Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38
DOI: https://doi.org/10.24867/30BE12Pejic

RAZVOJ I PARALELIZACIJA ŠAHOVSKE MAŠINE SA IMPLEMENTACIJOM SERVERA I ANDROID APLIKACIJE

CHESS MACHINE DEVELOPMENT AND PARALLELIZATION WITH SERVER IMPLEMENTATION AND ANDROID APPLICATION

Dejan Pejić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu prikazan je razvoj jednostavne šahovske mašine. Opisan je Minimax algoritam za pretragu stabla svih po-teza. Izvršena je implementacija serijske i paralelne verzije algoritma s ciljem poboljšanja mašine. Mašina je integrisana u server, dok je pored toga razvijena i korisnička Android aplikacija.*

Ključne reči: *Šahovska mašina, Minimax algoritam, Paralelno programiranje, OpenMP, Klijent-server, Pthreads, Android*

Abstract – *This paper presents the development of a simple chess machine. The Minimax algorithm for searching the tree of all propositions is described. Serial and parallel versions of the algorithm were implemented with the aim of improving the machine. The machine is integrated into the server, while a user-friendly Android application has also been developed.*

Keywords: *Chess engine, Minimax algorithm, Parallel programming, OpenMP, Client-server, Pthreads, Android*

1. UVOD

Šahovske mašine su sofisticirani kompjuterski programi dizajnirani za analizu i procenu šahovskih pozicija, obezbeđujući optimalne poteze kroz napredne algoritme. Jedan od najčešće korišćenih algoritama za pretragu je Minimaks. Minimaks algoritam je fundamentalni metod u teoriji igara koji se koristi za donošenje optimalnih odluka prolaskom kroz sve moguće poteze u igri. Sa velikim dobinama pretrage, koje su poželjne radi pronalaženja boljih poteza, dolazi do potrebe za proračunom velikog stabla potencijalnih poteza, što algoritam čini računski intenzivnim. Iz tog razloga neophodne su neke optimizacije ove pretrage radi rešavanja ogromne složenosti, a jedna od mogućnosti je paralelizacija pretrage. Paralelizacija omogućava značajna poboljšanja performansi raspodelom računarskog opterećenja na više procesora

Šahovske aplikacije, koje integrišu ove mašine, omogućavaju korisnicima da igraju, analiziraju i proučavaju partije. Ove aplikacije često imaju grafičke interfejse omogućavajući korisnicima da komuniciraju sa mašinom, analiziraju različite pozicije i pregledaju igre sa uvidima iz analize mašine.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vuk Vranjković, vanr. prof.

Moderne šahovske mašine uključuju veštačku inteligenciju i tehnike dubokog učenja kako bi dodatno poboljšali svoje performanske. Ideja ovog rada nije da se poredi sa njima, nego da se predstavi osnovni koncept razvoja i uticaj paralelizacije na performanse i snagu mašine.

U drugom poglavlju rada biće ukratko opisani osnovni koncepti u razvoju šahovske mašine sa posebnim fokusom na Minimaks algoritam i njegovu Negamaks verziju. U trećem poglavlju opisuje se paralelizacija Negamaks verzije algoritma korišćenjem OpenMP biblioteke za paralelno programiranje. U četvrtom poglavlju biće opisan razvoj servera uz pomoć Pthreads biblioteke i njemu prilagođene Android aplikacije. Na kraju, u petom poglavlju, biće predstavljeni rezultati celokupnog rada i diskusija o drugim načinima unapređenja mašine.

2. RAZVOJ MAŠINE I MINIMAX ALGORITAM

2.1. Osnovni koncepti mašine

Prilikom razvoja šahovske mašine osnova je implementacija šahovske table i figura. Šahovska tabla je reprezentovana pomoću niza kojim se modeluju 64 polja table. Slično tome i svaka figurica je predstavljena posebnim brojem, čime se omogućava njihovo razlikovanje. Nakon toga je omogućeno generisanje svih poteza, koji nije lagan proces usled raznolokih pravila kretanja figurica, a pored toga potrebno je obratiti pažnju i na specijalne poteze poput rokada, an pasana i promocije. Potez se čini 24-bitna vrednost koja predstavlja bitsko polje sa opisom raznih segmenata poteza. Što se tiče odigravanja poteza, moralo se osigurati da je svaki od njih legalan u toku partije, te omogućiti i detektovanje eventualnog ponavljanja poteza korišćenjem Zobristovog heširanja. Kako bi mašina pratila današnje standarde, implementiran je i FEN koncept.

