SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ZAVOD ZA ELEKTRONIKU, MIKROELEKTRONIKU,
RAČUNALNE I INTELIGENTNE SUSTAVE

DIPLOMSKI STUDIJ
PARALELNO PROGRAMIRANJE

Upute za izradu domaćih zadaća - MPI

Domagoj Jakobović, Karlo Knežević

Verzija: *1.1, 11.4.2022.*

Zagreb, travanj 2022.

SADRŽAJ

1. MPI

MPI (engl. *Message-Passing Interface*) jest sučelje specifikacije biblioteke za mehanizam razmjene poruka. MPI ponajviše predstavlja paralelni programski model razmjene poruka, gdje se podaci prebacuju iz adresnog prostora jednog procesa u adresni prostor drugoga, kroz surađujuće operacije na svakom procesu.

MPI je specifikacija, a ne implementacija. Postoji više MPI implementacija s obzirom na programski jezik. Sam MPI standard je specifikacija sučelja biblioteke koje će se koristiti kao funkcije, podrutine ili metode s obzirom na implementaciju MPI-a u nekom programskom jeziku. C, C++, Fortran-77 i Fortran-95 su dio MPI standarda. Cilj MPI-ja je razvoj široko upotrebljivog standarda za pisanje programa s principom razmjena poruka. Trenutna aktivna specifikacija standarda MPI je 4.0.

Neke od implementacija MPI specifikacije su MPICH, OpenMPI te MS-MPI.

1.1. Instalacija

Kako bi se mogao napisati, prevesti i izvršiti paralelan program koji koristi MPI funkcije potrebni su: biblioteka zaglavlja MPI funkcija, jezični prevoditelj, biblioteka implementiranih MPI funkcija te pokretač MPI programa. Kako bi se preveo MPI program, može se koristiti jezični prevoditelj namijenjen jeziku u kojem je program pisan ili program omotač (engl. *wrapper*) koji ima unaprijed postavljene zastavice za MPI biblioteke prilikom prevođenja. Nakon što se MPI program prevede, izvršni program pokreće se iz komandne ljuske naredbom *mpiexec*. Naredba *mpiexec* stvara i pokreće onoliko MPI procesa koliko je zadano u argumentima.

U nastavku slijedi opis instalacije MPI biblioteka za pojedine operacijske sustave.

1.1.1. Windows

Preporučena implementacija MPI specifikacije za Windows operacijski sustav jest MS-MPI. Sa stranice za preuzimanje preuzmite sljedeće instalacijske pakete: msmpisdk.msi

i msmpisetup.exe. Prije preuzimanja provjerite zadovoljava li računalo na koje namjeravate instalirati navedene programe ograničenja. Nakon preuzimanja oba instalacijska paketa, pokrenite oba paketa u neovisnom redoslijedu i pratite korake instalacije.

Nakon što su uspješno instalirana oba paketa, na računalu se nalaze dva direktorija: C:\Program Files\Microsoft MPI i C:\Program Files (x86)\Microsoft SDKs\MPI. Na prvoj putanji nalazi se pokretač MPI procesa (mpiexec), a na drugoj putanji nalazi se zaglavne biblioteke MPI funkcija i izvršne biblioteke MPI funkcija (lib i bin). Prilikom instalacije, putanje do pokretača MPI procesa, zaglavlja MPI funkcija te izvršnih biblioteka MPI funkcija za 32-bitnu i 64-bitnu arhitekturu dodane su u varijable okruženja.

