Paralelno programiranje - 2. laboratorijska vježba

Vedran Pintarić

14. svibnja 2017.

1 Uvod

Zadatak vježbe jest implementirati igru *connect 4* te implementirati pretraživanje stanja igre uz korištenje MPI biblioteke za paralelno programiranje tako da igrač može igrati protiv računala.

2 Implementacija algoritma

Prvotna implementacija pretraživanja je serijsko pretraživanje u dubinu ograničeno na neku maksimalnu dubinu.

2.1 Podjela

Ideja za paralelizaciju pretraživanja u dubinu jest ta da za svako novo stanje koje otvorimo stvaramo novi zadatak koji se može paralelno izvoditi. U tom slučaju bi u svakoj razini imali W^d zadataka, gdje je W širina ploče a d trenutna dubina pretraživanja.

2.2 Komunikacija

Podaci koje procesi međusobno trebaju izmijeniti u takvoj podjeli su podaci koji opisuju cijelo stanje koje proces treba istražiti. Za povratnu vrijednost je potreban samo jedan broj s pomičnim zarezom. Jedno stanje se može opisati pomoću cijelog stanja ploče, varijable koja govori tko je trenutno na potezu te varijable koja govori koja je trenutna dubina pretraživanja.

Table 1: Trajanje algoritma u ovisnosti o broju radnika

broj radnika	1	2	3	4	5	6	7	8
trajanje (s)	50.15	27.82	22.62	20.07	18.30	15.37	14.68	14.24

2.3 Aglomeracija

Kako nije realno napraviti podjelu na način da stvaramo nove zadatke na svakoj dubini algoritma potrebno je na drugačiji način stvarati zadatke. Odabrana je arhitektura *master* - *slave* gdje ce *master* provesti algoritam do neke "plitke" dubine te na toj dubini za daljnje pretraživanje stvoriti zadatke koje će dijeliti *slave* procesima. *Slave* procesi na kraju svojeg pretraživanja vraćaju rezultat *master-u* te traže sljedeći zadatak (ako takav postoji). Kada su svi zadaci gotovi *master* može ponovno provesti svoje pretraživanje do "plitke" dubine te sada, znajući rezultate od dubljih dubina, može provesti algoritam do kraja te vratiti krajnje rješenje.

2.4 Pridruživanje

Kako je za zadatak predviđeno 8 procesora, pomoću MPI-ja možemo napraviti 1 *master* proces te maksimalno 7 *slave* procesa. Kako imamo 7 *slave* procesa optimalni broj zadatak bi bio otprilike za red veličine veći od 7. Za širinu ploče $W = 7$ i na dubini $d = 2$ *master* proces bi stvorio $W^d = 49$ zadataka što nije niti previše grub ni previše fin broj zadataka za podijeliti na 7 procesa.

3 Ispitivanje

Algoritam je ispitan za širinu ploče 7, visinu ploče 6 te maksimalnu dubinu pretraživanja prostora stanja 9. Ispitujemo algoritam uz variranje broja *slave* procesa od 1 do 7. Svaki test je proveden na pronalaženju najboljeg prvog poteza (ploča je prazna).

Prikaz trajanja algoritma za pojedini broj radnika je prikazan u tablici 1. Ubrzanja i učinkovitosti u ovisnosti o broju radnika prikazana su u grafovima 1 i 2.

