Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti:	Olivera Radovanović SV46/2021 Miloš Bojanić SV9/2021 Marko Mitošević SV56/2021
Predmet:	Paralelno Programiranje
Tema projektnog zadatka:	Analiza serijskog i paralelnog algoritma za AI bees bota za igru sa hackathona

Sadržaj

1. Uvod

Razvoj paralelizacije ima dugu istoriju koja seže unazad decenijama. Sa napretkom tehnologije i sve većim brojem višejezgarnih procesora, paralelizacija je postala ključna za postizanje veće efikasnosti i performansi u izvršavanju računarskih zadataka. Tradicionalno, programiranje je bilo usmereno na sekventno izvršavanje instrukcija, ali su se paralelni modeli programiranja razvili kako bi iskoristili resurse modernih računara. Paralelizacija ima široku primenu u industriji, posebno u oblastima gde je obrada velikih količina podataka ili brza reakcija od ključne važnosti.

Jedna od oblasti u kojoj je paralelizacija široko rasprostranjena jeste industrija video igara. Paralelizacija se koristi za istovremeno izvršavanje različitih aspekata igre, kao što su fizika, veštačka inteligencija protivnika i prikazivanje grafike u realnom vremenu.

Algoritam za izračunavanje optimalnog poteza bota je dobar primer mogućnosti koje pruža paralelizacija stoga smo se i odlučili da paralizujemo bota kojeg smo napravili za svrhu hackathona. Već na samom događaju smo uvideli koliko vremenski zahtevan može biti serijski algoritam, što nam je pravilo veliki problem u testiranju ispravnosti algoritma. Budući da minimax algoritam u sebi sadrži veliki stepen grananja, on je veoma pogodan za paralelizaciju.

