	0	2016	0,00 %
DSP blokovi	0	2016	0,00 %
BRAM18 blokovi	468	2128	21,99 %

Uz navedene, definirana su još i dva toka za povezivanje CPU-a sa SDRAM memorijom koprocesora. Tokovi služe za prijenos skupa za učenje u memoriju koprocesora, te za prijenos polja identifikacijskih brojeva primjera i polja oznaka grupa koje se koriste pri grupiranju u podskupove.

Ova arhitektura uspješno je implementirana za rad na 200 MHz za *CompuetFreq* jezgru i 166 MHz za ostatak koprocesora. Širina svih ulaznih i izlaznih tokova je 32 bita, osim izlaznog toka *ComputeFreq*, koja iznosi 256 bitova. Uz navedene parametre, jezgra *ComputeFreq* može obraditi maksimalno 960×10^6 parova atribut-klasa u sekundi. Jezgra *GroupItems* može raspodijeliti najviše 966×10^6 elemenata u sekundi. Jezgre u idealnom slučaju mogu iskoristiti oko 25% raspoložive propusnosti SDRAM-a spojenog na FPGA. Zauzeće resursa u FPGA-u prikazano je u tablici 5.1.

5.5 Programsko sučelje koprocesora

Za rad s koprocesorom definirana su četiri temeljna sučelja: *writeLMem*, *readLMem*, *computeFreq* i *groupItems*. Namjena sučelja je definicija parametara jezgri i komponenti u koprocesoru, kao što su broj elemenata u podatkovnim tokovima, početne adrese za čitanje i pisanje u SDRAM koprocesora, usmjeravanje podatkovnih tokova pomoću multipleksora i demultipleksora, postavljanje vrijednosti skalarnih ulaza jezgara i postavljanja duljine izvršavanja pojedine jezgre (u broju ciklusa takta jezgre). Za svako sučelje MaxCompiler automatski generira skup funkcija koje omogućuju različite načine definicije parametra sučelja i pokretanja rada koprocesora. Ovdje su opisana sučelja i njihovi parametri, te su dani prototipovi osnovnih funkcija za pristup sučelju.

Sučelje ***writeLMem*** je sučelje za prijenos podataka iz CPU-a u SDRAM na koprocesoru. Ulazni parametri su: početna adresa u memoriji koprocesora (*param_address*), broj bajtova

koji se prenosi (*param_nbytes*) i pokazivač na polje u CPU memoriji iz kojeg se čitaju podaci (*instream_cpu_to_lmem*). Prototip osnovne funkcije za pristup sučelju je:

```
void DFDTC_writeLMem(
    int64_t param_address,
    int64_t param_nbytes,
    const uint32_t *instream_cpu_to_lmem);
```

Sučelje ***readLmem*** ima funkciju komplementarnu sučelju *writeLMem*. Ono služi za prijenos podataka iz SDRAMa na koprocesoru u memoriju CPU-a. Također prima tri parametra: početna adresa u memoriji koprocesora (*param_address*) iz koje se čita, broj bajtova koji se prenosi (*param_nbytes*) i pokazivač na polje u CPU memoriji u koje se upisuju podaci (*outstream_tocpu*). Prototip osnovne funkcije za pristup sučelju je:

```
void DFDTC_readLMem(
    int64_t param_address,
    int64_t param_nbytes,
    uint32_t *outstream_tocpu);
```

Sučelje ***computeFreq*** služi za pokretanje procesa izračuna matrice frekvencija jezgrom *ComputeFreq*. Unutar sučelja definiraju se duljine ulaznih i izlaznog podatkovnog toka, vrijeme izvršavanja jezgre, početne adrese ulaznih podatkovnih tokova i usmjeravanje tokova iz SDRAM-a u jezgru. Ulazni parametri su broj primjera za obradu (*param_items*), polje početnih adresa podatkovnih tokova u memoriji koprocesora (*param_addrs*) i pokazivač na polje u CPU memoriji u koje se upisuju izračunate matrice frekvencija (*outstream_freqMat*). Prototip osnovne funkcije za pristup sučelju je:

```
void DFDTC_computeFreq(
    int64_t param_items,
    const int64_t *param_addrs,
    uint32_t *outstream_freqMat);
```

Sučelje ***groupItems*** služi za pokretanje procesa grupiranja primjera iz izvornog skupa u niz izlaznih podskupova. Unutar sučelja definiraju se duljine ulaznih i izlaznih podatkovnih tokova, početne adrese ulaznih podatkovnih tokova, vrijeme izvršavanja jezgre i omogućavanje pojedinih parova ulaznih-izlaznih podatkovnih tokova i usmjeravanje tokova iz SDRAM-a u jezgru. Početne adrese izlaznih grupa prenose se jezgri kao skalarno polje parametara i koriste se za generiranje naredbi memorijskom kontroleru. Ulazni parametri su bit-mapa omogućenih podatkovnih tokova (*param_enableStream*), broj primjera za grupiranje (*param_items*), polje početnih adresa ulaznih podatkovnih tokova u memoriji koprocesora

(*param_inAddrs*) i polje početnih adresa izlaznih grupa u memoriji koprocesora (*param_outAddrs*). Prototip osnovne funkcije za pristup sučelju je:

```
void DFDTC_groupItems(
    uint16_t param_enableStream,
    int64_t param_items,
    const int64_t *param_inAddrs,
    const int64_t *param_outAddrs);
```

6 Hibridni algoritam za učenje stabla odluke DF-DTC

Hibridni algoritam za učenje stabla odluke DF-DTC (engl. *dataflow decision tree construction*) temelji se na algoritmu C4.5/EC4.5, a implementira podskup njegovih mogućnosti.

Prije definicije izvedbe hibridnog algoritma, izvedena je probna paralelizacija algoritma pomoću OpenMP API-ja [85], [86]. Paralelizaciji se pristupilo prema načelu minimalnih zahvata u izvorni tekst programa identificiranjem ključnih petlji i raspoređivanjem iteracija petlji na dretve. Prema potrebi, rađeni su dodatni zahvati da bi se izolirale petlje od interesa, te pojedine funkcije učinile višedretveno sigurnima.

Nakon probne paralelizacije pristupilo se definiciji i izvedbi hibridnog algoritma. Iste ključne petlje u ovom su slučaju zamijenjene funkcijskim pozivima prema koprocesoru putem sučelja koje pruža Maxeler SLiC API. Pri izvedbi hibridnog algoritma uvedene su optimizacije u strukturama podataka i algoritma da bi se postigla što bolja efikasnost rada heterogenog sustava. Princip minimalnih zahvata u izvorni tekst programa nije korišten pri implementaciji hibridnog algoritma.

6.1 Definicija mogućnosti DF-DTC

DF-DTC, temeljen na C4.5/EC4.5, implementira određeni dio mogućnosti i opcija koje pruža izvorni algoritam.

Algoritam izvodi gradnju stabla sa sljedećim parametrima:

- Pri podjeli skupa prema numeričkom atributu, skup se dijeli na dva podskupa.
- Pri podjeli skupa prema nominalnom atributu, skup se dijeli na onoliko podskupova koliko atribut ima jedinstvenih vrijednosti.

Algoritam ima sljedeća ograničenja:

- Radi na skupovima bez nedostajućih vrijednosti.
- Skup za učenje prenosi se iz CPU memorije u FPGA memoriju samo jedanput, prije početka izvršavanja procesa učenja.

6.2 Paralelizacija EC4.5

Pri analizi algoritma i proučavanju izvornog teksta programa C4.5 i EC4.5, prepoznate su tri općenite strategije za paralelizaciju algoritma:

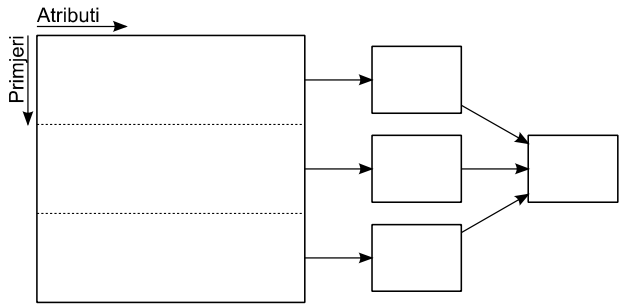
- strategija raspodjele primjera
- strategija raspodjele atributa
- strategija raspodjele čvorova.

Pod zahtjevom minimalnih zahvata u izvorni tekst programa, strategije raspodjela podataka i raspodjela atributa pogodne su samo za rad na nominalnim atributima. Algoritam različito postupa s nominalnim i numeričkim atributima. Radi efikasnosti algoritma, pri radu s numeričkim atributima koristi se sortiranje. Strategija raspodjele podatka zahtijevala bi implementaciju paralelnog algoritma za sortiranje, što bi bio preveliki zahvat u izvorni tekst programa. Strategija raspodjela atributa zahtijevala bi stvaranje dodatnih podatkovnih struktura koje bi omogućavale da svaka dretva ima svoju lokalnu kopiju atributa i indeksa prema podacima u memoriji. To je manji zahvat u izvorni tekst programa nego pri strategiji raspodjele podatka, ali i dalje veći nego predviđeno.

6.2.1 Strategija raspodjele primjera

Strategija raspodjela podataka implementira se samo na nominalnim atributima. Podaci podskupa koji se obrađuje rasporede se jednoliko po dretvama, kao što je ilustrirano na slici 6.1. Implementacija strategije obuhvaća izmjene u funkciji *ComputeFrequencies.* Isječak kritičnog dijela teksta programa dan je u ispisu 6.1.

Polje *Freq*, čiji se sadržaj mijenja u petlji, globalna je varijabla. Za potrebe paralelizacije petlje stvara se po jedna kopija polja za svaku radnu dretvu, te polje pokazivača *FreqMP* koje pokazuje na originalno polje, kao i na sve kopije. Svaka dretva pristupa svojoj kopiji pomoću



Slika 6.1. Strategija raspodjele primjera

```
/*  Determine the frequency of each class amongst cases
    with each possible value for the given attribute  */

ForEach(p, Fp, Lp)
{
    Case = Item[p];
    Freq[ DVal(Case,Att) ][ Class(Case) ] += Weight[p];
}
```

Ispis 6.1. Kritična petlja za paralelizaciju strategijom raspodjele primjera

funkcije *omp_get_thread_num*(), koja daje identifikacijski broj trenutne dretve: *FreqMP*[*omp_get_thread*()]. Originalnom polju *Freq* pristupa se preko indeksa 0 (*FreqMP*[0]=*Freq*) tako da glavna dretva uvijek pristupa originalnom polju. Nakon izvršavanja paralelizirane petlje, rezultati iz kopija polja *Freq* (*FreqMP*[1…*MP_THREAD*]) zbroje se u originalno polje (*FreqMP*[0]) prema sljedećoj formuli:

$$F_{v,c}^{0} \coloneqq F_{v,c}^{0} + \sum_{i=1}^{N_t-1} F_{v,c}^{i} \tag{6.1}$$

gdje je $F_{v,c}^{i}$ element polja *FreqMP*[i][v][c], v je indeks jedinstvene vrijednosti atributa, c je indeks klase primjera, a N_t je broj dretvi. Zbrajanje polja *FreqMP* izvedeno je slijedno, a vrijeme t potrebno za zbrajanje polja određeno je maksimalnim brojem dretvi N_t, brojem jedinstvenih vrijednosti atributa V, i brojem klasa u skupu podataka C:

$$t \propto N_t \, V \, C \tag{6.2}$$

U ispisu 6.2 dan je isječak teksta programa koji prikazuje paraleliziranu petlju, te zbrajanje rezultata u jedinstveno polje. Za paralelizaciju petlje, *ForEach* makronaredba nadomještena je

```
/* main loop */
#pragma omp parallel for if(Lp - Fp >= MP_CHUNK) private(p, v, c) \
    schedule(static)
for (p = Fp; p <= Lp; p++) {
    v = DVal(Item[p],Att);
    c = Class(Item[p]);
    FreqMP[omp_get_thread_num()][v][c] += Weight[p];
}

/* table reduction */
for (i = 1; i < MP_THREAD; ++i)
    for (v = 0; v <= MaxAttVal[Att]; ++v)
        for (c = 0; c <= MaxClass; ++c)
            FreqMP[0][v][c] += FreqMP[i][v][c];
```

Ispis 6.2. Kritična petlja paralelizirana strategijom raspodjele primjera

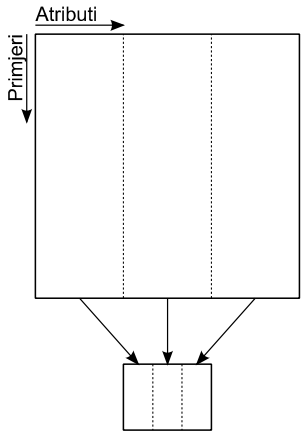
standardnom *for* petljom. Korištena je *parallel for* pragma s *if* klauzulom. Razlog korištenja *if* klauzule je sprečavanje da trošak stvaranja i upravljanja dretvama postane veći od dobitka višedretvenog izvođenja petlje. Uvjet za stvaranje radnih dretvi je da broj primjera u ulaznom polju bude veći ili jednak minimumu definiranim *MP_CHUNK* varijablom. Vrijednost varijable *MP_CHUNK* postavlja se iz komandne linije putem parametara pokretanja programa. Korišten je *static* raspored, tako da se svakoj dretvi na izvršavanje raspoređuje podjednako velik blok iteracija petlje. Privatne varijable su brojač petlje *p*, koji je ujedno i indeks primjera u skupu podataka, te varijable *v* i *c*, koje predstavljaju vrijednost atributa i klasu primjera *p*. Ostale korištene varijable ostaju zajedničke. Polje *Weight* se u petlji isključivo čita, a mogući sukobi u pristupu polju *FreqMP* razriješeni su indeksiranjem pomoću identifikacijskog broja dretve, koji osigurava da svaka dretva pristupa isključivo vlastitoj kopiji polja *Freq*.

6.2.2 Strategija raspodjele atributa

Raspodjela atributa također se implementira samo na nominalnim atributima. Svi nominalni atributi koji se obrađuju u čvoru raspodijele se po dretvama, kao što je ilustrirano na slici 6.2, dok se numerički atributi obrađuju slijedno.

Implementacija strategije obuhvaća izmjene u funkciji *FormTreeInt*. Isječak originalnog teksta programa koji je paraleliziran prikazan je u ispisu 6.3. U ovom slučaju cilj je na više dretvi rasporediti pozive funkcije *EvalDiscreteAtt*.

Prije izvođenja paralelizacije, petlja je podijeljena na pet dijelova tako da je svaka operacija u vlastitoj petlji. U svrhu odvajanja obrade nominalnih i numeričkih atributa, definirana su nova



Slika 6.2. Strategija raspodjele atributa

```
/*  For each available attribute, find the information and gain  */
ForEach(Att, 0, MaxAtt) {
    Gain[Att] = -Epsilon;

    if ( SpecialStatus[Att] == IGNORE ) continue;

    if ( MaxAttVal[Att] ) {
        /*  discrete valued attribute  */

        if ( SUBSET && MaxAttVal[Att] > 2 ) {
            EvalSubset(Att, Fp, Lp, Cases);
        } else if ( ! Tested[Att] ) {
            EvalDiscreteAtt(Att, Fp, Lp, Cases);
        }
    } else {
        /*  continuous attribute  */

        EvalContinuousAtt(Att, Fp, Lp);
    }

    /*  Update average gain, excluding attributes with very many values  */
    if ( Gain[Att] > -Epsilon &&
         ( MultiVal || MaxAttVal[Att] < 0.3 * (MaxItem + 1) ) ) {
        Possible++;
        AvGain += Gain[Att];
    }
}
```

Ispis 6.3. Kritična petlja za paralelizaciju strategijom raspodjele atributa

polja *DiscrAtt* i *ContinAtt* koja sadrže indekse nominalnih, odnosno numeričkih atributa, a petlje potom iščitavaju indekse atributa iz tih polja.

Od globalnih varijabli promijenjene su varijable *Freq* i *ValFreq*, kojima se pristupa u funkciji *ComputeFrequencies*, kako bi funkcija postala višedretveno sigurna (engl. *thread-safe*). Umjesto njih definirane su nove varijable *MpFreq* i *MpValFreq* koje definiraju onoliko kopija *Freq* i *ValFreq* koliki je maksimalni mogući broj dretvi, a *Freq* i *ValFreq* redefinirane su u makroe:

```
#define Freq    MpFreq[omp_get_thread_num()]
#define ValFreq MpValFreq[omp_get_thread_num()]
```

Od pet novih petlji, paralelizirana je jedino ona iz koje se poziva funkcija *EvalDiscrAtt*. Isječak izmijenjenog teksta programa koji uključuje paralelizaciju ključne petlje dan je u ispisu 6.4, a pareleliziranа petlja označena je crvenim tekstom. Korištena je *parallel for* pragma. Kao privatne varijable označene su *AttIdx* koja služi kao brojač petlje i indeks polja *DiscrAtt*, te *Att* koja predstavlja trenutnu vrijednost indeksa atributa. Korišten je *dynamic* raspored, tako da se svakoj dretvi dodjeljuje po jedna iteracija petlje čim ona završi s

```
/*  For each available attribute, find the information and gain  */

for (Att = 0; Att <= MaxAtt; ++Att)
    Gain[Att] = -Epsilon;

/* discrete valued attribute */
if (SUBSET)
    for (AttIdx = 0; AttIdx < DiscrAttCount; ++AttIdx) {
        Att = DiscrAtt[AttIdx];
        if (SpecialStatus[Att] != IGNORE)
            if (MaxAttVal[Att] > 2)
                EvalSubset(Att, Fp, Lp, Cases);
    }
else {
    #pragma omp parallel for private(AttIdx, Att) schedule(dynamic)
    for (AttIdx = 0; AttIdx < DiscrAttCount; ++AttIdx) {
        Att = DiscrAtt[AttIdx];
        if (SpecialStatus[Att] != IGNORE)
            if (!Tested[Att])
                EvalDiscreteAtt(Att, Fp, Lp, Cases);
    }
}

/* continuous attribute */
for (AttIdx = 0; AttIdx < ContinAttCount; ++AttIdx) {
    Att = ContinAtt[AttIdx];
    if (SpecialStatus[Att] != IGNORE)
        EvalContinuousAtt(Att, Fp, Lp);
}

/*  Update average gain, excluding attributes with very many values  */
for (Att = 0; Att <= MaxAtt; ++Att)
    if (SpecialStatus[Att] != IGNORE)
        if (Gain[Att] > -Epsilon
                && (MultiVal || MaxAttVal[Att] < 0.3 * (MaxItem + 1))) {
            Possible++;
            AvGain += Gain[Att];
        }
```

Ispis 6.4. Izmijenjen program s kritičnom petljom paraleliziranom strategijom raspodjele atributa

trenutnom. Neki atributi se preskaču ovisno o tome jesu li općenito ignorirani, ili su već odabrani kao testni atribut u nekom od predaka trenutnog čvora, pa *dynamic* raspored omogućuje punu zaposlenost dretvi i kada pojedine iteracije završe ranije.

6.2.3 Strategija raspodjele čvorova

Raspodjela čvorova je strategija koja od navedene tri najviše obećava u pogledu mogućeg ubrzanja. Gradnja stabla počinje korijenskim čvorom koji se izvršava u jednoj dretvi. Nakon odabira testnog atributa i podjele ulaznog skupa na podskupove, stvaraju se čvorovi djeca, koja se potom mogu rasporediti po dretvama i obrađivati paralelno.

Za implementaciju ove strategije koristi se pragma *omp task*. Paralelizacija se izvodi u funkciji *FormTreeInt*, koja implementira temeljni rekurzivni proces algoritma. Na pozive funkcije *FormTreeInt* dodaju se „*task*“ pragme, koje označavaju da se taj poziv paralelizira pomoću reda zadataka:

```
#pragma omp task
Node->Branch[v] = FormTreeInt(Fp, Ep, MaxError);
```

Pri izvršavanju programa, OpenMP API stavlja taj poziv funkcije u red. Kada se izvrše svi prethodni zadaci, dodjeljuje ga na izvršavanje prvoj slobodnoj dretvi.

Od ove strategije očekuje se najveće ubrzanje jer se ne paralelizira samo jedan zadatak u učenju stabla (izračun matrice frekvencija), već se paralelno izvršavaju svi zadaci koji su dio obrade čvora.

Analizom izvornog teksta programa zaključeno je da implementacija ove strategije nije u okviru predviđenih optimizacija. Originalna implementacija oslanja se na korištenje globalnih varijabli. Ukupno ima oko dvadeset globalnih varijabli, a uglavnom služe za prosljeđivanje podataka između funkcija, npr. kada je potrebno prenijeti podatke u obliku polja iz jedne funkcije u drugu. Proučavanjem teksta programa pronađeno je petnaest funkcija koje na taj način prosljeđuju podatke.

Za implementaciju ove strategije nužno je da sve funkcije koje sudjeluju u procesu obrade čvora budu višedretveno sigurne. Svaku funkciju koja koristi globalne varijable potrebno je dodatno analizirati i preinačiti, te pri tome vrlo vjerojatno i dodavati nove podatkovne strukture nužne da bi se postigla višedretvena sigurnost. Sve navedeno zahtijevalo bi znatno veće zahvate u tekst programa nego što je predviđeno.

6.2.4 Očekivano ubrzanje

Strategije raspodjele primjera i raspodjele atributa svode se na paralelizaciju izvršavanja funkcije *ComputeFrequencies*. U prvom slučaju raspodjelom na dretve iteracije ključne petlje u funkciji, a u drugom slučaju raspodjelom na dretve pojedinih poziva funkcije. U oba slučaja može se očekivati jednaka gornja granica ubrzanja određena Amdahlovim zakonom:

$$s(k) = \frac{1}{1 - p\left(1 - \frac{1}{k}\right)} \tag{6.3}$$

Tablica 6.1. Gornja granica ubrzanja učenja stabla prema Amdahlovom zakonu

Broj dretvi	Skup podataka / ubrzanje Census-Income p = 56,74 %	 Covertype p = 41,83%
1	1,00	1,00
2	1,40	1,26
3	1,61	1,39
4	1,74	1,46

gdje je k broj dretvi, a p dio programa koji je paraleliziran izražen u udjelu vremena izvršavanja na slijednom stroju. Očekivana gornja granica dobivenog ubrzanja učenja stabla prikazana je u tablici 6.1.

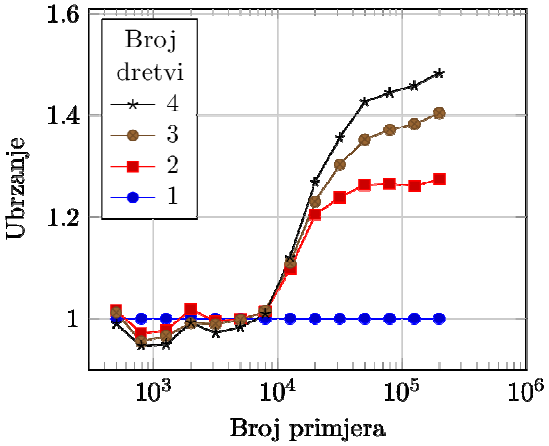
6.2.5 Rezultati

Ubrzanje izvršavanja algoritma izražava se omjerom vremena izvršavanja izvorne jednodretvene implementacije i izvedene višedtretvene:

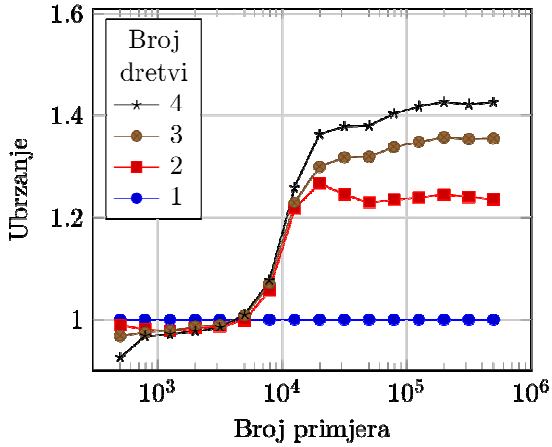
$$s_k = \frac{t_1}{t_k} \tag{6.4}$$

gdje je: t_1 vrijeme izvršavanja na jednoj dretvi, t_k vrijeme izvršavanja na k dretvi, a s_k ubrzanje postignuto izvršavanjem programa na k dretvi. Ovom se definicijom ubrzanja jednako vrijeme izvršavanja s jednom i sa više dretvi prikazuje kao jedinično ubrzanje, tj. ubrzanje jedan puta. Ako je izračunato ubrzanje veće od jedan, višedretvena implementacija izvršava se brže od jednodretvne, a ako je ubrzanje manje od jedan, višedretvena implementacija je sporija.

Ispitivanje paralelizacije provedeno je na istim uzorcima skupova kao i dinamička analiza opisana u poglavlju 4.3. Broj primjera u skupu, i broj uzoraka skupa dan je u tablici 4.2, a broj atributa u skupu s brojem uzoraka skupa dan je u tablici 4.3. Mjerenja ubrzanja provedena su na osobnom računalu s Intel Core 2 Q8400 CPU-om i 8 GiB RAM-a, pod operacijskim sustavom Linux, verzija 2.6.32. Korišten je prevodilac gcc, verzija 4.4.2. Parametri prevodioca su: „-O3 -pg -Wall -c -fmessage-length=0“. Parametri kolekcije su: „-pg -lm“.



