FPGA realizacija lukap algoritma upotrebom ćelijske strukture

Jordan B. Jovanović, Ognjen M. Vuković

***Sadržaj* — Sa naglim porastom bitskog protoka koji se može prenositi kroz medijume i kontinualnim rastom saobraćaja na Internetu, potreba za ruterima koji mogu na efikasan i brz način da izvrše usmeravanje IP paketa je veća nego ikada. Jednu od ključnih uloga u performansama rutera ima tzv. lukap funkcija, koja na osnovu odredišne IP adrese vrši pretragu lukap tabele da bi odredila na koji izlazni port rutera treba proslediti paket. Ova funkcija predstavlja potencijalnu tačku zagušenja povećanja realnog kapaciteta mreže. Iz tog razloga su razvijeni razni algoritmi koji treba ovu funkciju da učine što efikasnijom, da bi i ruter samim tim bio efikasniji. U ovom radu je izloženo jedno hardversko rešenje za lukap funkciju implementirano na Alterinom FPGA čipu.**

***Ključne reči* — FPGA, Lukap funkcija, Ruter**

I. UVOD

USLED sve zahtevnijih aplikacija koje zahtevaju veoma velike propusne opege (npr. prenos video signala u realnom vremenu), saobraćaj na Internetu se konstantno povećava. Da bi se izborilo sa povećanim saobraćajnim zahtevima implementiraju se linkovi sve većeg kapaciteta do nekoliko desetina gigabajta po sekundi sa tendencijom da se ti protoci podignu na terabitske brzine. Uz to neprestani rast korisnika Interneta zahteva sve veći broj IP adresa. Kako IPv4 adresni prostor polako postaje nedovoljan da opsluži toliki broj korisnika u toku je tranzicija na IPv6 koji će u adekvatnoj meri proširiti adresni prostor. Sa povećanjem broja bita IP adresa, shodno standardu IPv6, povećava se i broj bita adresnog prefiksa.

Kada se krajem prošlog veka [2] prešlo sa klasnog na besklasno adresiranje ukidanjem fiksne granice između adresnog prefiksa i *host* dela omogućena je racionalnija upotreba adresnog prostora, koji kod IPv4 predstavlja ograničen resurs. Besklasno adresiranje je takođe dovelo i do smanjenja broja zapisa u lukap tabeli, što je omogućeno agregacijom adresa na više nivoa. Ideja je bila da dodela adresa ima topološki smisao tako da se adrese mogu agregirati na više tačaka unutar hijerarhije Internet mreže. S obzirom na to da se u lukap tabeli može pronaći više zapisa koji odgovaraju IP adresi za koju se vrši pretraga, pristupa se pronalaženju adresnog prefiksa maksimalnog poklapanja i paket se prosleđuje na odgovarajući izlazni port rutera. Pronalaženje adresnog prefiksa maksimalne dužine (poklapanja) predstavlja najveći problem besklasnog adresiranja.

Zbog svega ovoga, obrada paketa u ruteru mora biti veoma brza i efikasna jer ruter sada pri ovim gigabitskim kapacitetima linkova mora da procesira milione paketa u sekundi i prosleđuje ih na odgovarajuće izlazne portove. Odatle se jasno vidi da je jedan od glavnih problema povećanja kapaciteta rutera upravo lukap funkcija koja predstavlja kritičnu tačku u dizajnu rutera. Zbog toga u poslednje vreme izuzetno puno pažnje je posvećeno ovom problemu. Razvijen je velik broj mahom softverskih algoritama koji u određenoj meri rešavaju ove probleme.

Softverski implementirani algoritmi zasnivaju svoj rad na *RISC* procesorima koji upravljaju lukap tabelom, smeštenoj u eksternu memoriju (*SRAM ili DRAM).* Sa stanovišta brzine lukap funkcije u ruteru, pokazalo se kao kritično vreme pristupa CPU-a eksternoj memoriji, što ujedno i predstavlja jednu od glavnih mana softverskih algoritama.