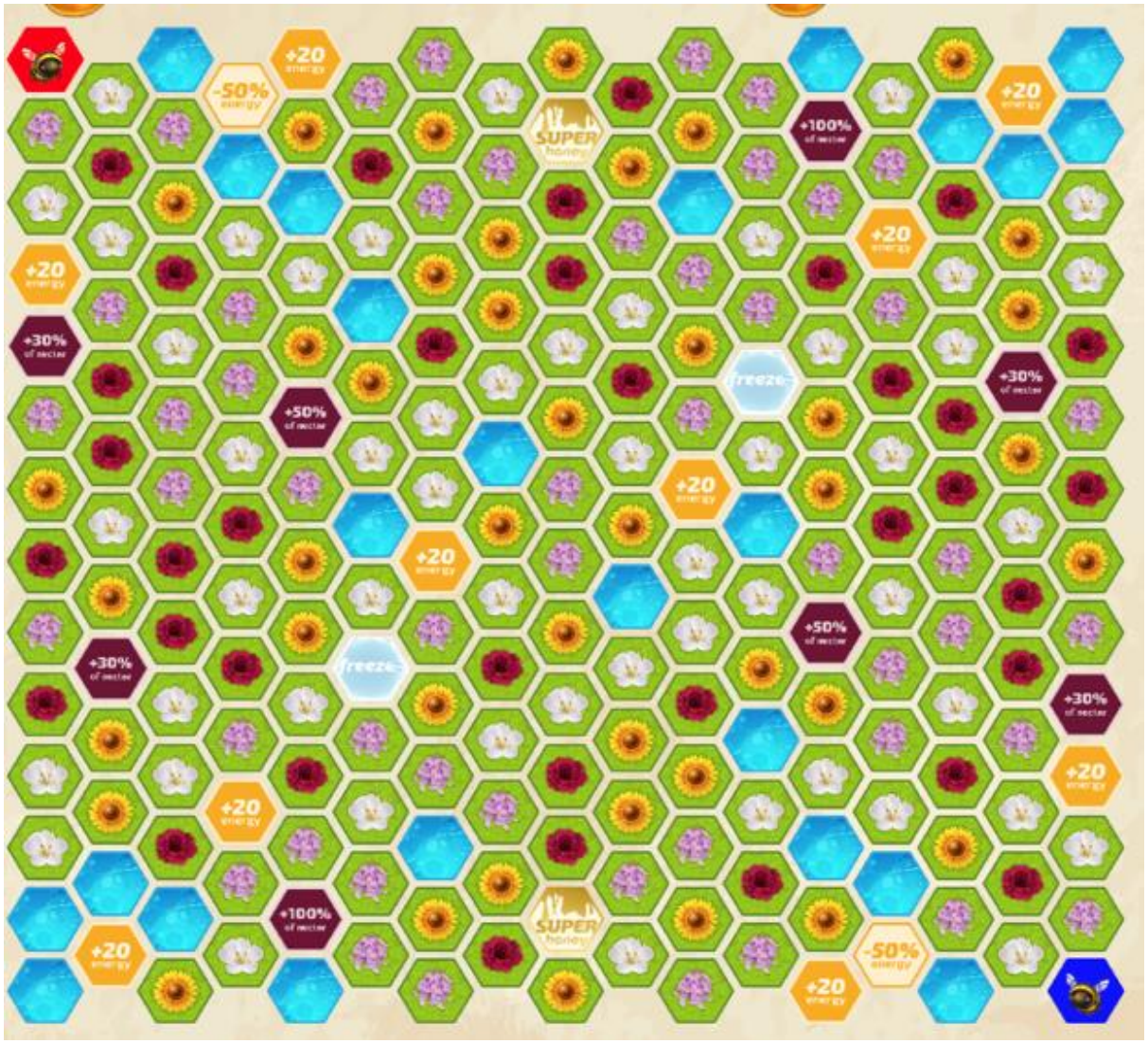
U ovom tekstu analiziraćemo performanse serijske implementacije ovog algoritma u programskom jeziku C++ i porediti je sa paralelnom implementacijom istog algoritma. Za paralelizaciju zadatka koristićemo apstrakciju zadatka (*tasks*), obezbeđenu od strane Intel-ove Threading Building Blocks biblioteke. Sva merenja vršićemo na Intel i5-8300h četvorojezgarnom (sa osam logičkih jezgara) procesoru i Windows 11 operativnom sistemu, koristeći verziju 11.2 g++ kompajlera. Takođe, za pokretanje same igre i komunikacije sa aplikacijom korišćene su pomoćne biblioteke CURL i json.hpp.

2. AI Bees igra

AI Bees igra je zasnovana na takmičenju između dva igrača, od kojih svaki kontroliše jednu pčelu, na mapi sastavljenoj od šestougaonih polja. Cilj svakog igrača je da skupi što više poena, koje može dobiti na razne načine.

2.1. Mapa

U gornjem levom ćošku se nalazi košnica prvog igrača, dok se košnice drugog igrača nalazi u suprotnom uglu, donjem desnom i ova polja predstavljaju početnu poziciju pčela kao i mesto na kojem pčele mogu da pretvore skupljeni nektar u med. Osim košnica na mapi se nalaze razna druga polja od kojih svaki ima svoju ulogu. Na slici 2.1. možete videti izgled mape.



Slika 2.1. Mapa igre

Polja se dele na:

- Cveće – postoje četiri vrste cveta, gde svaki daje određenu količinu nektara. Kada pčela jednom pokupi cveće, ono nestaje sa mape i polje postaje prazno i igrač dobija broj poena jednak broju nektara koji taj cvet donosi, ukoliko igrač kupljenjem cveta ne prelazi 100 nektara. U suprotnom igrač dobija onoliko bodova koliko nektara može da dobije a da pritom ne pređe 100. Na slici 2.2. su prikazane sve vrste cveća sa odgovarajućim brojem nektara koje pčela dobije kada pređe preko njih.

