Jedno rešenje programske podrške za razvoj konkurentnih algoritama i programa

Nikola Vranić, Branislav Atlagić, Pavle Savković

***Sadržaj* — U radu je predstavljena programska podrška za razvoj konkurentnih algoritama i programa. Opisana su 3 konkurentna algoritma koji su nastali na osnovu ove programske podrške.**

***Ključne reči* — Programske niti, konkurentni algoritmi.**

I. UVOD

Još od predstavljanja objektno orijentisanog programiranja nije bilo većih koraka ka novoj revoluciji u oblasti programiranja. Stalna potreba za sve bržim izvršavanjem programa, dovela je do povećanja radne frekvencije procesora i razvijanja raznih tehnika optimizacije instrukcija. Ovim se izvukao maksimum iz procesora sa jednim jezgrom. Poboljšanje tehnoloških procesa proizvodnje procesora i smanjivanje veličine tranzistora obezbeđuje brže izvršavanje programa, ali je skupo i pitanje je koliko će još moći da se smanjuje veličina tranzistora bez ugrožavanja njihove funkcionalnosti. Drugi smer razvoja je povećanje broja jezgara na samom procesoru. Ova tehnika zahteva novi pristup u programiranju, konkurentno programiranje. Da bi se uspešno iskoristio veći broj jezgara, zadatak se deli u logičke celine koje se izvršavaju unutar programskih niti (u daljem tekstu nit).

Za razliku od pisanja programa sa jednom niti, konkurentno programiranje deli program na logički nezavisne celine. Trenutno postoje dve tehnologije na osnovu kojih se razvija konkurentno programiranje: hyperthreading i multicore.

Hyperthreding tehnologija omogućava emulaciju više jezgara na jednom procesoru. Prednost hyperthreading tehnologije je što dozvoljava izvršavanje više instrukcija istovremeno i poseduje više skupova registara. Za ovu tehnologiju je karakateristično da ubrzava program u proseku od 5% do 15%, a u nekim slučajevima i do 40% [1]. Da bi se iskoristile pogodonosti ove tehnologije programi se moraju pažljivo razložiti na celine uz vođenje računa o komunikaciji između celina.

Multicore tehnologija, za razliku od hyperthreading tehnilogije, izvršava niti na više jezgara unutar jednog procesora. Ako se program razloži na celine moguće je povećati brzinu izvršavanja programa približno onoliko puta koliko postoji jezgara [2].

Obe tehnologije se odnose samo na programe koji su pisani na konkurentan način. Ni jedna ne garantuje ubrzanje programa koji su pisani u jednoj niti. Zbog toga je potrebno pisati konkurente programe za procesore sa jednim i više jezgara. Ovo predstavlja kvalitetni skok u tehnici programiranja.

Prilikom pisanja konkurentnih programa potrebno je promeniti način razmišljanja i rešavati probleme koji nisu bili prisutni tokom pisanja programa u jednoj niti. Da bi se olakšalo pisanje konkurentnih programa razvijaju se razne programske podrške. Jedna od njih je objašnjena u ovom radu. Cilj ove programske podrške jeste da olakša posao programeru i ubrza pisanje konkurentnih algoritama i programa.

II. OPIS PROGRAMSKE PODRŠKE

Analizom konkurentnih algoritama izdvojila se jedna velika klasa problema sa vrlo specifičnim karakteristikama. U toj klasi postoji skup ulaznih podataka nad kojima treba da se izvrši određeni algoritam više puta. Podatke je moguće izdeliti na nezavisne celine, tako da se nad svakom celinom izvršava algoritam u isto vreme. Pošto je izlaz iz algoritma ujedno i ulaz, obradu je potrebno izvršavati u iteracijama. Svakoj nezavisnoj celini podataka dodeljena je posebna nit u kojoj će se oni obrađivati. Ista obrada se izvršava nad svim podacima.

Problem koji se javlja u ovoj klasi konkurentnih algoritama je čekanje na kraj obrade svih podataka. Potrebno je biti siguran da su sve niti završile obradu i da program može dalje da nastavi rad. Zbog toga su razvijene klase za pomoć pri pisanju konkurentnih algoritama:

- osnovna klasa za rad sa nitima i
- izvedena klasa za rad sa nitima.