U ovom radu je predložen jedan hardverski algoritam, implementiran na Alterinom FPGA čipu [1] koji bi trebao da reši neke od mana softverski baziranih algoritama

Ostatak rada je organizovan na sledeći način. Drugi deo ovog rada daje pregled postojećih lukap algoritama, sa njihovim prednostima i manama. Treći deo se bavi opisom našeg rešenja lukap algoritma. U četvrtom delu biće izložene performanse ovog algoritma na Alterinom FPGA čipu CYCLONE II familije. Na kraju će biti dat zaključak u kome će biti dat pregled prednosti našeg algoritma, predložena dalje optimizacija, kao i moguća poboljšanja ovog algoritma.

II. OPIS POSTOJEĆIH ALGORITAMA

Sami algoritmi se mogu klasifikovati na više načina, npr. softverski ili hardverski, da li se problem rešava po dimenziji vrednosti ili veličine itd.

A. Binarna trie struktura

Kod ovog algoritma lukap tabela je organizovana u binarno stablo [3]. Pretraga se vrši tako što se u svakom koraku gleda sledeći bit adresnog prefiksa da bi se odredilo kojom izlaznom granom treba napustiti čvor. Neki čvorovi u sebi sadrže zapise, neki ne u zavisnosti od toga da li odgovarajući prefiks postoji u tabeli ili ne. Idući kroz stablo pamti se najbolje rešenje nađeno dotad sve dok se ne dođe do kraja stabla.

Update procedura je jednostavna jer se ona svodi na pretragu po stablu dok se ne dođe do traženog mesta gde se vrši promena, brisanje ili dodavanje zapisa.

Jordan B. Jovanović, Elektrotehnički fakultet u Beogradu, Srbija (e-mail: jocaj@sbb.co.yu).

Ognjen M. Vuković, Elektrotehnički fakultet u Beogradu, Srbija (e-mail: ognjen85@email.co.yu).

Mana ovog rešenja jeste brzina. Zahteva se veliki broj pristupa eksternoj memoriji u kojoj je smeštena lukap tabela, u najgorem slučaju broj pristupa jednak je broju bita adresnog prefiksa, što takođe dovodi do problema skalabilnosti sa IPv6 adresama koje imaju 128 bita. Još jedan veliki problem ovog algoritma jesu ogromni memorijski zahtevi, koji bi ga u slučaj IPv6 učilini ne praktičnim za serijsku implementaciju na ruterima.

B. *Komprimovana trie struktura*

U prethodno razmatranoj *trie* strukturi u deloviima stabla gde nema račvanja ne donosi se nikakva odluka, u tim stanjima se samo bespotrebno troše resursi vremena i memorije. Komprimovane trie strukture imaju za cilj da eliminišu redundantnost iz lukap tabela što postižu brisanjem (kompresijom) čvorova koji imaju samo jednu izlaznu granu [3]. Na ovaj način se smanjuju memorijski zahtevi, a pri tome se omogućava efikasno nalaženje potrebnih zapisa u tabeli.

Mana ove tehnike je što, u slučaju gusto popunjenih stabala, ne doprinosi velikom poboljšanju jer je broj potencijalnih putanja, koje se mogu komprimovati, veoma mali. Prilokom *update-a* neophodno je vršiti dekompresiju strukture i ponovnu komporesiju što u pogledu vremena predstavlja dodatnu manu.

C. *M-rne trie strukture*

Ovi algoritmi predstavljaju poboljšanje rada bazične binarne trie strukture. Ideja je je da se pretraga ne vrši sa po jednim bitom u svakom koraku, već sa većim brojem bita tzv. *stride*-ovima[3], čime bi se smanjio broj pristupa eksternoj memoriji. Primera radi ako bi se u svakom koraku koristilo po 4 bita, u slučaju IPv4 adrese imali bi umesto 32 pristupa memoriji svega 8 pristupa, a samim tim evidentnu uštedu u vremenu potrebnom za pretragu.

Usled toga što se prolazi kroz stablo vrše na osnovu *stride*-ova može se desiti da se originalna informacija ne nalazi u stablu što dovodi do problema prilikom brisanja ili dodavanja novog zapisa (npr. ukoliko je *stride* dužine 4 u stablu se nalaze samo čvorovi binarnog stabla koji predstavljaju adresne prefikse dužine koja je deljiva sa 4). Takođe, velika mana u ovom algoritmu bi bio prelaz na IPv6 adrese jer bi memorijski zahtevi postali preveliki za komercijalnu upotrebu. Jedna od modifikacija ovog algoritma predstavlja tzv. *LC Algoritam* [4] koji vrši transformaciju stabla, pri čemu *stride*-ovi nemaju konstantnu vrednost tako da stablo postigne minimalnu dubinu.