FEN je skraćenica od Forsajt-Edvardsonove notacije i to je standardna notacija za opisivanje pozicija u šahovskoj igri. FEN je važan jer olakšava prevođenje bilo koje šahovske pozicije u jedan red teksta. Olakšava proces ponvaljanja pozicije pomoću računara i omogućava igračima da pokreću partije iz željene pozicije [1].

Pored konzolne aplikacije koja je razvijena za testiranje mašine, omogućeno je da mašina bude kompatibilna i sa UCI protokolom (engl. *Universal Chess Interface*), tako da se mašina može učitati u već razvijene šahovske aplikacije sa korisničkim interfejsom koje koriste ovaj standard.

2.1. Minimaks algoritam

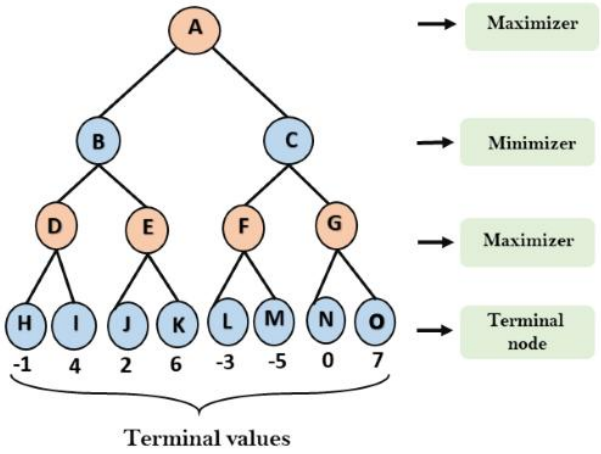
Algoritmi pretraživanja imaju tendenciju da koriste uzročno-posledični koncept – pretraga razmatra svaku moguću akciju koja joj je dostupna u datom trenutku; zatim razmatra naknadne poteze iz svakog od tih stanja, i tako redom pokušavajući da pronađe terminalna stanja koja zadovoljavaju ciljne uslove koji su joj dati. Po pronalasku terminalnog stanja ona sledi korake za koje zna da su neophodni za postizanje tog stanja.

Međutim, u takmičarskoj igri za više igrača, kada su uključeni drugi činioci koji imaju različite ciljeve na umu (obično one koji se direktno suprotstavljaju jedni drugima), stvari postaju komplikovanije. Čak i ako algoritam pretrage može da pronađe terminalno stanje, on obično ne može jednostavno da preduzme niz akcija koje će dostići to stanje, jer za svaku akciju koju algoritam preduzme ka svom cilju, protivnički igrač može preduzeti akciju koja će promeniti trenutno stanje (i to verovatno na način koji je nepovoljan za algoritam pretrage). To ne znači da je pretraga beskorisna u pronalaženju strategija za igre sa više igrača, već jednostavno zahteva dodatnu taktiku kako bi bila efektivna.

Za igre sa dva igrača, minimaks algoritam koristi činjenicu da dva igrača rade na suprotnim ciljevima i na taj način pravi pretpostavku o tome koja će buduća stanja biti dostignuta sa napretkom igre, a zatim nastavlja u skladu sa optimizovanjem svoje šanse za pobedu. Teorija iza minimaks algoritma je da njegov protivnik (engl. *minimizer*) pokušava da minimizira bilo koju vrednost koju algoritam (engl. *maximizer*) pokušava da maksimizira (shodno tome algoritam se i naziva „minimaks“). Dakle, program treba da napravi potez koji ostavlja svog protivnika u stanju da napravi najmanju štetu.

U idealnom slučaju, program istražuje svako moguće terminalno stanje igre počevši iz trenutnog stanja (kao i putanje koje je potrebno da dođe do bilo kog od njih) i svakom dodeljuje vrednosti (engl. *terminal values*). Za igre na poraz ili pobedu, kakva je i šah, postoje samo tri moguće vrednosti: pobeda, poraz ili nerešeno (kojima se često dodeljuju numeričke vrijednosti 1, -1 i 0 redom). Onda, počevši sa dna, program proračunava koji mogući ishod je najbolji za njegovog protivnika. Zatim pretpostavlja da će protvnik, ako se dostigne to stanje igre, napraviti potez koji je najbolji za njega (protivnika), a najgori za algoritam. Tako može da pretpostavi šta će njegov protivnik uraditi i da ima konkretnu predstavu o tome kakvo će biti konačno stanje igre ako se ta pretposlednja pozicija zaista i dostigne. Zatim, program može da tretira tu poziciju kao terminalnu, iako ona u suštini to nije. Ovaj postupak se može ponoviti na višem nivou, i tim redom do korena stabla. Na kraju, svakoj opciji koju algoritam trenutno ima na raspolaganju može biti dodeljena vrednost, kao da je terminalno stanje, a program jednostavno bira najvišu vrednost i preduzima tu akciju, odnosno odigrava taj potez [2].