A. Osnovna klasa za rad sa nitima

Ideja je da se razvije okruženje koje će obezbediti programeru funkcije za pokretanje niti, za signalizaciju kraja niti i za podizanje i spuštanje barijere. Pre pokretanja niti potrebno je napraviti objekat osnovne klase. Tada se svi parametri, potrebni za rad, postavljaju na početne vrednost i podiže se barijera. Zatim se pokreću niti, prosleđuju im se podaci i čeka se na njihov kraj, tj. čeka se spuštanje barijere.

Da bi se pokrenula nit programer treba da pozove funkciju za pokretanje niti i da preko argumenata prenese adresu funkcije u kojoj je opisan algoritam za obradu podataka, kao i adresu na kojoj se nalaze podaci. U

Nikola Vranić, master studije, Nikola.Vranic@KRT.neobee.net
Branislav Atlagić, mentor, Branislav.Atlagic@KRT.neobee.net
Pavle Savković, mentor, Pavle.Savkovic@KRT.neobee.net

Fakultet tehničkih nauka u Novom Sadu, Fruškogorska 11, Srbija (telefon 381-21-4801100)

osnovnoj klasi postoji lista u koju se uvezuje novi objekat koji opisuje upravo pokrenutu nit. Postoji i brojač koji ukazuje na broj aktivnih niti. Brojač se povećava pokretanjem nove niti, a smanjuje signalizacijom za kraj izvršavanja niti. Kada nit završi obradu potrebno je da pozove funkciju za kraj izvršavanja niti. Signaliziranjem kraja izvršavanja poslednje niti spušta se barijera.

B. *Izvedena klasa za rad sa nitima*

Izvedena klasa za rad sa nitima nasleđuje osnovnu klasu za rad sa nitima i obezbeđuje funkcije za pokretanje niti, signalizaciju za kraj niti, funkciju za pokretanje algoritma, signalizaciju za kraj algoritma, kao i funkcije za podizanje i spuštanje barijere. Za razliku od osnovne klase, niti izvedene klase se ne gase odmah po završetku algoritma. Kada niti završe izvršavanje algoritama prelaze u stanje čekanja u kome čekaju nove podatke za obradu. Pošto je poznato da pokretanje niti traje dugo, ovaj vid optimizacije je veoma poželjan.

Izvedena klasa razlikuje dve vrste niti: niti koje čekaju i aktivne niti. Aktivne su one niti u kojima se trenutno izvršava algoritam, sve ostale niti čekaju na podatke da bi pokrenule svoj algoritam.

TABELA 1: POREĐENJE IZMEĐU OSNOVNE I IZVEDENE KLASE

Broj potrebnih niti	***Broj novih niti osnovne klase***	***Broj novih niti izvedene klase***	***Broj niti koje čekaju***
2	2	2	0
4	4	2	0
10	10	6	0
8	8	0	2
1	1	0	9
Ukupno niti	25	10	-

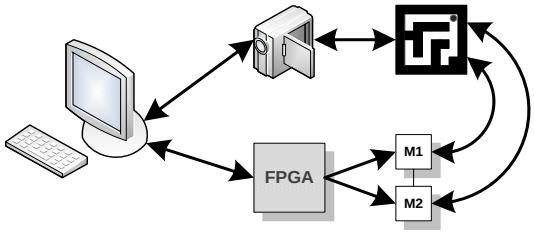
Iz tabele 1 jasno se može videti prednost izvedene klase. U prvoj iteraciji potrebne su dve niti da bi se obradili podaci. I jedna i druga klasa treba da pokrenu po dve niti. U drugoj iteraciji su potrebne 4 niti. Osnovna klasa, koja na kraju svake iteracije gasi niti, mora da pokrene 4 niti, dok izvedena klasa treba da pokrene samo 2 niti. U narednim iteracijama osnovna klasa mora da pokrene onoliko niti koliko je potrebno za obradu podataka, dok izvedena treba da pokrene samo onoliko koliko joj nedostaje do broja niti potrebnih za obradu podataka.

III. PRIMERI KONKURENTNIH ALGORITAMA

Konkurentni algoritmi koji će biti opisani u radu deo su jednog projekta za rešavanje lavirinta u realnom vremenu.