D. *Algoritmi pretrage po dimenziji dužine*

Ideja je da se vrši binarna pretraga po parametru dužine adresnog prefiksa [5]. Zamišljeno je da se krene od srednje dužine i izvrši provera da li postoji poklapajući prefiks te dužine. Ukoliko postoji onda se pretraga pomera na gornju polovinu (tj. duže prefikse), u suprotnom na donju polovinu (kraće prefikse).

Ovo rešenje ostavlja originalnu brzinu algoritma, ali povećava memorijske zahteve i takođe komplikuje *update* proceduru

E. *Hardverski Algoritmi*

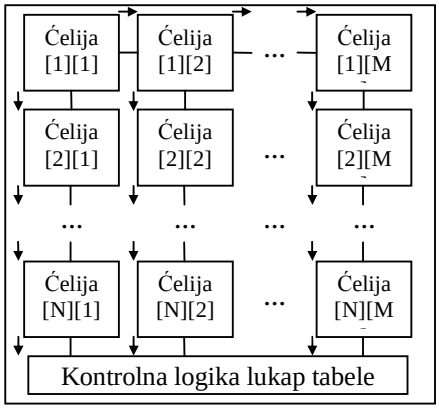
Većina hardverski baziranih algoritama svoj rad zasniva na tzv. *Content Addressable Memory CAM.*[6] Ove memorije su specijalizovane za lako pretraživanje (tj. da se u samo jednom ciklusu takta dobije traženi podatak). Tipična realizacija CAM memorije se zasniva na korišćenju brze SRAM memorije sa adekvatnom ulaznom logikom za komparaciju koja omogućava ovu brzu pretragu.

Korišćenje CAM memorija je ubedljivo najefikasnije rešenje u odnosu na bilo koji softverski algoritam. Dve mane ovog rešenja su cena i disipacija koja je veoma velika.

III. OPIS ALGORITAMA

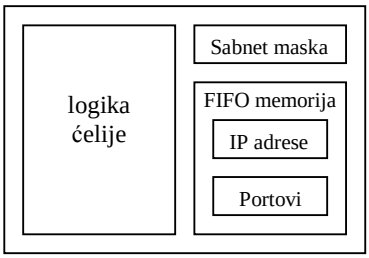
A. *Matrična struktura tabele rutiranja*

Osnovna ideja algoritma je podela memorijskog prostora lukap tabele na više manjih, nezavisnih tabela, kao i samostalna pretraga svake od njih. Na slici 3.1 data je osnovna blok šema.



Slika 3.1. Osnovna blok šema tabele rutiranja

Cela tabela rutiranja predstavljena je matričnom strukturom. Kao element matrice pojavljuje se ćelija. Struktura svih ćelija je identična što omogućava proizvoljan broj redova *N* i kolona *M* matrice. Kada stigne podatak za obradu, bilo da je reč o pretrazi, upisu novog zapisa, izmeni ili brisanju postojećeg, on se od prve ćelije (to je uvek Ćelija 1,1) prosleđuje odmah ćeliji desno i ćeliji ispod nje sa ciljem kvazi-paralelne obrade. Generalni sistem prosleđivanja u cilju kvazi-pararelne obrade je sledeći: sve ćelije prosleđuju podatke ispod dok samo ćelije u prvom redu matrice prosleđuju podatke desno od sebe susednim ćelijama ukoliko postoje. Princip prosleđivanja prikazan je strelicama na slici 3.1. Ovakvim postupkom podatak za obradu stigne do svake ćelije tačno jedanput. Nakon kvazi-paralelne obrade rezultati od svake ćelije stižu do kontrolne logike lukap tabele, koja obrađuje dolazne rezultate paralelno kako stižu i na kraju, u zavisnosti od trenutnog zadatka sistema, donosi adekvatnu odluku i prosleđuje rezultat na izlaz. U slučaju da je u pitanju pretraga, bira se rezultat koji ima najdužu sabnet masku i njegov zapis se uzima kao rezultat pretrage (rezultat pretrage je rezultujući izlazni port rutera). Ovako obezbeđujemo najprecizniju rutu.