Slika 6.3. Ubrzanje u odnosu na broj primjera – raspodjela primjera, Census-Income



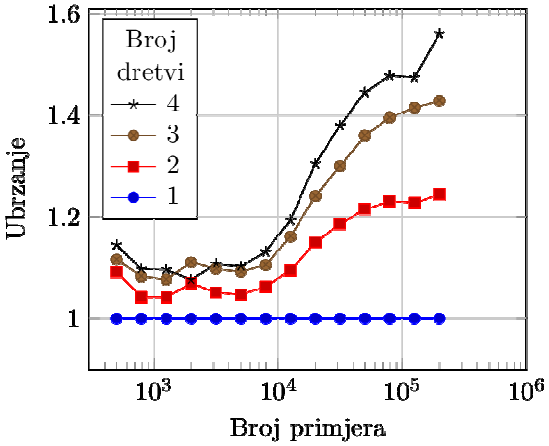
Slika 6.4. Ubrzanje u odnosu na broj primjera – raspodjela primjera, Covertype

6.2.5.1 Utjecaj broja primjera

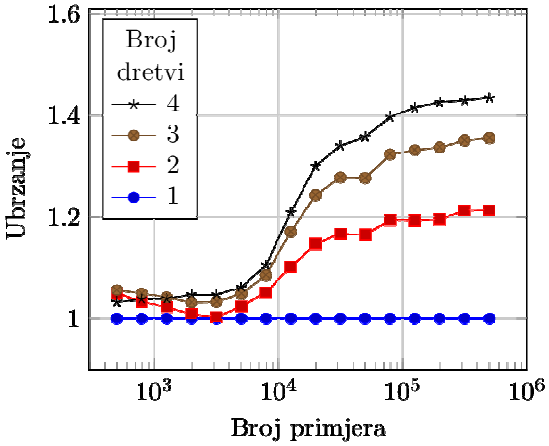
Za strategiju raspodjele primjera, iz slika 6.3 i 6.4 vidljivo je da za male skupove podataka – one manje od oko 5.000 primjera – višedretveno izvršavanje ima negativan utjecaj na performanse programa. Vrijeme višedretvenog izvršavanja programa u tom je slučaju nešto duže nego kod jednodretvnog izvršavanja što se pokazuje faktorom ubrzanja manjim od jedan. Minimalno izmjereno ubrzanje je 0,93 puta, uz izvršavanje na četiri dretve. Za skupove veće od 20.000–30.000 primjera dobije se puno ubrzanje, a najveće izmjereno ubrzanje iznosi

Tablica 6.2. Izmjerena ubrzanja u odnosu na veličinu skupa i broja dretvi za paralelizaciju strategijom raspodjele primjera

Broj dretvi / Broj primjera	Census-Income 1	2	3	4	Covertype 1	2	3	4
500	1,00	1,02	1,01	0,99	1,00	0,99	0,97	0,93
792	1,00	0,97	0,96	0,95	1,00	0,98	0,97	0,97
1256	1,00	0,98	0,97	0,95	1,00	0,98	0,98	0,97
1991	1,00	1,02	0,99	0,99	1,00	0,98	0,99	0,98
3155	1,00	1,00	0,99	0,97	1,00	0,99	0,99	0,98
5000	1,00	1,00	1,00	0,98	1,00	1,00	1,01	1,01
7924	1,00	1,01	1,01	1,01	1,00	1,06	1,07	1,08
12559	1,00	1,10	1,11	1,12	1,00	1,22	1,23	1,26
19905	1,00	1,20	1,23	1,27	1,00	1,27	1,30	1,36
31548	1,00	1,24	1,30	1,36	1,00	1,25	1,32	1,38
50000	1,00	1,26	1,35	1,43	1,00	1,23	1,32	1,38
79245	1,00	1,26	1,37	1,44	1,00	1,24	1,34	1,40
125594	1,00	1,26	1,38	1,46	1,00	1,24	1,35	1,42
199054	1,00	1,27	1,40	1,48	1,00	1,24	1,36	1,43
315479	–	–	–	–	1,00	1,24	1,35	1,42
500000	–	–	–	–	1,00	1,24	1,36	1,43



Slika 6.5. Ubrzanje u odnosu na broj primjera – raspodjela atributa, Census-Income



Slika 6.6. Ubrzanje u odnosu na broj primjera – raspodjela atributa, Covertype

1,48 puta, uz izvršavanje na četiri dretve. Izmjerene vrijednosti ubrzanja dane su u tablici 6.2.

Najvjerojatniji uzrok takvom ponašanju je mali broj primjera u podskupovima koji se pojavljuju kako se povećava dubina stabla. Algoritam počinje rad s jednim skupom pune veličine, koji se potom dijeli na manje podskupove. Kako se stablo gradi, broj podskupova se povećava, a njihova se veličina smanjuje. Zbog toga, relativno velik broj podskupova bude jednostavno obrađen slijedno jer je broj primjera u njemu ispod praga za višedretveno izvršavanje. Uz to, moguće je i da je prag veličine podskupa postavljen prenisko zbog čega bi

Tablica 6.3. Izmjerena ubrzanja u odnosu na veličinu skupa i broja dretvi za paralelizaciju strategijom raspodjele atributa

Broj dretvi / Broj primjera	Census-Income 1	2	3	4	Covertype 1	2	3	4
500	1,00	1,09	1,12	1,14	1,00	1,05	1,06	1,03
792	1,00	1,04	1,08	1,10	1,00	1,03	1,05	1,04
1256	1,00	1,04	1,08	1,10	1,00	1,02	1,04	1,04
1991	1,00	1,07	1,11	1,08	1,00	1,01	1,03	1,05
3155	1,00	1,05	1,10	1,11	1,00	1,00	1,03	1,05
5000	1,00	1,05	1,09	1,10	1,00	1,02	1,05	1,06
7924	1,00	1,06	1,11	1,13	1,00	1,05	1,09	1,10
12559	1,00	1,09	1,16	1,19	1,00	1,10	1,17	1,21
19905	1,00	1,15	1,24	1,30	1,00	1,15	1,24	1,30
31548	1,00	1,19	1,30	1,38	1,00	1,17	1,28	1,34
50000	1,00	1,21	1,36	1,45	1,00	1,17	1,28	1,36
79245	1,00	1,23	1,40	1,48	1,00	1,19	1,32	1,40
125594	1,00	1,23	1,41	1,47	1,00	1,19	1,33	1,41
199054	1,00	1,24	1,43	1,56	1,00	1,20	1,34	1,43
315479	–	–	–	–	1,00	1,21	1,35	1,43
500000	–	–	–	–	1,00	1,21	1,36	1,43

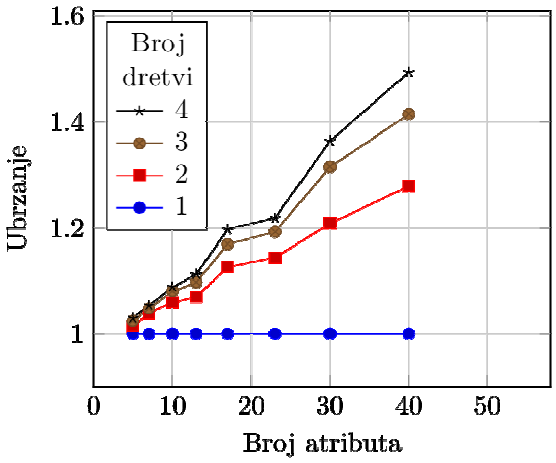
na značajnom broju podskupova trošak stvaranja i upravljanja dretvama poništio dobitak od višedretvenog izvršavanja.

Za paralelizaciju strategijom raspodjele atributa, iz slika 6.5 i 6.6 vidljivo je da bez obzira na veličinu skupa nema negativnog utjecaja višedretvenog izvršavanja na vrijeme izvršavanja programa. Minimalno izmjereno ubrzanje iznosi 1,00, uz izvršavanje na dvije dretve. To znači da trošak stvaranja i upravljanja dretvama ne premašuje dobitak od višedretvenog izvršavanja. Za skupove manje od 10.000 primjera ne postiže se značajno ubrzanje, a puno ubrzanje dobije se na skupovima iznad 50.000 primjera. Najveće izmjereno ubrzanje iznosi 1,56, uz izvršavanje na četiri dretve. Izmjerene vrijednosti ubrzanja dane su u tablici 6.3.

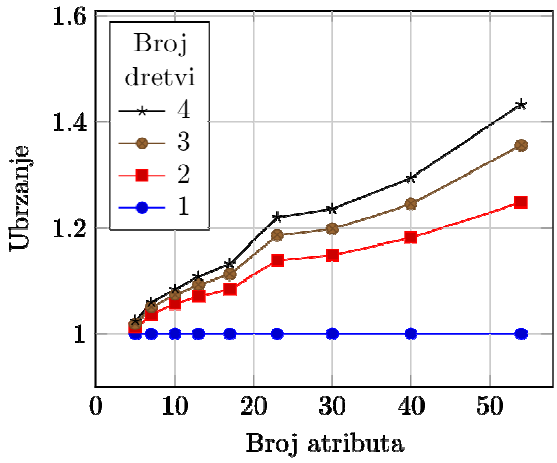
6.2.5.2 Utjecaj broja atributa

Budući da je ispitivanje skaliranja broja atributa provedeno na skupu s punim brojem primjera, ne pojavljuju se efekti malog broja primjera kakvi su bili vidljivi pri skaliranju veličine skupa.

Na slikama 6.7 i 6.8 prikazani su rezultati za strategiju raspodjele primjera, a izmjerene vrijednosti dane su u tablici 6.4. Vidljivo je da je ubrzanje u prvoj aproksimaciji raste približno linearno s brojem atributa. Budući da je broj atributa u skupu podataka ograničen, nije bilo moguće ispitati kada dolazi do zasićenja ubrzanja. Rezultati strategije raspodjele atributa prikazani su na slikama 6.9 i 6.10, s izmjerenim ubrzanjima danim u tablici 6.5. Kao i kod strategije raspodjele atributa, i u ovom slučaju ubrzanje raste približno linearno s brojem atributa. Dobivena ubrzanja nešto su veća nego kod strategije raspodjele primjera, što je u skladu s rezultatima utjecaja broja primjera.



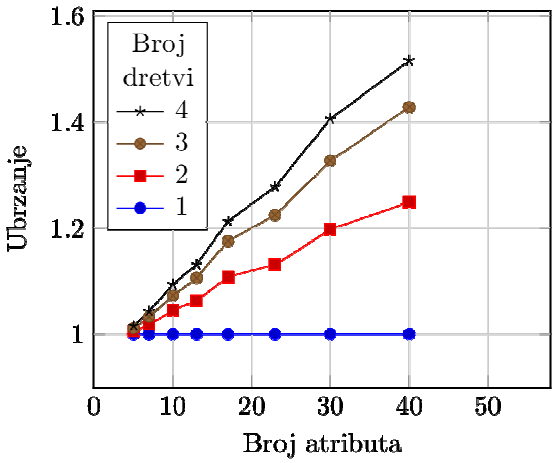
Slika 6.7. Ubrzanje u odnosu na broj atributa – raspodjela primjera, Census-Income



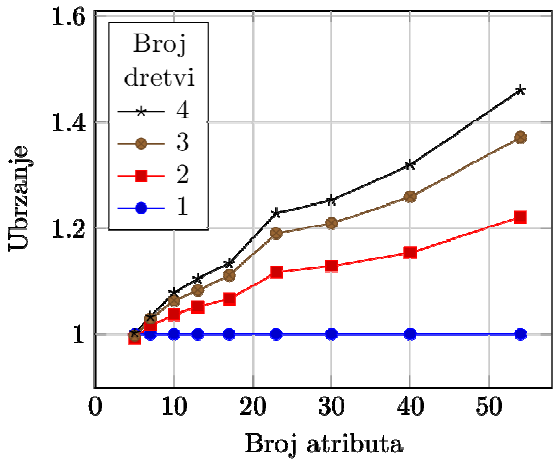
Slika 6.8. Ubrzanje u odnosu na broj atributa – raspodjela primjera, Covertype

Tablica 6.4. Izmjerena ubrzanja u odnosu na broj atributa i broj dretvi za paralelizaciju strategijom raspodjele primjera

Broj dretvi / Broj atributa	Census-Income 1	2	3	4	Covertype 1	2	3	4
5	1,00	1,02	1,01	1,03	1,00	1,02	1,01	1,02
7	1,00	1,05	1,04	1,05	1,00	1,05	1,04	1,06
10	1,00	1,08	1,06	1,09	1,00	1,07	1,06	1,08
13	1,00	1,10	1,07	1,11	1,00	1,09	1,07	1,11
17	1,00	1,17	1,13	1,20	1,00	1,11	1,08	1,13
23	1,00	1,19	1,14	1,22	1,00	1,19	1,14	1,22
30	1,00	1,31	1,21	1,36	1,00	1,20	1,15	1,24
40	1,00	1,41	1,28	1,49	1,00	1,25	1,18	1,29
54	–	–	–	–	1,00	1,36	1,25	1,43



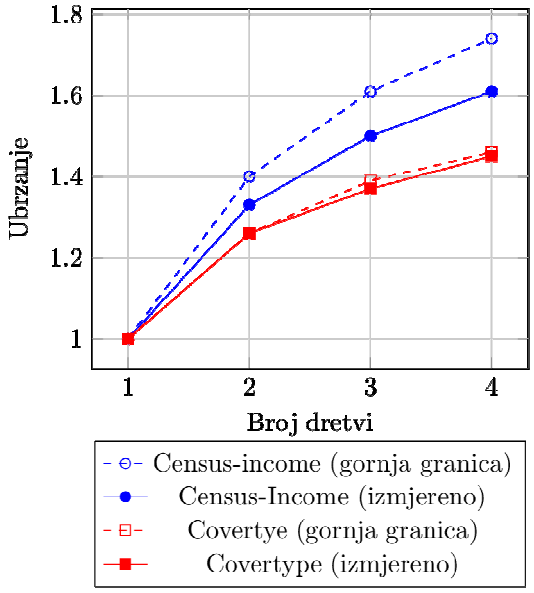
Slika 6.9. Ubrzanje u odnosu na broj atributa – raspodjela atributa, Census-Income



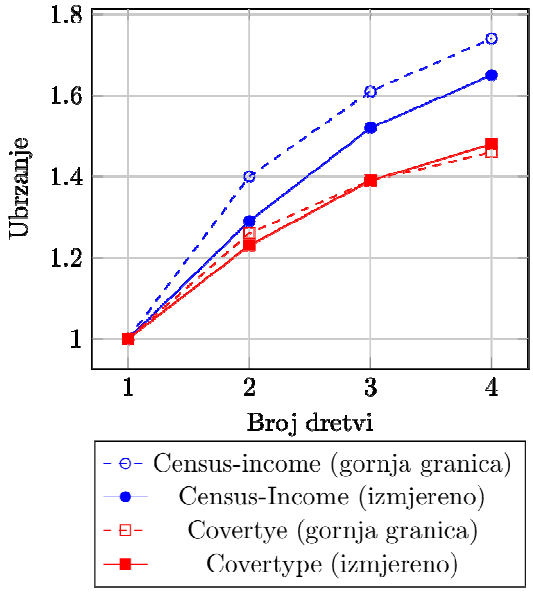
Slika 6.10. Ubrzanje u odnosu na broj atributa – raspodjela atributa, Covertype

Tablica 6.5. Izmjerena ubrzanja u odnosu na broj atributa i broj dretvi za paralelizaciju strategijom raspodjele atributa

Broj dretvi / Broj atributa	Census-Income 1	2	3	4	Covertype 1	2	3	4
5	1,00	1,00	1,01	1,02	1,00	0,99	1,00	1,00
7	1,00	1,02	1,03	1,04	1,00	1,02	1,03	1,03
10	1,00	1,05	1,07	1,09	1,00	1,04	1,06	1,08
13	1,00	1,06	1,11	1,13	1,00	1,05	1,08	1,11
17	1,00	1,11	1,18	1,21	1,00	1,07	1,11	1,13
23	1,00	1,13	1,23	1,28	1,00	1,12	1,19	1,23
30	1,00	1,20	1,33	1,41	1,00	1,13	1,21	1,25
40	1,00	1,25	1,43	1,52	1,00	1,15	1,26	1,32
54	–	–	–	–	1,00	1,22	1,37	1,46



Slika 6.11. Ubrzanje učenja stabla strategijom raspodjele primjera



Slika 6.12. Ubrzanje učenja stabla strategijom raspodjele atributa

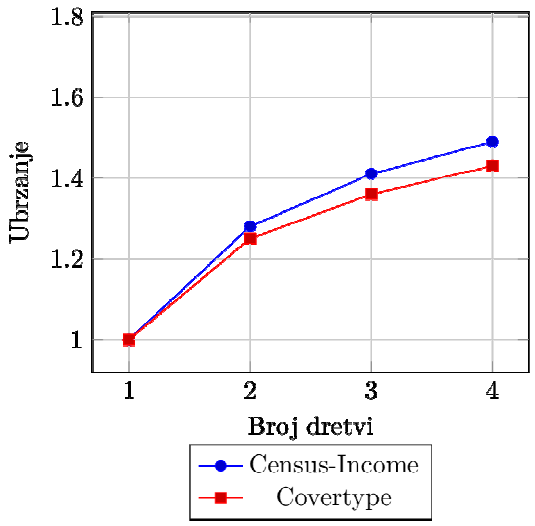
6.2.5.3 Utjecaj broja dretvi

Ispitivanje utjecaja broja dretvi provedeno je na cjelovitim skupovima Census-income i Covertype. Tijekom ispitivanja broj dretvi je mijenjan od jedne do četiri, s tim da je vrijeme izvršavanja na jednoj dretvi osnovica za usporedbu. Dobivena ubrzanja dana su u tablici 6.6.

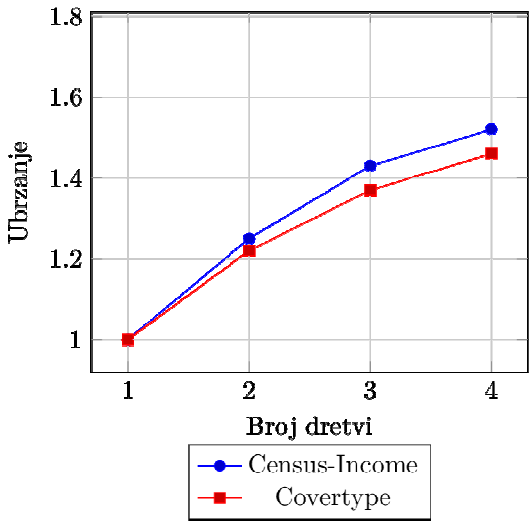
Na slikama 6.11 i 6.12 prikazano je ubrzanje učenja stabla u odnosu na broj dretvi za paralelizaciju strategijom raspodjele primjera, odnosno raspodjele atributa. Za skup Covertype dobiveno je ubrzanje jako blisko očekivanom. Za skup Census-income za obje strategije dobiveno ubrzanje nešto je niže od očekivanog prema Amdahlovom zakonu, s time da se za strategiju raspodjele atributa dobiju bolja ubrzanja nego za strategiju raspodjele primjera.

Tablica 6.6. Izmjerena ubrzanja u odnosu na broj dretvi

		Census-income		Covertype	
	Broj dretvi	**Ukupno**	**Učenje stabla**	**Ukupno**	**Učenje stabla**
Strategija raspodjele atributa	**1**	1,00	1,00	1,00	1,00
	2	1,25	1,29	1,22	1,23
	3	1,43	1,52	1,37	1,39
	4	1,52	1,65	1,46	1,48
Strategija raspodjele primjera	**1**	1,00	1,00	1,00	1,00
	2	1,28	1,33	1,25	1,26
	3	1,41	1,50	1,36	1,37
	4	1,49	1,61	1,43	1,45



Slika 6.13. Ukupno ubrzanje strategijom raspodjele podataka



Slika 6.14. Ukupno ubrzanje strategijom raspodjele atributa

Ukupno ubrzanje, prikazano na slikama 6.13 i 6.14, za obje strategije raste s brojem atributa. Prema očekivanju, ono je nešto niže od ubrzanja učenja stabla. Na oba ispitna skupa pokazano je da se strategijom raspodjele atributa postižu veća ukupna ubrzanja nego strategijom raspodjele atributa. Budući da se ispitivanje provodilo na CPU-u s četiri jezgre, nije bilo moguće ispitati performanse paraleliziranog programa na više od četiri dretve.

6.3 Hibridni algoritam

Hibridni algoritam zadržava obradu numeričkih atributa na CPU-u, dok se glavnina obrade nominalnih atributa izvodi na koprocesoru. U algoritmu su ključne petlje prepoznate u poglavlju 6.2 zamijenjene funkcijskim pozivima koprocesoru putem sučelja koje pruža Maxeler SLiC API. Pri izvedbi algoritma uvedene su nove podatkovne strukture i funkcije koje su potrebne za prihvat i obradu rezultata iz koprocesora. Uvedene su funkcije za upravljanje radom koprocesora, te za usklađivanje sadržaja podskupova u CPU-u i koprocesorskoj memoriji.

Tijek hibridnog algoritma za izgradnju stabla:

1. Podaci se učitaju u memoriju u matričnu strukturu. Uz učitane podatke, u skup se dodaje novi stupac u kojem se nalaze identifikacijski brojevi za svaki primjer.
2. Stvori se matrica nominalnih atributa u koju se pohrane vrijednosti nominalnih atributa, klase, te identifikacijski brojevi. Matrica nominalnih atributa učita se u memoriju koprocesora.

3. Za ulazni skup S:
    - Ispitaju se uvjeti za stvaranje lista, ako su ispunjeni stvori se list i zaustavi rekurzija.
    - Ispita se veličina skupa S:
        i. ako je $|S| \geq t$: na koprocesoru se pokrene obrada nominalnih atributa.
        ii. inače: obrada nominalnih atributa se provede na CPU.
    - Preuzmu se i na CPU-u obrade matrice frekvencija, izračunaju se informacijski dobitci.
    - Na CPU se obrade se numerički atributi, izračunaju se informacijski dobitci.
    - Odabere se testni atribut.
    - Ulazni skup se prema vrijednosti testnog atributa podijeli na podskupove $S_1, S_2, \ldots$
    - Na svakom podskupu rekurzivno se ponovi postupak iz točke 3, gradnje stabla je u dubinu (engl. *depth-first*).

Za cijelo vrijeme izvršavanja programa, cjeloviti ulazni skup nalazi se u CPU memoriji, uključujući vrijednosti nominalnih atributa. U memoriju koprocesora kopiraju se samo vrijednosti nominalnih atributa, te klase i identifikacijski brojevi primjera.

S obzirom na to da se skup za učenje nalazi u memoriji CPU-a i memoriji koprocesora, prilikom svake podjele skupa na podskupove provodi se usklađivanje njihovih sadržaja. Metoda usklađivanja je takva da minimizira prijenos podataka između CPU-a i koprocesora. To ograničenje posljedica je relativno male propusnosti veze CPU ↔ FPGA. Opetovanim prijenosom podataka ona postaje usko grlo i značajno smanjuje performanse cijelog sustava.

List se stvara u dva slučaja. Prvi je kad su svi primjeri u podskupu iste klase. Tada se stvori list s oznakom te klase. Drugi je kad nema dovoljno primjera za daljnju podjelu skupa. Tada se stvori list s oznakom klase najveće frekvencije u podskupu. Nakon što je dosegnut list, podskup koji mu pripada više ne sudjeluje u procesu izgradnje stabla. U tom slučaju se oslobađa koprocesorska memorija koju on zauzima, čime se taj podskup u stvari uklanja iz memorije. Uklanjanje lisnog podskupa izvršava se samo u memoriji koprocesora, dok u CPU memoriji ostaje netaknut.

Nakon završetka izgradnje stabla provodi se obrezivanje. Proces obrezivanja u izvornom je obliku i izvodi se u potpunosti na CPU-u.

6.4 Implementacija hibridnog algoritma

Za implementaciju hibridnog algoritma korišten je izvorni tekst programa C4.5 Release 8 [73] napisan u programskom jeziku C. Na program C4.5 primijenjena je zakrpa kojom su dodane mogućnosti programa EC4.5 [74]. Za implementaciju hibridnog algoritma tekst programa izmijenjen je na sljedeći način:

- Iz programa su uklonjene funkcije kojima su implementirani dijelovi izvornog algoritma koji nisu dio hibridnog algoritma.
- Matrica u koju se sprema skup za učenje je proširena.
- Dodane su funkcije u kojima se koriste pozivi prema koprocesoru.
- Dodane su pomoćne podatkovne strukture za rad s koprocesorom.

Kao što je određeno arhitekturom jezgre *ComputeFreqKernel*, izračun matrice frekvencija na FPGA izvršava se paralelno sa po šest atributa. Ako je ulazni skup manji od praga, cjelovita obrada provodi sa na CPU-u. Obrada numeričkih atributa na CPU-u paralelizirana je primjenom strategije raspodjele atributa, kao što je opisano u poglavlju 6.2.2. Ostala obrada (izračun informacijskog dobitka) izvršava se slijedno na CPU-u.

6.4.1 Podatkovne strukture

6.4.1.1 Skup za učenje

U glavnini su zadržane izvorne strukture podataka. Manje izmjene napravljene su jedino u matrici za pohranu skupa za učenje.

Skup za učenje pohranjuje se u matricu u kojoj redci predstavljaju primjere, a stupci atribute i klasu. Matrica je pohranjena po redcima, gdje je svaki redak pohranjen u posebno alocirano polje. Svi redci povezani su u matricu pomoću polja pokazivača u kojem svaki element pokazuje na jedan redak matrice. Matrica je proširena jednim dodatnim stupcem koji predstavlja identifikacijski broj primjera. Kao identifikacijski broj služi redni broj pod kojim

Tablica 6.7. *Union* struktura *AttValue* za opis vrijednosti atributa

Namjena	Naziv	Podatkovni tip
Nominalni atribut / klasa	_discr_val	short
Numerički atribut	_cont_val	float
Indeks jedinstvene vrijednosti numeričkog atributa	_pos_val	int
Identifikacijski broj primjera	_itemID_val	uint32_t

je redak iz datoteke učitan u memoriju, i on ostaje jedinstven za cijelo vrijeme izvršavanje programa.

Pojedini elementi matrice pohranjuju se *union* strukturom *AttValue*. U *union* strukturi definirani su podatkovni tipovi kojima se vrijednosti atributa, klasa i indeksa jedinstvene vrijednosti numeričkog atributa prikazuju u računalu. U strukturu je dodana definicija za prikaz vrijednosti identifikacijskih brojeva. Definicija *union* strukture *AttValue* dana je u tablici 6.7. Ukupna veličina strukture na ispitnoj računalnoj arhitekturi iznosi četiri bajta.

