SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1993

Evolucijski algoritam za hibridnu paralelnu okolinu

Petar Čolić

Zagreb, srpanj 2011.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Centralni mikroprocesor slijedno prolazi kroz strojni jezik programa i izvršava naredbe. Kako nikada neće postojati dovoljno brz mikroprocesor, logično je više procesora udružiti da bi brže izveli željeni algoritam. Zbog toga je potrebno slijedne algoritme paralelizirati. Na raspolaganju je više tipova mikroprocesora specijaliziranih za različite probleme, poput centralnog ili grafičkog mikroprocesora. Ali pojavljuje se problem prenosivosti programa između različitih platformi zbog njihove vrlo različite arhitekture. Paralelizacija algoritama i njihova prenosivost zahtijeva drugačiji pristup oblikovanju algoritama. OpenCL je sustav koji olakšava paralelizaciju algoritama i prvi je koji omogućuje njihovu prenosivost između različitih platformi.

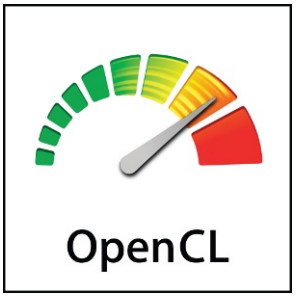
Paralelizaciju je pogodno koristiti na algoritmima optimizacije, naročito evolucijskim algoritmima. Evolucijski algoritmi uvijek rade sa većim brojem jedinki. Broj jedinki se zna kretati i do 100 000, pa je operacije nad njima poželjno paralelizirati.

U ovome radu opisan je standard OpenCL. Također su ukratko opisani evolucijski algoritami s naglaskom na genetsko programiranje. Zatim je prikazan vlastiti primjer implementacije genetskog programiranja, te paralelizacije putem OpenCL-a.

2. OpenCL

2.1. Općenito

OpenCL je skraćenica od *Open Computing Language*. Donedavno se za procesorski zahtjevne algoritme oslanjalo isključivo na centralne mikroprocesore (engl. *CPU*), a od 2007. je počelo iskorištavanje i specijaliziranih grafičkih mikroprocesora (engl. *GPU*) u općenitije svrhe. Zadnjih nekoliko godina koriste se višejezgreni CPU-ovi, a GPU-ovi su sami po sebi paralelizirane arhitekture. U oba slučaja se nameće potreba za paralelnim algoritmima. OpenCL usmjeren je upravo na paralelizaciju algoritama i heterogene platforme. Heterogenom platformom smatra se bilo kakav sustav s više različitih procesora (CPU, GPU, Cell, DSP. . . .). OpenCL je zamišljen kao API koji će iskorištavati sve dostupne resurse u takvom heterogenom sustavu.



Slika 2.1: OpenCL logo

Razvoj OpenCL-a je započela tvrtka Apple, a 2008. godine je upravljanje standardom prepušteno konzorciju Khronos koji okuplja sve relevantnije kompanije u području mikroprocesora i multimedije (Apple, AMD/ATI, IBM, Intel i Nvidia). 2009. godine su se pojavile prve implementacije, podrška za CPU i GPU u Mac OS X Snow Leopardu, Nvidia je podržala svoje GPU-ove kroz upravljačke programe (engl. *drivers*), a isto tako i AMD/ATI svoje CPU-ove i GPU-ove. Standard OpenCL usmjeren je na područje od ugradbene (engl. *embedded*) i potrošačke (engl. *consumer*) program-

ske opreme, pa sve do razine računarstva visokih performansi. Sustav je zadržan na niskoj razini koja je blizu samog sklopovlja, što omogućuje postizanje izrazito dobrih performansi. Sastoji se od sučelja za pristup sklopovlju, biblioteka i OpenCL C programskog jezika. Sustav je specifikacija otvorenog tipa, pa ga svaki proizvođač može implementirati. (1)

2.2. Arhitektura sustava OpenCL

Sustav se sastoji od hijerarhije 4 modela: platformskog, memorijskog, izvršnog i programskog.

2.2.1. Platformski model

Platformski model se sastoji od domaćina sa jednim ili više računalnih uređaja, npr. CPU i/ili GPU. Svaki računalni uređaj može imati više računalnih jedinica, npr. jezgra CPU-a ili multiprocesor GPU-a, a svaka računalna jedinica može imati više procesirajućih elemenata, poput procesora dretvi kod GPU-a.

2.2.2. Izvršni model

Izvođenje OpenCL aplikacije se odvija u dva dijela: programske jezgre (engl. *kernel*) se izvode na jednom ili više računalnih uređaja, a na domaćinu se izvodi program koji upravlja izvođenjem programskih jezgri, tj. brine se za stvaranje konteksta za izvođenje, stvaranje programskih slijedova i stvaranje memorijskih međuspremnika za pisanje i čitanje podataka iz memorije uređaja.

Pri pokretanju OpenCL jezgri se mora definirati (globalna) veličina problema, a za primjer možemo zamisliti broj redaka i stupaca matrice. Sustav zatim stvara indeksni prostor koji odgovara prethodno definiranim dimenzijama. Za svaki njegov element pokreće se po jedna radna jedinica, tj. OpenCL jezgra koja se izvodi nad elementom matrice. Radne jedinice su grupirane u radne grupe, a veličine radnih grupa se također mogu definirati pri pokretanju OpenCL jezgri kroz lokalne veličine problema. Navedeni princip omogućava bolju granulaciju problema.

Svaka radna jedinica se može jednoznačno identificirati u indeksnom prostoru pomoću globalnih indeksa (engl. *global ID*). Svaka radna grupa se također može identificirati pomoću indeksa grupa. Radne jedinice se također mogu identificirati i pomoću lokalnih indeksa i grupa kojima pripadaju, odnosno svaka radna jedinica ima i svoj

jednoznačni lokalni indeks u grupi u kojoj se nalazi.

2.2.3. Memorijski model

OpenCL poznaje 4 memorijska područja. Globalnoj memoriji mogu pristupati sve radne jedinice, a ona fizički odgovara radnoj memoriji CPU-a ili GPU-a. Konstantna memorija ostaje nepromijenjena za vrijeme izvođenja jezgri, a za nju se brine domaćin. Lokalna memorija je namijenjena radnim jedinicama unutar iste radne grupe, a njezina lokacija je ovisna o implementaciji; kod GPU-a je to priručna memorija mikroprocesora koja je znatno manja od radne memorije, ali joj je pristup daleko brži. Privatna memorija je namijenjena svakoj radnoj jedinici pojedinačno.