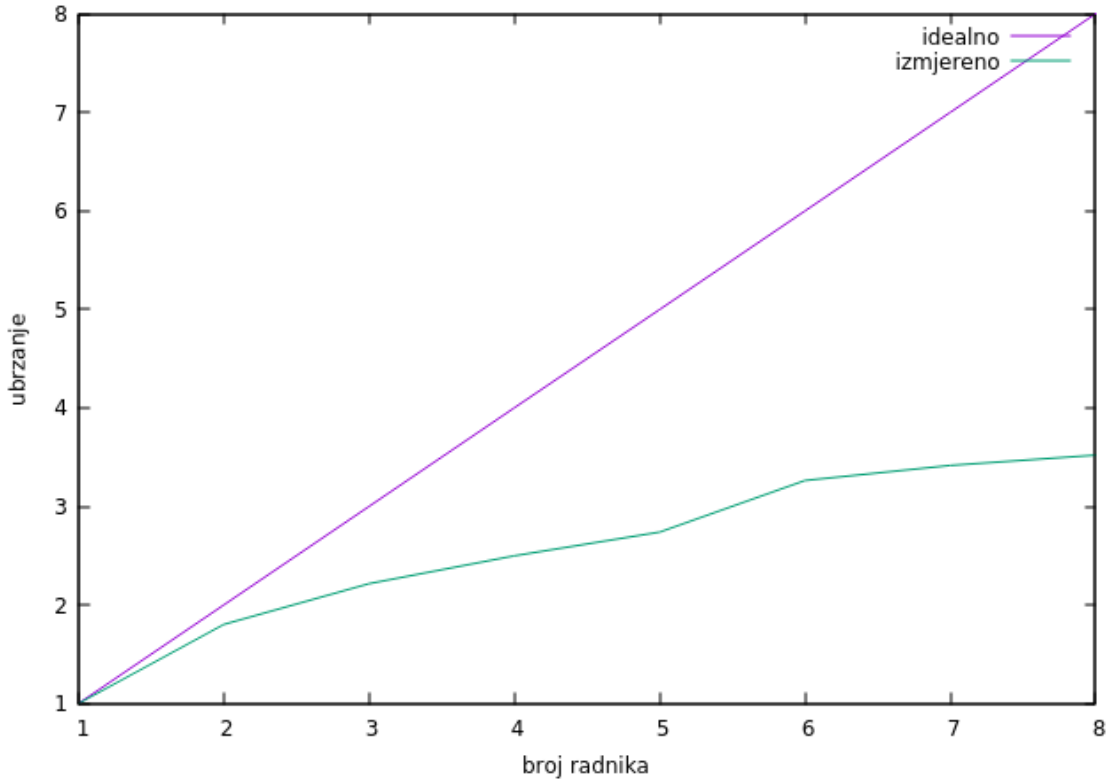

Figure 1: Ubrzanje u ovisnosti broja radnika

4 Zaključak

Zbog prilično velikih poruka koje *master* proces mora slati *slave* procesima kao zadatke algoritam ne dobiva puno na ubrzanju uz veći broj *slave* procesa. Unatoč tome, paralelizacija pretraživanja se definitivno isplati za veće maksimalne dubine gdje je veći naglasak na zadatke koje dobivaju *slave* procesi nego na poruke koje šalje *master* proces.

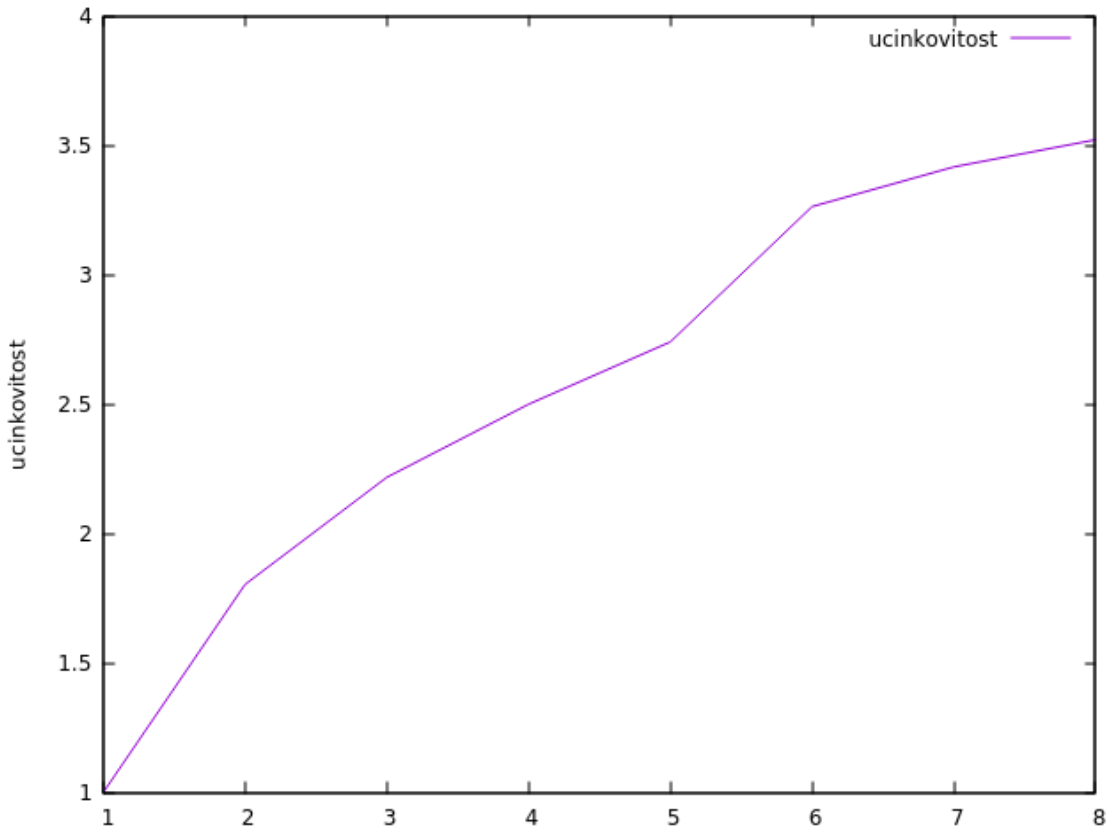

Figure 2: Efikasnost pojedinog radnika u ovisnosti broja radnika