Kako biste provjerili uspješnost instalacije MPI okoline, u naredbenu ljusku unesite sljedeću naredbu:

```
C:\Users\PP\Desktop> mpiexec
```

Ako je instalacija MPI okoline i postavljanje varijabli okruženja uspješno, tada biste kao odgovor trebali vidjeti sljedeći ispis:

```
Microsoft MPI Startup Program [Version 9.0.12497.9]

Launches an application on multiple hosts.

Usage:

mpiexec [options] executable [args]
[ : [options] exe [args] : ... ]
mpiexec -configfile <file name>

Common options:

-n <num_processes>
-env <env_var_name> <env_var_value>
-wdir <working_directory>
-hosts n host1 [m1] host2 [m2] ... hostn [mn]
-cores <num_cores_per_host>
-lines
-debug [0-3]
```

```
Examples:

mpiexec -n 4 pi.exe
mpiexec -hosts 1 server1 master : -n 8 worker

For a complete list of options, run mpiexec -help2
For a list of environment variables, run mpiexec -help3

You can reach the Microsoft MPI team via
email at askmpi@microsoft.com
```

U nastavku slijedi opis postavljanja integrirane razvojne okoline u kojoj ćete pisati MPI program. Pretpostavlja se da koristite C programski jezik i razvojnu okolinu Visual Studio. Unutar Visual Studija napravite prazan projekt za programski jezik C. Stvorite *main.c* datoteku i u nju kopirajte primjer Pozdrav svijete 1.2.

Primjetit ćete da su sve MPI funkcije, uključujući i MPI zaglavlje podcrtani. Uzrok ovih grešaka jest činjenica da razvojna okolina ne prepoznaje putanju do MPI zaglavlja, a time i do prototipova MPI funkcija korištenih u kodu. Kako bi ispravili ovu grešku, prevoditelju je potrebno zadati gdje se nalaze zaglavlja MPI funkcija (mpi.h).

Unutar programskog paketa (engl. *Visual Studio solution*), desnim klikom miša na projekt otvorite postavke projekta. Na početku prozora nalazi se konfiguracijsko polje projekta (engl. *configuration*). Preporuka je odabrati sve konfiguracije projekta, konfiguraciju za otklanjanje pogrešaka (engl. *debug*) i distribucijsku konfiguraciju (engl. *release*), (engl. *all configurations*). Ako odaberete samo jednu konfiguraciju, tada prilikom prevođenja i stvaranja izvršnog MPI programa koristite tu konfiguraciju.

U postavkama za C/C++, u polje dodatnih direktorija za uključivanje zaglavnih datoteka (engl. *additional include directories*) dodajte putanju do direktorija u kojem se nalazi *mpi.h* datoteka, primjerice C:\Program Files (x86)\Microsoft SDKs\MPI\Include. Nakon što ste dodali ovu putanju i potvrdili izmjene u postavkama (engl. *apply*), razvojna okolina ne bi smjela više pokazivati pogreške u programu. U ovom trenutku, zadovoljene su pretpostavke za uspješno prevođenje programa u objektnu datoteku.

Ako pokrenete prevođenje, primjetit ćete da i dalje postoje greške. U ovom trenutku greške javlja program povezivač (engl. *linker*) koji ne može povezati objekte datoteke prevedenog programa i izvršne biblioteke MPI funkcija u izvršnu datoteku. Razlog ovakvih grešaka jest činjenica da program povezivač ne zna gdje pronaći izvršne datoteke MPI funkcija. Vratite se ponovno u postavke projekta. U postavkama za

program povezivač, u polje dodatnih direktorija biblioteka (engl. *additional library directories*) dodajte putanju do izvršnih biblioteka MPI funkcija, primjerice C:\Program Files (x86)\Microsoft SDKs\MPI\Lib. Primjetit ćete da se unutar Lib direktorija nalaze direktoriji x86 i x64. Navedeni direktoriji sadrže prevedene implementacije MPI specifikacije za 32-bitnu i 64-bitnu arhitekturu. Odaberite postavke ulaza u program povezivač (engl. *input*) i u polje dodatnih međuovisnosti (engl. *additional dependencies*) odaberite relativnu putanju do one izvršne biblioteke koja je prilagođena arhitekturi za koju ćete prevesti program. Ako prevodite za 32-bitnu arhitekturu, tada je putanja x86/msmpi.lib, a u suprotnom je x64/msmpi.lib.