2.2.4. Programski model

Sustav OpenCL podržava dva programska modela: podatkovnu paralelizaciju (primarni) i paralelizaciju po poslovima. Do sada opisani model je podatkovna paralelizacija, a odvija se po principu SIMD, tj. ista operacija se izvršava nad različitim podacima. U modelu paralelizacije po poslovima programska jezgra se pokreće neovisno o indeksnom prostoru, a pretpostavljamo da jedna radna grupa sadrži samo jednu radnu jednicu. U ovom slučaju korisnik postiže paralelizam koristeći vektorske tipove podataka na OpenCL uređaju i stavljajući u red izvršavanje više poslova, te se brine oko sinkronizacije poslova. (2)

2.3. Komponente sustava

Sustav OpenCL omogućuje aplikacijama korištenje domaćina i dostupnih OpenCL uređaja kao jedno heterogeno paralelno računalo. Sustav se sastoji od sljedećih komponenata:

- OpenCL platformskog sloja: omogućuje programima na domaćinu otkrivanje OpenCL uređaja i njihovih karakteristika, te stvaranje OpenCL konteksta
- OpenCL izvršnog okruženja (engl. *runtime*): omogućuje programu na domaćinu upravljanje kontekstom
- OpenCL prevoditelja: služi za stvaranje programa koji sadrže programske jezgre koje se izvršavaju na OpenCL uređajima; tekst programa se piše u OpenCL C programskom jeziku

2.4. OpenCL C

OpenCL C je proširenje ISO C99 programskog jezika. ISO C99 nije samo proširen, već i neke njegove funkcionalnosti više nisu iskoristive, uglavnom zbog načina izvođenja OpenCL programskih jezgri. To su pokazivači na funkcije, mogućnost rekurzije, varijabilne duljine nizova, te neke druge specifičnosti. S druge strane, proširenja uključuju rad sa radnim jedinicama i grupama, vektorske tipove podataka, mogućnost sinkronizacije, te općenito neka proširenja za paralelizam, rad sa slikama, povezivanje sa OpenGL sustavom i sl. (2)

2.5. Arhitekture mikroprocesora

Za rad sa OpenCL-om poželjno je poznavati osnove rada mikroprocesora. Centralni mikroprocesor je nužan za izvršavanje operacijskog sustava i korisničkih aplikacija, ali u računalima se često nalaze i specijalizirani grafički mikroprocesori. OpenCL omogućava prenosivost istog teksta programa između CPU-a ili GPU-a, te je prvi takav sustav. Slika 2.2 (2) pokazuje strukturu CPU-a i GPU-a. Vidljivo je da je GPU više pogodan za paralelizaciju, te da veći udio tranzistora služi za procesiranje podataka.

Slika 2.2: Struktura CPU-a i GPU-a

2.5.1. Centralni mikroprocesor – CPU

Osnovni princip funkcioniranja CPU je uglavnom ostao isti do danas, a temelji se na Von Neumannovoj arhitekturi. Mikroprocesor dohvaća redom naredbu po naredbu iz memorije, izvodi je na aritmetičko-logičkoj jedinici, a dobiveni rezultat zapisuje nazad

na memoriju. Radi se o slijednom izvođenju naredbi, a za jednu naredbu je najčešće potrebno po nekoliko ciklusa mikroprocesora. Današnji CPU-ovi su izuzetno kompleksni kako bi se navedeni princip ubrzao. Neke od ugrađenih tehnologija su cjevovodi, superskalarnost, višedretvenost i velike količine priručne memorije. Zadnjih godina su se pojavili višejezgreni CPU-ovi, no teško to možemo smatrati "pravim" paralelizmom na koji programer može utjecati.

2.5.2. Grafički mikroprocesor – GPU

Grafički procesori su se pojavili dosta kasnije od CPU-a, tj. tek sredinom '90-ih godina. Kako su specijalizirani za prikaz 3D grafike i njihova arhitektura je u mnogočemu drugačija. Za razliku od CPU-a, GPU je već po arhitekturi paraleliziran, tj. sastoji se od puno (preko 100) procesora dretvi, te relativno male priručne memorije. Razlog je što se kod CPU-a puno tranzistora koristi za "logiku" procesora i priručnu memoriju, dok se kod GPU-a više tranzistora koristi za aritmetičko-logičke operacije. Zbog toga današnji GPU-ovi, sa jednakim brojem tranzistora kao i CPU-ovi, postižu daleko bolje performanse (za red veličine) u matematičkim kalkulacijama. Zadnjih nekoliko godina GPU se počeo koristiti i u općenitije svrhe (engl. *GP/GPU*) od prikaza 3D grafike, te ga danas kroz sustav poput OpenCL-a možemo koristiti kao svojevrstan matematički koprocesor CPU-a.

2.5.3. "Hibridne" i buduće arhitekture

Zadnjih godina su se pojavile, ali i počele najavljivati, arhitekture koje možemo nazvati "hibridima" između CPU-a i GPU-a. Za primjer možemo uzeti IBM-ov Cell procesor koji se sastoji od jezgre opće namjene na kojoj se izvršava operacijski sustav, te 8 specijaliziranih jezgri koje služe za matematičke kalkulacije. Jezgra opće namjene je u ulozi nadzornika nad preostalih 8 jezgri sa kojima je povezana kroz izrazito brzu sabirnicu. Intel je također najavio novu arhitekturu zvanu Larrabee koja bi se sastojala od većeg broja x86 jezgri.

Na temelju trenutne situacije dostupnih mikroprocesora možemo zaključiti kako se u budućnosti može očekivati veće iskorištavanje platformi koje se sastoje od neizbježnog mikroprocesora opće namjene za izvršavanje operacijskog sustava, te vrste specijaliziranog mikroprocesora koji bi služio ako dodatni akcelerator. Ništa ne isključuje i korištenje "hibridnog" mikroprocesora koji objedinjuje obje arhitekture kroz više različitih jezgri. (2)

2.6. JOCL - Java OpenCL

JOCL je skup Java biblioteka koje omogućuju korištenje OpenCL-a aplikacijama koje se izvode u JVM (engl. *Java Virtual Machine*). Sastoji se od dva dijela, biblioteka niske i visoke razine.

Biblioteke niske razine generirane su direktno iz Khronosovih OpenCL zaglavlja. Pružaju brzo, JNI (engl. *Java Native Interface*) bazirano, 1:1 mapiranje C funkcija. OpenCL biblioteke učitavaju se dinamički i pristupa im se preko pokazivača na funkcije.

Biblioteke visoke razine su "ručno" pisane koristeći biblioteke niske razine. Pojednostavljuju korištenje OpenCL-a uvođenjem složenijih naredbi odnosno složenijih Java razreda. Za brzi prijenos podataka između JVM i OpenCL teksta programa koriste se NIO bufferi. (3)

3. Evolucijski algoritmi

3.1. Općenito

Razvojem računarstva i povećanjem procesne moći računala, mnogi kompleksni problemi postali su rješivi. Rješavalo ih se uglavnom tehnikom grube sile (engl. *brute-force*), tj. pretraživanjem cjelokupnog prostora rješenja.

No inženjerski problemi današnjice su sve kompliciraniji i zahtjevniji. Rijetko su rješenja tih problema jednostavna i egzaktna. Na temeljima stohastike, svijet modernog računarstva se sve više okreće empirizmu, pa čak i u slučaju eksplicitno riješenih problema (najčešće zbog složenosti samih rješenja).

Evolucijski algoritmi su postupci optimiranja, učenja i modeliranja, koji se temelje na procesu evolucije u prirodi. Njihov nastanak uzrokovali su pokušaji primjene načela evolucije u prirodi pri rješavanju nekih problema. Temelj evolucijskih algoritama su pojave iz biološke evolucije – mutacija, rekombinacija, križanje, selekcija itd. Iako na prvi pogled ne postoji jasno vidljiva poveznica između spomenutih domena, nakon kraće analize ipak se dolazi do zaključka kako su principi evolucije – bolji opstaje, gori izumire – idealni za pronalaženje dovoljno dobrog rješenja zadanog problema.