Slika 2.2. Polja cveća sa odgovarajućim brojem poena

- Boostere – postoje četiri vrste boostera, gde svaki daje svoj specijalni efekat i donosi određen broj bodova. Prvi booster, sa slike 2.3. pčeli koja ga pokupi automatski dodeljuje 20 energije i donosi igraču onoliko poena koliko je energije dopunio kupljenjem ovog boostera. Ostala booster polja proporcionalno povećavaju nektar koji pčela poseduje i donose igraču onoliko poena koliko nektara su dobili kupljenjem ovog boostera. Na slici 2.3. su prikazana booster polja.

Slika 2.3. Booster polja sa odgovarajućim brojem poena

- Specijalna polja – postoje šest vrsti specijalnih polja. Košnica (**HIVE**) je polje na kom pčela može da pretvori nektar koji poseduje u med, gde se od 20 nektara može napraviti jedan med. Za svaki napravljen med igrač osvaja 30 poena. Pčela ne može da uđe u protivničku košnicu. Prazno polje (**EMPTY**) nema nikakav efekat i nastaje nakon što je na tom polju pokupljeno cveće, booster ili druga specijalna polja osim košnice i jezera.

Jezero (**POND**) je polje na koje ako stane pčela ona odmah umire i igrač automatski gubi. Na slici 2.4. su prikazane prve tri vrste specijalnih polja.

Slika 2.4. Specijalna polja

Poslednja tri polja imaju sličnu funkcionalnost kao booster polja, s obzirom da kada ih pčela jednom pokupi i aktivira njihov efekat ona nestaju i umesto njih se stvara prazno polje. **FREEZE** polje zamrzava protivničku pčelu na tri poteza i dodeljuje igraču koji ga je pokupio 100 poena. **MINUS_FIFTY_PCT_ENERGY** polje oduzima polovinu energije koje protivnik trenutno poseduje i dodelju igraču koji ga je pookupio 50 poena. **SUPER HONEY** polje dodeljuje 5 meda igraču, čime igrač dobija 150 poena. Na slici 2.5. su prikazane poslednje tri vrste specijalnih polja.

Slika 2.5. Specijalna polja

2.2. Ograničenja

- Igrač u svakom trenutku može da ima između 0 i 100 energije.
- Igrač ne može da ima više od 100 nektara.
- Igrač ne može da preskoči potez više puta od dozvoljenog broja (150).

2.3. Tok igre

Igra se odvija tako što igrači naizmenično igraju poteze dok jedan od uslova za kraj igre nije ispunjen. Igrač može da odigra jedan od četiri poteza:

- **MOVE** – Igrač se pomera u jednom od šest pravaca za određen broj polja i usput kupi sve sa polja kroz koje prolazi i na kojem završi. Za svako polje koje igrač pređe gubi dve energije.
- **CONVERT TO HONEY** – Igrač pretvara određenu količinu nektara u med, gde za svakih 20 nektara pčela napravi jedan med. Za svaki napravljen med igrač dobija 30 poena. Ovaj potez igrač može da odigra samo u svojoj košnici.
- **FEED WITH NECTAR -** Igrač hrani svoju pčelu određenom količinom nektara, gde za svaki nektar pčela dobija dve energije. Ovaj potez igrač može da odigra samo u svojoj košnici.
- **SKIP A TURN** – Igrač preskače svoj potez i dobija 5 energije.

2.4. Kraj igre

Igra se završava ukoliko je ispunjen jedan ili više od navedenih uslova:

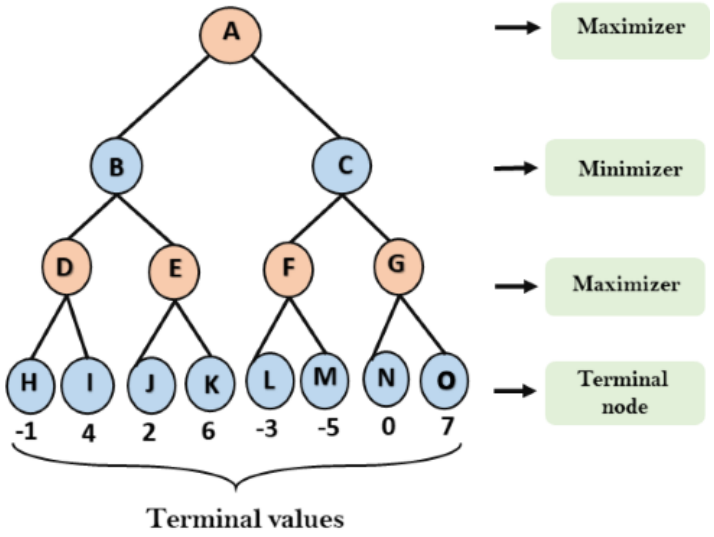
- Dostignut je maksimalan broj poteza (500). U tom slučaju pobeđuje igrač sa više poena.
- Na mapi više nema cveća. U tom slučaju pobeđuje igrač sa više poena.
- Ako igrač prekorači maksimalan broj SkipATurn akcija (150), pobeđuje protivnik.
- Ako igrač stane na polje tipa **POND** gubi igru.
- Ako igrač dođe do 0 energije. U ovom slučaju pobeđuje protivnik.

3. Osnovne karakteristike serijskog algoritma

Problem koji je bio zadat na hackathonu je pisanje algoritma koji će igrati optimalne poteze umesto jednog od igrača. S obzirom da se prethodno predstavljena igra može svesti na igru nulte sume (*zero-sum*) - one u kojima je dobitak jednog igrača tačno odgovara gubitku drugog igrača, jedan od pogodnih algoritama za rešenje ovog zadatka, koji smo i mi odlučili da koristimo, jeste minimax algoritam.

3.1. Minimax algoritam

Minimax algoritam je algoritam koji kroz rekurziju simulira sve moguće poteze za određeni broj narednih poteza i pomoću heuristika određuje koji od prethodno simuliranih poteza je optimalan za trenutnu poziciju. Na slici 3.1. je grafički prikazan ovaj algoritam.



Slika 3.1. Stablo minimax algoritma

U minimax algoritmu igrač za koga se računa potez se zove minimizer, dok se protivnik naziva maximizer. Cilj minimizera je da dobije najmanju moguću brojevnu vrednost - koja predstavlja najbolji mogući potez za našeg igrača. Cilj maximizera je da dobije najveću moguću brojevnu vrednost - koja predstavlja najbolji mogući potez za protivnika. Svaki čvor u stablu predstavlja jedan odigran potez, gde deca čvora igraju potez na osnovu stanja igre u roditeljskom čvoru. Kada se dosegne maksimalna dubina radi se evaluacija zatečenog stanja igre. Zatim se, u zavisnosti čiji je potez bio u roditeljskom čvoru, bira najbolji ili najgori rezultat, kao i potez kojim se došlo do izabranog stanja i prosleđuje roditeljskom čvoru. Ovaj proces se nastavlja dok se ne stigne do korenog čvora čime je osigurano da je od svih mogućih poteza izabran optimalni.

3.2. Heuristika

Cilj heuristike je da se brzo dođe do rešenja koje je dovoljno dobro za problem koji se rešava. To rešenje ne mora biti nužno najbolje, ili čak može biti samo aproksimacija tačnog rešenja. I pored toga takvo rešenje je vredno zato što za njegovo nalaženje nije potrošeno mnogo vremena.

U kontekstu igara nulte sume, heuristika se koristi da bi se izračunalo koliko je trenutno stanje povoljno ili nepovoljno za datog igrača. Neke od mogućih heuristika uključuju broj poena igrača, broj figura, trenutna pozicija na tabli itd. Zbirom svih heuristika dobijamo konačno rešenje koje dalje prosleđujemo minimax algoritmu.

Ponašanje bota je onoliko dobro koliko su dobro podešene heuristike. Dobra postava heuristika može biti vrlo težak posao, s obzirom da tokom igranja poteza čovek većinu svog proces razmišljanja čini podsvesno. Zbog toga je bitno da se bot marljivo testira sa različitim vrednostima heuristika dok se ne dođe do zadovoljavajućeg rešenja.