Slika 3.2. Struktura ćelije

Kao što je prikazano na slici 3.2. ćelija se sastoji od dve FIFO memorije, jedne za IP adresne prefikse i jedne za odgovarajuće adrese izlaznih portova. Tu je naravno i logika ćelije koja omogućava njeno odgovarajuće funkcionisanje. Zbog uštede resursa uvedeno je da svi zapisi u okviru jedne iste ćelije imaju istu dužinu sabnet maske. Za ovo rešenje odlučeno je iz razloga što se dobija dosta, a gubi jako malo. Najveća prednost je ušteda resursa jer bi u suprotnom slučaju bila potrebna još jedna FIFO memorija po ćeliji koja bi čuvala odgovarajuće sabnet maske, pa bi to značajno povećalo kapacitet dizajna. Potrebno je i napomenuti da dužina sabnet maske u okviru ćelije nije inicijalno definisana nego se formira po potrebi tokom upisa u lukap tabelu. To znači da kada se ukaže potreba za upisom novog zapisa, a ne postoji mogućnost upisa u neku od ćelija sa već formiranom sabnet maskom (ako ih uopšte ima), tada se jednoj od praznih ćelija zadaje da definiše svoju sabnet masku i izvrši upis. Nakon formiranja sabnet maske, svaki novi zapis te sabnet maske biće upisivan u tu ćeliju do popunjenja FIFO memorije.

B. Funkcija pretrage

Kada stignu podaci na ulaz lukap tabele oni će se prosleđivati dalje po već objašnjenom principu i svaka ćelija će samostalno započeti sa kvazi-paralelnom pretragom svoje FIFO memorije. Kada ćelija završi pretragu, poslaće svoje rezultate susednim ćelijama, koje će zatim to proslediti dalje do kontrolne logike lukap tabele. Nakon slanja rezultata ćelija će ući u stanje čekanja da ceo sistem završi. Poslati rezultati se sastoje od signala koji označava da li je bilo poklapanja, od dužine sabnet maske ćelije, rezultujućeg izlaznog porta, kao i od identifikacije te ćelije.

Kada kontrolna logika lukap tabele dobije rezultate od svih ćelija, tada na osnovu najduže sabnet maske uspešnih poklapanja, tj. najpreciznije rute određuje rezultujući izlazni port i prosleđuje ga na izlaz. Nakon toga kontrolna logika lukap tabele signalizira ćelijama da je sistem završio i da je lukap tabela spremna za novi zadatak.

Broj potrebnih taktova za pretragu zavisi od popunjenosti memorija pojedinih ćelija. Ako uzmemo najgori slučaj, koji podrazumeva popunjenost svih ćelija dubine FIFO memorije *K* tada dobijamo:

$$T_s = N+M+K \qquad (3.1.)$$

Formula pokazuje da se kombinovanjem ova tri parametra može postići željeni rezultat.

C. Funkcija upisivanja u Lukap tabelu

Kada na ulaz stignu podaci za upis novog zapisa, prva ćelija koja prima podatke proverava popunjenost svoje FIFO memorije, kao i dužinu svoje sabnet maske. Ako se sabnet maska poklapa i ima još mesta u njenoj FIFO memoriji ta ćelija signalizira sledećim ćelijama da se kandiduje za upis i prosleđuje ulazne podatke. U slučaju da sabnet maska nije bila ni formirana, što znači da je FIFO memorija prazna, ćelija se takođe kandiduje za upis, ali i dodatno signalizira da je njena kandidatura proistekla iz razloga što je prazna tj. da nema formiranu sabnet masku. U svim ostalim slučajevima ćelija će samo proslediti dalje podatke bez kandidature. Sve ostale ćelije kada prime podatak provere da li se neka ćelija pre njih kandidovala za upis. U slučaju da nije, pokrenuće proces provere kao i prva ćelija, dok će u suprotnom slučaju samo proslediti podatke dalje. Potrebno je naglasiti da se sve ovo realizuje u trajanju od jednog takta po ćeliji.

Kada stignu podaci sa svih kolona u kontrolnu logiku lukap tabele, povratnom informacijom se signalizira jednoj od kandidovanih ćelija da upiše adresu. Ciljna ćelija se bira na osnovu kriterijuma da se prvo popuni ćelija koja već ima formiranu sabnet masku, a u slučaju da takva ne postoji, dozvoljava se jednoj ćeliji da formira svoju sabnet masku i upiše novi zapis. Nakon toga signalizira se ćelijama kao i spoljnim sistemima spremnost za novi zadatak.