Prve ideje o mogućoj primjeni ovakvih algoritama javljaju se još pedesetih godina 20. stoljeća, ali zbog skromnih mogućnosti tadašnjih računala ostaju nerealizirane i nepoznate široj znanstvenoj javnosti. Šezdesetih godina neovisno su razvijena tri postupka zasnovana na načelima evolucije u prirodi: evolucijsko programiranje (L. J. Fogel), evolucijske strategije (Rechenberg, Schwefel) i genetski algoritmi (J. H. Holland). Nešto kasnije počinje i razvoj genetskog programiranja (J. Koza). Evolucijski algoritmi spadaju u šire područje znanosti o spoznaji (engl. *cognitive science*), a uže u područje inteligentnih algoritama (engl. *computational intelligence*). (4)

3.2. Genetsko programiranje

Iako se teorijski razvijao paralelno sa ostalim evolucijskim metodama, koncept genetskog programiranja je svoju pravu vrijednost dosegao tek u devedesetima, kada ga je utvrdio i primjenio John R. Koza, doajen računarske znanosti sa sveučilišta Stanford. Otada se uočava još veća progresija u razvoju, kako u istraživačkim timovima, tako i kod samog Koze.

Postoje brojne sličnosti, ali i neke bitne razlike od genetskih algoritama. Temeljna je ideja ista: stvoriti neku populaciju početnih rješenja. Djelovanjem genetskih operatora i ocjenjivanjem pojedinih rješenja kroz određeni broj generacija pronalazi se optimalno rješenje.

3.2.1. Osnovni pojmovi i definicije

Genetsko programiranje je automatiziran optimizacijski postupak razvoja računalnih programa, čija je namjena rješavanje većinom složenih problema iz područja računarstva. Koncept je zasnovan na općim idejama iz teorije genetskih algoritama. Najjednostavnije rečeno, konačni cilj genetičkog programiranja je univerzalni računalni program koji nalazi optimalno rješenje određenog problema. (5)

3.2.2. Formalna predodžba računalnog programa

Bez obzira na činjenicu da se genetskom programiranju može pristupiti na više različitih načina i iz više različitih perspektiva, uglavnom se radi o univerzalnom pristupu računalnom programu kao formalnom stablu u kontekstu teorije grafova. Svaki računalni program može se prikazati kao stablo, gdje unutarnji čvorovi stabla imaju ulogu operatora, a listovi ulogu operanada. Pritom su skup operatora (engl. *function set F*) i operanada (engl. *terminal set T*) unaprijed definirani konačni skupovi.

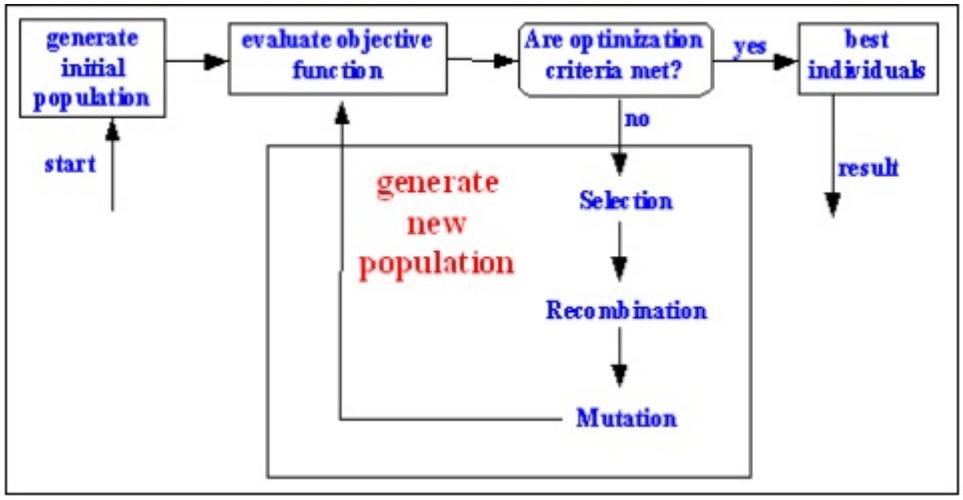
U skladu s općenitom teorijom evolucijskih algoritama, možemo reći da kod GP-a ulogu kromosoma imaju stabla. Upravo su svojstva stabala kao strogih matematičkih objekata, poput jednostavnog rekurzivnog obilaska, odredila razvojni put genetskog programiranja u naglašenom smjeru.

3.2.3. GP kao poseban slučaj evolucijskog algoritma

Iterativni postupak, na kojem se bazira genetsko programiranje, u sebi sadrži jasno vidljive elemente općenitog pseudokoda evolucijskih algoritama, a to su:

1. generacija inicijalne populacije rješenja,
2. analiza svake jedinke populacije i populacije općenito,
3. odabir genetskog operatora i provedba odgovarajućih akcija nad pojedinom jedinkom.

Detaljniji pseudokod prikazan je na slici 3.1. (4)



Slika 3.1: Pseudokod evolucijskih algoritama

3.2.4. Populacija

U biološkom smislu, populacija je skup jedinki iste vrste koje žive u istom prostoru. Razmnožavanjem unutar populacije, ali i umiranjem, veličina populacije je konstantna kroz generativni proces. Što se tiče genetskog programiranja, jedinku predstavlja računalni program, a prostor egzistencije jedinki iz iste populacije je jedna iteracija u optimizacijskom algoritmu. Prema tome, govori se o populaciji računalnih programa jedne iteracije algoritma.

3.2.5. Funkcija dobrote

U cilju rješavanja problema evolucijskim algoritmom, jako je bitno da bolja rješenja ostaju, a ona lošija ne ostaju u daljnjem razmatranju. Upravo dobro definirana funkcija dobrote obavlja tu ulogu. Na temelju raznih parametara, ona određuje dobrotu (engl. *fitness*) pojedine jedinke.

Definiranje funkcije dobrote je jedan od ključnih problema genetskog programiranja. Budući da je njeno evaluiranje prisutno u svakom trenutku generativnog procesa, potrebno je da bude što "bolja", i što jednostavnija.

3.2.6. Genetski operatori

Evolucijski aspekt genetskog programiranja se očituje u načinu optimiranja promatranog programa, gdje su, u ulogama genetskih operatora, prisutne metode reprodukcije i križanja (engl. *crossover*). U nekim slučajevima se javljaju i mutacija, permutiranje i sl.

Reprodukcija

Reprodukcija je jednostavno kopiranje odabrane jedinke i njezino umetanje u novu populaciju. Parametri: vjerojatnost odabira reprodukcije kao genetskog operatora i vjerojatnost odabira pojedine jedinke.

Križanje

Križanje je analogno biološkoj spolnoj reprodukciji. Naime, u tom postupku dolazi do zamjene nekih podstabala odabranih dviju jedinki. Najučestaliji oblik je obavljanje zamjene na dva slučajno odabrana podstabla. Parametri: vjerojatnost odabira križanja kao genetskog operatora, vjerojatnost odabira pojedine jedinke i vjerojatnost odabira pojedinog čvora jedinke kao korijena podstabla.

3.2.7. Selekcija

Tijekom generiranja populacija odvija se selekcija, i to na temelju vjerojatnosti odabira genetskog operatora, te dobrote jedinki. Uobičajeno je da selekcija direktno ovisi o dobroti. Često se koriste proporcionalna (engl. *Roulette-wheel selection*), te turnirska selekcija.

3.2.8. Kriterij prekidanja generativnog postupka

S obzirom na činjenicu da koncept genetskog programiranja s dobro definiranom funkcijom dobrote nad određenim problemom, i adekvatnim skupovima F i T, zadovoljavajuće brzo dolazi do rješenja problema, u velikom broju slučajeva generativni postupak se prekida pri ispunjenju određenog logičkog predikata. Dodatno se uzima i konstanta

G, kao supremum broja iteracija, odnosno broja generacija populacije. Također, zadaje se i supremum M broja jedinki u populaciji, te supremum D dubine generiranih stabala.