A. *Rešavanje lavirinta u realnom vremenu*

Sistem koji se posmatra se sastoji od : fizičkog modela lavirinta, kuglice, kamere, platforme za brz razvoj prototipova zasnovane na FPGA (Field-programmable gate array) integrisanom kolu i programske podrške na operativnom sistemu Windows [4], Sl. 1. Fizički model lavirinta se sastoji od ploče na kojoj se nalaze zidovi i polazi. Naginjanjem celog lavirinta uz pomoć servo motora omogućeno je pomeranje kuglice u željenom pravcu. Lavirint se snima odozgo kamerom koja je povezana na personalni računar USB(Universal Serial Bus) spregom. Celim sistemom upravlja PC podrška. Ona preko kamere dobija sliku lavirinta, a njim upravlja pomoću motora. FPGA integrisano kolo je iskorišćeno za prilagođavanje signala od PC-a ka servo motorima. Na ploči lavirinta postoji jedan otvor kroz koji kuglica može da prođe. Cilj je da kuglica pronađe put do izlaza i da prođe kroz njega.



Sl. 1. Izgled sistema

Lavirint se može podeliti na ćelije. Ćelije crne boje predstavljaju zid, a ćelije bele boje prolaze. Sve ćelije su kvadratnog oblika, stranice 22 mm. Ova fiksna dužina je odabrana da bi se kuglica, prečnika 21 mm, mogla kretati unutar lavirinta. Zidovi su visoki 15 mm.

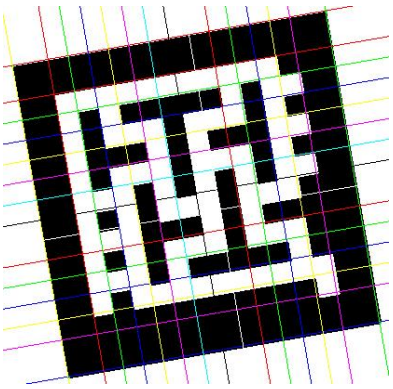
Da bi rešavanje moglo da se izvrši u realnom vremenu algoritmi su razvijeni tako da iskoriste sve pogodnosti konkurentnog izvršavanja. Realizovani su sledeći algoritmi:

- algoritam za otkrivanje lavirinta,
- algoritam za otkrivanje kuglice i
- algoritam za pronalaženje izlazne putanje.

B. *Algoritam za otkrivanje lavirinta*

Cilj algoritma za otkrivanje lavirinta je da otkrije zidove i prolaze lavirinta na slici lavirinta preuzete sa kamere. Ulaz u algoritam je dužina stranice ćelije u pikselima (pixels) i oblast u kojoj je otkriven lavirint. Izlaz iz algoritma je matrica čiji elementi opisuju zid ili prolaz lavirinta. Algoritam je realizovan primenom osnovne klase za rad sa nitima.

Na početku izvršavanja algoritma preračunava se broj redova i kolona ćelija u lavirintu. Zatim se stvara mreža za pretragu, Sl. 2. Mreža se sastoji od horizontalnih i vertikalnih pravi. Algoritam kao ulaz preuzima parametre dve susedne horizontalne prave, parametre svih vertikalnih pravih na slici i broj reda koga opisuju horizontalne prave. Od dve horizontalne i dve vertikalne prave formiraju se ćelije, a zatim se prebrojavaju crne piksele unutar njih. Na osnovu broja crnih tačaka u ćeliji se odlučuje da li se radi o zidu ili prolazu. Rezultat se upisuje u matricu zidova i prolaza.



Sl. 2. Mreža za pretragu

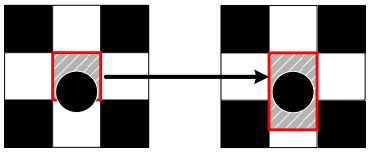
Kako je analiza ćelija unutar svakog reda nezavisna operacija, pokreće se onoliko niti koliko ima redova u lavirintu. Treba napomenuti da matrica zidova i prolaza nije deljena promenljiva. Svaka nit pristupa samo delu matrice koji je definisan ulaznim parametrima algoritma koju nit izvršava. Na ovaj način je obezbeđena potpuna istovremenost otkrivanja lavirinta.

