Projekat iz Ekspertskih sistema

Više-niti simulator rešavanja ukrštenih reči

zasnovan na algoritmima Veštačke inteligencije

Stefan Dragišić

Elektrotehnički fakultet Beograd

1. Opis algoritma.

Algoritam je zasnovan na rekurzivnom Backtracking algoritmu pretraživanja uz modifikaciju raspoređivanja poslova na više niti radi efikasnijeg nalaženja svih mogućih rešenja.

Problem sinhronizacije je rešen tako što svaka nit dobija potpuno novu, duboku kopiju table, nad kojom ona traži dalja rešenja, i ni na koji način ne može uticati na druge niti.

Kada algoritam detektuje novi pravac pretraživanja, pokuša da pretragu u novom pravcu dodeli novoj niti ukoliko ima slobodnih niti na raspolaganju. Ukoliko nema nastavlja pretragu u svojoj niti.

Broj niti u aplikaciji se automatski skalira u zavisnosti od broja hardverskih niti (jezgara u procesoru).

Primećuje se da se najveća brzina postiže ukoliko je broj niti jednak broju hardverskih niti, za bilo koji veći broj niti dosta vremena se gubi na promenu konteksta i brzina pretrage opada, čak do te mere da može biti sporija od jedno-nitnog moda.

U zavisnosti od hardvera može se postići ubrzanje nalaženja rešenja od čak 10 puta u odnosu na jedno-nitni mod.

Jedno-nitni mod je efikasniji od više-nitnog u slučajevima kada postoji jako malo reči u rečniku i mali broj varijabli na tabli, zbog režiskih troškova koje zahteva često pravljenje novih niti i duboko kopiranje stanja na tabli.

**Moguće modifikacije:**

Uz jednostavne prepravke aplikacija se može modifikovati da deli zadatke između više računara i tako implementira Grid computing mrežu sa efikasno nalaženje svih rešenja u velikim ukrštenicama sa jako velikim rečnikom.

Takođe se primećuje da bi se uz minimalne modifikacije i uključivanje odgovarajućih biblioteka postigao ogroman nivo paralelizma koristeci GPU umesto CPU za pretragu rešenja, s’obzirom da niti brzo stiču potrebu da podele nove poslove drugim nitima, a da je sama sinhornizacija već obezbeđena.

1. Algoritam

**backtrack** (Assignment asm){

if varQueue is Empty then return;

var = varQueue.PopVar();

while(var.tryAndRemoveNextDomain){

if(Constrain.validate(var,asm){

asm.complete\_vars.add(var);

if(is solution) then print(solution)

if(threadQueue.hasFreeThreads())

start new WorkerThread(**backtrack**(DeepCopy(asm)));

} else {

continue in current thread -> **backtrack**(DeepCopy(asm));

remove (asm.complete\_vars.remove(var));

}

}

}