6.4.1.2 Privremena matrica nominalnih atributa

Kao što je opisano u poglavlju 5.3.3, skup za učenje se u koprocesoru pohranjuje u transponiranu matricu. U svrhu pripreme skupa za učenje za pohranu u memoriju koprocesora, koristi se privremena matrica u koju se prepišu nominalni atributi. Privremena matrica je transponirana. Svaki stupac je posebno polje, a stupci su povezni u matricu poljem pokazivača u kojem svaki element pokazuje na jedan stupac matrice. Privremena matrica je usklađena sa značajkama matrice u koprocesoru. Duljine stupaca su, prema potrebi, dopunjene na višekratnik od 96 bajtova, a elementi matrice su u zapisu 32 bita bez predznaka, tj. podatkovnog tipa *uint32_t*. Prilikom prepisivanja nominalnih atributa u privremenu matricu, njihove vrijednosti se prevode u zadani podatkovni tip. Podaci se u koprocesor učitavaju iz privremene matrice. Nakon što je prijenos završen, memorija pridijeljena privremenoj matrici se oslobađa.

6.4.2 Izračun matrica frekvencija

Izračun matrica frekvencija također se izvodi na dva načina, ovisno o vrsti atributa koji se obrađuje. Za numeričke atribute, izračun se provodi na CPU-u, izvornim postupkom.

Za nominalne atribute, izračun se provodi na koprocesoru, pozivom odgovarajućih funkcija koje upravljaju radom koprocesora: Iz ključne petlje kojom se pokreće izračun matrice frekvencija za sve atribute, u posebnu petlju je izdvojen dio koji pokreće izračun za numeričke atribute:

```
ForEach(Att, 0, MaxAtt)
    if (SpecialStatus[Att] != IGNORE)
        if (MaxAttVal[Att])
            if (!Tested[Att])
                EvalDiscreteAtt(Att, Fp, Lp, Cases);
```

Ta petlja je zamijenjena jednim funkcijskim pozivom:

```
EvalDiscreteAttDF(srcSet, Cases, Fp, Lp);
```

U pozvanoj funkciji izvršavaju se pripremne radnje:

- pakiranje atributa skupa u grupe od po šest atributa koji se istovremeno obrađuju jednim pozivom koprocesora
- pokretanje koprocesora za svaku grupu
- obrada prihvaćenih rezultata.

Pakiranje atributa u grupe uključuje pripremu polja s početnim adresama atributa i pripremu polja za prihvat izračunatih matrica frekvencija. Za pripremu početnih adresa i za prihvat rezultata iz koprocesora koristi se po jedno polje za svakih šest atributa. Uz sama polja za prihvat prilikom pakiranja stvara se mapa polja u koju se bilježi kojem atributu je pridijeljen dio polja za prihvat rezultata.

Pripremljena polja za svaku grupu koriste se kao parametri za sučelje *computeFreq*, koje je opisano u poglavlju 5.5. Koriste se neblokirajući pozivi sučelja koji omogućuju da se prihvaćeni rezultati jedne grupe počnu obrađivati na CPU-u čim su dostupni, a istovremeno se pokreće obrada slijedeće grupe na koprocesoru.

Obrada rezultata prihvaćenih iz koprocesora obuhvaća sve operacije potrebne za izračun informacijskog dobitka i faktora informacijskog dobitka. Dobiveni rezultati za svaki atribut prosljeđuju se natrag glavnoj funkciji za izgradnju stabla, gdje se koriste za odabir testnog atributa.

6.4.3 Grupiranje

Grupiranje, odnosno podjela ulaznog skupa na podskupove, izvodi se na dva različita načina ovisno o vrsti testnog atributa. Bitan dio procesa grupiranja je usklađivanje sadržaja podskupova na CPU-u i na koprocesoru.

Podjela ulaznog skupa na podskupove (grupiranje) izvodi se na dva načina, ovisno o testnom atributu. Za nominalni testni atribut, podjela se izvršava na koprocesoru pozivom *GroupItems* sučelja uz testni atribut u ulozi polja oznaka grupa. S obzirom na to da je cjeloviti skup prisutan u memoriji, za sinkronizaciju nije potrebna komunikacija s koprocesorom. Stoga se na CPU-u koristi izvorni način grupiranja.

Podjela prema numeričkom atributu uključuje komunikaciju s koprocesorom jer u memoriji koprocesora nisu prisutni nominalni atributi. Iz koprocesora se čitaju identifikacijski brojevi

primjera radi utvrđivanja njihova redoslijeda u memoriji koprocesora. Na temelju identifikacijskih brojeva stvara se polje oznaka grupa koje se u koprocesoru koristi za grupiranje primjera.

Tijek postupka grupiranja prema numeričkom atributu:

1. Stvori se pomoćno polje identifikacijskih brojeva i oznaka grupe. Za svaki primjer iz ulaznog skupa:
    - prepiše se identifikacijski broj
    - upiše se oznaka grupe: „1“ za vrijednosti manje ili jednake pragu, „2“ za vrijednosti veće od praga.
2. Pomoćno polje sortira se prema identifikacijskom broju.
3. Pročitaju se identifikacijski brojevi iz memorije koprocesora i pohrane u posebno polje oznaka grupe.
4. Identifikacijski brojevi iz koprocesora zamijene se odgovarajućim oznakama grupe iz tablice primjer-grupa. Za svaki primjer:
    - binarnim pretraživanjem pronađe se odgovarajući identifikacijski broj u pomoćnom polju
    - oznaka grupe uz pronađeni identifikacijski broj iz pomoćnog polja prepiše se u polje oznaka grupa.
5. Oznake grupa upišu se u memoriju koprocesora.
6. Skup nominalnih atributa u memoriji koprocesora grupira se prema primljenim oznakama grupa pozivom *groupItems* sučelja.

Nakon odabira testnog atributa, prvo se pokrene postupak grupiranja na koprocesoru. Ovisno o vrsti atributa, razlikuje se priprema. Ako se grupira prema nominalnom atributu, iz matrice frekvencija izračuna se broj primjera u svakoj odredišnoj grupi, što se koristi za alociranje memorije koprocesora. Ako se grupira prema numeričkom atributu, provede se prije opisani postupak. Isječak teksta programa za pripremu i pokretanje grupiranja:

```
if (MaxAttVal[BestAtt]) /* Discrete valued attribute. */
    stripeToValFreqInt(BestAtt, valFreqTmp);
else /* Continuous valued attribute. */
    setupGroupByContin(srcSet, Fp, Lp, BestAtt, Node->Cut, valFreqTmp);

splitSetAlloc(groupSink, BestAtt, valFreqTmp);
GroupItemsDF(srcSet, groupSink, BestAtt, Lp - Fp + 1);
```

Pri grupiranju primjera u memoriji CPU-a, funkcija za grupiranje poziva se iz petlje neposredno prije rekurzivnog poziva kojim se postupak gradnje stabla pokreće na stvorenom podskupu:

```
ForEach(v, 1, Node->Forks) {
    Ep = Group(v, Kp, Lp, Node);
    if (Kp <= Ep) {
        Node->Branch[v] = FormTreeInt(Fp, Ep, MaxError, &srcSubset);
        Node->Errors += Node->Branch[v]->Errors;
        MaxError -= Node->Branch[v]->Errors;
        if (MaxError <= 0)
            break;
    } else
        Node->Branch[v] = Leaf(Node->ClassDist, BestClass, 0.0, 0.0);
}
```

Zbog navedene izvedbe grupiranja prvo se mora provesti grupiranje na koprocesoru da ne bi došlo do narušavanja usklađenosti sadržaja skupova.

6.4.4 Obrada podskupa na CPU i na koprocesoru

Operacije izračuna matrice frekvencija i grupiranja objedinjene su u postupak obrade podskupa. Obrada podskupa izvodi se za svaki čvor stabla. Budući da postoji trošak komunikacije između CPU-a i koprocesora, moguće je da za dovoljno male podskupove taj trošak bude veći od vremena provedbe cjelokupne obrade programski na CPU-u. U svrhu usmjeravanja obrade na CPU ili na koprocesor, prije početka obrade ispituje se veličina ulaznog podskupa. Ako je broj primjera u podskupu manji od zadanog praga, sva obrada za taj podskup, odnosno čvor, izvodi se na CPU-u. U protivnom se koristi koprocesor na način koji je opisan ranije u poglavlju.

Kada je obrada podskupa (čvora) usmjerena na CPU, tada se obrada cijelog podstabla provodi na CPU-u, bez sudjelovanja koprocesora. To je posljedica činjenice da izlazni podskupovi čvora stabla mogu jedino biti manji od ulaznog podskupa. S obzirom na to da koprocesor više ne sudjeluje pri obradi takvog podstabla, sinkronizacija sadržaja podskupova između CPU-a i koprocesora nije potrebna i ne provodi se. Koprocesor tijekom ostatka izvođenja algoritma ne pristupa podacima vezanim uz takvo podstablo, pa se taj dio memorije koprocesora oslobađa.

Za manje podskupove očekuje se da je vrijeme obrade na koprocesoru veće od vremena obrade na CPU-u, budući da rad s koprocesorom uključuje dodatne vremenske troškove. Optimalna vrijednost praga ovisi o više faktora: brzini obrade na koprocesoru, kašnjenjima u komunikaciji između CPU-a i koprocesora, stupnju paralelizacije obrade na CPU-u (broj raspoloživih dretvi) itd. Prag za usmjeravanje obrade mora biti odabran tako da se nalazi u

području u kojem su vremena obrade na CPU-u i koprocesoru približno jednaka. Prag se zadaje kao korisnički parametar prilikom pokretanja programa i ostaje nepromijenjen za cijelo vrijeme izvršavanja.

7 Vrednovanje performansi heterogenog računalnog sustava i hibridnog algoritma

7.1 Postupak vrednovanja performansi

Postupak vrednovanja performansi sastoji se od dva dijela:

- vrednovanje performansi pojedinih jezgara u izolaciji
- vrednovanje performansi hibridnog algoritma izvršavanog na heterogenom sustavu.

U tijeku postupka vrednovanja performansi provode se slijedeći postupci:

- mjerenje i analiza utjecaja broja atributa
- mjerenje i analiza utjecaja veličine skupa (broja primjera)
- mjerenje i analiza utjecaja broja dretvi (samo za višedretvene programske implementacije)
- usporedba performansi s postojećim programskim implementacijama algoritma C4.5.

Postupak vrednovanja provodi se na radnoj stanici s Intel Xeon E5-1650 CPU-om, 16 GiB DDR3-1600 RAM-a i Maxeler Vectis-Lite FPGA koprocesorom. Programi se izvršavaju pod CentOS 6.5 Linux operacijskim sustavom, verzije 2.6.32.

U postupku vrednovanja mjeri se cjelokupno vrijeme izvršavanja programa, te vremena izvršavanja ključnih dijelova programa: učitavanje podataka, prijenosa podataka na FPGA (samo za hibridni algoritam), izgradnja stabla i obrezivanje stabla. Ukupno vrijeme učenja stabla je zbroj vremena izgradnje i obrezivanja stabla. Uz izgradnju stabla, mjere se još i posebno vrijeme izračuna matrice frekvencija i vrijeme grupiranja. Mjerenje vremena izvedeno je unutar programa uzimanjem vremenskih oznaka (engl. *timestamp*) na ključnim mjestima u programu, iz kojih se potom izračunava proteklo vrijeme. Svako mjerenje ponavlja se deset puta i iz dobivenih rezultata izračunava se prosjek u obliku aritmetičke sredine.

U postupku se koristi niz sintetičkih skupova podataka i jedan realni skup podataka. Koriste se skupovi koji sadrže najmanje 500×10^3 elemenata, a najviše 500×10^6 elemenata i nemaju

nepoznatih ili nedostajućih vrijednosti. U javno dostupnim repozitorijima pronađen je samo jedan realni skup koji zadovoljava zadane uvjete: skup Covertype iz UCI repozitorija [45]. Ostali korišteni skupovi generirani su pomoću alata *datagen* [87]. Sintetički skupovi su generirani samo s numeričkim atributima. Matični sintetički skup dobiven je programom *datagen* sa sljedećim parametrima:

- ukupno 500 atributa, atributi su binarni
- ukupno 2.000.000 primjera
- 20 klasa.

Naredba za pokretanje programa je:

```
./datgen -O2000000 -R20 -d2/2 -C0/200 -D0/100
```

gdje parametar „-O“ definira broj primjera, „-R“ broj klasa, „-d“ raspon vrijednosti atributa, a „-C“ i „-D“ raspone broja konjunkcija, odnosno disjunkcija korištenih za generiranje pravila kojim se niz generiranih atributa razvrstava u određenu klasu.

Provode se dva ispitivanja: ispitivanje utjecaja broja primjera i utjecaja broja atributa. Ispitni skupovi različitih brojeva primjera, odnosno atributa, generirani su uzorkovanjem iz matičnog skupa. Parametri ispitnih skupova dani su u tablicama 7.1 i 7.2. Za ispitivanje utjecaja broja primjera, broj atributa u skupu je fiksiran na 200, dok je za ispitivanje utjecaja broja atributa broj primjera fiksiran na 1.000.000.

Za realni skup Covertype, parametri su određeni značajkama skupa. Za ispitivanje utjecaja broja primjera, broj atributa je fiksiran na 54, a za ispitivanje utjecaja broja atributa, broj primjera fiksiran je na 1.000.000. Ispitni skupovi generirani su uzorkovanjem ili

Tablica 7.1. Parametri sintetičkih skupova za ispitivanje utjecaja broja primjera

Broj primjera	**Broj ispitnih skupova** (N_{at} = 200)
20.000	125
50.000	50
100.000	25
200.000	12
500.000	5
1.000.000	3
2.000.000	3

Tablica 7.2. Parametri sintetičkih skupova za ispitivanje utjecaja broja atributa

Broj atributa	**Broj ispitnih skupova** (N_{pr} = 1.000.000)
5	100
10	50
20	25
50	10
100	5
200	3
500	3

Tablica 7.3. Parametri skupa *Covertype* za ispitivanje utjecaja broja primjera

Broj primjera	Broj uzoraka (N_{at} = 54)
20.000	500
50.000	200
100.000	100
200.000	50
500.000	20
1.000.000	10
2.000.000	5

Tablica 7.4 Parametri skupa *Covertype* za ispitivanje utjecaja broja atributa

Broj atributa	Broj uzoraka (N_{pr} = 1.000.000)
5	200
10	100
20	50
50	20
100	10
200	5
500	3

ponavljanjem, ovisno o parametrima ispitnog skupa. Za ispitne skupove kojima je zadani broj primjera ili atributa veći nego što je u matičnom skupu, slučajno se biraju primjeri, odnosno atributi koji se ponavljaju. Parametri ispitnih skupova generiranih iz realnog skupa Covertype dani su u tablicama 7.3 i 7.4.

7.2 Vrednovanje jezgre *ComputeFreq* u izolaciji

Prije nego je odlučena konačna izvedba heterogenog sustava, performanse pojedinih jezgara opisanih u poglavlju 5.3.1 vrednovane su u izolaciji. U svrhu ispitivanja, za svaku jezgru definiran je posebni koprocesor i na njima su provedena mjerenja vremena izvršavanja jezgri na nizu skupova podataka.

Mjerenja na jezgri SS provedena su na sintetičkom skupu podataka sa 64 atributa i 4×2^{20} primjera. Vrijednosti atributa i klasa su slučajni cijeli brojevi, raspona vrijednosti od 0 do 63 i jednolike raspodjele. Vremena su mjerena na podskupovima kojima je mijenjan broj atributa i broj primjera. Broj atributa je iznosio od 1 do 64, u nizu 2^i, gdje je i = 0, 1, …, 6. Broj primjera je iznosio od 2048 do 4×2^{20}, u nizu 2^i, gdje je i = 11, 12, …, 22.

Na jezgrama MS i MS2 provedena su dva skupa mjerenja. Dva različita pristupa potrebna su zbog činjenice da jezgre obrađuju po šest atributa istovremeno. Cilj prvog skupa mjerenja je ispitati performanse jezgre za nepotpuno iskorištenje podatkovnih tokova, stoga se broj atributa mijenja od 1 do 6. Kod drugog skupa mjerenja cilj je ispitati performanse pri potpunom iskorištenju podatkovnih tokova jezgre. Stoga je broj atributa višekratnik broja šest i mijenja se od 12 do 1536, u nizu 6×2^i, gdje je i = 1, 2, …, 7. Kod oba skupa mjerenja, broj primjera mijenja se u nizu 2^i, gdje je i = 11, 12, …, 22, tj. od 2048 do 4×2^{20}.

Iz izmjerenih vremena izvršavanja izračunati su protoci jezgara. Protok se izračunava formulom:

$$T_{a,i} = \frac{a\ i}{t_{a,i}} \tag{7.1}$$

gdje je a broj atributa, i broj primjera, $t_{a,i}$ izmjereno vrijeme izvršavanja, a $T_{a,i}$ je izračunat protok jezgre za skup s a atributa i i primjera.

Za mjerenje su korišteni dijelovi programa EC4.5 koji služe za učitavanje skupa u memoriju i izračun matrice frekvencija. Izvornim funkcijama su na odgovarajuća mjesta dodana mjerenja vremena. Uz izvorne, dodane su nove funkcije koje za izračun matrice frekvencija koriste koprocesor, kao i funkcije za prijenos skupa u memoriju koprocesora. Sva opisana mjerenja provode se i na izvornoj programskoj implementaciji. Rezultati programske implementacije koriste se kao osnovica za usporedbu.

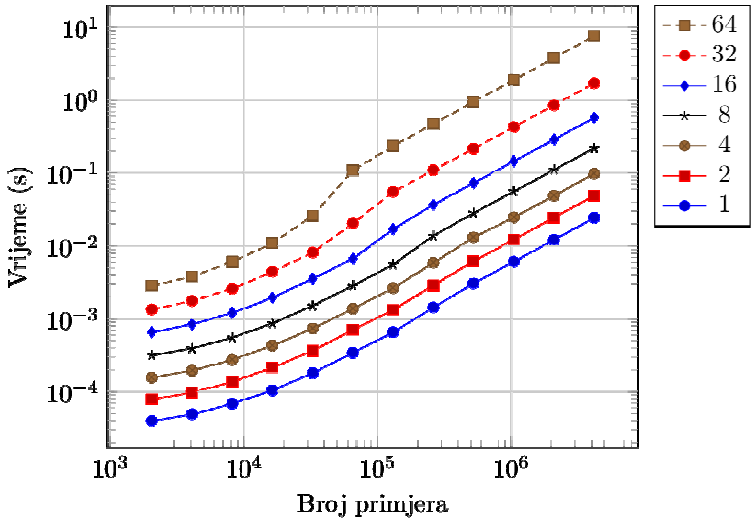
Mjeri se vrijeme izvršavanja funkcije *ComputeFrequencies* koja implementira izračun matrice frekvencija parova atribut-klasa, te frekvencije vrijednosti atributa koje se izračunavaju iz matrice frekvencija zbrajanjem po stupcima. Za mjerenja na programskoj implementaciji, koristi se paralelizirana izvedba. Broj dretvi postavljen je na jednu, za usporedbu s jezgrom SS, te na šest, za usporedbu s jezgrama MS i MS2. Za mjerenja na koprocesoru implementirana je nova funkcija u kojoj se koristi koprocesor za izračun matrice frekvencija, a izračun frekvencija vrijednosti atributa izveden je programski.

7.2.1 Programska implementacija

Mjerenja su provedena na dvije varijante programske implementacije: jednodretvenoj i višedretvenoj. Rezultati mjerenja jednodretvene implementacije osnovica su za usporedbu s rezultatima jezgre SS, dok su rezultati višedretvene osnovica za usporedbu s rezultatima jezgara MS i MS2.

7.2.1.1 Jednodretvena

Vremena izvršavanja programske implementacije izračuna matrice frekvencija nalaze se u tablici 7.5 i prikazana su na slici 7.1. Za skupove s više od 64×2^{10} primjera, vrijeme izvršavanja raste približno linearno s povećanjem broja primjera. U području veličine skupova 128×2^{10} – 256×2^{10} primjera i 8 atributa, do 32×2^{10} – 64×2^{10} primjera i 64 atributa pojavljuje se skok u vremenu izvršavanja.

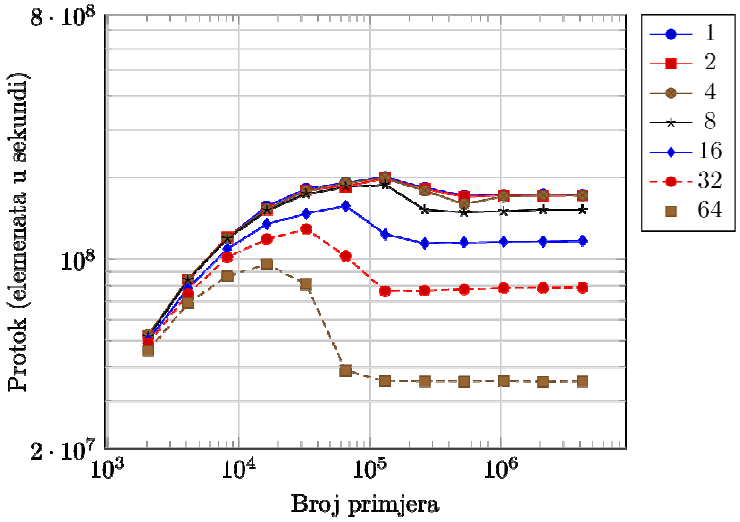


Slika 7.1. Vrijeme izračuna matrice frekvencija na CPU-u s jednom dretvom

Protok funkcije, prikazan na slici 7.2, bolje ilustrira taj pad u performansama. Protok funkcije raste dok se ne dosegne vrh pri veličinama skupa od 16×2^{10} do 128×2^{10}, ovisno o broju atributa. S daljnjim rastom broja primjera, protok funkcije pada i brzo se stabilizira na približno konstantnu vrijednost. Konstantni protok odgovara linearnom skaliranju vremena izvršavanja s brojem primjera. Najveći izmjereni konstantni protok iznosi 173×10^6 elemenata u sekundi za skup sa jednim atributom, a najniži je is $35{,}4\times10^6$ elemenata u sekundi za skup sa 64 atributa. Protoci funkcije dani su u tablici 7.6.

Tablica 7.5. Vremena izračuna matrice frekvencija na CPU-u s jednom dretvom

Broj primjera	Broj atributa / Vrijeme izvršavanja						
	1	2	4	8	16	32	64
2.048	39,62 µs	78,74 µs	156,1 µs	316,6 µs	651,2 µs	1,339 ms	2,859 ms
4.096	48,97 µs	97,27 µs	194,7 µs	392,2 µs	841,7 µs	1,766 ms	3,807 ms
8.192	68,39 µs	136,1 µs	273,6 µs	551,0 µs	1,202 ms	2,582 ms	6,065 ms
16.384	104,5 µs	214,7 µs	428,3 µs	866,7 µs	1,945 ms	4,420 ms	10,94 ms
32.768	180,1 µs	364,4 µs	735,0 µs	1,505 ms	3,556 ms	8,116 ms	25,95 ms
65.536	341,7 µs	705,5 µs	1,372 ms	2,841 ms	6,663 ms	20,45 ms	108,1 ms
131.072	654,0 µs	1,319 ms	2,632 ms	5,561 ms	16,97 ms	55,01 ms	235,7 ms
262.144	1,427 ms	2,885 ms	5,858 ms	13,78 ms	36,63 ms	109,4 ms	473,5 ms
524.288	3,056 ms	6,161 ms	13,11 ms	28,14 ms	72,70 ms	216,4 ms	947,4 ms
1.048.576	6,080 ms	12,19 ms	24,57 ms	55,80 ms	144,7 ms	428,8 ms	1,889 s
2.097.152	12,10 ms	24,49 ms	48,83 ms	110,1 ms	288,7 ms	855,6 ms	3,800 s
4.194.304	24,30 ms	48,82 ms	97,81 ms	220,5 ms	574,6 ms	1,707 s	7,575 s

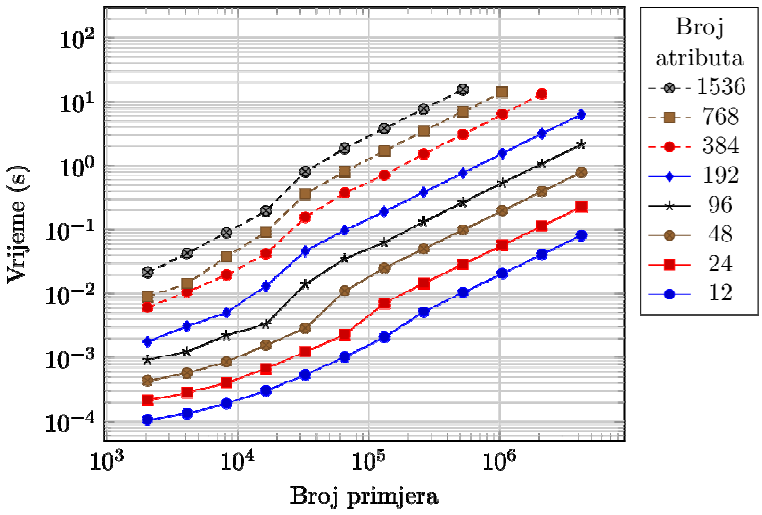


Slika 7.2. Protok funkcije za izračun matrice frekvencija na CPU-u sa jednom dretvom

Izmjereni pad u performansama programske implementacije posljedica je međudjelovanja sustava priručnih memorija u CPU-u i glavne SDRAM memorije. Kod manjih skupova manja je vjerojatnost promašaja priručne memorije – pokušaja dohvata podatka koji nije učitan u priručnu memoriju već se mora dohvatiti iz sporije glavne memorije. Dodatni faktor koji igra ulogu su podatkovne strukture za pohranu skupa za učenje. U postupku izračuna matrice frekvencija čitaju se vrijednosti istog atributa različitih primjera. U memoriji računala skup je pohranjen kao niz polja primjera u kojima su pohranjene vrijednosti atributa, kao što je opisano u poglavlju 6.4.1. To vodi uzastopnim pristupima memorijskim lokacijama koje nisu

Tablica 7.6. Protoci funkcije za izračun matrice frekvencija na CPU-u sa jednom dretvom

	Broj atributa / Protok (×10^6 elemenata u sekundi)						
Broj primjera	**1**	**2**	**4**	**8**	**16**	**32**	**64**
2.048	2,654	2,869	3,063	3,266	3,680	3,840	3,889
4.096	5,103	5,437	5,914	6,501	6,931	7,388	7,344
8.192	9,574	10,58	11,28	12,11	13,04	13,53	13,37
16.384	17,95	18,94	18,83	20,20	21,51	21,88	22,53
32.768	27,86	28,93	29,98	30,69	32,44	32,20	32,58
65.536	38,68	39,05	39,89	41,07	42,27	42,69	43,42
131.072	49,26	48,76	49,16	51,06	50,99	51,23	49,74
262.144	56,01	55,57	56,77	56,60	56,82	56,34	56,74
524.288	60,98	60,63	60,12	60,71	60,65	61,33	60,85
1.048.576	63,72	63,35	63,28	63,38	63,39	63,60	63,53
2.097.152	64,91	65,13	64,89	64,96	64,83	64,92	64,93
4.194.304	65,79	65,91	65,81	65,75	65,79	65,78	65,76



Slika 7.3. Vrijeme izračuna matrice frekvencija na CPU-u sa šest dretvi

nužno u bliskom susjedstvu. Što je broj atributa veći, to je veća međusobna udaljenost istog atributa susjednih primjera, što povećava vjerojatnost promašaja priručne memorije.