3.2.9. Utvrđivanje i opisivanje rješenja

Jednom kada se ispuni kriterij prekida generativnog procesa, iz populacije rješenja se odabire najkvalitetnije. Moguće su i poželjne daljnje analize istog, kao i ponavljanje cijelog postupka zbog nezadovoljstva dobivenim rješenjem. Bolje rješenje se uglavnom pokušava naći redefiniranjem funkcije dobrote ili povećanjem supremuma broja iteracija. (6)

4. Programsko ostvarenje

Slijedi opis vlastite implementacije genetskog programiranja i OpenCL-a. Projekt je napravljen u programskom jeziku Java, u programerskom okruženju Eclipse. Specifikacija računala na kojemu je aplikacija testirana je:

- AMD Athlon64 X2 5600+
- NVIDIA GeForce 8800GT
- 4GB DDR2 RAM
- OS Windows 7 64bit

Grafička kartica ima mogućnost paralelnog pokretanja 512 radnih jedinica. Procesor je dvojezgreni, radnoga takta 2.8GHz.

4.1. Optimizacijski problem

Prvo treba definirati problem odnosno zadatak. Radi se o problemu optimizacije mrava. Cilj svakog mrava je pojesti što više biljaka. Mravi i biljke nalaze se u pravokutnom "svijetu". Bridovi pravokutnika predstavljaju zidove kroz koje mravi ne mogu proći. Unutrašnjost pravokutnika ispunjena je poljima. U jednom polju u jednom trenutku može se nalaziti mrav, biljka ili ništa, odnosno polje može biti prazno. Mrav ima svoje usmjerenje i u jednom trenutku vidi samo što je u polju ispred njega. Usmjerenje može biti prema gore, dolje, lijevo ili desno. U jednoj vremenskoj jedinici mrav se može pomaknuti za jedno polje unaprijed, te okrenuti lijevo ili desno za 90 stupnjeva. Mrav će pojesti biljku ako dođe na njeno polje. Zadatak je stvoriti odnosno simulirati mrava, ili točnije, cijelu populaciju mrava koja će svoju zadaću obavljati što bolje.

4.2. Rješenje

Mravi su optimizirani genetskim programiranjem. Postupak se može podijeliti na četiri glavna koraka:

1. Stvori inicijalnu populaciju mrava.
2. Evaluiraj populaciju.
3. Čuvaj dobre jedinke i nad njima obavi operatore križanja i mutacije.
4. Ponavljaj 2-3.

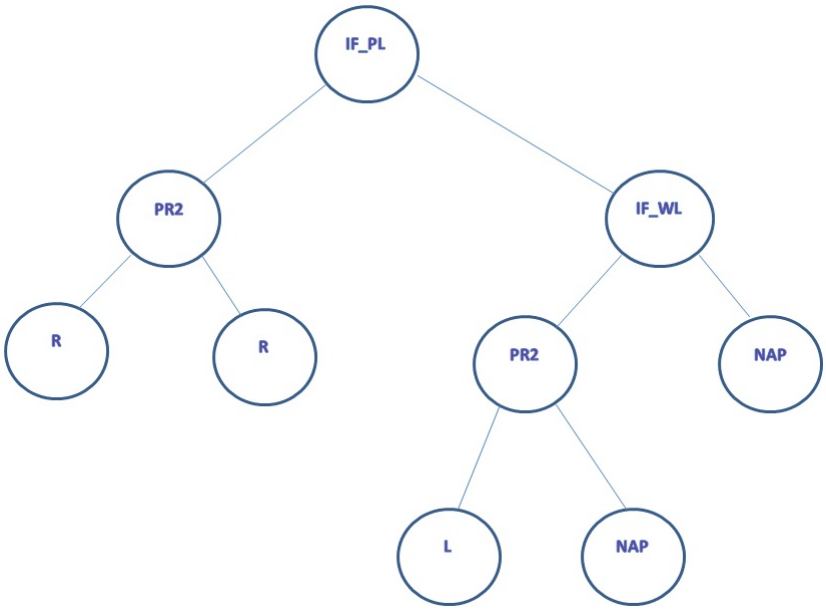
4.2.1. Prikaz jedinke

Svaki mrav je predstavljen kao stablo. Stablo predstavlja zaseban računalni program koji definira ponašanje mrava. Listovi stabla su operandi odnosno akcije, a ostali čvorovi operatori odnosno funkcije. Skup svih mogućih čvorova odnosno akcija i funkcija čine:

1. pomak unaprijed,
2. okret u lijevo,
3. okret u desno,
4. "IF_PL" => funkcija grananja, ispituje da li je ispred mrava biljka,
5. "IF_WL" => također funkcija grananja, ispituje da li je ispred mrava zid ili drugi mrav,
6. "Switch" => Složena funkcija grananja, ispituje sve četiri moguće situacije,
7. "PR2" => slijedno izvodi dvije naredbe,
8. "PR3" => slijedno izvodi tri naredbe.

Skup akcija čine čvorovi 1-3, a skup funkcija čvorovi 4-8. U čvorovima 4 i 5, u slučaju da je uvjet ispunjen, prelazi se na lijevo, a u suprotnom, na desno dijete (podstablo). Switch ima četvero djece, ispituje se što se nalazi ispred mrava. Ispred mrava može biti prazno polje, zid, biljka ili pak drugi mrav. S obzirom na rezultat ispitivanja, pokreće se određeno dijete. PR2 ima dvoje djece i ona se aktiviraju slijedno s lijeva na desno. Isto tako radi i PR3, samo što ima troje djece.

Zapis stabla u računalu ostvaren je kao niz brojeva. Svaki broj označava jedan čvor. Broj 1 odgovara prethodno navedenom čvoru 1, broj 2 čvoru 2 itd. Djeca nekog čvora zapisana su, s lijeva na desno, direktno poslije samog čvora. Znači zapis ide "u dubinu". Prvi broj u nizu je ujedno i korijen stabla.



Slika 4.1: Primjer stabla

Obilazak stabla, tj. čitanje niza brojeva obavlja se s lijeva na desno. Svaki broj odnosno čvor ima svoju težinu. Težina je određena brojem djece. Tako čvorovi 1-3 imaju težinu 0, dok čvor 6 ima težinu 4. Ukupna težina nekog čvora je težina podstabla kojemu je taj čvor korijen. Obilazak počinje s korijenom koji mora biti funkcija, u obzir dolaze čvorovi 4-8. Aktivira se korijen. Ako treba "skočiti" na drugo dijete, računa se ukupna težina prvog djeteta te se za taj iznos pomaknemo u nizu, tako da sljedeći čvor bude upravo drugo dijete.

Generiranje početne populacije mrava odnosno stabala, odvija se uz ograničenja. Maksimalan broj čvorova u stablu je 20, a minimalan 5. Odabir čvorova u početnoj populaciji je slučajan. Pazi se samo da je broj čvorova unutar granica te da nema "praznih" grananja. Slika 4.1 pokazuje stablo "473357211". Nakon što su generirani, mravi dobivaju svoje polje tj. poziciju u svijetu.

4.2.2. Evaluiranje populacije

Uz mrave, generiraju se i biljke koje također dobivaju svoja polja u svijetu. Odabir tih polja je slučajan za svaku novu generaciju mrava, što znači da je svijet promjenjiv. Nakon što su mravi i biljke generirani i pozicionirani, počinje godina. Mravlja godina sastoji se od 250 dana. U jednom danu mrav može napraviti jednu akciju. Evaluacija se odvija na kraju svake godine.