Ovakav algoritam pokazuje veliku prednost po broju taktova potrebnih za realizaciju upisa. Broj potrebnih taktova je fiksan i on iznosi:

$$T_A = N + M \qquad (3.2.)$$

Gde je *N* broj ćelija u jednoj koloni a *M* broj ćelija u jednom redu.

D. Funkcije izmene i brisanja

Kada stigne zahtev na ulaz Lukap tabele on će se prosleđivati dalje u cilju kvazi-paralelne pretrage po već objašnjenom principu. Svaka ćelija upoređuje svoju sabnet masku i sabnet masku zahteva i u slučaju da su iste započinje pretragu svoje FIFO memorije. Kada završi pretragu, ćelija šalje svoje rezultate susednim ćelijama, koje će to proslediti do kontrolne logike lukap tabele. Nakon slanja rezultata ćelija ulazi u stanje čekanja ostatka sistema. Rezultati se sastoje od identifikacije te ćelije, kao i od signala koji označava uspešno poklapanje pa u isto vreme signalizira i da je zahtev izvršen.

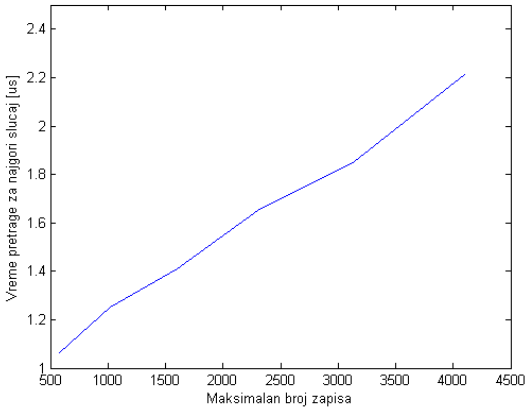
Kada kontrolna logika lukap tabele dobije rezultate od svih ćelija, tada u slučaju da joj je stigao signal za uspešno poklapanje i izvršenje zahteva signalizira uspešnost i obaveštava ćelije da je sistem spreman za novi zadatak.

Broj potrebnih taktova za pretragu poklapanja zavisi od popunjenosti memorija pojedinih ćelija, kao i od njihove sabnet maske. Pošto se ni u najgorem slučaju unapred ne znaju sabnet maske ćelija nije moguće izvesti opštu formulu, ali se može zaključiti da taj broj može biti samo manji ili teoretski jednak broju taktova potrebnih za funkciju pretrage.

IV. REZULTATI SIMULACIJA

Za potrebe testiranja i simulaciju dizajna, korišćen je Alterin FPGA čip iz porodice *Cyclone II* (konkretno EP2C70F896C6) a kao softversko okruženje *Quartus II Web edition* verzija 8.0. Simulacija je rađena sa postepenim povećavanjem broja ćelija pa samim tim i

maksimalnim brojem zapisa. Svaka ćelija je imala istu dubinu FIFO memorije koja iznosi 64. Praćeni su ključni parametri: maksimalna učestanost takta (koja određuje brzinu rada) i zauzeće čipa (određuje maksimalnu veličinu lukap tabele).



Slika 4.2. – Grafik zavisnosti maksimalnog vremena pretrage od maksimalnog broja zapisa

Slika 4.2. pokazuje zavisnost vremena pretrage za najgori slučaj. U situaciji popunjenja čipa, što odgovara 4096 zapisa, vreme potrebno da se pretraži cela lukap tabela iznosi 2.21 µs. U cilju smanjenja potrebnog vremena može se koristiti jedan od raspoloživih bržih čipova.