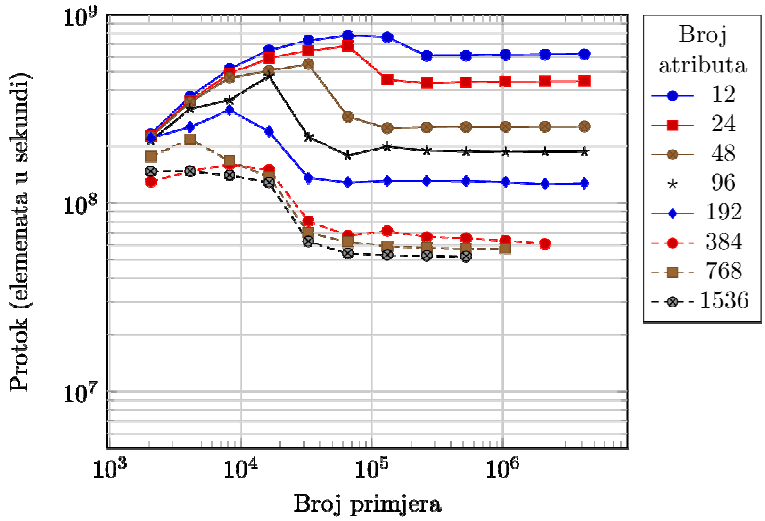
7.2.1.2 Višedretvena

Mjerenja na višedretvenoj implementaciji provedena su sa šest dretvi, što je ograničeno brojem jezgara u CPU-u ispitnog računala. Budući da jezgre MS i MS2 imaju šest ulaznih podatkovnih tokova za atribute, moguća je izravna usporedba rezultata.

Izmjerena vremena izvršavanja dana su u tablici 7.7 i prikazana na slici 7.3. Kao i na jednodretvenoj programskoj implementaciji, vidljiv je skok u vremenu izvršavanja. U ovoj implementaciji skok se pojavljuje u području veličine skupova 16×2^{10} – 32×2^{10} za skupove s

Tablica 7.7. Vremena izračuna matrice frekvencija na CPU-u sa šest dretvi

	Broj atributa / Vrijeme izvršavanja							
Broj primjera	**12**	**24**	**48**	**96**	**192**	**384**	**768**	**1.536**
2.048	105,3 µs	215,0 µs	434,9 µs	916,9 µs	1,769 ms	6,049 ms	8,887 ms	21,32 ms
4.096	133,5 µs	280,4 µs	571,0 µs	1,237 ms	3,096 ms	10,59 ms	14,37 ms	42,51 ms
8.192	190,1 µs	403,6 µs	847,8 µs	2,228 ms	5,027 ms	19,56 ms	37,46 ms	89,34 ms
16.384	302,8 µs	668,8 µs	1,557 ms	3,321 ms	13,08 ms	41,85 ms	91,15 ms	195,6 ms
32.768	538,0 µs	1,219 ms	2,867 ms	13,99 ms	46,18 ms	156,3 ms	360,6 ms	803,1 ms
65.536	1,014 ms	2,296 ms	10,92 ms	35,06 ms	97,45 ms	373,2 ms	808,1 ms	1,860 s
131.072	2,066 ms	6,949 ms	25,08 ms	62,94 ms	191,5 ms	706,2 ms	1,702 s	3,795 s
262.144	5,184 ms	14,48 ms	49,73 ms	132,1 ms	382,5 ms	1,518 s	3,467 s	7,658 s
524.288	10,35 ms	28,61 ms	98,80 ms	267,1 ms	766,7 ms	3,079 s	6,987 s	15,44 s
1.048.576	20,48 ms	56,80 ms	197,3 ms	537,8 ms	1,553 s	6,359 s	14,01 s	–
2.097.152	40,80 ms	113,3 ms	394,1 ms	1,071 s	3,176 s	13,21 s	–	–
4.194.304	81,27 ms	226,3 ms	787,0 ms	2,139 s	6,303 s	–	–	–



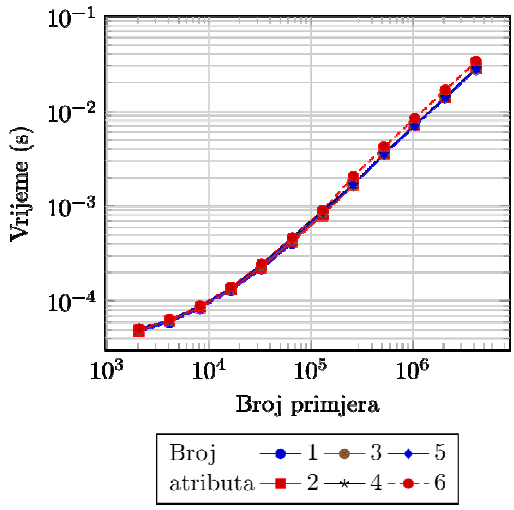
Slika 7.4. Protok funkcije za izračun matrice frekvencija na CPU-u sa šest dretvi

96 i više atributa, te u području 32×2^{10} – 256×2^{10} za skupove s manje od 96 atributa.

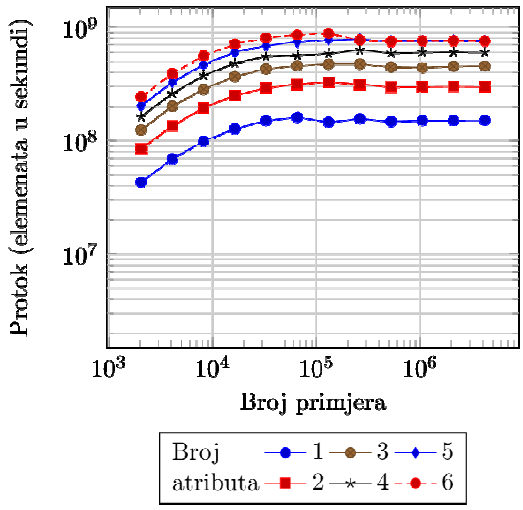
Skok u vremenu izvršavanja očituje se kao pad protoka funkcije, prikazan na slici 7.4. Protok funkcije za manji broj atributa – 192 i manji – sličan je kao kod jednodretvene implementacije. Za manje skupove protok raste dok ne dostigne maksimum i potom brzo padne na vrijednost koja ostaje približno konstantna. Povećanjem broja atributa smanjuje se veličina skupa na kojoj se dostiže maksimum, kao i sam iznos maksimuma. Za skupove s 384 i više atributa nema izraženog rasta protoka kao za manji boj atributa, već se protok s jedne približno konstantne vrijednosti smanji na drugu. Također, vidljivo je da grafovi protoka za te skupove čine tijesnu skupinu. Iz toga se može zaključiti da se daljnjim povećanjem broja

Tablica 7.8. Protoci funkcije za izračun matrice frekvencija na CPU-u sa šest dretvi

	Broj atributa / Protok (×10^6 elemenata u sekundi)							
Broj primjera	**12**	**24**	**48**	**96**	**192**	**384**	**768**	**1.536**
2.048	233,4	228,6	226,0	214,4	222,3	130,0	177,0	147,5
4.096	368,1	350,6	344,4	317,8	254,0	148,6	218,9	148,0
8.192	517,1	487,1	463,8	352,9	312,9	160,8	168,0	140,8
16.384	649,4	587,9	505,2	473,6	240,4	150,3	138,1	128,7
32.768	730,9	645,2	548,6	224,8	136,2	80,51	69,79	62,67
65.536	775,3	685,0	288,2	179,4	129,1	67,44	62,29	54,13
131.072	761,5	452,7	250,8	199,9	131,4	71,27	59,15	53,05
262.144	606,8	434,6	253,0	190,5	131,6	66,31	58,07	52,58
524.288	607,9	439,8	254,7	188,5	131,3	65,39	57,63	52,16
1.048.576	614,3	443,1	255,1	187,2	129,7	63,33	57,49	–
2.097.152	616,8	444,2	255,4	188,0	126,8	60,96	–	–
4.194.304	619,3	444,9	255,8	188,3	127,8	–	–	–



Slika 7.5. Vrijeme izračuna matrice frekvencija na CPU-u sa šest dretvi uz nepotpuno iskorištenje dretvi



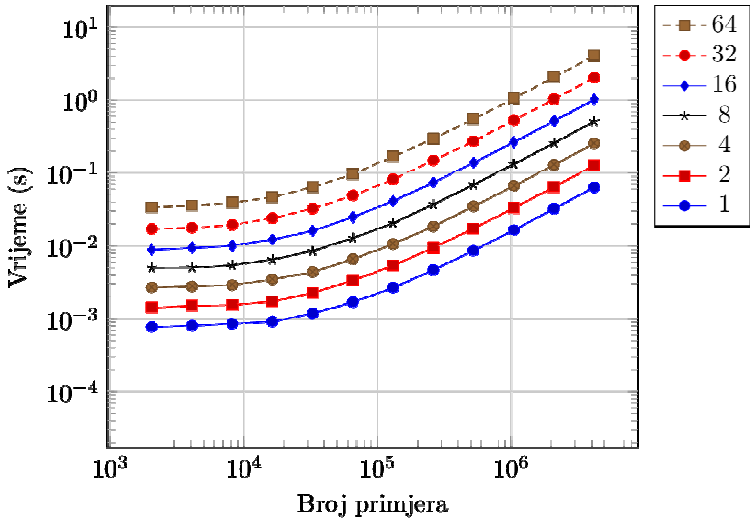
Slika 7.6. Protok funkcije za izračun matrice frekvencija na CPU-u sa šest dretvi uz nepotpuno iskorištenje dretvi

atributa neće bitno smanjivati protok funkcije. Iznosi protoka dani su u tablici 7.8. Maksimalni konstantni protok iznosi 617×10^6 elemenata u sekundi za skupove s 12 atributa, a minimalni iznosi $52{,}2\times10^6$ elemenata u sekundi, za skupove s 1536 atributa.

Na slikama 7.5 i 7.6 prikazano je vrijeme izvršavanja, odnosno protok funkcije, za slučaj nepotpunog iskorištenja dretvi. Broj atributa je mijenjan od jedan do šest, gdje skup sa šest atributa služi kao osnovica za usporedbu. Iz prikaza vremena izvršavanja vidljivo je da nema razlike s obzirom na broj atributa, što je očekivan rezultat. Jedina razlika vidi se u malom porastu vremena izvršavanja za skup sa šest atributa, gdje za skupove s 256×2^{10} i više

Tablica 7.9. Vremena izračuna matrice frekvencija na CPU-u sa šest dretvi uz nepotpuno iskorištenje dretvi

	Broj atributa / Vrijeme izvršavanja					
Broj primjera	**1**	**2**	**3**	**4**	**5**	**6**
2.048	59,63 µs	60,99 µs	61,30 µs	63,03 µs	62,79 µs	63,31 µs
4.096	47,81 µs	48,60 µs	49,52 µs	50,52 µs	50,71 µs	50,59 µs
8.192	83,23 µs	84,50 µs	86,96 µs	87,80 µs	87,98 µs	88,21 µs
16.384	129,5 µs	131,4 µs	134,1 µs	136,8 µs	136,3 µs	139,2 µs
32.768	218,9 µs	224,9 µs	230,4 µs	238,6 µs	238,8 µs	245,6 µs
65.536	408,7 µs	419,2 µs	430,9 µs	466,8 µs	445,5 µs	462,1 µs
131.072	901,9 µs	800,0 µs	834,9 µs	895,5 µs	852,9 µs	899,0 µs
262.144	1,688 ms	1,681 ms	1,667 ms	1,673 ms	1,677 ms	2,056 ms
524.288	3,564 ms	3,547 ms	3,551 ms	3,552 ms	3,503 ms	4,215 ms
1.048.576	7,004 ms	7,052 ms	7,184 ms	7,001 ms	6,991 ms	8,365 ms
2.097.152	13,94 ms	14,01 ms	14,01 ms	13,94 ms	13,83 ms	16,70 ms
4.194.304	27,82 ms	28,20 ms	27,82 ms	28,06 ms	27,80 ms	33,49 ms



Slika 7.7. Vrijeme izvršavanja jezgre *ComputeFreq* SS

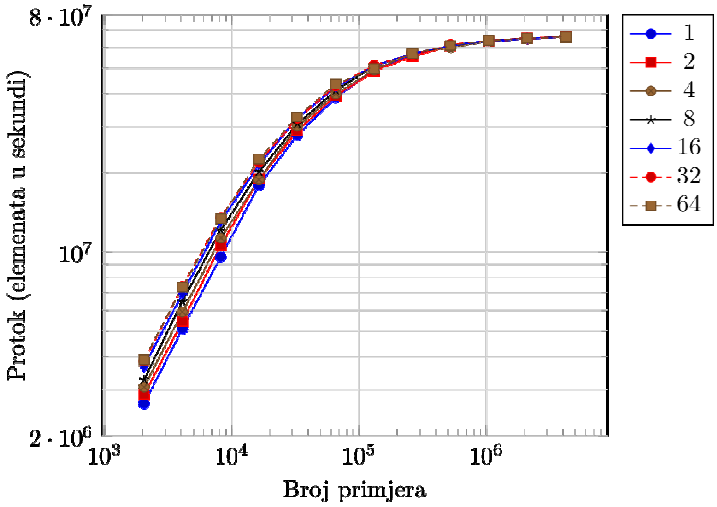
primjera počinju dolaziti do izražaja promašaji priručne memorije. U skladu s jednakim vremenom izvršavanja s rastom broja atributa, protok raste s porastom broja atributa. Izmjerena vremena izvršavanja dana su u tablici 7.9, dok su izračunati protoci dani u prilogu A, tablica A.1.

7.2.2 Jezgra *ComputeFreq* SS

Jezgra SS ispitivana je pod istim uvjetima kao jednodretvena programska implementacija. Izmjerena vremena izvršavanja jezgre dana su u tablici 7.10 i prikazana na slici 7.7. Na slici je vidljivo da sa skupove s više od 128×2^{10} primjera vrijeme izvršavanja raste približno linearno s brojem primjera. Za skupove s manjim brojem primjera, vrijeme izvršavanja je približno

Tablica 7.10. Vremena izračuna matrice frekvencija na jezgri *ComputeFreq* SS

	Broj atributa / Vrijeme izvršavanja						
Broj primjera	**1**	**2**	**4**	**8**	**16**	**32**	**64**
2.048	764,4 µs	1,378 ms	2,619 ms	5,023 ms	8,955 ms	17,27 ms	34,51 ms
4.096	805,4 µs	1,501 ms	2,845 ms	5,180 ms	9,648 ms	19,36 ms	37,80 ms
8.192	904,6 µs	1,740 ms	3,367 ms	6,314 ms	12,04 ms	23,36 ms	46,67 ms
16.384	1,154 ms	2,231 ms	4,306 ms	8,340 ms	16,03 ms	31,76 ms	64,22 ms
32.768	1,650 ms	3,280 ms	6,350 ms	12,28 ms	24,47 ms	48,17 ms	97,13 ms
65.536	2,658 ms	5,272 ms	10,48 ms	20,67 ms	40,44 ms	81,38 ms	161,7 ms
131.072	4,622 ms	9,334 ms	18,68 ms	36,74 ms	73,25 ms	148,4 ms	292,9 ms
262.144	8,635 ms	17,28 ms	34,46 ms	68,86 ms	136,1 ms	274,8 ms	548,7 ms
524.288	16,48 ms	33,03 ms	66,02 ms	132,4 ms	264,8 ms	529,6 ms	1,062 s
1.048.576	32,25 ms	64,85 ms	128,6 ms	258,3 ms	515,1 ms	1,032 s	2,060 s
2.097.152	63,76 ms	127,4 ms	254,8 ms	509,8 ms	1,021 s	2,040 s	4,074 s
4.194.304	126,6 ms	253,4 ms	506,7 ms	1,013 s	2,028 s	4,052 s	8,108 s



Slika 7.8. Protok jezgre *ComputeFreq* SS

konstantno. Za razliku od programske implementacije, kod izvršavanja na koprocesoru ne pojavljuje se pad performansi na većim skupovima. Razlog tome je da koprocesor nema sustav priručne memorije, te da je korištena podatkovna struktura transponirana u odnosu na onu iz CPU-a. Transponirana pohrana skupa za učenje omogućuje slijedno čitanje vrijednosti atributa, što je najpovoljnije s obzirom na iskorištenje propusnosti SDRAM memorije.

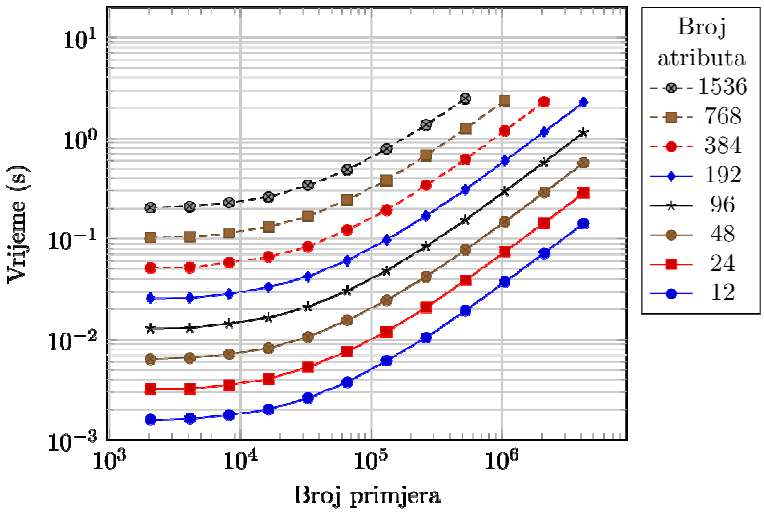
Sa slike 7.8 koja prikazuje protok jezgre, vidljivo je da su za sve veličine skupova izmjereni protoci malo razlikuju. Razlike u izmjerenom protoku gotovo nestaju za veće skupove. Izmjerene razlike su vjerojatnije uslijed pogreške mjerenja vremena, nego uslijed stvarne razlike u protoku, s obzirom na to da se veće rasipanje u rezultatima pojavljuje na manjim skupovima. Protoci jezgre dani su u tablici 7.11.

Iz izmjerenih rezultata vidljivo je da vrijeme izvršavanja skalira linearno s brojem atributa. Linearno skaliranje vidljivo je u vremenu izvršavanja kao konstantni pomak između krivulja

Tablica 7.11. Protoci jezgre *ComputeFreq* SS

	Broj atributa / Protok (×10^6 elemenata u sekundi)						
Broj primjera	**1**	**2**	**4**	**8**	**16**	**32**	**64**
2.048	2,654	2,869	3,063	3,266	3,680	3,840	3,889
4.096	5,103	5,437	5,914	6,501	6,931	7,388	7,344
8.192	9,574	10,58	11,28	12,11	13,04	13,53	13,37
16.384	17,95	18,94	18,83	20,20	21,51	21,88	22,53
32.768	27,86	28,93	29,98	30,69	32,44	32,20	32,58
65.536	38,68	39,05	39,89	41,07	42,27	42,69	43,42
131.072	49,26	48,76	49,16	51,06	50,99	51,23	49,74
262.144	56,01	55,57	56,77	56,60	56,82	56,34	56,74
524.288	60,98	60,63	60,12	60,71	60,65	61,33	60,85
1.048.576	63,72	63,35	63,28	63,38	63,39	63,60	63,53
2.097.152	64,91	65,13	64,89	64,96	64,83	64,92	64,93
4.194.304	65,79	65,91	65,81	65,75	65,79	65,78	65,76

za različite brojeve atributa, dok je iz protoka vidljiv kao tijesna grupa krivulja. Za skupove s 256×2^{10} i više primjera, protok je približno konstantan, i iznosi $62{,}4 \times 10^6$ primjera u sekundi, što je blizu teoretskom maksimumu od $66{,}6 \times 10^6$ primjera u sekundi. Slabije performanse na manjim skupovima posljedica su vremenskog troška komunikacije CPU-a i koprocesora. Taj vremenski trošak je neovisan o veličini skupa, što se vidi kao približno vodoravni dio krivulje u prikazu vremena izvršavanja na slici 7.7. Budući da se isti niz funkcijskih poziva mora izvršiti za obradu svakog atributa, vremenski trošak skalira linearno s brojem atributa.



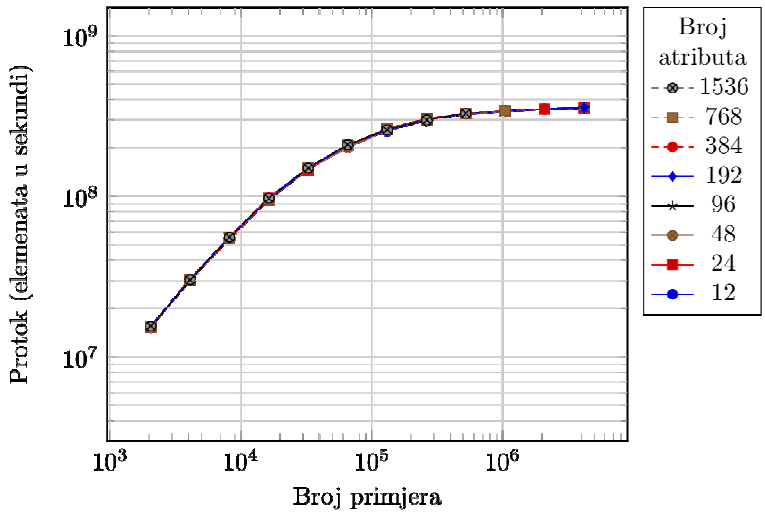
Slika 7.9. Vrijeme izvršavanja jezgre *ComputeFreq* MS

7.2.3 Jezgra *ComputeFreq* MS

Jezgra *ComputeFreq* MS ispitivana je pod istim uvjetima kao višedretvena programska implementacija. U tablici 7.12 dana su vremena izvršavanja jezgre, koja su grafički prikazana na slici 7.9. Iz grafa je vidljivo da vrijeme izvršavanja raste približno linearno s veličinom skupa za skupove sa 128×2^{10} i više primjera. Za manji broj primjera, vrijeme izvršavanja

Tablica 7.12. Vremena izračuna matrice frekvencija na jezgri *ComputeFreq* MS

	Broj atributa / Vrijeme izvršavanja							
Broj primjera	**12**	**24**	**48**	**96**	**192**	**384**	**768**	**1.536**
2.048	1,597 ms	3,220 ms	6,385 ms	12,85 ms	25,69 ms	51,28 ms	102,4 ms	203,3 ms
4.096	1,634 ms	3,248 ms	6,559 ms	13,00 ms	25,89 ms	51,89 ms	104,1 ms	209,2 ms
8.192	1,780 ms	3,544 ms	7,133 ms	14,39 ms	28,32 ms	58,37 ms	113,5 ms	227,2 ms
16.384	2,014 ms	4,043 ms	8,187 ms	16,47 ms	33,04 ms	65,85 ms	131,8 ms	260,3 ms
32.768	2,626 ms	5,338 ms	10,55 ms	21,02 ms	42,13 ms	83,68 ms	167,0 ms	337,4 ms
65.536	3,758 ms	7,576 ms	15,54 ms	30,34 ms	60,90 ms	123,1 ms	243,9 ms	483,3 ms
131.072	6,153 ms	11,97 ms	24,43 ms	48,17 ms	97,41 ms	192,2 ms	381,3 ms	779,6 ms
262.144	10,40 ms	20,69 ms	42,10 ms	83,98 ms	169,3 ms	338,4 ms	675,4 ms	1,356 s
524.288	19,35 ms	38,62 ms	77,18 ms	153,9 ms	308,5 ms	616,5 ms	1,236 s	2,467 s
1.048.576	37,08 ms	74,18 ms	147,9 ms	296,3 ms	594,5 ms	1,184 s	2,374 s	–
2.097.152	72,08 ms	144,1 ms	288,2 ms	576,0 ms	1,154 s	2,307 s	–	–
4.194.304	142,0 ms	284,0 ms	567,9 ms	1,136 s	2,272 s	–	–	–



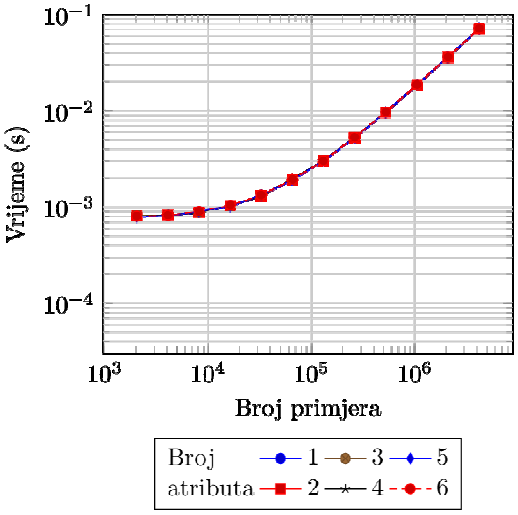
Slika 7.10. Protok jezgre *ComputeFreq* MS

asimptotski se približava određenoj minimalnoj vrijednosti.