4. Paralelni algoritma

Budući da minimax algoritam proizvodi grananje, prirodna prva ideja je da se svaka grana pokrene u posebnoj niti. Međutim, budući da broj grana može biti za više stepena veći od broj procesorskih jezgara u jednom računaru, pogotovo sa većom dubinom minimax algoritma, vrlo brzo može do *stack overflow*-a. Takođe, u slučajevima u kojima ne dođe do greške, program u proseku radi mnogo duže, zbog velikog broja niti koje zavise od procesa drugih niti da bi se izvršile. Da bi se izbegao ovaj problem, kao i zadržali pozitivni aspekti paralelizacije, dovoljno je paralelizovati početnu dubinu minimax algoritma.

Što se računanja heuristika tiče, pri pokušaju paralelizacije naleteli smo na sličan problem kao i pri paralelizaciji samog minimax algoritma, zbog velikog broja evaluacija koje treba da se izvrše u isto vreme. Ukoliko bi sama igra bila jednostavnija i imala manje mogućih poteza uporedno bi i proces evaluacije bio jednostavniji, pa bi paralelizacija evaluacije mogla biti validna opcija.

5. Implementacija algoritma

5.1. Implementacija serijskog algoritma

Pre nego što proba da se pomeri sa trenutnog polja, pčela prvo proveri da li se nalazi u svojoj košnici. Ukoliko ima određen broj nektara, proveri da li joj je ponestalo energije. Ukoliko jeste, prvo pretvori određen broj nektara u energiju. U suprotnom, pretvori maksimalnu moguću količinu nektara u med.

Budući da se pčela može kretati u jednom od 6 smerova po heksagonalnoj mapi, sam smer kretanja nam predstavlja prvo račvanje u minimax algoritmu. U funkciji FindBestMove se preko for petlje razmatraju svi pravci u kojima pčela može da se pomera, a zatim se za svaku stranu proverava za koliko polja bi bilo optimalno da se pčela pomeri.

Ovaj poces se ponavlja dok se ne dođe do zadate dubine gde se trenutna pozicija pčele obračunava uz pomoć raznih heuristika i svakoj poziciji se dodeljuje neka brojevna vrednost. Minimax algortimom, kao što je već navedeno, optimalna vrednost se propagira nazad kroz rekurziju i najbolji potez se odigra tako što se prosledi API-u pomoću post request-a.

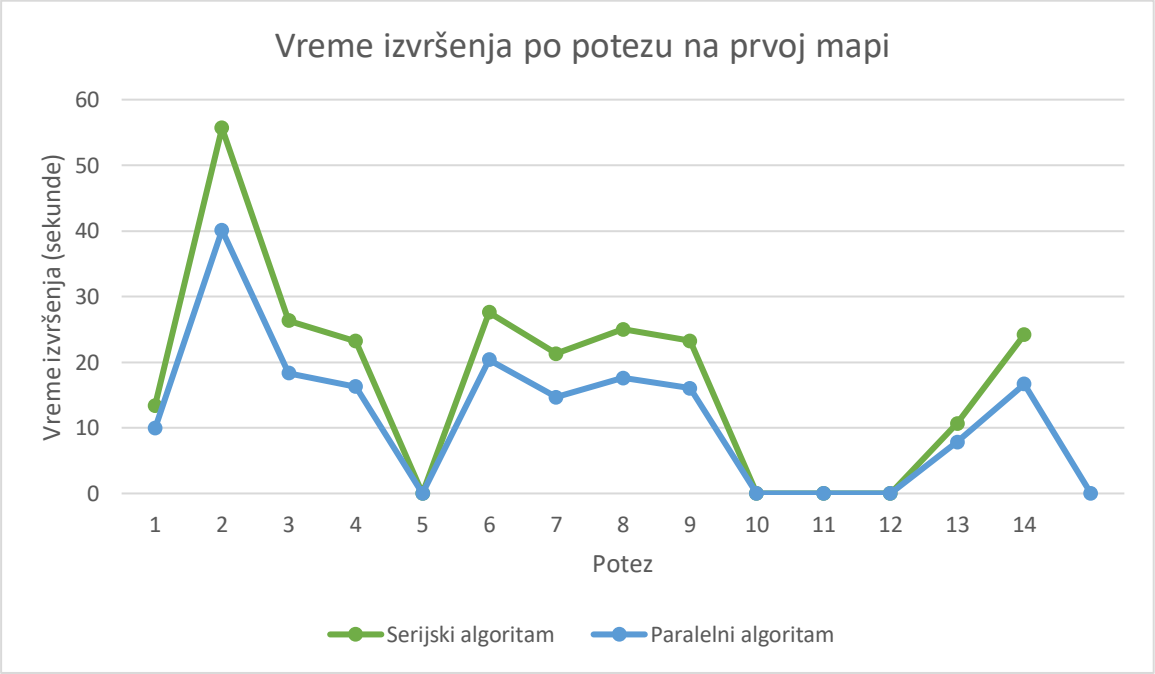
5.2. Implementacija paralelnog algoritma