Na slici 1 dat je primer pretrage putem minimaks algoritma. Crveni čvorovi maksimiziraju rezultat, dok je plavima čvorovima cilj da ga minimiziraju. Kada se porede svi terminali čvorovi, vidi se da je najveća vrednost 7, koja se može dostići ukoliko bi se crveni igrač pri prvom biranju odlučio za čvor C. Međutim, u tom slučaju bi se prepustio riziku da pri svom biranju plavi igrač izabere čvor F. U tom slučaju, crveni bi na kraju mogao da dođe do vrijednosti -3 ili -5, koje su nepovoljnije u odnosu na najgori izbor iz druge grupe. Zato će po minimaks algoritmu, crveni igrač izabrati čvor B, što bi ga kasnije u najgorem slučaju dovelo da bira između -1 i 4. Ako protivnik ne odigra kako se očekuje, proračun se može ponovo pokrenuti, pri čemu će rezultat biti dobar kao što je i predviđeno (ili čak i bolji).



Slika 1. *Primer pretrage po Minimaks algoritmu [3]*

Negamaks algoritam je pojednostavljena i elegantnija verzija mimaks algoritma koja koristi prednosti simetrije između maksimiziranja i minimiziranja vednosti u igrama sa dva igrača, kao što je šah. Funkcioniše pod pretpostavkom da u igri važi koncept nulte sume, odnosno da je dobitak jednog igrača gubitak drugog. Samim tim rezultat jednog igrača je negacija drugog igrača. U programu se poziva rekurzivno.

Ova verzija algoritma je iskorišćena za pretragu u šahovskoj mašini iz rada. Pošto se ne primenjuju druge optimizacije, algoritam pronalazi poteze za pristojno vreme pri najvećoj dubini pretrage jednakoj 5.

```
int negaMax( int depth ) {
    if ( depth == 0 ) return evaluate();
    int max = -oo;
    for ( all moves)  {
        score = -negaMax( depth - 1 );
        if( score > max )
            max = score;
    }
    return max;
}
```

Slika 2. *Pseudokod rekurzivnog Negamaks algoritma*

3. PARALELIZACIJA NEGAMAKS ALGORITMA

Dobra šahovska mašina ne treba samo da pronađe najbolji potez, već je poželjno da to uradi za što kraće vreme. Za dubinu pretrage jednakoj 5 prilikom testiranja rada mašine, u zavisnosti od pozicije mašini je bilo potrebno 2 – 18 sekundi da pronađe najbolji potez. To je suviše vremena, pa se iz tog razloga pretraga ograničila na konačno trajanje od 4 sekunde. Za to vreme se pri

komplikovanim pozicijama ne može pronaći adekvatan potez, pa je iz tog razloga potrebno ubrzati pretragu kako bi mašina održala svoj potencijal. S obzirom na stabloliku strukturu Negamaks algoritma, jasno je da je on odličan kandidat za paralelizaciju.

3.1. Implementacija paralelnog Negamaks algoritma upotrebom OpenMP biblioteke

Koren stabla pretrage je početna pozicija pretrage i iz te pozicije se vrši paralelizacija. Mnogi programeri posvećuju posebnu funkciju za pretraživanje korena [4], što je i ovde slučaj. Za koren pretrage implementirana je funkcija *rootNegaMax*, unutar koje se paralelno pozivaju rekurzivne funkcije *negaMax*. Pri korenu stabla formira se lista svih mogućih poteza, koji se potom do svojih terminalnih čvorova analiziraju u različitim nitima.

Paralelizacija je postignuta upotrebom modela deljene memorije pomoću OpenMP biblioteke i generalno se implementira dodavanjem *pragma* direktiva u kodu. Za paralelizaciju Negamaks pretrage iskorićena je direktiva *pragma omp parallel num_threads(tc) reduction(+: nodes) reduction(||: playableMove)*, gde parametar *tc* predstavlja broj niti koje koriste za izvršavanje pretrage, a redukcije služe da kombinuju rezultate iz više niti u jedinstvenu vrednost. Promenljiva *nodes* predstava broj posećenih terminalnih čvorova i zahteva se od strane UCI protokola, dok je *playableMoves* promenljiva tipa *bool* i daje informaciju o tome da li je tokom pretrage pronađen bar jedan legalan potez.