C. Algoritam za otkrivanje kuglice

Algoritam za otkrivanje kuglice unutar lavirinta sličan je algoritmu za otkrivanje lavirinta. Pokreće se onoliko niti koliko ima redova u lavirintu. Umesto otkrivanja zidova i prolaza, algoritam traži kuglicu unutar prolaza lavirinta. Ulaz u algoritam je slika lavirinta i slika lavirinta i kuglice u njemu. Izlaz iz algoritma je pozicija kuglice u lavirintu.

Algoritam za otkrivanje kuglice na slici lavirinta i kuglice prolazi kroz ćelije, koje opisuju prolaze lavirinta, i broji koliko novih crnih tačaka ima u odnosu na broj tačaka u istim ćelijama sa slikom lavirinta. Ukoliko razlika prelazi određenu vrednost ta ćelija se označava kao ćelija u kojoj se možda nalazi kuglica. Da bi se potvrdila pozicija kuglice potrebno je otkriti crnu kružnu oblast unutar ćelije [5]. Kada se i taj kriterijum ispuni prijavljuje se pozicija kuglice.

Postoje slučajevi kada se kuglica ne nalazi u potpunosti u jednoj ćeliji, već delom prelazi u susednu ćeliju. Zato se prilikom otkriti crnog kružnog oblika oblast pretrage proširuje. Prvo se ispituje da li postoji prolaz gore, dole, levo i desno. Ukoliko postoji prolaz, pretraga se proširuje za polovinu dužine ćelije u određenom smeru, Sl. 3.



Sl. 3. Primer proširivanja oblasti pretrage na dole

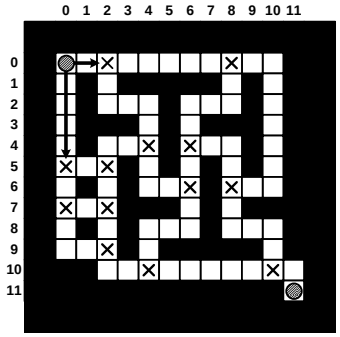
Kada se kuglica nalazi između dva reda ili dve kolone može se desiti da se otkriju dve pozicije kuglice. Ovaj problem je rešen tako što se prilikom prijavljivanje pozicije kuglice u lavirintu prijavljuje koliko crnih tačaka je otkriveno u kružnoj oblasti. Za najbolju poziciju se uzima ona u čijem opisu se nalazi više crnih tačaka. Konkurentan pristup promenljivoj koja sadrži poziciju kuglice ostvaren je pomoću semafora.

Algoritam za otkrivanje kuglice je u potpunosti paralelan osim kada se pronađu dve pozicije kuglice. Ovo ne predstavlja veliki problem, jer najviše dve niti mogu da se nadmeću za deljenu promenljivu.

D. Algoritam za pronalaženje izlazne putanje

Algoritam zadužen za pronalaženje izlazne putanje treba da otkrije najbolji put od trenutne pozicije kuglice u lavirintu do izlaza iz lavirinta. Ovaj algoritam je zasnovan na dvosmernim i višesmernim ćelijama.

Dvosmerna ćelija je ćelija lavirinta iz koje kuglica može da krene u najviše 2 smera. Višesmerna ćelija je ćelija lavirinta iz koje kuglica može da krene u više od 2 smera. Na Sl. 4 su krstićima označene višesmerne ćelije. Šrafiranim kružićem označene su početna pozicija (0,0) i izlaz iz lavirinta (11,11). Sve ostale ćelije su dvosmerne ćelije.



Sl. 4. Dvosmerne i višesmerne ćelije, početna pozicija i izlaz iz lavirinta.