Funkcija dobrote je prilično jednostavna. Za svakog se mrava pamti koliko je bi-

ljaka pojeo u tekućoj godini. Prosječan rezultat generacije se računa kao omjer broja pojedenih biljaka i broja mrava. Dobrota pojedinog mrava je omjer pojedenih biljaka dotičnog mrava i prosječnog rezultata generacije. Ako mrav ima dobrotu 3.00, znači da je pojeo čak tri puta više biljaka od prosjeka. Dobrota 0.50 značila bi da je mrav ispodprosječan.

4.2.3. Stvaranje nove populacije

Preživljavanje mrava, tj. ulazak u novu populaciju ovisi o njegovoj dobroti. Ukoliko je dobrota određenog mrava 0.87, upravo je tolika šansa (87%) da će ući u novu populaciju. Ako je dobrota 3.00, mrav će se tri puta kopirati u novu generaciju.

Uz ovakvo preslikavanje, može se zaključiti da nova populacija ne mora biti jednake veličine kao stara. Obično veličina nove populacije odstupa do 15% od željene veličine. Da veličina populacije ne bi previše odstupala od željene, dobrota svakog mrava prije preslikavanja se množi sa omjerom željene i stvarne veličine populacije. Tako će zapravo nova populacija veličinom odstupati uvijek od željene veličine, a ne od veličine stare populacije.

4.2.4. Genetski operatori

Nad novom generacijom mrava obavljaju se operatori križanja i mutacije. Aplikacija najprije učita koje su odabrane vjerojatnosti križanja i mutacije, te ih onda primjenjuje.

U procesu križanja sudjeluju dva mrava. Slučajno se odabire podstablo jednog i zamjenjuje sa slučajno odabranim podstablom drugoga mrava. Pritom se mora paziti da nova stabla budu u dozvoljenim granicama veličine. Broj križanja je fiksan. Ako je veličina populacije POP=100 i vjerojatnost križanja C=50%, doći će do točno 25 (POP*C/2) procesa križanja. Odabir mrava za križanje je slučajan, no pazi se da se ne odabiru isti mravi više puta.

Mutacija stabla je zamjena jednog čvora skupa sa pripadajućim podstablom sa novim slučajno generiranim podstablom. Također se pazi da novo stablo bude dozvoljene veličine. Broj mutacija nije fiksan. Vjerojatnost mutacije svakog mrava jednaka je željenoj odnosno odabranoj vrijednosti.

4.2.5. Paralelizacija

Aplikacija je ubrzana paralelizacijom čestih operacija. Svakog dana u godini, obilaskom stabla, računaju se nove pozicije svih mrava. Ako je broj mrava 1000, te ako

se promatra period od 100 godina, dolazi do ukupno 25 milijuna računanja novih pozicija, s time da je u jednom danu potrebno izračunati pozicije svih mrava da bi se moglo prijeći u sljedeći dan. Paralelizacijom ove operacije, teorijski, postiže se znatno ubrzanje. Ubrzanje bi trebalo biti proporcionalno broju mrava.

Paralelizaciju omogućuje OpenCL, a JOCL biblioteke omogućuju korištenje OpenCL-a u Javi.

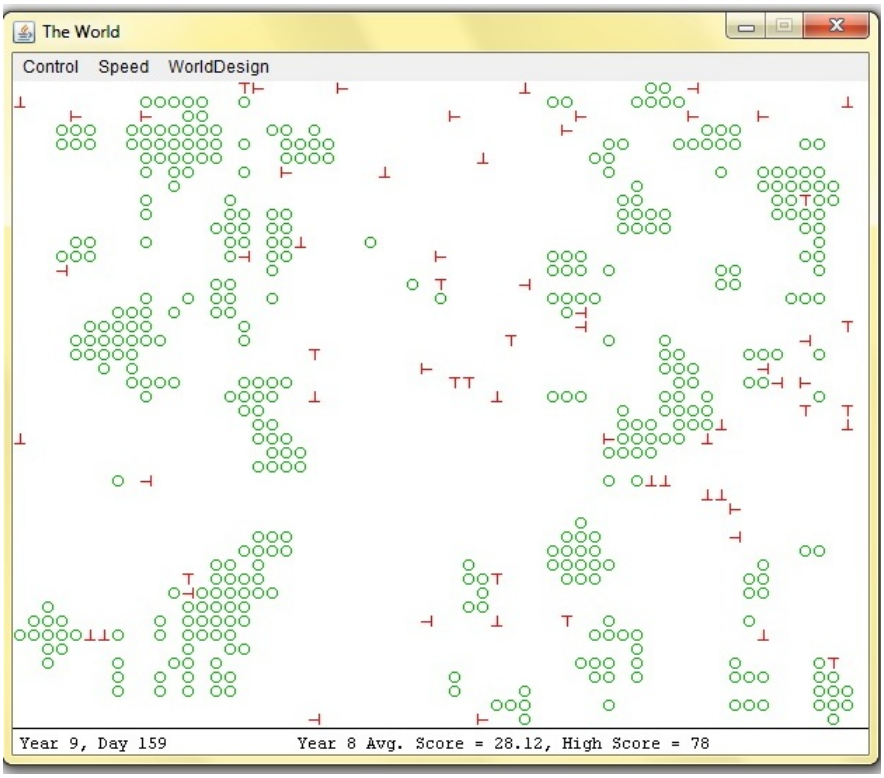
Grafički prikaz mrava, te korisničko sučelje preuzeto je s projekta Davida Ecka (7). Prije pokretanja GUI-a, inicijalizira se OpenCL uređaj. Inicijalizirat će se uređaj koji ima najveću moć paralelizacije. To je u ovom slučaju grafička kartica. Moć paralelizacije se mjeri radnom veličinom odnosno brojem dretvi koje uređaj može paralelno izvoditi.

Tekst programa za izračun nove pozicije prilagođen je i premješten u zasebnu OpenCL C datoteku koja predstavlja funkciju OpenCL jezgre. Funkcija kao argumente prima tri spremnika i jednu konstantu. Veličina spremnika ovisi o broju mrava i radnoj veličini uređaja. Određuje se najmanjim višekratnikom radne veličine koji je veći ili jednak broju mrava. Taj višekratnik predstavlja globalnu radnu veličinu ili globalnu veličinu problema. Prvi spremnik sadrži usmjerenje i informaciju što se nalazi ispred mrava za sve mrave na početku jednoga dana. Drugi spremnik sadrži stabla, odnosno odgovarajući niz od 20 brojeva, svih mrava. U treći spremnik se spremaju rezultati odnosno novo usmjerenje i informacija o pomaku svakog mrava. Zadnji argument je broj mrava. Budući da je on promjenjiv, a veličina spremnika fiksna, informacija o trenutnom broju mrava služi da bi se mogle zaustaviti radne jedinice čiji je indeks veći od broja mrava. Budući da je polje mrava jednodimenzionalno, funkcija je također jednodimenzionalna. To znači da se stvara jednodimenzionalni indeksni prostor kojemu sa svojim indeksom pristupaju radne jedinice uređaja.