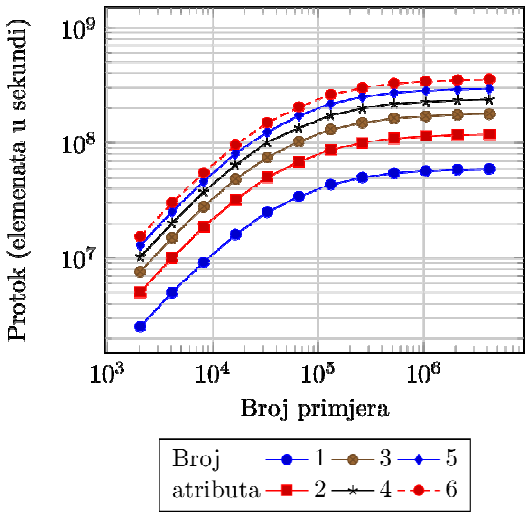
Za sve veličine skupova vrijeme izvršavanja skalira linearno s brojem atributa. To se dobro vidi na grafu protoka jezgre prikazanom na slici 7.10. Grafovi protoka za sve brojeve atributa čine tijesnu skupinu u kojoj se ne mogu razlikovati pojedinačne krivulje protoka. Iz vrijednosti protoka, koji su dani u tablici 7.13, vidi se da za jednak broj primjera u skupu nema bitne razlike u protoku za različiti broj atributa. Izmjerene razlike posljedica su mjerne nesigurnosti. S porastom broja primjera raste i protok jezgre, dok se za skupove s $512{\times}2^{10}$ primjera može protok smatrati približno konstantnim i iznosa $340{\times}10^6$ elemenata u sekundi. Ta izmjerena vrijednost blizu je najvećoj teoretskoj propusnosti od $360{\times}10^6$ elemenata u

Tablica 7.13. Protoci jezgre *ComputeFreq* MS

	Broj atributa / Protok (×10^6 elemenata u sekundi)							
Broj primjera	**12**	**24**	**48**	**96**	**192**	**384**	**768**	**1.536**
2.048	15,39	15,27	15,40	15,29	15,31	15,33	15,35	15,47
4.096	30,07	30,26	29,97	30,24	30,37	30,31	30,19	30,07
8.192	55,23	55,48	55,13	54,65	55,53	53,89	55,39	55,36
16.384	97,65	97,26	96,05	95,48	95,20	95,54	95,42	96,66
32.768	149,7	147,3	149,1	149,6	149,3	150,4	150,7	149,1
65.536	209,3	207,6	202,4	207,3	206,6	204,3	206,3	208,3
131.072	255,6	262,6	257,5	261,2	258,3	261,7	264,0	258,2
262.144	302,3	304,0	298,9	299,7	297,3	297,4	298,1	297,0
524.288	325,1	325,8	326,1	326,9	326,3	326,5	325,8	326,5
1.048.576	339,3	339,2	340,2	339,6	338,6	340,1	339,2	–
2.097.152	349,1	349,1	349,2	349,5	349,1	349,1	–	–
4.194.304	354,4	354,4	354,5	354,4	354,5	–	–	–



Slika 7.11. Vrijeme izvršavanja jezgre *ComputeFreq* MS uz nepotpuno iskorištenje podatkovnih tokova



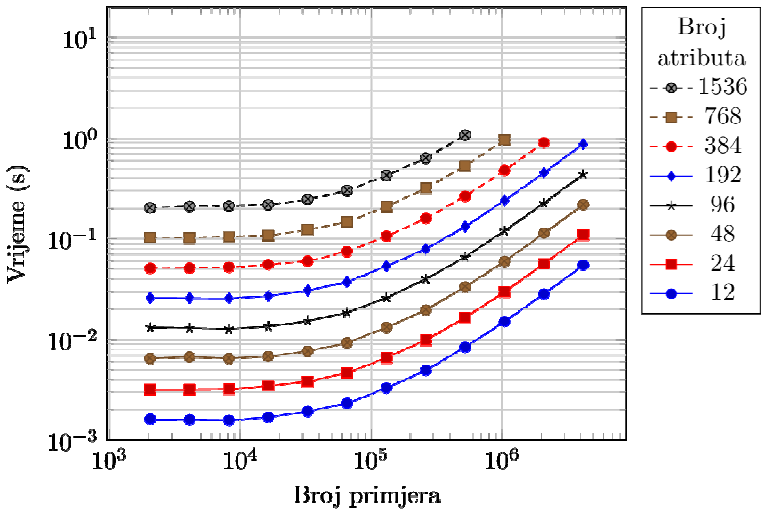
Slika 7.12. Protok jezgre *ComputeFreq* MS uz nepotpuno iskorištenje podatkovnih tokova

sekundi. Povećanjem broja primjera u skupu propusnost jezgre asimptotski se približava teoretskoj najvećoj propusnosti.

Na slikama 7.11 i 7.12 prikazani su vremena izvršavanja, odnosno protoci jezgre MS uz nepotpuno iskorištenje podatkovnih tokova atributa. Broj atributa mijenjan je od jedan do šest. Vrijeme izvršavanja jednako je za sve vrijednosti atributa, što je u skladu s arhitekturom i načinom rada jezgre. Protok jezgre raste s povećanjem broja atributa, što je u skladu s jednakim vremenom izvršavanja za svaki broj atributa. Vremena izvršavanje jezgra dana su u

Tablica 7.14. Vremena izvršavanja jezgre *ComputeFreq* MS uz nepotpuno iskorištenje podatkovnih tokova

	Broj atributa / Vrijeme izvršavanja					
Broj primjera	**1**	**2**	**3**	**4**	**5**	**6**
2.048	810,4 µs	809,9 µs	814,1 µs	800,5 µs	793,8 µs	800,6 µs
4.096	815,1 µs	822,2 µs	820,2 µs	819,8 µs	814,8 µs	817,1 µs
8.192	894,7 µs	881,5 µs	882,6 µs	877,0 µs	887,9 µs	898,9 µs
16.384	1,028 ms	1,026 ms	1,020 ms	1,025 ms	1,016 ms	1,031 ms
32.768	1,315 ms	1,313 ms	1,322 ms	1,303 ms	1,325 ms	1,322 ms
65.536	1,925 ms	1,920 ms	1,924 ms	1,948 ms	1,903 ms	1,929 ms
131.072	3,040 ms	3,026 ms	3,021 ms	3,032 ms	3,019 ms	3,009 ms
262.144	5,279 ms	5,282 ms	5,278 ms	5,272 ms	5,236 ms	5,277 ms
524.288	9,624 ms	9,647 ms	9,654 ms	9,656 ms	9,659 ms	9,664 ms
1.048.576	18,48 ms	18,52 ms	18,54 ms	18,54 ms	18,54 ms	18,55 ms
2.097.152	36,00 ms	36,00 ms	35,97 ms	36,04 ms	36,04 ms	36,03 ms
4.194.304	70,98 ms	70,97 ms	70,99 ms	70,96 ms	70,98 ms	70,96 ms



Slika 7.13. Vrijeme izvršavanja jezgre *ComputeFreq* MS2

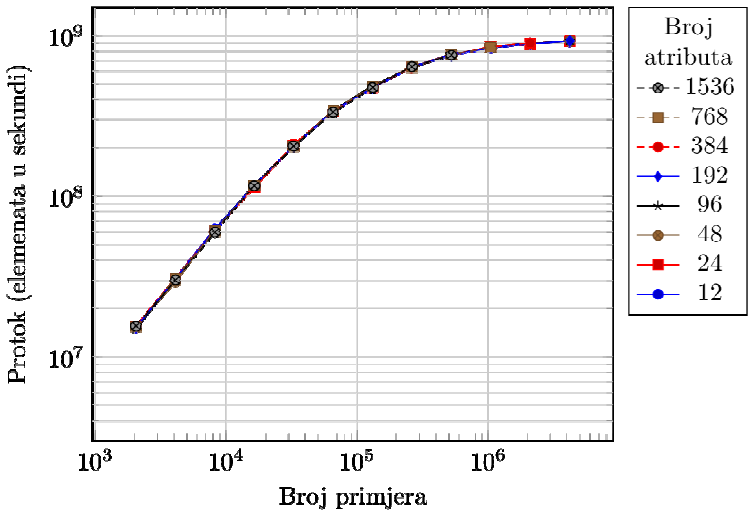
tablici 7.14, a izračunati protoci jezgre dani su u prilogu A, tablici A.2.

7.2.4 Jezgra *ComputeFreq* MS2

Jezgra *ComputeFreq* MS2 ispitivana je pod istim uvjetima kao višedretvena programska implementacija. Na slici 7.13 i u tablici 7.15 prikazani su rezultati mjerenja vremena izvršavanja jezgre. Kao i kod jezgara SS i MS vremena izvršavanja približno su konstantna za manje skupove. Vrijeme izvršavanja je konstantno za skupove do $16{\times}2^{10}$ primjera, a približno linearno raste za skupove s više $256{\times}2^{10}$ primjera i više. Također, vrijeme izvršavanja linearno skalira s brojem atributa.

Tablica 7.15. Vremena izračuna matrice frekvencija za jezgru *ComputeFreq* MS2

	Broj atributa / Vrijeme izvršavanja							
Broj primjera	**12**	**24**	**48**	**96**	**192**	**384**	**768**	**1.536**
2.048	1,611 ms	3,178 ms	6,456 ms	13,15 ms	25,89 ms	50,72 ms	102,5 ms	202,7 ms
4.096	1,591 ms	3,179 ms	6,669 ms	12,98 ms	25,76 ms	50,99 ms	101,8 ms	209,9 ms
8.192	1,573 ms	3,201 ms	6,457 ms	12,73 ms	25,60 ms	52,04 ms	104,0 ms	211,6 ms
16.384	1,690 ms	3,454 ms	6,776 ms	13,42 ms	26,97 ms	55,25 ms	107,9 ms	217,5 ms
32.768	1,923 ms	3,815 ms	7,615 ms	15,28 ms	30,70 ms	60,02 ms	123,1 ms	246,3 ms
65.536	2,312 ms	4,646 ms	9,236 ms	18,45 ms	37,14 ms	74,50 ms	147,2 ms	302,4 ms
131.072	3,289 ms	6,571 ms	13,17 ms	25,86 ms	53,40 ms	106,0 ms	209,2 ms	425,1 ms
262.144	4,927 ms	9,843 ms	19,58 ms	39,81 ms	79,01 ms	158,8 ms	317,6 ms	631,8 ms
524.288	8,360 ms	16,48 ms	33,29 ms	65,68 ms	132,1 ms	265,0 ms	529,4 ms	1,070 s
1.048.576	15,02 ms	29,56 ms	59,44 ms	119,7 ms	239,3 ms	476,3 ms	955,5 ms	–
2.097.152	28,23 ms	56,29 ms	112,8 ms	225,5 ms	452,0 ms	900,5 ms	–	–
4.194.304	54,62 ms	108,9 ms	218,1 ms	436,1 ms	873,2 ms	–	–	–

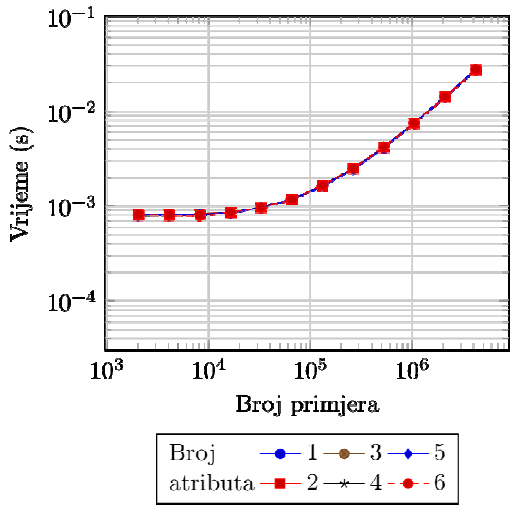


Slika 7.14. Protok jezgre *ComputeFreq* MS2

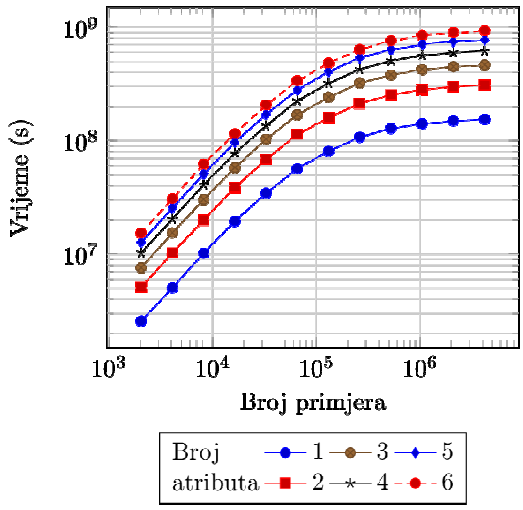
Linearno skaliranje broja atributa potvrđuje se na grafu protoka jezgre, prikazanom na slici 7.14, gdje su krivulje protoka za pojedini broj atributa nerazlučive jedne od druge. Protok jezgre raste s povećanjem broja atributa, i to približno linearno, za skupove s do $64{\times}2^{10}$ primjera. Za veće skupove dolazi do zasićenja protoka, što u prikazu vremena izvršavanja vidljivo kao linearni rast. Približno konstantna vrijednost protoka procijenjena je iz vrijednosti protoka za skupove s $2{\times}10^{20}$ i $4{\times}2^{20}$ primjera i iznosi $906{\times}10^6$ elemenata u sekundi. To je blizu teoretski najvećem protoku, koji iznosi $960{\times}10^6$ elemenata u sekundi. Prema iznosima protoka, danim u tablici 7.16, vidljivo je da i kod skupova s najviše primjera protok i dalje

Tablica 7.16. Protoci jezgre *ComputeFreq* MS2

	Broj atributa / Protok ($\times10^6$ elemenata u sekundi)							
Broj primjera	**12**	**24**	**48**	**96**	**192**	**384**	**768**	**1.536**
2.048	15,25	15,47	15,23	14,95	15,18	15,50	15,34	15,51
4.096	30,89	30,92	29,48	30,28	30,53	30,84	30,89	29,97
8.192	62,49	61,42	60,90	61,76	61,43	60,44	60,49	59,45
16.384	116,4	113,9	116,1	117,1	116,6	113,9	116,6	115,7
32.768	204,5	206,2	206,6	205,9	204,9	209,6	204,4	204,4
65.536	340,2	338,6	340,6	340,9	338,8	337,8	341,8	332,8
131.072	478,3	478,7	477,5	486,4	471,3	474,4	481,1	473,5
262.144	638,5	639,2	642,6	632,1	637,0	633,6	633,9	637,2
524.288	752,6	763,2	755,9	766,3	761,6	759,7	760,5	752,9
1.048.576	837,6	851,3	846,7	840,8	841,0	845,3	842,7	–
2.097.152	891,4	894,0	892,1	892,6	890,7	894,2	–	–
4.194.304	921,4	924,0	922,9	923,1	922,2	–	–	–



Slika 7.15. Vrijeme izvršavanja jezgre *ComputeFreq* MS2 uz nepotpuno iskorištenje podatkovnih tokova



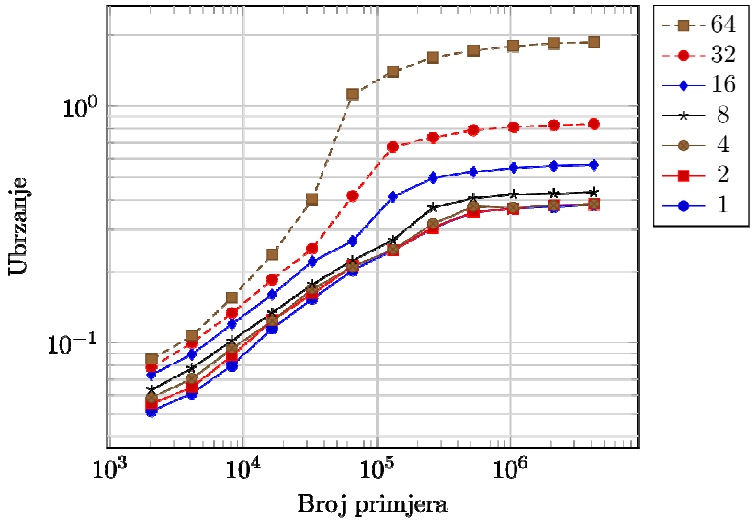
Slika 7.16. Protok jezgre *ComputeFreq* MS2 uz nepotpuno iskorištenje podatkovnih tokova

raste, te se može očekivati da se za veće skupove on asimptotski približava teoretskoj najvećoj vrijednosti 960×10^6 elemenata u sekundi.

Na slikama 7.15 i 7.16 prikazana su vremena izvršavanja jezgre, odnosno protoci jezgre za nepotpuno iskorištenje podatkovnih tokova, tj. za broj atributa od jedan do šest. Kao i kod jezgre MS nema razlike u vremenu izvršavanja za različiti broj atributa, što se vidi i iz tablice 7.17. U skladu s vremenom izvršavanja koje ne ovisi o broju atributa, protok jezgre raste s porastom broja atributa. Izračunati protoci jezgre dani su u prilogu A, tablica A.3.

Tablica 7.17. Vremena izvršavanja jezgre *ComputeFreq* MS2 uz nepotpuno iskorištenje podatkovnih tokova

	Broj atributa / Vrijeme izvršavanja					
Broj primjera	**1**	**2**	**3**	**4**	**5**	**6**
2.048	799,8 µs	803,6 µs	811,8 µs	797,5 µs	805,7 µs	804,2 µs
4.096	811,6 µs	802,0 µs	799,0 µs	803,0 µs	803,8 µs	799,4 µs
8.192	808,4 µs	824,7 µs	813,8 µs	803,5 µs	809,7 µs	792,3 µs
16.384	841,3 µs	848,8 µs	855,4 µs	848,4 µs	848,2 µs	854,0 µs
32.768	954,7 µs	959,6 µs	956,5 µs	970,1 µs	962,2 µs	957,0 µs
65.536	1,161 ms	1,162 ms	1,166 ms	1,168 ms	1,176 ms	1,169 ms
131.072	1,631 ms	1,645 ms	1,634 ms	1,641 ms	1,621 ms	1,638 ms
262.144	2,449 ms	2,468 ms	2,446 ms	2,472 ms	2,439 ms	2,474 ms
524.288	4,100 ms	4,149 ms	4,156 ms	4,148 ms	4,152 ms	4,161 ms
1.048.576	7,467 ms	7,455 ms	7,459 ms	7,481 ms	7,456 ms	7,442 ms
2.097.152	14,07 ms	14,09 ms	14,10 ms	14,09 ms	14,10 ms	14,11 ms
4.194.304	27,28 ms	27,22 ms	27,17 ms	27,25 ms	27,27 ms	27,21 ms



Slika 7.17. Ubrzanje izračuna matrice frekvencije jezgrom *ComputeFreq* SS

7.2.5 Usporedba performansi

7.2.5.1 Jezgra ComputeFreq SS

Za usporedbu rezultata izračunata su ubrzanja kao omjeri vremena izvršavanja izračuna matrice frekvencija na koprocesoru s vremenima izvršavanja jednodretvene programske implementacije. Izračunati omjeri dani su u tablici 7.18 i prikazani na slici 7.17.

Izvršavanje izračuna matrice frekvencija općenito je sporije jezgrom SS, nego programskom implementacijom. Rezultat je očekivan, s obzirom da koprocesor radi na 333 MHz, a podatke čita iz DDR3-800 SDRAM-a koji radi na frekvenciji 303 MHz. Za usporedbu, programska implementacija izvršava se na CPU-u koji radi na 3,2 GHz i čita podatke iz DDR3-1600

Tablica 7.18. Ubrzanja jezgre *ComputeFreq* SS

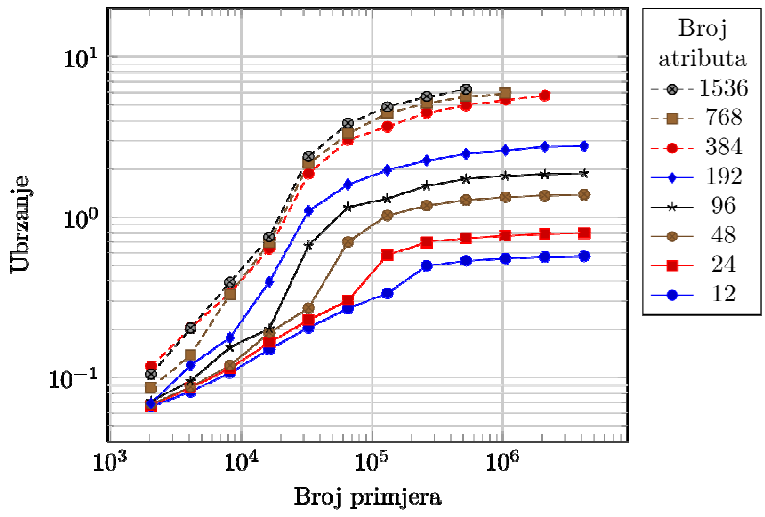
Broj primjera	Broj atributa / Ubrzanje						
	1	2	4	8	16	32	64
2.048	0,051	0,055	0,058	0,063	0,073	0,078	0,085
4.096	0,061	0,065	0,070	0,078	0,089	0,100	0,107
8.192	0,080	0,088	0,094	0,102	0,120	0,133	0,155
16.384	0,115	0,124	0,123	0,134	0,160	0,184	0,235
32.768	0,153	0,161	0,168	0,176	0,220	0,249	0,403
65.536	0,202	0,210	0,209	0,223	0,269	0,416	1,120
131.072	0,246	0,245	0,247	0,271	0,413	0,672	1,398
262.144	0,305	0,306	0,317	0,372	0,496	0,735	1,601
524.288	0,355	0,356	0,376	0,407	0,526	0,791	1,718
1.048.576	0,369	0,368	0,371	0,422	0,547	0,813	1,788
2.097.152	0,375	0,380	0,378	0,426	0,558	0,828	1,839
4.194.304	0,381	0,384	0,384	0,432	0,563	0,837	1,856

SDRAM-u koji radi na 800 MHz. Usprkos tome, zahvaljujući efikasnoj podatkovnoj strukturi za pohranu skupa u memoriji koprocesora, izvršavanje na koprocesoru je brže za skupove sa 64 atributa i više od 128×2^{10} primjera. U tim uvjetima postižu se najbolje performanse jezgre i dostiže ubrzanje od 1,8 puta u odnosu na programsku implementaciju.

7.2.5.2 Jezgre *ComputeFreq* MS i MS2

Za usporedbu rezultata izračunata su ubrzanja kao omjeri vremena izvršavanja višedretvene programske implementacije i vremena izvršavanja jezgara *ComputeFreq* MS i *ComputeFreq* MS2. Programska implementacija izvršavana je sa šest dretvi, što odgovara broju jezgara u CPU-u ispitnog računalnog sustava.

Na slici 7.18, prikazano je ubrzanje jezgre MS u odnosu na višedretvenu programsku implementaciju, a u tablici 7.19 dane su vrijednosti ubrzanja. Izvršavanje jezgre velikim je dijelom sporije nego kod programske implementacije, osobito za skupove s malim brojem primjera. Za mali broj atributa – manje od 48 – jezgra je sporija od programske implementacije bez obzira na broj primjera. Tek s 48 i više atributa ubrzanje se diže iznad jediničnog, i to za skupove s više od 128×2^{10} primjera. Najveće ubrzanje dobiveno je za skupove s 384 i više atributa. Budući da su vrijednosti ubrzanja za te skupove bliske, može se zaključiti da je u tom području i gornja granica ubrzanja ove jezgre. Najveće ubrzanje je 6,26 puta, a postignuto je na skupu s 1.536 atributa i 512×2^{10} primjera.

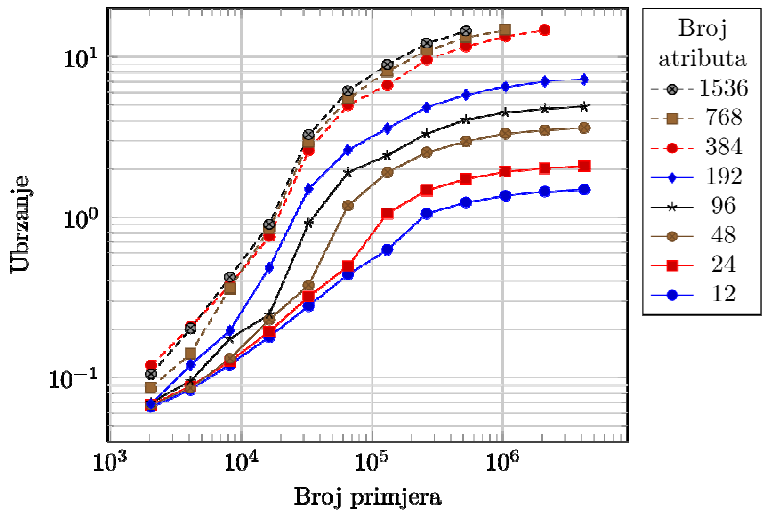


Slika 7.18. Ubrzanje izračuna matrice frekvencije jezgrom *ComputeFreq* MS

Tablica 7.19. Ubrzanja jezgre *ComputeFreq* MS

Broj primjera	Broj atributa / Ubrzanje							
	12	**24**	**48**	**96**	**192**	**384**	**768**	**1.536**
2.048	0,066	0,067	0,068	0,071	0,069	0,118	0,087	0,105
4.096	0,082	0,086	0,087	0,095	0,120	0,204	0,138	0,203
8.192	0,107	0,114	0,119	0,155	0,177	0,335	0,330	0,393
16.384	0,150	0,165	0,190	0,202	0,396	0,636	0,691	0,751
32.768	0,205	0,228	0,272	0,665	1,096	1,868	2,159	2,380
65.536	0,270	0,303	0,702	1,155	1,600	3,029	3,313	3,847
131.072	0,336	0,580	1,027	1,307	1,966	3,673	4,463	4,868
262.144	0,498	0,699	1,181	1,573	2,259	4,486	5,133	5,648
524.288	0,535	0,741	1,280	1,734	2,485	4,994	5,654	6,258
1.048.576	0,552	0,766	1,334	1,815	2,612	5,371	5,901	–
2.097.152	0,566	0,786	1,367	1,859	2,754	5,727	–	–
4.194.304	0,572	0,797	1,386	1,882	2,775	–	–	–

Za jezgru *ComputeFreq* MS2 ubrzanje je prikazano na slici 7.19, a iznosi ubrzanja dani su u tablici 7.20. I ova jezgra velikim je dijelom sporija od programske implementacije. Bolje iskorištenje podatkovnih tokova koje pruža arhitektura ove jezgre, omogućuje dobitak u brzini izvođenja i za skupove s malim brojem atributa. Skup s 12 atributa dostiže jedinično ubrzanje – brzinu izvođenja jednaku programskoj implementaciji – za skupove 256×2^{10} primjera i više. Kao i kod jezgara SS i MS, povećanjem broja atributa povećava se i postignuto ubrzanje. Kao i kod jezgre MS, male razlike u ubrzanjima za skupove s 384 i više atributa upućuju da je u tom području gornja granica mogućeg ubrzanja. Najveće ubrzanje izmjereno je na skupu sa 384 atributa i 2×2^{20} primjera, a iznosi 14,7 puta.