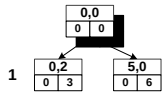
Algoritam pronalazi višermerne ćelije tako što se kreće po dvosmernim ćelijama. Pronalaženje izlazne putanje se vrši u iteracijama, te je zbog toga iskorišćena izvedena klasa za rad sa nitima. Nakon svake iteracije odabira se višesmerna ćelija na osnovu sledećih kriterijuma istog prioriteta:

1. Da li se višesmerna ćelija ponavlja u putanji
2. Da li postoji bolja putanja do višesmerne ćelije
3. Da li je pronađen izlaz iz lavirinta

Ukoliko višesmerna ćelija ne ispunjava ni jedan kriterijum nad njom se pokreće algoritam za pronalaženje nove višesmerne ćelije, tj. šalju se novi podaci prvoj slobodnoj niti na čekanju, odnosno pokreće se nova nit ukoliko nema niti na čekanju. Podaci se svakoj niti šalju preko poštanskog sandučeta. Svaka nit može da pristupi samo svom poštanskom sandučetu.

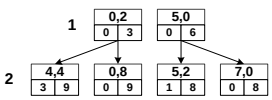
Putanje se porede po broju promena putanje i po broju ćelija u putanji. Broj promena putanje je jači kriterijum i on se prvo proverava, zatim se proverava broj ćelija u putanji. Putanja je bolja ako sadrži manje promena putanje ili ćelija. Sledi opis algoritma za pronalaženje izlazne putanje, na osnovu lavirinta sa Sl. 4.

Iz početne pozicije moguće je doći do višesmerne ćelije (0,2) i (5,0). Put od (0,0) do (0,2) sadrži 3 ćelije i 0 promena smera. Put od (0,0) do (5,0) sadrži 6 ćelija i 0 promena smera. Pozicija, broj ćelija i broj promene smerova se mogu predstaviti grafički. Crnom senkom se označava prva višesmerna ćelija. U pravougaoniku gore upisana je pozicija višesmerne ćelije. U levom kvadratiću dole upisan je broj promena smera, a u desnom broj pređenih ćelija na putanji. Sa leve strane upisan je redni broj iteracije, Sl. 5.



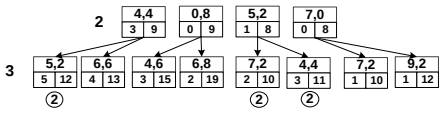
Sl. 5. Prva iteracija

Obe višesmerne ćelije prolaze odabiranje i nad njima se pokreće algoritam za pronalaženje višesmerne ćelije. Jedno pokretanje algoritma za pronalaženje višesmerne ćelije i jedno odabiranje zajedno čine iteraciju. Druga iteracija je predstavljena na Sl. 6.



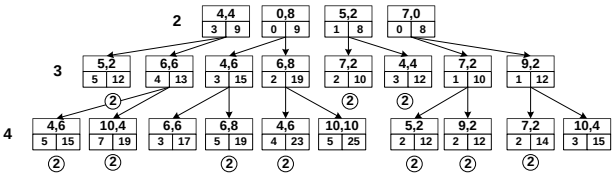
Sl. 6. Druga iteracija

Pronađene su 4 nove višesmerne ćelije. Sve 4 višesmerne ćelije prolaze odabiranje i nad sve 4 se pokreće algoritam za pronalaženje višesmerne ćelije.



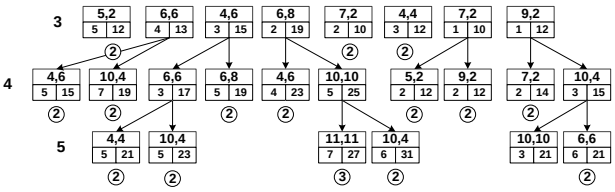
Sl. 7. Treća iteracija

U trećoj iteraciji je pronađeno 8 višesmernih ćelija. Od 8 višesmernih ćelija 5 prolaze odabiranje, a ostale ne. Tri ćelije koje ne prolaze odabiranje ispunjavaju kriterijum 2, postoji višesmerna ćelija sa boljom putanjom. Višesmerna ćelija sa pozicijom (5,2) je zaustavljena, jer postoji u drugoj iteraciji višesmerna ćelija sa boljom putanjom. Višesmerna ćelija u drugoj iteraciji u svojoj putanji ima manji broj promene putanje. Slično je i sa višesmernim ćelijama sa pozicijom (7,2) i (4,4).