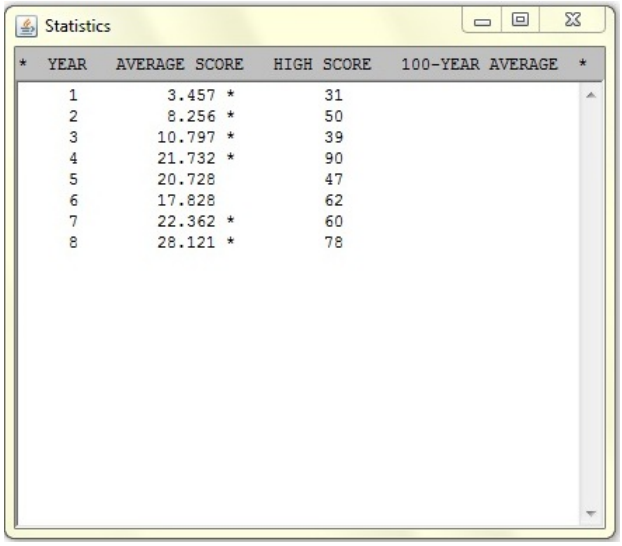
Na uređaju se postavi red (engl. *Queue*) operacija koje uređaj treba izvršiti. Operacije uključuju pripremu spremnika i jezgrene OpenCL funkcije. Nakon što su se operacije izvršile, rezultate dobavljamo iz trećeg spremnika.

4.2.6. Upute za korištenje programa

Kompletan GUI je napravljen u Javi. Pri pokretanju aplikacije pojavljuje se prozor koji ima samo jedan gumb "*Start the world*". U alatnoj traci nalazi se gumb "*Applet*" koji nudi opcije poput "*Restart*","*Reload*","*Stop*","*Clone*" itd. Svijet pokrećemo klikom na "*Start the world*".



Slika 4.2: Prozor "The World"



Slika 4.3: Prozor "Statistics"

Stvaraju se dva nova prozora. "*The World*" (slika 4.2) i "*Statistics*" (slika 4.3). Prvi prozor prikazuje svijet. Mravi su prikazani crvenom bojom, u obliku slova "T", gdje vršak tj. donji dio slova predstavlja usmjerenje. Biljke su prikazane kao zeleni kružići. Na dnu prozora nalazi se redni broj godine, trenutni dan, te prosječan i najbolji rezultat prošle godine. Na alatnoj traci možemo namjestiti razne opcije. Možemo promijeniti

vjerojatnost križanja i mutacije, željenu veličinu populacije, te broj biljaka. Može se definirati način stvaranja biljaka i mrava na početku godine, tako da njihov raspored ne bude potpuno slučajan. Imamo opcije kontrole vremena "*Go*" i "*Pause*", te možemo odabrati brzinu izvođenja aplikacije. Početni parametri su sljedeći:

- Raspored biljaka - u gomilama
- Broj biljaka - 500
- Nakon što je biljka pojedena - opet se stvara na nekoj lokaciji
- Željena populacija - 100
- Mravi su rođeni - na slučajnoj poziciji
- Vjerojatnost mutacije - 1%
- Vjerojatnost križanja - 75%

Prozor "*Statistics*" nam pruža uvid u statističke podatke. Podaci su organizirani u tablicu. Redak tablice predstavlja jednu godinu. Za svaku godinu ispisuje se prosjek određene godine, najbolji rezultat u toj godini, te stogodišnji prosjek do te godine.

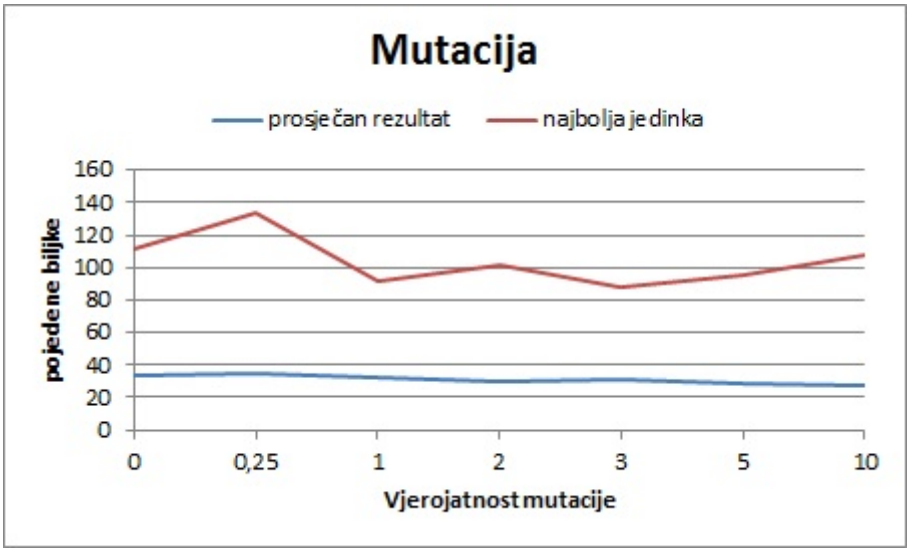
Iz aplikacije se izlazi klikom na opciju "*End the world*", te biranjem opcije "*Quit*" u početnom prozoru.

4.3. Rezultati

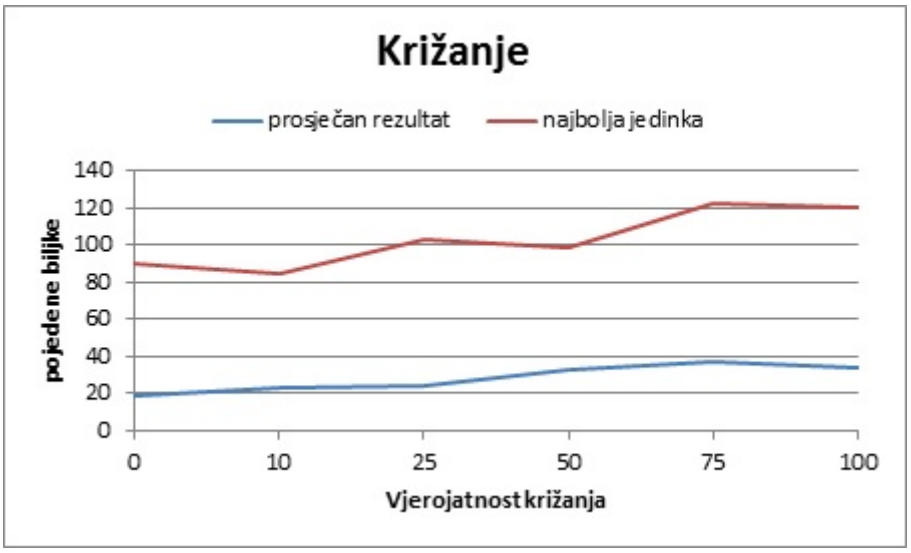
Zasebno se razmatraju rezultati optimizacije genetskim programiranjem, te rezultati ubrzanja putem OpenCL-a.

4.3.1. Genetsko programiranje

Ispitani parametri vezani uz genetsko programiranje su vjerojatnost križanja i mutacije. Za mjerenje optimalnih vrijednosti parametara, potrebno je odrediti kriterij zaustavljanja algoritma. Budući da je algoritam prilično učinkovit, zaustaviti će se ako u posljednjih deset godina nije postignut bolji od trenutno najboljeg rezultata. Određene vrijednosti parametara ispitane su deset puta. Slika 4.4 pokazuje utjecaj vjerojatnosti mutacije na prosječan rezultat najboljih generacija i najboljih jedinki prije zaustavljanja algoritma. Najbolji rezultat postignut je uz vjerojatnost mutacije 0.25%. Daljnjim porastom vjerojatnosti, prosječan rezultat opada.