Nakon što ste izmijenili postavke za program povezivač, možete uspješno stvoriti prvi MPI izvršni program.
Nakon što stvorite izvršni MPI program, pokrenite 5 MPI procesa koji će ispisivati poruku pozdrava svijetu:

```
C:\Users\PP\Desktop> mpiexec.exe -n 5 .\HelloWorld.exe
```

U tom slučaju pojavio se sljedeći ispis:

```
Hello world from processor PP, rank 2 out of 5 processors
Hello world from processor PP, rank 3 out of 5 processors
Hello world from processor PP, rank 1 out of 5 processors
Hello world from processor PP, rank 4 out of 5 processors
Hello world from processor PP, rank 0 out of 5 processors
```

Ako na operacijskom sustavu Windows koristite drugu razvojnu okolinu, tada u razvojnoj okolini koju koristite morate podeciti putanje do zaglavne MPI datoteke (mpi.h) i do izvršne biblioteke MPI specfikacije (msmpi.lib).

Ako ste instalirali neku drugu implementaciju MPI specifikacije, tada provjerite jesu li ispravno postavljene varijable okruženja i ispravno postavite prethodno navedene putanje u razvojnoj okolini koju koristite.

Ako na operacijskom sustavu Windows koristite neki drugi programski jezik (Python, Java, C#), tada morate preuzeti MPI biblioteke prilagođene za navedene programske jezike.

1.1.2. Linux i OS X

Za potrebe rada na ovim operacijskim sustavima, preporuča se koristiti implementacije MPICH ili OpenMPI. Na stranicama dotičnih implementacija dostupne su upute

za postavljanje radne okoline, ovisno o odabranom programskom jeziku i razvojnoj okolini.

1.2. Primjeri

Hello world

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);

  // Get the number of processes
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // Get the rank of the process
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  // Get the name of the processor
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name, &name_len);

  // Print off a hello world message
  printf("Hello world from processor %s, rank %d"
    " out of %d processors\n",
    processor_name, world_rank, world_size);

  // Finalize the MPI environment.
  MPI_Finalize();
}
```

Slanje i primanje poruka

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
```

```
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // We are assuming at least 2 processes for this task
  if (world_size < 2) {
    fprintf(stderr, "World␣size␣must␣be␣greater␣than␣1␣for␣%s\n",
        argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  int number;
  if (world_rank == 0) {
    // If we are rank 0, set the number to -1 and send it to
       process 1
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  }
  else if (world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
       MPI_STATUS_IGNORE);
    printf("Process␣1␣received␣number␣%d␣from␣process␣0\n",
       number);
  }
  MPI_Finalize();
}
```

2. Prva domaća zadaća

Zadatak prve domaće zadaće jest uporabom MPI-a izraditi simulaciju raspodijeljenog problema *n* filozofa.

Slično uobičajenoj inačici problema, model obuhvaća *n* filozofa koji sjede za okruglim stolom na kojemu se nalazi hrana. Na raspologanju je *n* vilica, od kojih za potrebe prehrane svaki filozof mora koristiti dvije. Svaki filozof koristi točno određene vilice (njegovu 'lijevu' i 'desnu'), tj. svaku pojedinačnu vilicu mogu koristiti samo dva susjedna filozofa. Za razliku od uobičajenog rješenja sa zajedničkim spremnikom, filozofi moraju biti implementirani kao procesi koji komuniciraju isključivo razmjenom poruka (raspodijeljena okolina). Iz istog razloga, vilice ne stoje na stolu nego se uvijek nalaze kod nekog od filozofa (procesa). Program se mora moći pokrenuti u proizvoljnom broju procesa ($n > 1$), a ispis je potrebno prilagoditi tako da svaki proces/filozof ispisuje promjene stanja uz uvlačenje teksta (*tabs*) proporcionalno indeksu procesa.