Upotrebom Intelove Threading Building Blocks biblioteke, konkretno raspoređivača poslova *task group* možemo da paralelno računamo optimalan broj koraka za svaku stranu kojom pčela može da ide. Naime, iteracija prethodno navedene for petlje iz serijskog algoritma je izvučena u zasebnu funkciju koju pokrećemo korišćenjem raspoređivača posla. Na ovaj način smo raspon algoritma minimaxa smanjili na jednu šestinu u poređenu sa serijiskom verzijom.

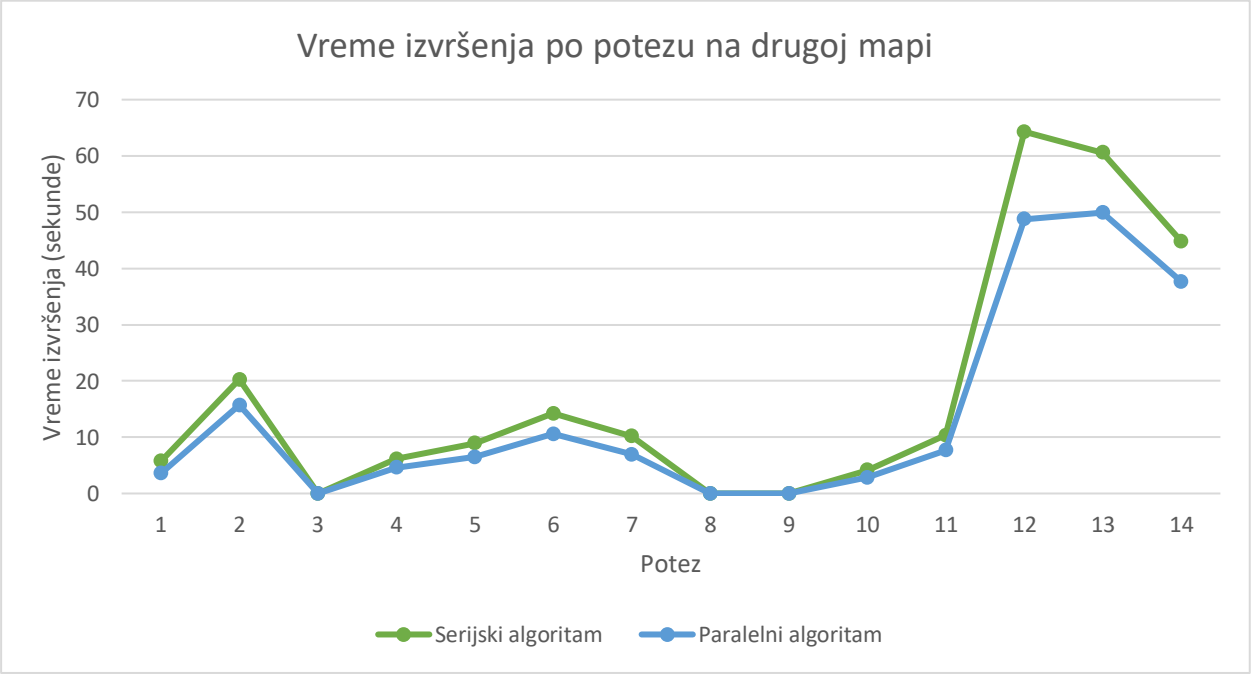
U praksi, budući da zbog prirode mape za neke strane treba mnogo više vremena za izračunavanje od drugih, neke niti će mnogo brže završiti svoj proces. Usled ove činjenice, ubrzanje na prosečnom računaru će biti poprilično manje, iako i dalje i te kako primetno u poređenju sa serijskim algoritmom.