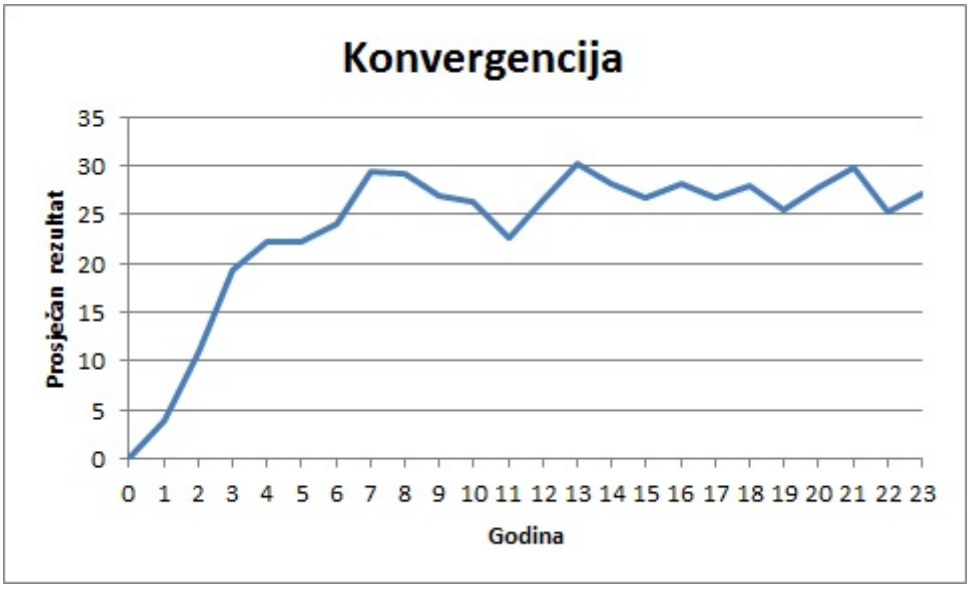
Slika 4.4: Utjecaj vjerojatnosti mutacije



Slika 4.5: Utjecaj vjerojatnosti križanja

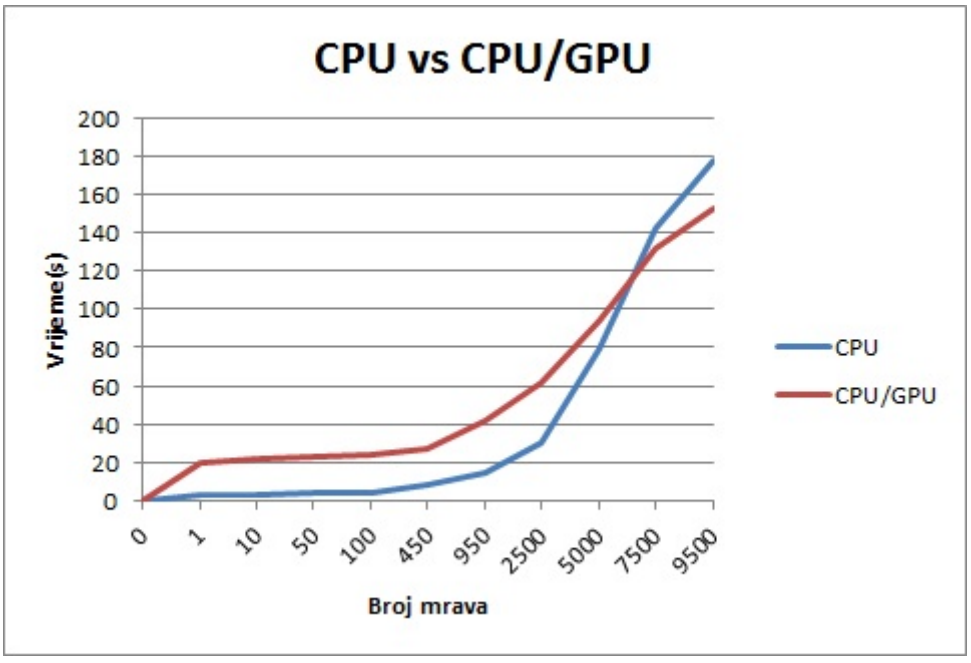
Na slici 4.5 vidi se odnos vjerojatnosti križanja sa prosječnim i najboljim rezultatom. Rezultat je proporcionalan vjerojatnosti križanja sve do maksimuma koji je postignut sa vjerojatnošću od 75%. Nakon toga rezultat je u laganom padu.

Uz optimalne parametre, algoritam brzo konvergira prema zadovoljavajućim rezultatima. Konvergencija je prikazna na slici 4.6. Algoritmu je dovoljno svega tri do četiri godine za osjetno poboljšanje rezultata. Nakon toga rezultat je, uz manje oscilacije, u blagom porastu.



Slika 4.6: Prikaz konvergencije

4.3.2. OpenCL



Slika 4.7: Odnos CPU i CPU/GPU

Slika 4.7 pokazuje vrijeme potrebno za izvođenje sto godina algoritma uz određeni broj mrava. Prikazani su rezultati bez paralelizacije (CPU), te rezultati koristeći OpenCL (CPU/GPU). Vidljivo je da je OpenCL u ovom slučaju isplativ za veći broj mrava. Ubrzanje je postignuto nakon brojke od 7000 mrava. Uz daljnje povećanje broja mrava, prednost paralelizacije je sve veća.

Ovakvi rezultati ne moraju uvijek biti slučaj. Ubrzanje ponajprije ovisi o konkretnom CPU i GPU koji izvode aplikaciju. Slike 4.8 i 4.9 (8) pokazuju trenutni poredak po performansama komercijalnih CPU-ova i GPU-ova. Da bi što pravednije ocijenili ubrzanje postignuto OpenCL-om, poželjno je da CPU i GPU budu u istoj kategoriji.

Gaming CPU Hierarchy Chart	
Intel	**AMD**
Core i7-2600, -2600K Core i7-965, -975 Extreme, -980X Extreme, -990X Extreme Core i7-970, -960 Core i5-2500, -2500K	
Core i7-860, -870, -875K, -920, -930, -940, -950, Core i5-750, -760 Core 2 Extreme QX9775, QX9770, QX9650 Core 2 Quad Q9650 Core i3-2100, -2120	Phenom II X4 Black Edition 975
Core 2 Extreme QX6850, QX6800 Core 2 Quad Q9550, Q9450, Q9400 Core i5-650, -655K, -660, -661, -670, -680	Phenom II X6 1100T BE, 1090T BE, 1075T Phenom II X4 Black Edition 970, 965, 955
Core 2 Extreme QX6700 Core 2 Quad Q6700, Q9300, Q8400, Q6600, Q8300 Core 2 Duo E8600, E8500, E8400, E7600 Core i3 -530, -540, -550	Phenom II X6 1055T Phenom II X4 945, 940, 920, 910, 910e, 810 Phenom II X3 Black Edition 720, 740 Athlon II X4 645, 640, 635, 630 Athlon II X3 455, 450, 445, 440, 435
Core 2 Extreme X6800 Core 2 Quad Q8200 Core 2 Duo E8300, E8200, E8190, E7500, E7400, E6850, E6750	Phenom II X4 905e, 805 Phenom II X3 710, 705e Phenom II X2 565 BE, 560 BE, 555 BE, 550 BE, 545 Phenom X4 9950 Athlon II X4 620 Athlon II X3 425
Core 2 Duo E7200, E6550, E7300, E6540, E6700 Pentium Dual-Core E5700, E5800, E6300, E6500, E6600, E6700 Pentium G9650	Phenom X4 9850, 9750, 9650, 9600 Phenom X3 8850, 8750 Athlon II X2 265, 260, 255 Athlon 64 X2 6400+
Core 2 Duo E4700, E4600, E6600, E4500, E6420 Pentium Dual-Core E5400, E5300, E5200	Phenom X4 9500, 9550, 9450e, 9350e Phenom X3 8650, 8600, 8550, 8450e, 8450, 8400, 8250e Athlon II X2 240, 245, 250 Athlon X2 7850, 7750 Athlon 64 X2 6000+, 5600+
Core 2 Duo E4400, E4300, E6400, E6320 Celeron E3300	Phenom X4 9150e, 9100e Athlon X2 7550, 7450, 5050e, 4850e/b Athlon 64 X2 5400+, 5200+, 5000+, 4800+
Core 2 Duo E5500, E6300 Pentium Dual-Core E2220, E2200, E2210 Celeron E3200	Athlon X2 6550, 6500, 4450e/b, Athlon X2 4600+, 4400+, 4200+, BE-2400
Pentium Dual-Core E2180 Celeron E1600	Athlon 64 X2 4000+, 3800+ Athlon X2 4050e, BE-2300