Algoritam 1 Proces filozofa *i*

misli (slučajan broj sekundi)
nabavi vilice
jedi (slučajan broj sekundi)

Primjer ispisa procesa 1 prikazan je u nastavku.

```
mislim
trazim vilicu (0)
jedem
```

Rješenje koje treba implementirati opisano je u radu [1] (poglavlje 4) te [2], a ukratko je objašnjeno u nastavku. Svaka od *n* vilica može biti čista ili prljava, te se u jednom trenutku može nalaziti samo kod jednog filozofa (naravno). Na početku, sve su vilice *prljave*. Također, vilice su na početku podijeljene tako da se svaka vilica, koju mogu dijeliti dva susjedna filozofa, nalazi kod filozofa s nižim rednim brojem

(indeksom procesa). Slijedom navedenog, filozof s indeksom 0 na početku ima dvije vilice, a filozof s indeksom $n-1$ niti jednu. Svi filozofi slijede ova pravila:

1. Filozof jede ako je gladan i ako ima obje vilice (bez obzira jesu li čiste ili prljave ;)
2. Nakon jela, obje korištene vilice postaju prljave.
3. Ako filozof želi jesti, šalje zahtjeve za vilicama koje nisu kod njega i čeka na odgovor.
4. Ako filozof ne jede, a postoji zahtjev za *prljavom* vilicom koja se nalazi kod njega, vilicu čisti i šalje je susjedu.
5. Svaki filozof pamti zahtjeve za vilicama koje je dobio od svojih susjeda.
6. Dok filozof misli, udovoljava svakom zahtjevu za vilicom koja je trenutno kod njega.

Iz navedenih pravila je vidljivo da filozof ne udovoljava zahtjevu za *čistom* vilicom - zahtjev će biti udovoljen tek kad vilica postane prljava (nakon jela). Isto tako, ukoliko filozof misli (trenutno nije gladan), obvezan je odmah (što prije) udovoljiti zahtjevima drugih filozofa. Predloženo programsko rješenje može se opisati pseudokodom 2.

Algoritam 2 Filozof *i*

```
Proces(i)
{ misli (slucajan broj sekundi);             // ispis: mislim
    i 'istovremeno' odgovaraj na zahtjeve!   // asinkrono, s povremenom provjerom
  dok (nemam obje vilice) {
    posalji zahtjev za vilicom;              // ispis: trazim vilicu (i)
    ponavljaj {
      cekaj poruku (bilo koju!);
      ako je poruka odgovor na zahtjev       // dobio vilicu
        azuriraj vilice;
      inace ako je poruka zahtjev            // drugi traze moju vilicu
        obradi zahtjev (odobri ili zabiljezi);
    } dok ne dobijes trazenu vilicu;
  }
  jedi;                                      // ispis: jedem
  odgovori na postojeće zahtjeve;            // ako ih je bilo
}
```

U zadanom algoritamskom rješenju, "istovremeno" odgovaranje na zahtjeve potrebno je izvesti tako da filozof povremeno provjerava postoji li neki zahtjev (neka

poruka) upućen njemu. Ova funkcionalnost može se postići funkcijom `MPI_Iprobe` koja trenutno (neblokirajuće) provjerava postoji li neka poruka upućena promatranom procesu (predavanja, poglavlje 2.8). Tek nakon što funkcija detektira postojanje dolazne poruke, ista se treba primiti s `MPI_Recv`. U našem primjeru, interval provjeravanja postojanja dolaznih poruka nije kritičan i može biti jednak 1 sekundu.

2.1. Predaja zadaće

Do datuma određenog na stranicama predmeta potrebno je predati komprimirane izvorne kodove rješenja domaće zadaće. U zip arhivi (ne rar) očekuju se isključivo izvorni kodovi.