6. Analiza performansi

Naši algoritmi su testirani na dve mape koje su nam bile ponuđene sa različitim nivoom paralelizacije. Na svakoj mapi je testirano vreme izvršavanja poteza serijskog, a zatim i paralelnog algoritma. Na sledećim slikama je pomoću grafikona vizuelno prikazana razlika u prosečnom vremenu izvršavanja spomenutih algoritama.



Slika 6.1. Grafikon prosečnog vremena izvršenja serijskog i paralelnog algoritma na prvoj mapi



Slika 6.2. Grafikon prosečnog vremena izvršenja serijskog i paralelnog algoritma na drugoj mapi

Iz prethodno prikazanih grafikona može se primetiti da paralelni algoritam u proseku dostiže ubrzanje od preko 20% prilikom izračunavanja najboljeg poteza. Takođe, potezi koji se izvrašavaju u zanemarljivom vremenu predstavljaju poteze ConvertToHoney i FeedWithNectar za koje nije potrebno da se pokreće minimax algoritam za izračunavanje najbolje pozicije na koju treba pčela da se pomeri, jer se provera za ove dve funkcije izršava pre poziva minimax algoritma.

Testiranjanjem na prvoj mapi dobijeno je da paralaleni algoritam radi za 29% brže od serijskog, dok ubrzanje na drugoj mapi iznosilo 23%. Svi navedeni rezultati su postignuti paralelizovanjem prve dubine minimaxa, dok se paralelizacijom evaluacije i ostalih dubina minimaxa dobijaju rezultati gori od serijskog za približno 30%. Rezultati su pokazali da paralelizacija donosi značajne prednosti u pogledu efikasnosti i brzine izvršavanja.

7. Zaključak

Paralelizacija je ključna u industriji jer omogućava efikasno korišćenje resursa i poboljšava performanse računarskih sistema. Kroz paralelno izvršavanje zadatka, maksimalno iskorištavamo raspoloživa procesorske jezgra i logičke procesore. To dovodi do smanjenja vremena čekanja, ubrzanja izvršavanja i poboljšane skalabilnosti.

Uz pravilno planiranje, sinhronizaciju i upravljanje resursima, paralelizacija pruža mogućnost da se izvuče maksimum iz modernih računara i postigne visoka efikasnost u izvršavanju zahtevnih zadataka. Ovaj proces može biti izuzetno zahtevan, budući da podrazumeva široko znanje o arhitekturi računara i duboko razumevanje problema. Često, kao što je viđeno i u ovom algoritmu, može doći do preterane paralelizacije koja je kontraproduktivna i izaziva usporenje programa. Usled navedenih poteškoća, treba izbagavati preteranu paralelizaciju, pogotovo u procesima koji su izuzetno isprepletani sa drugim procesima, jer u protivnom se može izazvati dugo čekanje na završenje procesa ili u najgorem slučaju međusobno blokiranje.

Paralelizacija je ključna za postizanje bolje efikasnosti i performansi u računarstvu. Implementacija paralelizacije Minimax algoritma u C++ jeziku uz pomoć Intelovog alata Threading Building Blocks pokazala je značajno ubrzanje u donošenju odluka i boljem korišćenju resursa. Ova tehnika ostaje važan alat za postizanje visoke efikasnosti u industriji i omogućava nam da iskoristimo pun potencijal modernih računarskih sistema.