Slika 4.8: Tablica CPU-ova

Graphics Card Hierarchy Chart		
GeForce	**Radeon**	**Intel**
Discrete: GTX 590	Discrete: HD 6990	
Discrete: GTX 580	Discrete: HD 5970	
Discrete: GTX 295, GTX 480, GTX 570	Discrete: HD 4870 X2, 6970	
	Discrete: HD 4850 X2, 5870, 6950	
Discrete: GTX 470, GTX 560 Ti	Discrete: HD 5850, 6870	
Discrete: 9800 GX2, GTX 285, GTX 460 1GB, GTX 465	Discrete: HD 6850	
Discrete: GTX 260, GTX 275, GTX 280, GTX 460 768 MB, GTX 460 SE, GTX 550 Ti	Discrete: HD 4870, HD 5770, HD 4890, HD 5830, 6790 Mobility: HD 5870	
Discrete: 8800 Ultra, 9800 GTX, 9800 GTX+, GTS 250, GTS 450	Discrete: HD 3870 X2, HD 4850, HD 5750 Mobility: HD 4850, HD 5850	
Discrete: 8800 GTX, 8800 GTS 512 MB Go (mobile): GTX 280M, GTX 285M	Discrete: HD 4770 Mobility: HD 4860	
Discrete: 8800 GT 512 MB, 9800 GT Go (mobile): 9800M GTX, GTX 260M (112), GTS 360M (GDDR5)	Discrete: HD 4830, HD 5670, HD 6670 Mobility: HD 5770, HD 5750	
Discrete: 8800 GTS 640 MB, 9600 GT, GT 240 (GDDR5) Go (mobile): 9800M GTS, GTX 160M	Discrete: HD 2900 XT, HD 3870, HD 5570 (GDDR5), HD 6570 (GDDR5)	
Discrete: 8800 GS, 9600 GSO, GT 240 (DDR3) Go (mobile): GTX 260M (96), GTS 150M, GTS 360M (DDR3)	Discrete: HD 3850 512 MB, HD 4670, HD 5570 (DDR3), HD 6570 (DDR3) Mobility: HD 3870, HD 5730, HD 5650	
Discrete: 8800 GT 256 MB, 8800 GTS 320 MB, GT 440 GDDR5 Go (mobile): 8800M	Discrete: HD 2900 PRO, HD 3850 256 MB, 5550 (GDDR5) Mobility: HD 3850	
Discrete: 7950 GX2, GT 440 DDR3	Discrete: X1950 XTX, HD 4650 (DDR3), 5550 (DDR3)	
Discrete: 7800 GTX 512, 7900 GTO, 7900 GTX, GT 430	Discrete: X1900 XT, X1950 XT, X1900 XTX	
Discrete: 7800 GTX, 7900 GT, 7950 G, GT 220 (DDR3)	Discrete: X1800 XT, X1900 AIW, X1900 GT, X1950 PRO, HD 2900 GT, HD 5550 (DDR2)	
Discrete: 7800 GT, 7900 GS, 8600 GTS, 9500 GT (GDDR3), GT 220 (DDR2) Go (mobile): 7950 GTX	Discrete: X1800 XL, X1950 GT, HD 4650 (DDR2), HD 6450 Mobility X1800 XT, HD 4650, HD 5165	

Slika 4.9: Tablica GPU-ova

5. Zaključak

Postupak optimizacije genetskim programiranjem, u vrlo kratkom vremenu, postiže zadovoljavajuće rezultate. Zahtjevnije optimizacije poželjno je ubrzati postupkom paralelizacije određenog dijela algoritma. OpenCL se pokazao idealnim za taj postupak.

OpenCL je prvi sustav koji omogućuje izvođenje istog teksta programa na različitim platformama, olakšava postupak paralelizacije, a njegova primjena može varirati od korisničkih aplikacija pa sve do računarstva visokih performansi. Danas su na raspolaganju mikroprocesori različitih arhitektura, pa se nameće pitanje kako ih što bolje i jednostavnije iskoristiti. OpenCL popunjava upravo tu prazninu. U budućnosti se očekuje korištenje GPU-ova u općenitije svrhe, naravno uz postojanje neophodnog CPU-a. Također se može očekivati i češća pojava "hibridnih" mikroprocesora koji sadrže veći broj jezgri, bile one opće namjene ili specijalizirane.

LITERATURA

[1] "Opencl." http://en.wikipedia.org/wiki/OpenCL, 2011.

[2] "Nvidia cuda programming guide." http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf, 2007.

[3] "Java opencl." http://jogamp.org/jocl/www/, 2011.

[4] "Evolutionary algorithms." http://www.scholarpedia.org/article/Evolutionary_algorithms, 2008.

[5] "Genetic programming – wikipedia, the free encyclopedia." http://en.wikipedia.org/wiki/Genetic_programming, 2011.

[6] R. J. Koza, *What is Genetic Programming (GP)?, How Genetic Programming Works*, 2007.

[7] D. Eck, "A demonstration of the genetic algorithm." http://math.hws.edu/xJava/GA/, 2001.

[8] "Toms hardware." http://www.tomshardware.com/, 2011.

Evolucijski algoritam za hibridnu paralelnu okolinu

Sažetak

Evolucijski algoritmi su postupci optimiranja koji se temelje na mehanizmu evolucije u prirodi. Kod genetskog programiranja kromosom predstavlja program koji je rješenje zadanog problema. Tipična struktura podataka koja se kod genetskog programiranja koristi za prikaz kromosoma jest stablo. Standard OpenCL omogućuje paralelizaciju algoritama i njihovo pokretanje na različitim platformama. Optimizacija virtualnih mrava postignuta je genetskim programiranjem, te je ubrzana OpenCL-om. Ubrzanje je proporcionalno broju mrava.

Ključne riječi: evolucija, algoritmi, genetsko, programiranje, OpenCL, paralelizacija, hibridi.

Evolutionary algorithm for hybrid parallel platforms

Abstract

Evolutionary algorithms are optimization procedures based on the mechanism of evolution in nature. In genetic programming chromosome represents a program, solution of the problem. Typical data structure used in genetic programming to display chromosomes is a tree. Standard OpenCL offers algorithm parallelization and their run on different platforms. Optimization of virtual ants is achieved by genetic programming, and accelerated by OpenCL. Acceleration is proportional to the number of ants.

Keywords: evolution, algorithm, genetic, programming, OpenCL, parallel, hybrid.