Prilikom paralelizacije, radi poboljšavanja performansi programa za svako jezgro napravljene su lokalne kopije podataka neophodnih za realizaciju igre i pretrage. Međutim, jedan podatak ostaje deljen, a to su informacije o najboljem potezu i njegovoj terminalnoj vrednosti. Istovremeni pristup više niti mogao bi dovesti do greške sa generisanjem poteza, pa je zato pristup ovom podatku stavljen unutar *#pragma omp critical* direktive, koja obezbeđuje da samo jedna nit izvršava određeni deo koda u isto vreme.

3.2. Rezultati poređenja serijske i paralelizovane implementacije Negamaks algoritma

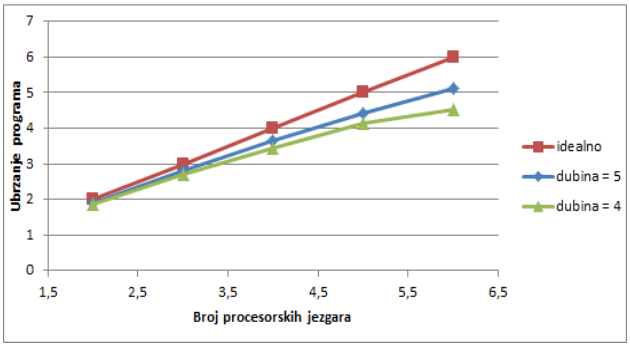
Za merenje vremena izvršavanja programa paralelizovane verzije programa korišćena je funkcija *omp_get_wtime* iz OpenMP bilioteke. To vreme se poredi sa vremenom izvršavanja serijske verzije, čime se dobija odnos koji predstavlja ubrzanje programa dobijeno paralelizacijom.

Po prirodi algoritma i načina implementacije algoritma bolji rezultati paralelizacije se očekuju pri većim dubinama pretrage. Ipak, bez drugih optimizacija, pretraga stabla već pri dubini 6 postaje neprihvatljiva za jedan šahovski program, a za veoma male dubine se dobijaju lošiji potezi. Zbog toga se kao optimalno pokazalo testiranje programa za dubine 4 i 5.

Brzina pretrage se značajno razlikuje od pozicije do pozicije. U situacijama kada je algoritam u šah–poziciji on uglavnom nema mnogo poteza na raspolaganju, pa je pri tome i serijska verzija optimalna i izvršava se u trajanju do 1 sekunde. Isto važi i za završnicu igre, kada je sa malim brojem figurica trajanje pretrage reda stotina, pa čak i desetina milisekundi. Zato je fokus na početku i sredini igre gde je trajanje dosta duže. Testiranje je izvršeno tako što je analiziran veliki broj različitih pozicija na određenom broju procesora. Za svaku poziciju se nalazi njeno ubrzanje, a potom je izračunato i srednje ubrzanje za odgovarajući broj procesora. U tabeli 1 i na slici 3 prikazani su dobijeni rezultati.

Tabela 1. *Rezultati paralelizacije algoritma*

Broj procesora	2	3	4	5	6
Ubrzanje (dubina = 4)	1,88	2,70	3,45	4,13	4,53
Ubrzanje (dubina = 5)	1,92	2,83	3,66	4,42	5,11



Slika 3. Prikaz prosečnog ubrzanja programa

5. SERVER I ANDROID APLIKACIJA

Šahovska mašina je prvobitno implementirana kao konzolna aplikacija, pri čemu je fokus bio na osnovnoj funkcionalnosti i performansama algoriritma pretrage Ipak, za potpuno korisničko iskustvo, važno je integrisati grafički korisnički interfejs (GUI). Zato je izabrana realizacija klijent-server arhitekture, trenutno u lokalnoj varijanti. Arhitektura zasnovana na serveru predstavlja dobar izbor za šahovsku mašinu, jer se klijentima omogućava daljinski pristup. S druge strane, Android aplikacija je dobra klijentska platforma zbog široke upotrebe mobilnih uređaja sa Android operativnim sistemom i svim njihovim mogućnostima.

Slika 4. *Prijava na server*

Slika 5. *Početni meni*

U server je integrisana razvijena šahovska mašina, koja samim tim čini uslugu kojima će pristupati klijenti. Server je realizovan pomoću Pthreads biblioteke kojom je omogućeno da se za svakog klijenta kreira posebna nit unutar koje se putem poruka vrši interakcija sa klijentom. Klijentska aplikacija realizovana je unutar Android

studija, pisana Java programskim jezikom i XML-ом за GUI.