Slika 7.19. Ubrzanje izračuna matrice frekvencije jezgrom *ComputeFreq* MS2

Tablica 7.20. Ubrzanja jezgre *ComputeFreq* MS2

Broj primjera	Broj atributa / Ubrzanje							
	12	24	48	96	192	384	768	1.536
2.048	0,065	0,068	0,067	0,070	0,068	0,119	0,087	0,105
4.096	0,084	0,088	0,086	0,095	0,120	0,208	0,141	0,203
8.192	0,121	0,126	0,131	0,175	0,196	0,376	0,360	0,422
16.384	0,179	0,194	0,230	0,247	0,485	0,757	0,845	0,899
32.768	0,280	0,320	0,377	0,916	1,504	2,604	2,929	3,261
65.536	0,439	0,494	1,182	1,900	2,624	5,009	5,487	6,148
131.072	0,628	1,058	1,904	2,433	3,586	6,657	8,134	8,926
262.144	1,052	1,471	2,540	3,318	4,841	9,556	10,920	12,120
524.288	1,238	1,735	2,968	4,066	5,801	11,620	13,200	14,430
1.048.576	1,363	1,921	3,320	4,492	6,486	13,350	14,660	–
2.097.152	1,445	2,013	3,493	4,748	7,026	14,670	–	–
4.194.304	1,488	2,077	3,608	4,903	7,217	–	–	–

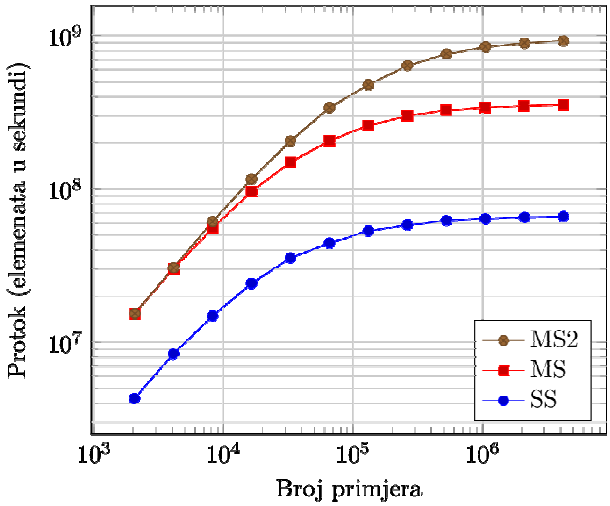
7.2.6 Diskusija rezultata vrednovanja jezgre *ComputeFreq*

U tablici 7.21 sažete su osnovne značajke jezgara za izračun matrice frekvencije, dok su na slici 7.20. izravno uspoređene njihove performanse. Prikazani protoci dobiveni su izračunavanjem aritmetičke sredine protoka izmjerenih za različiti broj atributa. Izračunate vrijednosti dane su u prilogu A, tablica A.4.

Najbolje performanse pokazuje jezgra MS2. Iako je frekvencija takta jezgre MS2 niža od takta jezgre MS, korištenjem vremenski pomaknutih parcijalnih matrica frekvencije ta razlika uspješno je kompenzirana i ostvaren je dobitak u brzini izvođenja. Sve tri jezgre postižu najbolje performanse bliske teoretskom maksimumu, no tek za skupove s većim brojem primjera – preko 200.000. Za skupove s manjim brojem atributa, performanse se smanjuju do te mjere da postaju ovisne samo o broju podatkovnih tokova atributa. Vidljivo je da se za manji broj primjera – ispod 10.000 – nema značajne razlike u performansama između jezgara MS i MS2. To upućuje da pod tim uvjetima performanse nisu određene samim jezgrama, već da veći značaj imaju drugi čimbenici u sustavu.

Tablica 7.21. Osnovne značajke varijanti jezgre *ComputeFreq*

Jezgra	SS	MS	MS2
Broj podatkovnih tokova atributa	1	6	6
Radna frekvencija	333 MHz	300 MHz	200 MHz
Teoretski najveći protok (elemenata u sekundi)	$66{,}6\times10^6$	360×10^6	960×10^6
Najveći izmjereni protok (elemenata u sekundi)	$62{,}4\times10^6$	340×10^6	906×10^6



Slika 7.20. Prosječni protoci jezgara SS, MS i MS2

Iz vremena izvršavanja pojedinih jezgara (slike 7.7, 7.9 i 7.13), vidljivo je da je za skupove s manjim brojem atributa – manje od 50.000 – vrijeme izvršavanja približno konstantno. Vrijeme izvršavanja u tom području ne ovisi o veličini skupa, što također upućuje na to da drugi čimbenici u sustavu pod tim uvjetima imaju znatno veći utjecaj na performanse. Vremena izvršavanja jezgara ne razlikuju se bitno za skup s jednim atributom i s do 4096 primjera (v. tablice 7.10, 7.14 i 7.17), iz čega se može zaključiti da je to minimalno vrijeme koje je potrebno za izvršavanje bilo koje radnje na koprocesoru. Prosječno vrijeme izvršavanja u tim uvjetima iznosi 801,1 µs.

Navedeno minimalno vrijeme izvršavanja posljedica je kašnjenja u komunikaciji između CPU-a i FPGA-a, koje uključuje vrijeme izvršavanja programskog sučelja, kašnjenja pri pokretanju jezgre unutar FPGA-a, te vremena potrebnog za prijenos matrice frekvencija iz koprocesora u FPGA. Tek kada vrijeme samog izvršavanja jezgre postane usporedivo s navedenim vremenom kašnjenja, počinje rasti ukupno vrijeme izvršavanja, što je vidljivo kao približavanje iznosa protoka određenoj maksimalnoj vrijednosti. Posljedica toga je da se dobitak u brzini izvođenja hibridnog algoritma može očekivati tek za skupove kod kojih je vrijeme izvršavanja na CPU-u veće od minimalnog vremena izvršavanja na koprocesoru. Za skupove s malim brojem atributa i primjera na CPU-u je potrebno nekoliko desetaka do nekoliko stotina mikrosekundi (tablice 7.5 i 7.9), dok je za iste skupove na koprocesoru vrijeme izvršavanja oko 800 µs.

Iz vremena izvršavanja pojedinih jezgara (slike 7.7, 7.9 i 7.13) jasno se vidi linearna ovisnost vremena izvršavanja o broju atributa. Dodatna potvrda linearnog skaliranja dobivena je iz

protoka jezgara (slike 7.8, 7.10 i 7.14) koje ne pokazuju ovisnost o broju atributa. Ta značajka zajednička je za sve tri jezgre i ona im daje određenu prednost nad programskom implementacijom. Performanse programske implementacije izvršavane na CPU-u osjetljive su na broj atributa u skupu i postaju znatno lošije pri velikom broju atributa, što je jasno vidljivo iz protoka funkcija (v. slike 7.2 i 7.4). U skladu s time, ubrzanje izvršavanja na jezgrama u odnosu na programske implementacije veće je za skupove s većim brojem atributa (v. slike 7.17, 7.18, 7.19.).

Unatoč implementaciji matematički jednostavne operacije koja je prije svega ograničena brzinom pristupa memoriji, jezgre imaju dobre performanse. Jezgra MS2 postiže najveće ubrzanje od 14,7 puta u odnosu na programsku implementaciju sa šest dretvi. Za ocjenu prosječnog ubrzanja izračunata je otežana aritmetička sredina, otežana brojem elemenata u skupu, prema formuli:

$$S = \frac{\sum_a \sum_n a\ n\ s_{a,n}}{\sum_a \sum_n a\ n} \tag{7.2}$$

gdje su: S otežana srednja vrijednost ubrzanja, a $s_{a,n}$ izmjereno ubrzanje za skup s a atributa i n primjera. Otežana srednja vrijednost ubrzanja iznosi 3,00 puta.

Izračun matrice frekvencije jezgrom MS2 mnogo je efikasniji od izračuna na CPU-u. Efikasnost je ocijenjena brojem ciklusa takta CPU-a, odnosno jezgre, koji je utrošen na obradu jednog elementa. Formula prema kojoj se izračunava efikasnost je:

$$E_{a,n} = \frac{m\ f}{F_{a,n}} \tag{7.3}$$

gdje je: $E_{a,n}$ efikasnost, $F_{a,n}$ protok funkcije, odnosno jezgre za skup s a atributa i n primjera, m broj dretvi, odnosno podatkovnih tokova tributa i f frekvencija takta CPU-a, odnosno jezgre. Jedinica kojom se izražava efikasnost jest „ciklusa takta po elementu“. Za usporedbu su uzete najbolje izmjerene performanse CPU-a i jezgre MS2.

Programska implementacija izvršavala se na šest dretvi na CPU-u s radnom frekvencijom 3,2 GHz. Najveći izmjereni protok funkcije iznosi $874{,}8 \times 10^6$ elemenata u sekundi, što daje efikasnost od 22,0 ciklusa takta po elementu. Jezgra MS2 ima šest podatkovnih tokova atributa i radi na 200 MHz. Najveći izmjereni protok iznosi $924{,}0 \times 10^6$ elemenata u sekundi, što daje efikasnost od 1,30 ciklusa takta po elementu. Od navedenih 1,30 ciklusa takta, 1,25

ciklusa je posljedica arhitekture jezgre kojom je određeno da se obrađuje 4 elementa svakih 5 ciklusa takta.

Vrednovanjem jezgara u izolaciji pokazano je da je FPGA dobra računalna platforma za izračun matrice frekvencija s obzirom na postignute performanse i efikasnost. No, značajni utjecaj imaju i drugi čimbenici: kašnjenje i propusnost komunikacije između CPU-a i FPGA-a. Njihov utjecaj jako je izražen za skupove s manjim brojem primjera, gdje su performanse više određene kašnjenjima u komunikaciji nego vremenom izvršavanja jezgre. Iz navedenog, vidljivo je da je mogući dobitak u brzini izvršavanja algoritma ovisan o veličini skupa za učenje i to početnog skupa, kao i svakog podskupa koji se dobije u tijekom izvršavanja algoritma. Općenito, bolje performanse hibridnog algoritma očekuju se za veće skupove, a osobito za skupove s velikim brojem atributa.

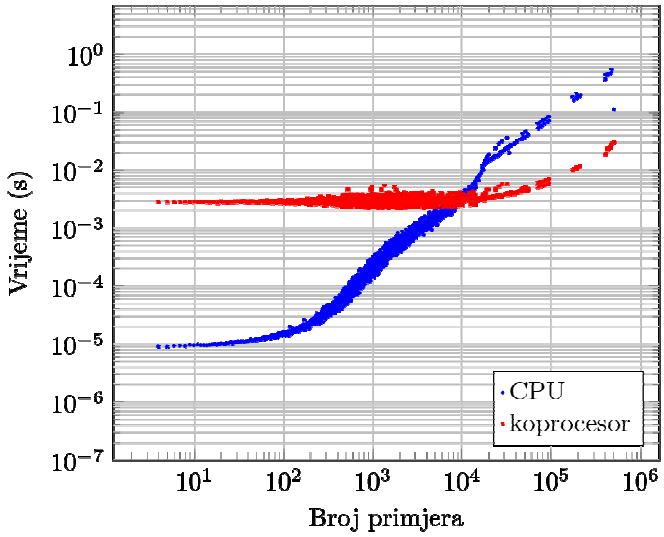
7.3 Vrednovanje hibridnog algoritma DF-DTC

7.3.1 Obrada podskupa na CPU-u i na koprocesoru

Prilikom izvršavanja programa, kao što je opisano u poglavlju 6.4.4, prije obrade svakog podskupa provjerava se koliko primjera on sadrži. Ako je broj primjera ispod određenog praga, obrada podskupa izvršava se u potpunosti na CPU-u. Radi određivanja optimalne vrijednosti praga provedena je analiza veličine podskupova i vremena obrade u realnim uvjetima izvršavanja programa.

U svrhu ispitivanja, program je prilagođen tako da u datoteku bilježi veličine ulaznog skupa i vremena obrade u svakom čvoru. Prilikom ispitivanja mjeri se ukupno vrijeme obrade čvora, te posebno vrijeme obrade nominalnih atributa i vrijeme grupiranja. U ukupno vrijeme obrade čvora uključeno je i vrijeme obrade nominalnih atributa. Ispitivanje je provedeno na uzorku realnog skupa Covertype sa 500.000 primjera. Provedena su dva ispitivanja: ispitivanje na izvornoj programskoj implementaciji EC4.5 i ispitivanje na implementaciji hibridnog algoritma DF-DTC. Pri ispitivanju na DF-DTC, prag za usmjeravanje obrade podskupa onemogućen je postavljanjem na nulu. Na taj su način svi podskupovi obrađivani na koprocesoru.

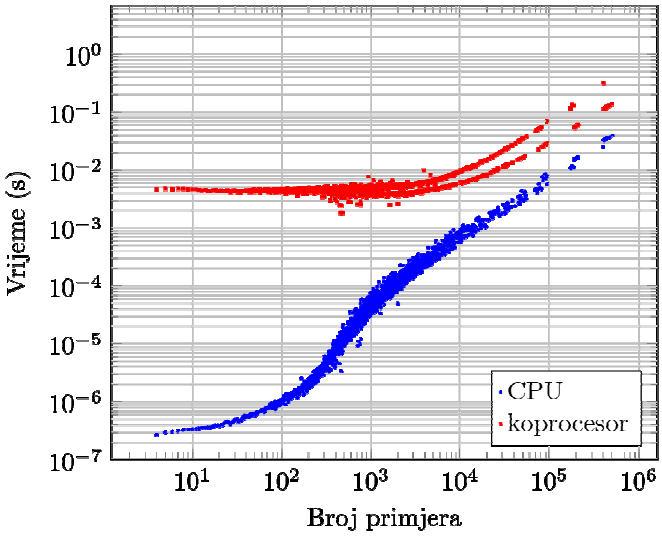
Na slici 7.21 prikazano je vrijeme izračuna matrice frekvencija u odnosu na veličinu obrađivanog podskupa. Kao što je predviđeno u poglavlju 7.2, za manje podskupove koprocesor ima lošije performanse od CPU-a. Na slici je vidljiv isti obrazac vremena izvršavanja kao kod jezgre u izolaciji – za manje podskupove vrijeme izvršavanja je



Slika 7.21. Vrijeme izračuna matrice frekvencija za nominalne atribute

konstantno (v. sliku 7.13). Vrijeme izvršavanja počinje rasti za skupove s preko 10.000 primjera, što upućuje na to da tek tada vrijeme izvršavanja jezgre postaje usporedivo s troškom komunikacije između CPU-a i koprocesora. Rastom veličine skupa iznad te točke povećava se i razlika u vremenu izvođenja, tj. povećava se ubrzanje izračuna matrice frekvencija, što je u skladu s rezultatima dobivenim ispitivanjem jezgre u izolaciji. Vremena izračuna matrice frekvencija na CPU-u i na koprocesoru preklapaju se za podskupove s brojem primjera između 5000 i 15.000. Za podskupove s preko 15.000 primjera koprocesor je brži od CPU-a.

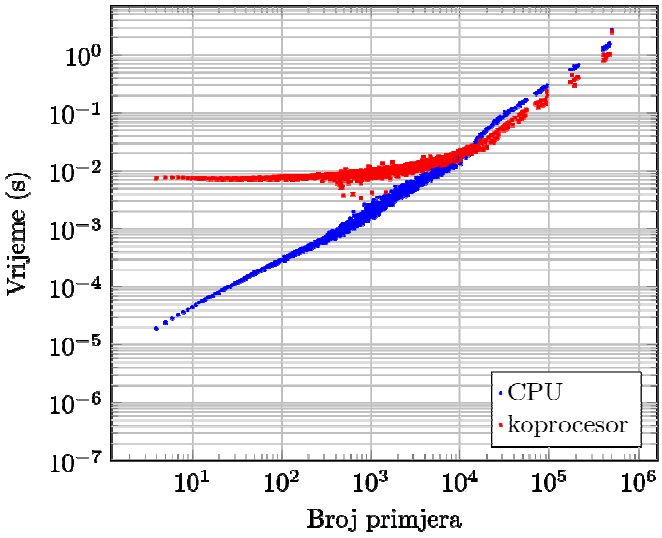
Drugi proces koji bitno utječe na performanse algoritma je grupiranje. Na slici 7.22 prikazana su izmjerena vremena grupiranja na CPU-u i na koprocesoru u odnosu na veličine obrađivanih podskupova. Vrijeme grupiranja na CPU-u manje je od vremena grupiranja na koprocesoru, što je očekivano s obzirom na način izvedbe tog procesa. U programskoj implementaciji izvršavanoj na CPU-u, grupiranje se izvodi promjenom položaja pojedinih pokazivača u polju pokazivača na primjere. Primjeri pri tome ne mijenjaju svoj položaj u memoriji. Grupiranje na koprocesoru izvodi se kopiranjem sadržaja ulaznog podskupa na odgovarajuća mjesta u memoriji. Iz izmjerenih vremena grupiranja i izračuna matrice frekvencija vidljivo je i da je za grupiranje na koprocesoru potrebno više vremena nego za izračun matrice frekvencija, što je posljedica obrasca pristupa memoriji. Jezgra *ComputeFreq* iz memorije isključivo čita, a pojedini podatkovni tokovi čitaju se sa uzastopnih memorijskih lokacija. Jezgra *GroupItems* iz memorije čita i piše. Dok je čitanje iz memorije po istom obrascu kao kod jezgre *ComputeFreq*, pri pisanju se u općem slučaju nasumično pristupa ograničenom broju lokacija,



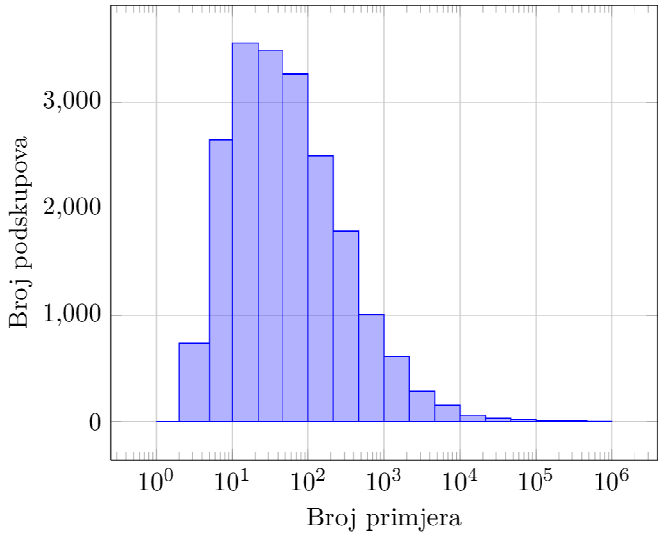
Slika 7.22. Vrijeme grupiranja

koji je određen brojem jedinstvenih vrijednosti atributa. S obzirom na radne značajke memorije, obrazac pristupa pri pisanju je nepovoljan.

Ukupno vrijeme obrade podskupa prikazano je na slici 7.23. U ukupno vrijeme uključena su sva vremena obrade za nominalne i numeričke atribute, te vrijeme grupiranja skupa. Obrada uz pomoć koprocesora brža je od obrade korištenjem samo CPU-a kada podskup ima najmanje 15.000 primjera, što je u skladu s rezultatima dobivenim za izračun matrice frekvencija. Za to područje izračunata je ocjena ubrzanja obrade podskupa. Ubrzanje je ocijenjeno aritmetičkom sredinom ubrzanja postignutog za pojedine veličine podskupova,



Slika 7.23. Ukupno vrijeme obrade podskupa



Slika 7.24. Raspodjela veličina podskupova

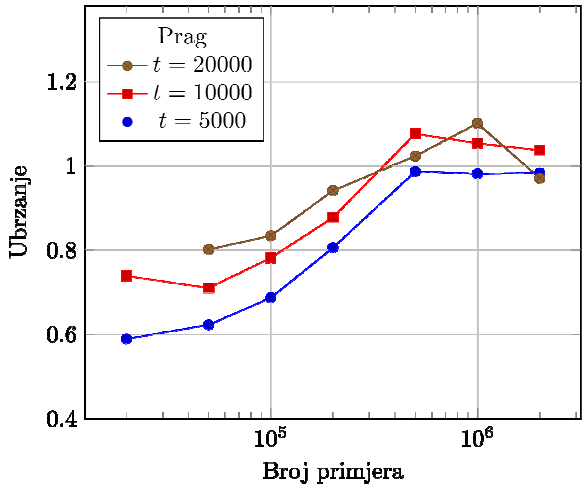
uzimajući u obzir samo podskupove s najmanje 15.000 primjera. Prosječno ubrzanje iznosi 1,60 puta. Budući da je to postignuto ubrzanje moguće samo za dio podskupova koji se pojavljuju tijekom izgradnje stabla, dobivena vrijednost može se uzeti kao gornja granica mogućeg ubrzanja algoritma DF-DTC.

U tablici 7.22 dana je, a na slici 7.24 prikazana je raspodjela veličina podskupova koji se pojavljuju tijekom gradnje stabla. Iz raspodjele je jasno vidljivo da većina podskupova ima mali broj primjera. Medijan veličine podskupa iznosi 42 primjera. Budući da se ubrzanje obrade podskupa postiže tek za podskupove s barem 15.000 primjera, raspodjela veličina podskupova nepovoljna je za performanse algoritma DF-DTC.

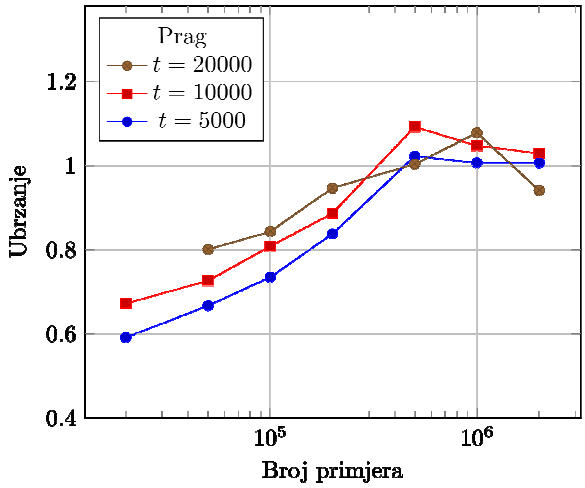
Iako prosječno ubrzanje jezgre *ComputeFreq* iznosi 3,00 puta, zbog ostalih procesa koji se izvode za vrijeme obrade podskupa očekuju se slabije ukupne performanse. U procesu obrade podskupa, najveći utjecaj ima izvršavanje grupiranja koje sa izvršava sporije od izračuna matrice frekvencije i umanjuje dobitak u brzini izvršavanja obrade. Budući da se obrada manjih podskupova izvršava programski, raspodjela

Tablica 7.22. Raspodjela veličina podskupa

Interval		Broj podskupova
–	1	0
2	4	739
5	9	2643
10	21	3557
22	45	3485
46	99	3262
100	214	2497
215	463	1793
464	999	1005
1.000	2.153	613
2.154	4.641	290
4.642	9.999	155
10.000	21.543	65
21.544	46.415	38
46.416	99.999	25
100.000	215.442	8
215.443	464.158	9
464.159	–	2



Slika 7.25. Ubrzanje gradnje stabla u odnosu na broj primjera za skup Covertype



Slika 7.26. Ukupno ubrzanje u odnosu na broj primjera za skup Covertype

veličine podskupova dodatno umanjuje dobitak. Iz ubrzanja za pojedine veličine podskupova gornja granica ukupnog ubrzanja procijenjena je na 1,60 puta.

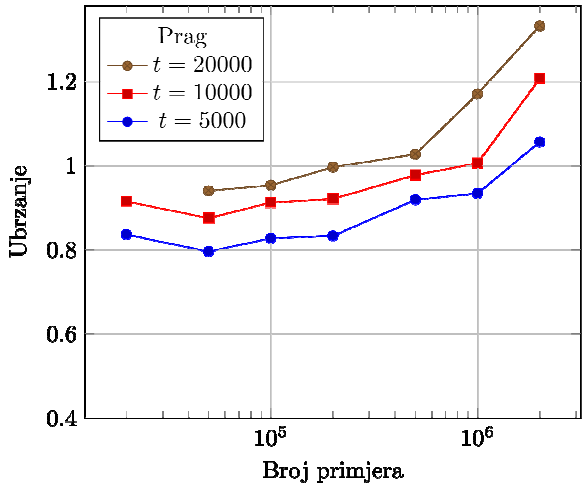
7.3.2 Analiza utjecaja broja primjera

Analiza utjecaja broja primjera na oba skupa podataka provedena je sa tri vrijednosti praga za usmjeravanje podskupa na obradu na CPU-u ili na koprocesoru. Vrijednosti praga su 5000, 10.000 i 20.000 primjera. Za vrijednost praga 20.000, uzorci skupova s 20.000 primjera izostavljeni su iz ispitivanja, budući da bi se u tom slučaju cijeli proces gradnje stabla izvršavao programski na CPU-u.