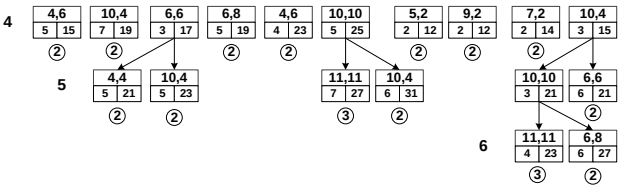
Sl. 8. Četvrta iteracija

U četvrtoj iteraciji, Sl. 8, pronađeno je 10 višesmernih ćelija. Od 10 višesmernih ćelija 3 prolaze odabiranje. Ostale zadovoljavaju kriterijum 2, postoji višesmerna ćelija sa boljom putanjom.



Sl. 9. Peta iteracija

U petoj iteraciji, Sl. 9, pronađeno je 6 višesmernih ćelija. Od 6 višesmernih ćelija samo jedna prolazi odabiranje. Višesmerna ćelija sa pozicijom (11,11) ispunjava kriterijum 3, pronađena je izlazna putanja.



Sl. 10. Šesta iteracija

U šestoj iteraciji, Sl. 10, pronađene su 2 višesmerne ćelije. Ni jedna višesmerna ćelija ne prolazi odabiranje. Višesmerna ćelija sa pozicijom 6,8 zadovoljava kriterijum 2, postoji višesmerna ćelija sa boljom putanjom. Višesmerna ćelija sa pozicijom (11,11) zadovoljava kriterijum 3, pronađena je izlazna putanja.

Pošto nema višesmerne ćelije nad kojom bi se dalje pokrenuo algoritam za otkrivanje višesmerne ćelije, algoritam za pronalaženje izlazne putanje se završava. Za konačnu najbolju izlaznu putanju uzima se ona koja je pronađena u šestoj iteraciji, 4 promene putanje i 23 ćelije.

IV. ISPITIVANJE

Svi algoritmi ispitani su automatski pomoću CPPUnit [5] okruženja za automatsko ispitivanje. Izvedeno je i ispitano preko 100 slučajeva za koje je utvrđeno da se mogu dogoditi. Sva ispitivanja su uspešno završena. Tokom 250 sati ispitivanja sistema u praksi nije zabeležena ni jedna greška, odnosno sistem je radio kao što je predviđeno. Da bi se izbeglo nesvesno iskorišćavanje osobina sistema od strane njegovog autora, ispitivanja su u praksi vođena od strane ispitivača koji nisu bili direktno uključeni u razvoj sistema, nasumičnim postavljanjem kuglice u lavirint.

V. ZAKLJUČAK

Uspešno je realizovana programska podrška za razvoj konkurentnih algoritama i programa. Takođe je uspešno razvijeno par algoritama u čijoj se osnovi nalazi opisana programska podrška.

Različiti operativni sistemi poseduju različite mehanizme pozivanja niti. Plan je da se programska podrška prilagodi svim poznatijim operativnim sistemima. Na taj način bi se dobila programska podrška koja ne zavisi od sistema na kome se izvršava.

Takođe, kako svi algoritmi koriste istu osnovu za paralelizaciju, programska podrška bi se mogla proširiti tako da se svaka nit izvršava na posebnom jezgru unutar procesorskih sistema sa više jezgara.

ZAHVALNICA

Ovaj rad je delimicno finansiran od Ministarstva za nauku Republike Srbije, projekat 12004, od 2009. god.

LITERATURA

[1] Herb Sutter, James Larus, Software and the Concurrency Revolution, Microsoft, electronic publication, http://portal.acm.org/citation.cfm?id=1095421
[2] Shameem Akhter, Jason Roberts, Multi-Core Programming, Intel Press, 2006.
[3] Joe Duffy, Concurrent Programing on Windows, Person Education, INC, Boston, 2008.
[4] Yali Amit, 2D Object Detection and Recognitin, The MIT Press, Cambrige Massacusetts, London, 2002.
[5] Mike Dickheiser, Game Programming Gems 6, Charles River Media, 2006.

ABSTRACT

The paper describes framework for development of concurrent algorithms and programs. It describes three concurrent algorithms that were made on top of this framework.

ONE FRAMEWORK SOLUTION FOR REALIZATION CONCURRENT ALGORITHMS AND PROGRAMS

Nikola Vranić
Branislav Atlagić
Pavle Savković