Tabela 4.1 – Brzina pretrage različitih lukap algoritama

Lukap algoritmi	**Brzina pretrage**
Binarna trie Struktura	8.92 µs
Pretraga po dužini prefiksa	7.08 µs
M-arne trie strukture	2.99 µs
LC trie	1.98 µs
Ćelijska struktura	2.2 µs

U tabeli 4.1 dato je poređenje različitih lukap algoritama u funkciji vremena potrebnog za pretraživanje lukap tabele. Prilikom formiranja tabele, vreme potrebno za pretragu dobijeno je analizom najgoreg slučaja, prikazanom u radu [3]. Iz tabele 4.1 se može primetiti da jedino LC trie algoritam može brže obavljati pretragu svoje lukap tabele, ali treba napomenuti da mane ovog algoritma opisane u II delu ga čine neupotrebljivim za rutere kod kojih je *update* proces učestan. Takođe, treba napomenuti da algoritam opisan u ovom radu je implementriran na Cyclone II čipu. Isti algoritam implementiran na novijem čipu, Stratix III familije, bi dao vreme potrebno za brzinu pretrage od samo 1.7 µs, što je bolje za nekih 0.3 µs u poređenju sa LC trie algoritmom.

Maksimalan broj zapisa, koji može da podrži *Cyclone II* čip koji smo i koristili, iznosi 4096 što je situacija sa 8x8 ćelija dubine FIFO memorije 64. U slučaju potrebe za većim brojem zapisa može se koristiti ili veći čip ili više ovakvih čipova.

V. ZAKLJUČAK

U okviru ovog rada prikazali smo jednu hardversku realizaciju lukap funkcije na Alterinom FPGA čipu *Cyclone II* familije. Iako ovaj čip po performansama spada u skromnije, načelne prednosti hardverske realizacije lukap algoritama su vidljive. Prednost ove implementacije se ogleda u tome što se kompletna realizacija lukap funkcije nalazi na jednom čipu, čime je ukinuta potreba da se svaki put pristupa eksternoj memoriji kako bi se pribavio podatak iz lukap tabele. Ovaj algoritam ne radi kompresiju zapisa, što značajno pojednostavljuje *update* proceduru, koja zna da bude jako česta na *backbone* ruterima. Nezavisnost rada algoritma od broja bita adresnog prefiksa takođe iskazuje još jednu značajnu prednost, koja omogućava skalabilnost prikazanaog algoritma i njegovo jednostavno proširenje na IPv6 adrese.

Kako je u ovom radu prikazana samo preliminarna verzija ovog novog algoritma ostavljeno je još dosta mesta za dodatnu optimizaciju. Među najaktuelnijim idejama za unapređenje ovog algoritma jeste dodavanja osobine da svaka ćelija nakon završetka svoje pretrage i slanja rezultata susednim ćelijama, ne mora čekati na završetak celog sistema, već odmah može preuzeti sledeći zadatak i momentalno krenuti u njegovo izvršavanje. Ovo unapređenje će smanjiti vreme pretrage sukcesivnih dolazećih zahteva (smanjenjem vremena čekanja sledećeg zahteva koji stiže na obradu).

LITERATURA

[1] M. Petrović, A. Smiljanić, *Programiranje Alterinih FPGA čipova*, Akademska Misao, 2008.

[2] www.ietf.org

[3] M. Á. Ruiz-Sánchez, E. W. Biersack, W. Dabbous "*Survey and Taxonomy of IP Address Lookup Algorithms"*, Network IEEE, 8-23 vol.15, Mar/Apr 2001.

[4] S. Nilsson, G. Karlsson "*IP-Address Lookup Using LC-Tries*", IEEE J. Selected Areas Comm. 17 (6), 1999.

[5] M. Waldvogel., G. Varghese, J. Turner, B. Plattner, "*Scalable high speed IP routing table lookups*" in *Proc. ACM SIGCOMM '97*, Sept

[6] T. Hayashi, T. Miyazaki "*High-speed Table Lookup Engine for IPv6 Longest Prefix Match*", GLOBECOM '99., 1576-1581 vol.2, Rio de Janeireo, Brazil

ABSTRACT

With the rapid increase of medium capacities and the continuous growth of internet traffic, the need for faster and more efficient routers that would perform very fast IP packet forwarding is higher than ever. One of the key roles in the performance of routers is the lookup function, which performs a search, based on destination IP address, of lookup table in order to determine the outgoing port of the router where IP packet should be forwarded to. The lookup function has become a potential bottleneck for high performance routers. Therefore, many lookup algorithms have been developed to make the IP router performance more efficient. In this paper we propose a hardware based solution for the lookup function implemented on an Altera FPGA chip.

FPGA IMPLEMENTATION OF LOOKUP ALGORITHM USING A CELL STRUCTURE

Jordan B. Jovanović, Ognjen M. Vuković