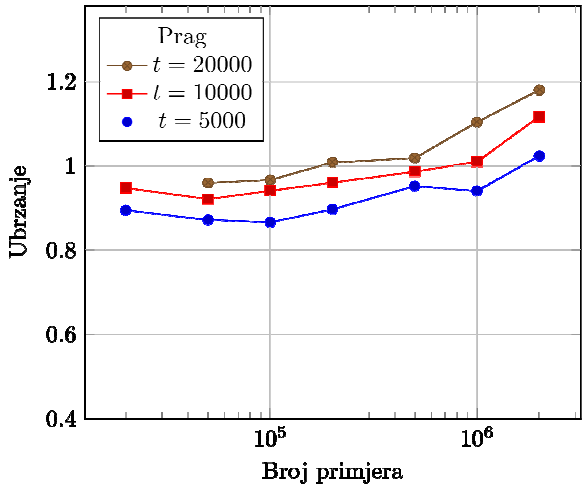
Na slikama 7.25 i 7.26 prikazana su ubrzanja gradnje stabla, te ukupnog algoritma DF-DTC postignuta na skupu Covertype. Za skupove s manje od 500.000 primjera dobiveno je ubrzanje manje od jediničnog, što znači da je algoritam DF-DTC sporiji od programske implementacije EC4.5. Tek s 500.000 i više primjera ubrzanje je veće od jediničnog i to za

Tablica 7.23. Ubrzanja algoritma DF-DTC na skupu Covertype

	Gradnja stabla			Ukupno		
Broj primjera	t = 5.000	t = 10.000	t = 20.000	t = 5.000	t = 10.000	t = 20.000
20.000	0,5893	0,7387	–	0,5913	0,6724	–
50.000	0,6227	0,7103	0,8014	0,6670	0,7265	0,8007
100.000	0,6869	0,7816	0,8340	0,7350	0,8084	0,8431
200.000	0,8058	0,8774	0,9411	0,8378	0,8863	0,9468
500.000	0,9865	1,076	1,023	1,022	1,092	1,003
1.000.000	0,9812	1,054	1,101	1,007	1,047	1,078
2.000.000	0,9830	1,037	0,9706	1,006	1,029	0,9409



Slika 7.27. Ubrzanje gradnje stabla u odnosu na broj primjera za sintetički skup



Slika 7.28. Ukupno ubrzanje u odnosu na broj primjera za sintetički skup

vrijednosti praga 10.000 i 20.000. Za prag 5000, ubrzanje je blizu jediničnom, ali ga ne dostiže.

Izmjereni iznosi ubrzanja gradnje stabla i ukupnog ubrzanja, dani u tablici 7.23, malo se razlikuju, što pokazuje da dodatni trošak pripreme i prijenosa skupa za učenje u memoriju koprocesora nema velik utjecaj na performanse algoritma. Vrijednosti ubrzanja općenito imaju uzlazni trend, tj. povećanjem broja primjera poboljšavaju se performanse. Međutim, za skupove s 500.000 i više primjera uzlazni trend se gubi.

Ubrzanja dobivena na sintetičkom skupu dana su u tablici 7.24 i prikazana na slikama 7.27 i 7.28. Za sve tri vrijednosti praga ubrzanje ima uzlazni trend, koji zadržava i nakon što prijeđe jediničnu vrijednost. Najbolja ubrzanja dobivena su s pragom iznosa 20.000. U tom slučaju jedinično ubrzanje postiže se u području između 200.000 i 500.000 primjera. Za razliku od skupa Covertype, kod sintetičkog skupa dobivene su veće razlike između ubrzanja gradnje

Tablica 7.24. Ubrzanja algoritma DF-DTC na sintetičkom skupu

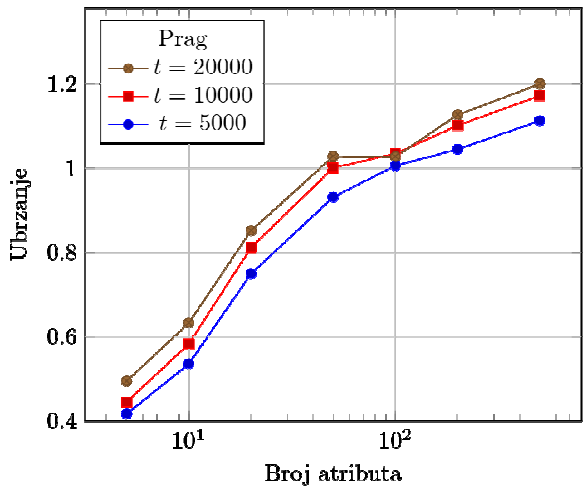
	Gradnja stabla			**Ukupno**		
Broj primjera	t = 5.000	t = 10.000	t = 20.000	t = 5.000	t = 10.000	t = 20.000
20.000	0,8363	0,9149	–	0,8944	0,9475	–
50.000	0,7959	0,8753	0,9402	0,8716	0,9210	0,9595
100.000	0,8274	0,9123	0,9534	0,8659	0,9407	0,9663
200.000	0,8332	0,9210	0,9964	0,8967	0,9601	1,008
500.000	0,9189	0,9777	1,027	0,9517	0,9862	1,018
1.000.000	0,9338	1,006	1,170	0,9402	1,010	1,104
2.000.000	1,056	1,207	1,332	1,024	1,117	1,180

stabla i ukupnog ubrzanja. Udio vremena gradnje stabla u ukupnom vremenu izvršavanja programa manji je, pa priprema i prijenos skupa u memoriju koprocesora ima veći utjecaj na ukupne performanse sustava. Najveće postignuto ubrzanje gradnje stabla iznosi 1,33 puta, dok je najveće ukupno ubrzanje 1,18 puta.

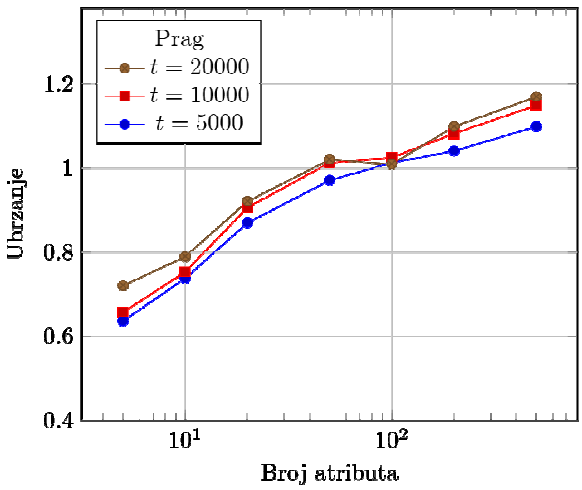
Rezultati za skup Covertype i za sintetički skup bitno se razlikuju. Razlike su posljedica značajki skupova. Skup Covertype ima 54 atributa, od kojih su deset numerički. Iako se obrada numeričkih atributa izvršava na CPU-u, sinkronizacija podskupova na koprocesoru i CPU-u uzrokuje vremenski trošak koji umanjuje ubrzanje gradnje stabla. Ukupni broj atributa također ima utjecaj na performanse. Prema rezultatima ispitivanja jezgre *ComputeFreq* u izolaciji bolje performanse očekuju se na skupovima s većim brojem atributa i s većim brojem primjera. Sintetički skup ima 200 atributa i bolje performanse postignute na njemu u skladu su s time. Na oba skupa vidljivo je da ubrzanje raste s brojem primjera i da ono prelazi jedinicu za skupove između 200.000 i 500.000 primjera. Rast ubrzanja s povećanjem broja primjera u također je u skladu s radnim značajkama jezgre.

7.3.3 Analiza utjecaja broja atributa

Za skup Covertype, ubrzanje raste s brojem atributa, što je vidljivo sa slika 7.29 i 7.30, te tablice 7.25. Za mali broj atributa, performanse DF-DTC značajno su lošije od programske implementacije EC4.5. Iz vremena izračuna matrice frekvencija za jezgru i za programsku implementaciju (tablica 7.9, odnosno 7.17) za skup s pet atributa i 1.000.000 primjera ubrzanje je 0,94 puta. Budući da nema dobitka u brzini izračuna matrice frekvencije, zbog



Slika 7.29. Ubrzanje gradnje stabla u odnosu na broj atributa za skup Covertype



Slika 7.30. Ukupno ubrzanje u odnosu na broj atributa za skup Covertype

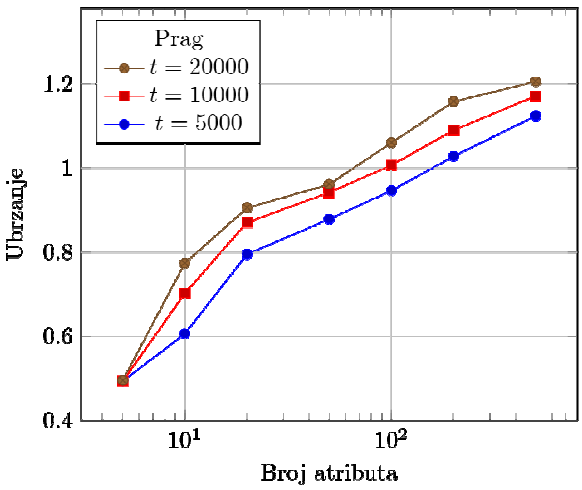
Tablica 7.25. Ubrzanje algoritma DF-DTC na skupu Covertype

Broj atributa	Gradnja stabla			Ukupno		
	t = 5.000	t = 10.000	t = 20.000	t = 5.000	t = 10.000	t = 20.000
5	0,4167	0,4453	0,4946	0,6362	0,6570	0,7204
10	0,5354	0,5829	0,6326	0,7378	0,7530	0,7892
20	0,7493	0,8115	0,8517	0,8696	0,9062	0,9207
50	0,9315	1,001	1,028	0,9708	1,012	1,020
100	1,006	1,035	1,027	1,013	1,025	1,009
200	1,045	1,102	1,127	1,041	1,082	1,099
500	1,113	1,172	1,200	1,099	1,149	1,169

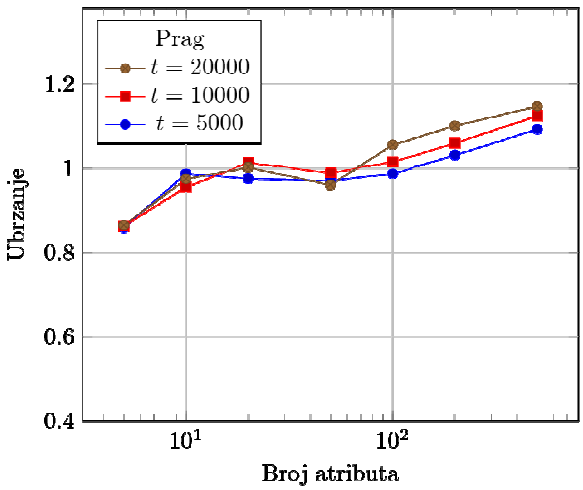
vremenskih troškova ostalih operacija na koprocesoru performanse algoritma DF-DTC bitno su lošije od izvorne programske implementacije EC4.5. Ubrzanje gradnje stabla, a i cijelog programa, postiže jediničnu vrijednost za skupove s 50 atributa i nastavlja rasti s povećanjem broja atributa.

Ubrzanja dobivena za sintetički skup, prikazana na slikama 7.32 i 7.31, slična su onima dobivenim za skup Covertype. Budući da za sintetički skup izgradnja stabla ima manji udio u ukupnom vremenu izvršavanja, rast ukupnog ubrzanja manje je izražen nego kod skupa Covertype. Za mali broj atributa izgradnja stabla u DF-DTC sporija je nego u programskoj implementaciji EC4.5, a usporedive performanse postignute su tek za skupove sa 100 atributa.

Broj atributa ima sličan utjecaj za oba ispitna skupa: ubrzanje raste s povećanjem broja atributa, što je u skladu s radnim značajkama jezgre *ComputeFreq*. Za skup Covertype rast ubrzanja je izraženiji, ali su performanse za skupove s manje od 50 atributa lošije nego za



Slika 7.32. Ubrzanje gradnje stabla u odnosu na broj atributa za sintetički skup



Slika 7.31. Ukupno ubrzanje u odnosu na broj atributa za sintetički skup

Tablica 7.26. Ubrzanja algoritma DF-DTC na sintetičkom skupu

	Gradnja stabla			Ukupno		
Broj atributa	t = 5.000	t = 10.000	t = 20.000	t = 5.000	t = 10.000	t = 20.000
5	0,4933	0,4928	0,4944	0,8589	0,8631	0,8642
10	0,6057	0,7017	0,7733	0,9868	0,9548	0,9743
20	0,7947	0,8706	0,9055	0,9755	1,013	1,002
50	0,8785	0,9421	0,9613	0,9709	0,9891	0,9603
100	0,9463	1,007	1,060	0,9869	1,015	1,055
200	1,028	1,090	1,158	1,031	1,059	1,101
500	1,124	1,172	1,206	1,093	1,124	1,147

sintetički skup. Za skupove sa 100 i više atributa, razlike u performansama su neznatne, što je vidljivo iz tablica 7.25 i 7.26. Utjecaj iznosa praga manje je izražen nego pri ispitivanju utjecaja veličine skupa, ali i dalje vidljiv. S većim pragom dobivene su bolje performanse.

Najveća ubrzanja postignuta su za skup s 500 atributa, uz prag 20.000 primjera. Ubrzanje izgradnje stabla iznosi 1,20 za skup Covertype i 1,21 za sintetički skup. Ukupno ubrzanje je 1,17 za skup i 1,15 za sintetički skup. Za oba skupa pokazano je da je promjena ubrzanja s promjenom boraj atributa u skladu s radnim značajkama jezgre *ComputeFreq*: s povećanjem broja atributa povećava se ubrzanje gradnje stabla, a time i ukupno ubrzanje.

7.4 Diskusija rezultata vrednovanja algoritma DF-DTC

Od tri varijante jezgre *ComputeFreq*, samo varijanta MS2 postiže performanse kojima se može očekivati postizanje ubrzanja izgradnje stabla. Stoga se ta varijanta jezgre koristi u koprocesoru. Ubrzanje izračuna matrice frekvencije ovisi o broju atributa i broju primjera u skupu. Povećanjem broja atributa ili primjera raste i ubrzanje, s time da veći utjecaj ima broj atributa. Ugrubo, jedinično ili veće ubrzanje a postiže se za skupove s barem 20.000 primjera i barem 3.000.000 elemenata. Za skupove s do 12 atributa ubrzanje ne prelazi 1,5 puta i u tom području ne očekuje se dobitak pri učenju stabla odluke. Najveće postignuto ubrzanje izračuna matrice frekvencija iznosi 14,7 puta, dok prosječno ubrzanje iznosi 3,00 puta.

Navedena ograničenja jezgre uzimaju se u obzir pri izvedbi algoritma DF-DTC. U algoritam je ugrađen parametar kojim se određuje hoće li se određeni podskup obrađivati na koprocesoru ili programski na CPU-u. Parametar je definiran kao minimalni broj primjera koji podskup mora sadržavati, a podešavanjem parametra nastoji se osigurati da se FPGA koprocesor koristi samo kada daje dobitak u brzini izvršavanja. Pri vrednovanju su korištene tri vrijednosti praga: 5.000, 10.000 i 20.000. Za veći prag dobivene su i bolje performanse, pa su skladu s time za prag t = 20.000 dobivene najbolje performanse.

Konačne performanse algoritma DF-DTC izvršavanog na heterogenom računalnom sustavu uspoređene su s programskom implementacijom EC4.5 u kojoj je paraleliziran izračun matrice frekvencija. Oba programa izvršavana su sa šest dretvi na CPU-u sa šest jezgara. Ispitan je utjecaj broja primjera i broja atributa na performanse DF-DTC

Ispitivanjem performansi DF-DTC, utvrđeno je da broj atributa ima veći utjecaj na ukupne performanse sustava. Za manji broj atributa – manje od 50 – DF-DTC je sporiji od programske implementacije, s najmanjim ukupnim ubrzanjem od 0,59 puta. Za barem 50 atributa i najmanje 500.000 primjera, postižu se ubrzanja veća od jedan puta. Najveće izmjereno ubrzanje iznosi 1,33 puta za gradnju stabla, te 1,18 puta za cijeli program.

Dobivene performanse imaju više uzroka:

- vrijeme potrebno za komunikaciju između CPU-a i koprocesora
- vrijeme potrebno za grupiranje primjera u memoriji koprocesora
- vremenski trošak dodatnih operacija koje nisu prisutne u programskoj implementaciji EC4.5
- raspodjela veličine podskupova koji se pojavljuju tijekom gradnje stabla.

Propusnost veze između CPU-a i FPGA-a iznosi oko 2 GB/s, ali ona nije čimbenik koji određuje vrijeme utrošeno na komunikaciju između njih. Za ovaj algoritam mnogo veći utjecaj imaju kašnjenja koja se pojavljuju pri pokretanju pojedinih radnji na koprocesoru. Kašnjenja su posljedica vremena potrebnog za pokretanje i izvršavanje programskog sučelja, te vremena pokretanja same jezgre u koprocesoru. Minimalno vrijeme potrebno za izvršavanje jedne radnje procijenjeno je na 800 µs. Ako se ista radnja na CPU-u izvršava u manje vremena, ukupni učinak korištenja koprocesora bit će usporenje algoritma.

Postupak grupiranja različit je na CPU-u i na koprocesoru. Na CPU-u grupiranje se izvodi promjenom položaja pokazivača u polju pokazivača na primjere, a primjeri se ne premještaju po memoriji. Grupiranje na koprocesoru svodi se na kopiranje sadržaja iz jednog izvorišnog polja u više odredišnih polja. Broj odredišnih polja određen je brojem jedinstvenih vrijednosti atributa. Budući da je u općem slučaju raspored primjera s obzirom odredišna polja slučajan, pristup za pisanje u SDRAM također je slučajan, što je nepovoljno za performanse.

Radi usklađivanja sadržaja podskupova, uz grupiranje izvršavaju se dodatne radnje koje nisu prisutne u programskoj implementaciji EC4.5. Pri grupiranju prema vrijednosti numeričkog atributa potrebno je pročitati identifikacijske brojeve primjera iz koprocesora, te u koprocesor

upisati polje oznaka grupa prema kojima se potom provodi grupiranje. Svaka od tih radnji zahtijeva barem navedeno minimalno vrijeme i smanjuje ukupne performanse sustava.

Najveći utjecaj na performanse ima raspodjela veličina podskupova koji se stvaraju tijekom izvođenja programa. Kako tijek algoritma napreduje, početni skup dijeli se na sve manje podskupove i u jednom trenutku oni postanu dovoljno mali da vrijeme izvršavanja radnji na koprocesoru bude ograničeno na minimalno trajanje od 800 μs. Radi sprečavanja smanjenja performansi uveden je prag za usmjeravanje obrade na CPU. No većina podskupova, kao što se vidi na slici 7.24, ima mali broj primjera, pa se većina vremena izvršavanja programa provede na CPU-u. Zbog toga se dobitak u brzini izvođenja dobije tek za skupove s većim brojem primjera (preko 500.000), gdje je vrijeme provedeno u izvršavanju na koprocesoru dovoljno veliko da dođe do izražaja.

Najbolje performanse dobivene su uz prag od 20.000 primjera. U pretpostavku da se skup uvijek dijeli na dva podskupa jednake veličine, skup s 1.000.000 primjera u najboljem slučaju već u šestoj razini stabla bude podijeljen na podskupove koji su manji od praga. Iz raspodjele veličine podskupova dobivenih za uzorak skupa Covertype s 54 atributa i 500.000 primjera, od 20.186 podskupova koji se pojavljuju u tijeku procesa učenja, samo ih 83 ima 20.000 ili više primjera. Zbog takvog odnosa broja skupova koji se obračunu na koprocesoru i na CPU-u – približno 1:240 –većina procesa izgradnje stabla odvija se bez sudjelovanja koprocesora. U konačnici se dobiju mala ubrzanja i to tek za skupove s velikim brojem primjera.

Iz rezultata ispitivanja jezgre *ComputeFreq* u izolaciji, vidljivo je da je FPGA prikladna platforma za implementaciju izračuna matrice frekvencije. Izvedbom na FPGA-u postižu se performanse bolje od programske implementacije izvršavanoj na CPU-u, osobito za skupove s većim brojem primjera. Da bi se performanse koje nudi FPGA učinkovito iskoristile za izvedbu učenja stabla odluke, potrebno je uvesti manje promjene u jezgre i veće promjene u hibridni algoritam.

Glavna ideja vodilja je da se jednom radnjom uvijek obrađuje dovoljno primjera da minimalno vrijeme izvršavanja jezgre ne dolazi do izražaja. Za izračun matrice frekvencija to znači da se jednim pozivom obrađuje niz podskupova. Pri grupiranju, uz provedbu grupiranja više skupova jednim pozivom programskog sučelja, potrebno je i smanjiti nasumičnost pristupa pri pisanju u memoriju. Cilj je postići što dulje nizove pisanja u uzastopne memorijske lokacije i tako bolje iskoristiti raspoloživu propusnost SDRAM-a. Da bi se mogle iskoristiti navedene promjene u radu koprocesora, potrebno je promijeniti redoslijed izgradnje

stabla iz izgradnje u dubinu u izgradnju u širinu, te izvesti planiranje razmještaja podskupova u memoriji koprocesora u grupe za obradu jednim pozivom sučelja.

8 Zaključak

Heterogeni računalni sustavi u posljednjih desetak godina postaju sve raširenji u primjeni u računarstvu visokih performansi. Razvojem naprednih FPGA sklopova i komercijalnih rješenja za izvedbu koprocesora, koja uz sklopovlje pružaju i razvojnu okolinu i programsku sučelje, FPGA-ovi postaju održiva platforma za razvoj korisnički definiranih koprocesora. Jedno od područja koje može imati velike koristi od primjene FPGA koprocesora jest dubinska analiza podataka. Količine prikupljenih podataka već dulje vrijeme neprekidno rastu. Stoga računalne metode analize koje omogućuju izvlačenje informacija i znanja iz skupova podataka postaju sve važnije.

Motivacija ovog rada bila je pomoću FPGA-a ubrzati jedan široko korišteni algoritam za dubinsku analizu podataka – algoritam za učenje stabla odluke C4.5. Algoritmi za učenje stabla odluke općenito su memorijski intenzivni, što znači da imaju velik broj pristupa memoriji (čitanja ili pisanja) u odnosu na broj računskih operacija izvedenih za vrijeme izvršavanja algoritma. FPGA-ovi su najpogodniji za implementaciju računskih postupaka koji se mogu predstaviti dubokom protočnom strukturom. Takvi postupci najčešće su računski intenzivni, tj. imaju velik broj računskih operacija u odnosu na broj pristupa memoriji. Navedene činjenice su i razlog malom broju objavljenih FPGA implementacija učenja stabla odluke. FPGA sklopovi već su dokazana podloga za izvršavanje algoritama za dubinsku analizu podataka, osobito računski intenzivnih algoritama. Pojavom i komercijalnom dostupnošću sustava u kojim su FPGA-ovi opremljeni brzim i širokim memorijskim sabirnicama otvorene su mogućnosti korištenja i za memorijski intenzivne algoritme, kao što je učenje stabla odluke.

U ovom radu predstavljen je novi heterogeni računalni sustav i hibridni algoritam za učenje stabla odluke. Heterogeni računalni sustav izveden je pomoću komercijalno dostupne platforme za izgradnju FPGA koprocesora Maxeler Vectus-Lite i vezanih programskih alata. Oblikovanje i izvedba sustava i hibridnog algoritma DF-DTC provedena je metodologijom programsko-sklopovskog suoblikovanja.

Ključan korak u izvedbi koprocesora odabir je dijela algoritma koji će se izvršavati na njemu. U tu svrhu provedena je dinamička analiza dvije implementacije algoritma C4.5: izvorne

implementacije algoritma i njegove poboljšane inačice EC4.5. Iz analize je zaključeno da se većina vremena učenja stabla troši na tri procesa: izračun matrice frekvencija za nominalne atribute, nalaženje najbolje podjele skupa za numeričke atribute, te formiranje podskupova na temelju testa atributa. U svrhu ispitivanja različitih strategija paralelizacije s potencijalnom primjenom u izvedbi FPGA akceleratora, razrađena je programska paralelizacija algoritma EC4.5. Definirane su tri strategije paralelizacije: paralelna obrada različitih primjera podskupa, paralelna obrada različitih atributa i paralelna obrada različitih čvorova, odnosno podskupova. Strategija paralelne obrade različitih atributa prepoznata je kao najprikladnija za implementaciju na FPGA-u.

Na temelju rezultata dinamičke analize, te probne paralelizacije, metodologijom programsko-sklopovskog suoblikovanja razrađena je arhitektura FPGA koprocesora i hibridni algoritam DF-DTC. U sklopovlju je implementiran vremenski najzahtjevniji proces obrade nominalnih atributa – izračun matrice frekvencija. Ostali dijelovi su implementirani programski i izvršavaju se na CPU-u, osim procesa podjele skupa na podskupove koji se izvršava na CPU-u i na koprocesoru. Razrađene su tri varijante jezgre za izračun matrice frekvencije. Izračun matrice frekvencija izveden je prema modelu podatkovnog toka, a glavna operacija koja je izvedena u jezgri je prebrojavanje ulaznih kombinacija vrijednost atributa-klasa. Konačna arhitektura koprocesora sadrži varijantu jezgre koja izračunava matrice frekvencija za više atributa istovremeno, te jezgru za provedbu grupiranja primjera.

Hibridni algoritam DF-DTC oblikovan je za rad s FPGA koprocesorom. Algoritam koristi proširenu podatkovnu strukturu za pohranu skupa za učenje, te pomoćne strukture za rad s koprocesorom. U glavnoj memoriji računala algoritam radi s cjelovitim skupom, dok se u memoriju koprocesora pohranjuju samo nominalni atributi. Budući da se skup za učenje nalazi u dva odvojena memorijska sustava, prilikom svake podjele skupa na podskupove provodi se i usklađivanje njihova sadržaja. Obrada numeričkih atributa izvršava se isključivo na CPU-u. Obrada nominalnih atributa može se odvijati na CPU-u i na koprocesoru, a odabir programske ili sklopovske obrade ovisi o veličini obrađivanog podskupa.