Po otvaranju aplikacije od korisnika se očekuje da se poveže i unosom imena prijavi na server, pri čemu se sa serverske strane za njega kreira posebna nit. Njemu se tada otvara glavni meni pri čemu može da bira između nekoliko opcija:

- igra protiv servera, odnosno realizovane šahovske mašine nazvane Ždralin,
- igra protiv nekog drugog dostupnog igrača i
- analiza pozicije na osnovu unesenog FEN-a.

Slika 6. *Biranje figurica*

Slika 7. *Tok partije*

Pri izboru igre protiv servera korisnik može da bira boju svojih figurica. To nije moguće kada želi igrati protiv drugog korisnika, već je aplikacija implementirana tako da bele figurice ima onaj koji pošalje zahtev za igru. Zahtev se može poslati aktivnom korisniku koji u tom momentu nije u nekoj drugoj partiji ili analizi pozicije.

Tokom igre korisnik ima mogućnost da predajom prekine partiju. Dodatno, prilikom igre između dva igrača, može se ponuditi i remi, koji mora biti prihvaćen od strane drugog korisnika.

Kada dođe do šah pozicije polje napadnutog kralja bude obojeno crvenom, a ukoliko prilikom tome nastupi i mat, polje pobedničkog kralja bude obojeno žutom bojom.

Slika 8. *Kraj partije*

Slika 9. *Analiza pozicije*

Usled remija polja oba kralja budu obojena sivom bojom. Bez obzira na ishod igre, korisnik na kraju može ponuditi revanš. U revanš partiji korisnici igraju sa figuricama druge boje.

Za analizu pozicije dovoljan je samo unog pozicije u FEN formatu. Ukoliko je FEN ispravan, server analizira datu poziciju i odgovara najboljim potezom koje se označi i na samoj šahovskoj tabli.

5. ZAKLJUČAK

U ovom radu opisan je razvoj šahovske mašine i njoj prilagođene aplikacije. Pored implementacije logike igre, najvažniji segment je pretraga za najboljim potezom, što je istovremeno i vremenski najkritičniji postupak usled velikog stabla mogućih poteza.

Za pretagu stabla iskorišćena je Negamaks verzija Minimaks algoritma, koji je fundamentalni algoritam za ovu svrhu. Sa malo većim dubinama pretrage, rad algoritma počinje da traje nezahvalno dugo, što uslovljava poimenu neke optimizacije. Zato je pomoću OpenMP biblioteke primenjena paralelizacija korišćenog Negamaks algoritma u cilju dobijanja snažnije mašine, koja će za kraće vreme doći do isto kvalitetnih poteza.

Paralelizacija se pokazala uspešno sa zadovoljavajućim i korisnim ubrzanjem. Takođe, njeni rezultati pokazuju da je ubranje bliže idealnom sa većom dubinom pretrage. Ipak, za veće dubine mašina postaje nedovoljno efikasna i za serijsku i za paralelnu realizaciju. Kako bi se to poboljšalo, morale bi se primeniti i neke druge optimizacije kao što su Alfa-beta pretraga, upotreba transpozicionih tabela, primena iterativnog produbljivanje (engl. *iterative deepening*) i slično. Od njih je najznačanijia Alfa-beta pretraga koja podrazumeva smanjivanje broja terminalnih čvorova tako što se eleminišu grane, za koje se zna da ne mogu uticati na konačnu odluku.

Što se tiče Android aplikacije, pored dosadašnjeg ishoda i tu postoje velike mogućnosti poboljšanja: implementacija baze podataka, omogućavanje registracije i rangiranja igrača, postavljanje aplikacije na *Google* prodavnicu kako bi bila na raspolaganju svim korisnima itd. Uz to, ovaj klijent-server sistem bi se mogao proširiti na sledeći nivo konfigurisanjem servera da bude globalno vidljiv, kako bi mu pristupili korisnici sa raznih lokacija.

4. LITERATURA

[1] https://www.chess.com/terms/fen-chess (pristupljeno u septembru 2024)
[2] https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html (pristupljeno u septembru 2024)
[3] https://www.javatpoint.com/mini-max-algorithm-in-ai (pristupljeno u septembru 2024)
[4] https://www.chessprogramming.org/Root (pristupljeno u septembru 2024)

Kratka biografija:

Dejan Pejić rođen je u Bijeljini 2000. god. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Embeded sistemi i algoritmi odbranio je 2023.god. kontakt: sarajevskapejic@gmail.com