Radi ocjene performansi razrađen je postupak vrednovanja heterogenog računalnog sustava i hibridnog algoritma DF-DTC. Pri vrednovanju koristi se jedan sintetički i jedan realni skup za učenje iz kojih su generirani uzorci skupova različitih veličina. Vrednovanje se sastoji od tri ispitivanja: ispitivanje obrade podskupa, ispitivanje utjecaja broja primjera i ispitivanje utjecaja broja atributa u skupu za učenje. Sva ispitivanja provedena su na implementaciji

algoritma DF-DTC i na programskoj implementaciji EC4.5, koja je prilagođena za izvršavanje na više dretvi. Implementacija EC4.5 služila je kao osnovica za usporedbu performansi. Pri ispitivanju su mjereni ukupno vrijeme izvršavanja programa i vremena izvršavanja ključnih dijelova algoritma. Prosječno ubrzanje obrade nominalnih atributa hibridnim algoritmom iznosi 3,00 puta u usporedbi s programskom implementacijom. Za cjelokupno izvršavanje programa najveće ubrzanje iznosi 1,18 puta, a postignuto je za skupove s 200 atributa i 2.000.00 primjera. Analizom rezultata vrednovanja nađen je uzrok malim poboljšanjima u brzini algoritma, te su dobivene smjernice za daljnja poboljšanja.

Razrada, te izvedba i analiza radnih značajki FPGA koprocesora pokazala je prikladnost korištenja FPGA-a kao platforme za ubrzanje učenja stabla odluke, ali uz dodatne izmjene hibridnog algoritma kojima će se postići bolje iskorištenje koprocesora. Daljnje istraživanje može biti usmjereno u više pravaca:

- unapređenje koprocesora i algoritma DF-DTC
- implementacija obrade numeričkih atributa na FPGA koprocesoru
- poopćavanje DF-DTC u programski okvir za učenje stabla odluke.

Koprocesor i hibridni algoritam mogu se unaprijediti prema smjernicama proizašlim iz rezultata vrednovanja. Unapređenjem se može postići bolje iskorištenje mogućnosti FPGA-a, a time i bolje ukupne performanse. Drugi smjer istraživanja usmjeren je na obradu numeričkih atributa u FPGA koprocesoru. Obrada numeričkih atributa zahtijeva rješavanje problema sortiranja na sklopovlju koje je prilagođeno za rad prema modelu toka podataka. Efikasna implementacija sortiranja omogućila bi izvođenje obrade numeričkih atributa, a time i svih vremenski intenzivnih dijelova algoritma, u koprocesoru. Treći smjer je najopsežniji: budući da su algoritmi za učenje stabla odluke zasnovani na istom temeljnom algoritmu, hibridni algoritam može se prilagoditi za korištenje drugih metoda izbora testnog atributa. Daljnje istraživanje može biti usmjereno na njegovo poopćavanje s krajnjim ciljem da obuhvati cijelu klasu algoritama za učenje stabla odluke.

Bibliografija

[1] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture," *Found. Trends Electron. Des. Autom.*, vol. 2, no. 2, pp. 153–253, 2008.

[2] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed., vol. 54. Morgan Kaufmann, 2011, p. 664.

[3] L. Rokach and O. Maimon, *Data Mining with Decision Trees: Theroy and Applications*. River Edge, NJ, USA: World Scientific Publishing, 2008.

[4] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1–37, Dec. 2007.

[5] J. R. Quinlan, "Discovering rules by induction from large collections of examples," in *Expert systems in the micro electronic age*, D. Michie, Ed. Edinburgh: Edinburgh University Press, 1979.

[6] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Belmont: Wadsworth, 1984.

[7] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo: Morgan Kaufmann, 1993, p. 302.

[8] S. Ruggieri, "Efficient C4.5," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 2, pp. 438–444, 2002.

[9] S. Ruggieri, "YaDT: yet another decision tree builder," in *16th IEEE International Conference on Tools with Artificial Intelligence*, 2004, pp. 260–265.

[10] M. Aldinucci, S. Ruggieri, and M. Torquati, "Porting Decision Tree Algorithms to Multicore using FastFlow," p. 18, Jun. 2010.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software," *ACM SIGKDD Explor. Newsl.*, vol. 11, p. 10, 2009.

[12] J. Shafer, R. Agrawal, and M. Mehta, "SPRINT: A Scalable Parallel Classifier for Data Mining," in *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB)*, 1996, pp. 544–555.

[13] M. V. Joshi, G. Karypis, and V. Kumar, "ScalParC: a new scalable and efficient parallel classification algorithm for mining large datasets," in *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, 1998, pp. 573–579.

[14] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, “PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce,” *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1426–1437, 2009.

[15] P. Škoda, B. Medved Rogina, and V. Sruk, “FPGA implementations of data mining algorithms,” in *MIPRO, 2012 Proceedings of the 35th International Convention*, 2012, pp. 362–367.

[16] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, “An FPGA Implementation of Decision Tree Classification,” in *2007 Design, Automation & Test in Europe Conference & Exhibition*, 2007, pp. 1–6.

[17] R. J. R. Struharik and L. A. Novak, “Evolving Decision Trees in Hardware,” *J. Circuits, Syst. Comput.*, vol. 18, no. 06, pp. 1033–1060, Oct. 2009.

[18] G. Chrysos, P. Dagritzikos, I. Papaefstathiou, and A. Dollas, “HC-CART: A parallel system implementation of data mining classification and regression tree (CART) algorithm on a multi-FPGA system,” *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 1–25, Jan. 2013.

[19] Mark L. Chang, “Device Architecture,” in *Reconfigurable Computing: The Theory and Practice of FPGA-based Computation*, S. Hauck and A. DeHon, Eds. Morgan Kaufmann, 2008, pp. 3–27.

[20] B. Medved Rogina, P. Škoda, K. Skala, and I. Michieli, “Metastability Testing at FPGA Circuit Design using Propagation Time Characterization,” *Radioelectron. Informatics J.*, vol. 51, no. 4, pp. 4–8, 2010.

[21] P. J. Ashenden, *The Designer’s Guide to VHDL*, 3rd ed. Morgan Kaufmann, 2010.

[22] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*, 2nd ed. Prentice Hall, 2003, p. 496.

[23] “Virtex-6 Family Overview,” 2012. http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. [2014-02-03].

[24] “Virtex-6 FPGA Configurable Logic Block User Guide,” 2012. http://www.xilinx.com/support/documentation/user_guides/ug364.pdf. [2014-02-03].

[25] “Virtex-6 FPGA DSP48E1 Slice User Guide,” 2011. http://www.xilinx.com/support/documentation/user_guides/ug369.pdf. [2014-02-03].

[26] “Virtex-6 FPGA Memory User Guide,” 2013. http://www.xilinx.com/support/documentation/user_guides/ug363.pdf. [2014-02-03].

[27] “Virtex-6 FPGA Clocking Resources User Guide,” 2014. http://www.xilinx.com/support/documentation/user_guides/ug362.pdf. [2014-02-03].

[28] S. Lopez-Estrada and R. Cumplido, “Decision Tree Based FPGA-Architecture for Texture Sea State Classification,” in *2006 IEEE International Conference on Reconfigurable Computing and FPGA’s (ReConFig 2006)*, 2006, pp. 1–7.

[29] W. Jiang and V. K. Prasanna, “A FPGA-based Parallel Architecture for Scalable High-Speed Packet Classification,” in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2009, pp. 24–31.

[30] J. R. Struharik, “Implementing decision trees in hardware,” *2011 IEEE 9th Int. Symp. Intell. Syst. Informatics*, pp. 41–46, 2011.

[31] J. Oberg, K. Eguro, R. Bittner, and A. Forin, “Random decision tree body part recognition using FPGAs,” in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 330–337.

[32] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, “Pipelined Decision Tree Classification Accelerator Implementation in FPGA (DT-CAIF),” *IEEE Trans. Comput.*, vol. 0, pp. 1–1, 2013.

[33] O. Pina-Ramirez, R. Valdes-Cristerna, and O. Yanez-Suarez, “An FPGA Implementation of Linear Kernel Support Vector Machines,” in *2006 IEEE International Conference on Reconfigurable Computing and FPGA’s (ReConFig 2006)*, 2006, pp. 1–6.

[34] K. Irick, M. DeBole, V. Narayanan, and A. Gayasen, “A Hardware Efficient Support Vector Machine Architecture for FPGA,” in *2008 16th International Symposium on Field-Programmable Custom Computing Machines*, 2008, pp. 304–305.

[35] M. Ruiz-Llata, G. Guarnizo, and M. Yebenes-Calvino, “FPGA implementation of a support vector machine for classification and regression,” in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–5.

[36] C. Kyrkou, T. Theocharides, and C.-S. Bouganis, “An embedded hardware-efficient architecture for real-time cascade Support Vector Machine classification,” in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2013, pp. 129–136.

[37] J. Zhu and P. Sutton, “FPGA Implementations of Neural Networks - A Survey of a Decade of Progress,” *Lect. Notes Comput. Sci.*, vol. 2778, pp. 1062–1066, 2003.

[38] A. R. Omondi and J. C. Rajapakse, *FPGA implementations of neural networks*. 2006, pp. 1–360.

[39] J. Misra and I. Saha, “Artificial neural networks in hardware: A survey of two decades of progress,” *Neurocomputing*, vol. 74, no. 1–3, pp. 239–255, Dec. 2010.

[40] P. Škoda, T. Lipić, Á. Srp, B. Medved Rogina, K. Skala, and F. Vajda, “Implementation Framework for Artificial Neural Networks on FPGA,” in *MIPRO, 2011 Proceedings of the 34th International Convention*, 2011, pp. 274–278.

[41] K. Cheung, S. R. Schultz, and W. Luk, “A Large-Scale Spiking Neural Network Accelerator for FPGA Systems,” in *Artificial Neural Networks and Machine Learning – ICANN 2012*, 2012, pp. 113–120.

[42] L.-W. Kim, S. Asaad, and R. Linsker, “A Fully Pipelined FPGA Architecture of a Factored Restricted Boltzmann Machine Artificial Neural Network,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 1, pp. 1–23, Feb. 2014.

[43] D. Heath, S. Kasif, and S. Salzberg, “Induction of Oblique Decision Trees,” in *Proceedings of the 13th International Joint Conference on Artiffcial Intelligence*, 1993, pp. 1002–1007.

[44] D. Levi, “HereBoy: a fast evolutionary algorithm,” *Proceedings. Second NASA/DoD Work. Evolvable Hardw.*, 2000.

[45] K. Bache and M. Lichman, “UCI Machine Learning Repository.” University of California, Irvine, School of Information and Computer Sciences, 2013.

[46] J. D. Bakos, “High-Performance Heterogeneous Computing with the Convey HC-1,” *Comput. Sci. Eng.*, vol. 12, no. 6, pp. 80–87, Nov. 2010.

[47] R Development Core Team, *R: A Language and Environment for Statistical Computing*, vol. 1. 2011, p. 409.

[48] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley-Interscience, 1998, p. 736.

[49] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, pp. 273–297, 1995.

[50] J. C. Platt, “Sequential minimal optimization: A fast algorithm for training support vector machines,” in *Advances in kernel methods: Support vector learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1998, pp. 185–209.

[51] T. Serafini, G. Zanghirati, and L. Zanni, “Gradient projection methods for quadratic programs and applications in training support vector machines,” *Optim. Methods Softw.*, vol. 20, no. 2–3, pp. 353–378, Apr. 2005.

[52] D. Anguita, A. Boni, and S. Ridella, “A digital architecture for support vector machines: theory, algorithm, and FPGA implementation,” *IEEE Trans. neural networks*, vol. 14, no. 5, pp. 993–1009, Jan. 2003.

[53] R. Pedersen and M. Schoeberl, “An Embedded Support Vector Machine,” in *2006 International Workshop on Intelligent Solutions in Embedded Systems*, 2006, pp. 1–11.

[54] M. Schoeberl, “JOP: A Java Optimized Processor for Embedded Real-Time Systems,” Vienna University of Technology, 2005.

[55] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. P. Graf, "A Massively Parallel FPGA-Based Coprocessor for Support Vector Machines," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, 2009, pp. 115–122.

[56] K. Cao, H. Shen, and H. Chen, "A parallel and scalable digital architecture for training support vector machines," *J. Zhejiang Univ. Sci. C*, vol. 11, no. 8, pp. 620–628, May 2010.

[57] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to Platt's SMO Algorithm for SVM Classifier Design," *Neural Comput.*, vol. 13, no. 3, pp. 637–649, Mar. 2001.

[58] M. Papadonikolakis and C.-S. Bouganis, "A Heterogeneous FPGA Architecture for Support Vector Machine Training," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 211–214.

[59] S. Wang, Y. Peng, G. Zhao, and X. Peng, "Accelerating on-line training of LS-SVM with run-time reconfiguration," in *2011 International Conference on Field-Programmable Technology*, 2011, pp. 1–6.

[60] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Process. Lett.*, vol. 9, no. 3, pp. 293–300, 1999.

[61] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Upper Saddle River, NJ: Prentice Hall, 1988, p. 304.

[62] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, "Algorithmic transformations in the implementation of K-means clustering on reconfigurable hardware," in *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays - FPGA '01*, 2001, pp. 103–110.

[63] M. Gokhale, J. Frigo, K. Mccabe, J. Theiler, C. Wolinski, and D. Lavenier, "Experience with a hybrid processor: K-means clustering," *J. Supercomput.*, vol. 26, no. 2, pp. 131–148, 2003.

[64] X. Wang and M. Leeser, "K-means Clustering for Multispectral Images Using Floating-Point Divide," in *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*, 2007, pp. 151–162.

[65] T. Maruyama, "Real-time K-Means Clustering for Color Images on Reconfigurable Hardware," in *18th International Conference on Pattern Recognition (ICPR'06)*, 2006, pp. 816–819.

[66] T. Saegusa and T. Maruyama, "An FPGA implementation of real-time K-means clustering for color images," *J. Real-Time Image Process.*, vol. 2, no. 4, pp. 309–318, Nov. 2007.

[67] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “An efficient k-means clustering algorithm: analysis and implementation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.

[68] G. A. Covington, C. L. G. Comstock, A. A. Levine, J. W. Lockwood, and Y. H. Cho, “High Speed Document Clustering in Reconfigurable Hardware,” in *2006 International Conference on Field Programmable Logic and Applications*, 2006, no. August, pp. 1–7.

[69] K. Nagarajan, B. Holland, A. D. George, K. C. Slatton, and H. Lam, “Accelerating Machine-Learning Algorithms on FPGAs using Pattern-Based Decomposition,” *J. Signal Process. Syst.*, vol. 62, no. 1, pp. 43–63, Jan. 2009.

[70] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, “FPGA implementation of K-means algorithm for bioinformatics application: An accelerated approach to clustering Microarray data,” in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 248–255.

[71] J. Singaraju and J. A. Chandy, “Active storage networks for accelerating K-means data clustering,” in *Proceedings of the 7th international conference on Reconfigurable computing: architectures, tools and applications*, 2011, pp. 102–109.

[72] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design*. Boston, MA: Springer, 2009.

[73] J. R. Quinlan, “C4.5 Release 8,” 1993. http://rulequest.com/Personal/c4.5r8.tar.gz. [23-06-2013].

[74] S. Ruggieri, “Patch from C4.5 release 8 to EC4.5 Beta 1.0 for Linux platforms.” http://www.di.unipi.it/~ruggieri/YaDT/EC45Beta10.zip. [2013-06-23].

[75] Machine Learning Group at the University of Waikato, “Weka 3: Data Mining Software in Java,” 2014. http://www.cs.waikato.ac.nz/ml/weka/. [2014-02-05].

[76] U. M. Fayyad and K. B. Irani, “On the handling of continuous-valued attributes in decision tree generation,” *Mach. Learn.*, vol. 8, no. 1, pp. 87–102, Jan. 1992.

[77] S. Ruggieri, “YaDT 1.2.5,” 2010. http://www.di.unipi.it/~ruggieri/YaDT/YaDT1.2.5.zip. [2014-02-05].

[78] S. L. Graham, P. B. Kessler, and M. K. Mckusick, “gprof: a Call Graph Execution Profiler,” *ACM SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, Jun. 1982.

[79] “Convey Computer HC Series,” 2014. http://www.conveycomputer.com/products/hcseries/. [2014-05-06].

[80] “Maxeler Technologies,” 2014. http://www.maxeler.com/. [2014-02-10].

[81] “Nallatech FPGA Accelerated Computing Solutions,” 2014. http://www.nallatech.com/opencl-fpga.html. [2014-05-12].

[82] O. Pell and V. Averbukh, “Maximum Performance Computing with Dataflow Engines,” *Comput. Sci. Eng.*, vol. 14, no. 4, pp. 98–103, Jul. 2012.

[83] P. Škoda, V. Sruk, and B. Medved Rogina, “Frequency Table Computation on Dataflow Architecture,” in *MIPRO, 2014 Proceedings of the 37th International Convention*, 2014, pp. 357–361.

[84] *MaxCompiler: Manager Compiler Tutorial, version 2013.2.2*. Maxeler Technologies, 2013.

[85] OpenMP Architecture Review Board, “OpenMP Application Program Interface,” 2011. http://www.openmp.org/mp-documents/OpenMP3.1.pdf. [2014-02-05].

[86] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, vol. 10. 2008, p. 353.

[87] G. Melli, “The datgen Dataset Generator, version 3.1,” 1999. http://www.datasetgenerator.com. [2014-06-09].

[88] *Multiscale Dataflow Programming, version 2013.2.2*. Maxeler Technologies, 2013.

Popis oznaka

Oznaka	Značenje
ALU	Aritmetički-logička jedinica, engl. *arithmetic logic unit*
CLB	Programirljivi logički blok, engl. *configurable logic block*
CPU	Glavni procesor, engl. *central processing unit*
DF-DTC	Hibridni algoritma za učenje stabla odluke, engl. *dataflow decision tree construction*
DFE	FPGA koprocesor, engl. *dataflow engine*
FF	Bistabil D-tipa, engl. *flip-flop*
FIFO	Memorijski spremnik kod kojeg je prvi upisani podatak onaj koji se prvi čita, engl. *first in first out*
FMem	Brza memorija unutar FPGA
FPGA	Programirljivo polje logičkih elemenata, engl. *field programmable gate array*
LMem	SDAM spojen na FPGA
LUT	Pregledna tablica, engl. *lookup table*
MAC	Združeno množenje i zbrajanje, engl. *multiply-accumulate*
PCIe	PCI express sabirnica
RAM	Memorija s nasumičnim pristupom, engl. *random access memory*.
SDRAM	Dinamički RAM koji se koristi u računalnim sustavima kao glavna radna memorija, engl *synchornous dynamic RAM.*

Prilog A Izmjerene vrijednosti protoka funkcija i jezgara za izračun matrice frekvencija

Tablica A.1. Protoci funkcije za izračun matrice frekvencija na CPU sa šest dretvi uz nepotpuno iskorištenje dretvi

	Broj atributa / Protok ($\times 10^6$ elemenata u sekundi)					
Broj primjera	**1**	**2**	**3**	**4**	**5**	**6**
2.048	42,84	84,28	124,1	162,1	201,9	242,9
4.096	68,69	134,3	200,5	260,0	326,2	388,2
8.192	98,43	193,9	282,6	373,2	465,6	557,2
16.384	126,5	249,5	366,5	479,1	600,9	706,4
32.768	149,7	291,4	426,7	549,4	686,0	800,5
65.536	160,4	312,7	456,3	561,5	735,6	851,0
131.072	145,3	327,7	471,0	585,5	768,4	874,8
262.144	155,3	311,8	471,9	626,8	781,7	765,2
524.288	147,1	295,6	443,0	590,4	748,4	746,4
1.048.576	149,7	297,4	437,9	599,1	750,0	752,1
2.097.152	150,5	299,5	449,2	601,7	758,1	753,3
4.194.304	150,8	297,5	452,3	597,9	754,5	751,4

Tablica A.2. Protoci jezgre *ComputeFreq* MS uz nepotpuno iskorištenje podatkovnih tokova

	Broj atributa / Protok ($\times 10^6$ elemenata u sekundi)					
Broj primjera	**1**	**2**	**3**	**4**	**5**	**6**
2.048	2,527	5,057	7,547	10,23	12,90	15,35
4.096	5,025	9,964	14,98	19,99	25,14	30,08
8.192	9,157	18,59	27,84	37,37	46,13	54,68
16.384	15,94	31,94	48,18	63,94	80,66	95,32
32.768	24,91	49,90	74,36	100,6	123,6	148,7
65.536	34,05	68,28	102,2	134,6	172,2	203,8
131.072	43,12	86,63	130,2	172,9	217,1	261,4
262.144	49,66	99,26	149,0	198,9	250,3	298,1
524.288	54,48	108,7	162,9	217,2	271,4	325,5
1.048.576	56,75	113,3	169,7	226,2	282,8	339,1
2.097.152	58,25	116,5	174,9	232,8	291,0	349,2
4.194.304	59,09	118,2	177,3	236,4	295,4	354,7

Tablica A.3. Protoci jezgre *ComputeFreq* MS2 uz nepotpuno iskorištenje podatkovnih tokova

	Broj atributa / Protok (×10^6 elemenata u sekundi)					
Broj primjera	**1**	**2**	**3**	**4**	**5**	**6**
2.048	2,561	5,097	7,569	10,27	12,71	15,28
4.096	5,047	10,21	15,38	20,40	25,48	30,74
8.192	10,13	19,87	30,20	40,78	50,59	62,04
16.384	19,48	38,60	57,46	77,25	96,58	115,1
32.768	34,32	68,29	102,8	135,1	170,3	205,5
65.536	56,45	112,9	168,6	224,4	278,6	336,5
131.072	80,35	159,4	240,6	319,5	404,3	480,0
262.144	107,0	212,4	321,5	424,3	537,4	635,9
524.288	127,9	252,8	378,5	505,7	631,4	756,1
1.048.576	140,4	281,3	421,8	560,7	703,2	845,4
2.097.152	149,0	297,7	446,3	595,3	743,9	891,9
4.194.304	153,8	308,2	463,1	615,7	769,0	924,8

Tablica A.4. Prosječne vrijednosti protoka jezgara *ComputeFreq* SS, MS i MS2

	Jezgra		
Broj primjera	**SS**	**MS**	**MS2**
2.048	4,295	15,35	15,30
4.096	8,380	30,19	30,48
8.192	14,80	55,08	61,05
16.384	24,16	96,16	115,8
32.768	35,44	149,4	205,8
65.536	44,36	206,5	338,9
131.072	53,16	259,9	477,7
262.144	58,29	299,3	636,8
524.288	62,10	326,1	759,1
1.048.576	63,79	339,5	843,6
2.097.152	65,19	349,2	892,5
4.194.304	65,90	354,5	922,7

Popis slika

Popis tablica

Popis ispisa

Životopis

Peter Škoda rođen je u Mariboru, R. Slovenija, 1980. godine. 2006. godine diplomirao je elektrotehniku na Fakultetu elektrotehnike i računarstva u Zagrebu. Za vrijeme posljednje godine studija, 2005. godine, počinje raditi u tvrtci Xylon d.o.o., Zagreb, kao inženjer za razvoj FPGA sustava. 2006. godine odlazi na civilno odsluženje vojnog roka u Udrugu za pomoć osobama s mentalnom retardacijom Međimurske županije, Čakovec. Nakon završetka civilnog roka, 2007. godine zapošljava se u Brodarskom institutu d.o.o., na radnom mjestu mlađeg istraživača u sektoru Eksperimentalna hidrodinamika, gdje radi na održavanju i razvoju elektroničkih mjernih uređaja. Od 2008. godine zaposlen je kao znanstveni novak na Zavodu za elektroniku Instituta Ruđer Bošković. Iste godine upisuje poslijediplomski doktorski studija Fakulteta elektrotehnike i računarstva. Tijekom poslijediplomskog studija radio je na EU FP7 projektu „Embedded Computer Engineering Learning Platform E2LP“, tijekom kojeg je istraživao primjenu FPGA sklopova u računarstvu, s naglaskom na područje dubinske analize podataka. Trenutno je istraživač na HRZZ projektu „Algoritmi strojnog učenja za otkrivanje znanja u složenim strukturama podataka“. Objavio je četiri znanstvena rada na međunarodnim konferencijama i jedan znanstveni rad u časopisu. Član je međunarodne strukovne udruge *Institute of Electrical and Electronics Engineers* (IEEE).

Popis objavljenih radova

[1] Medved Rogina, B., Škoda, P., Skala, K., Michieli, I., “Metastability Testing at FPGA Circuit Design using Propagation Time Characterization,” *Radioelectron. Informatics J.*, Vol. 51, No. 4, 2010., str. 4–8.

[2] Škoda, P., Lipić, T., Srp, Á., Medved Rogina, B., Skala, K., Vajda, F., “Implementation Framework for Artificial Neural Networks on FPGA,” *MIPRO, 2011 Proceedings of the 34th International Convention*, 2011., str. 274–278.

[3] Kiss, N., Patai, G., Hanak, P., Lipić, T., Škoda, P., Gjenero, L., Dubravic, A., Michieli, I., “Vital Fitness and Health Telemonitoring of Elderly People,” *MIPRO, 2011 Proceedings of the 34th International Convention*, 2011., str. 279-284.

[4] Škoda, P., Medved Rogina, B., Sruk, V., “FPGA implementations of data mining algorithms,” *MIPRO, 2012 Proceedings of the 35th International Convention*, 2012., str. 362–367.

[5] Škoda, P. , Sruk, V., Medved Rogina, B., “Frequency Table Computation on Dataflow Architecture,” *MIPRO, 2014 Proceedings of the 37th International Convention*, 2014., str. 357–361.

Biography

Peter Škoda was born in Maribor, Slovenia, in 1980. In 2006 he received M.Eng. in electrical engineering at the Faculty of electrical engineering and computing at the University of Zagreb. During the final year at the University, in 2005, he began working in Xylon d.o.o., Zagreb, at position of junior FPGA development engineer. In 2006 he leaves for civilian service. After completing civilian service, in 2007, he starts working at Brodarski institute, in sector Experimental hydrodynamics, as engineer for maintenance and development of electronic measurement systems. Since 2008 he is employed at Ruđer Bošković Institute, at position of Research assistant. In 2008 he enrolled Postgraduate doctoral study at Faculty of electrical engineering and computing at University of Zagreb. During the course of the doctoral study he worked EU FP 7 project "Embedded Computer Engineering Learning Platform E2LP", during which he researched applications of FPGA devices in computing systems, with focus on applications in data mining. Currently he is researcher on project "Machine learning algorithms for insightful analysis of complex data structures" financed by the HRZZ. He published four papers at international science conferences, and one paper in a science journal. He is a member of international professional association Institute of Electrical and Electronics Engineers (IEEE).