3. Druga domaća zadaća

Zadatak druge domaće zadaće jest uporabom MPI-a ostvariti program za igranje uspravne igre "4 u nizu" (*connect 4*) za jednog igrača (čovjek protiv računala).

Igra je istovjetna igri križić-kružić u kojoj je cilj napraviti niz od 4 igračeva znaka, s tom razlikom da se odvija na 'uspravnom' 2D polju u kojemu se novi znak može staviti samo na polje ispod kojega već postoji neki znak ili se stavlja na dno polja (djeluje gravitacija). Standardne dimenzije igračeg polja su 6 polja u visinu i 7 u širinu, mada je veličina proizvoljna (minimalno 6x7).

Opis slijednog algoritma. Neka metoda rješavanja bude djelomično pretraživanje prostora stanja, u obliku stabla, do neke zadane dubine od trenutnog stanja. Dakle, ne pokušavamo naučiti strategiju, već se za svaki potez računala obavlja pretraga podstabla i odabire sljedeće stanje (*brute force* pristup). Za svako se stanje u stablu određuje vrsta:

- stanje je 'pobjeda' ako računalo ima 4 u nizu (definiramo vrijednost 1)
- stanje je 'poraz' ako igrač ima 4 u nizu (definiramo vrijednost -1)
- inače, stanje je neutralno, a vrijednost stanja će ovisiti o stanjima u podstablu (ako se podstabla pretražuju)

Potragu za 4 u nizu treba obaviti samo s polja posljednjeg odigranog poteza.

Nakon pretraživanja stabla do zadane dubine, primjenjuju se sljedeća rekurzivna pravila:

1. ako je neko stanje 'pobjeda' i ako se u njega dolazi potezom računala, tada je i nadređeno stanje također 'pobjeda' (jer računalo iz nadređenog stanja uvijek može pobijediti, potezom koji vodi u stanje pobjede);

2. ako je stanje 'poraz' i ako se u njega dolazi potezom igrača (protivnika), tada je i nadređeno stanje 'poraz' (jer iz nadređenog stanja igrač može jednim potezom pobijediti);

3. ako su sva podstanja nekog stanja 'pobjeda' ili 'poraz', tada je i nadređeno stanje iste vrste

Osim ovim pravilima, svaki se mogući potez računala (odnosno stanje u koje se tim potezom dolazi) ocjenjuje promatranjem broja i dubine pobjedničkih stanja u podstablu u koje taj potez vodi. Mjera kvalitete poteza (jednog podstabla) definira se rekurzivno kao zbroj vrijednosti neposrednih podstanja, podijeljen s brojem mogućih poteza na toj dubini (brojem podstanja), odnosno:

$$vrijednost\ stanja = (zbroj\ vrijednosti\ moguih\ poteza)/(broj\ moguih\ poteza) \tag{3.1}$$

Izraz 3.1 može biti zamijenjen i nekom drugom, boljom ocjenom poteza.

Broj mogućih poteza se može smatrati konstantnim (jednakim brojem stupaca, npr. 7) uz pretpostavku o neograničenosti polja u visinu, a u stvarnim uvjetima potrebno je uzeti u obzir samo moguće poteze (ako je neki stupac popunjen). Računalo tada odabire onaj potez koji ne vodi u stanje 'poraz' (ako ima izbora) a koji ima najveću vrijednost (vrijednosti su po opisanoj definiciji u intervalu $[-1, 1]$). Eventualna dodatna pojašnjenja dana su na predavanjima.

Ostvarenje paralelnog algoritma. Program treba imati minimalno tekstno sučelje u obliku prikaza stanja polja i upita igrača o potezu. Računanje vrijednosti pojedinog poteza treba načiniti raspodijeljeno, a minimalna dubina pretraživanja stabla n je 4 (složenost je 7^n). Minimalni broj zadataka paralelnog algoritma je stoga broj mogućih poteza (7), no taj broj je potrebno povećati (npr. dijeljenjem pri većoj dubini) poradi boljeg ujednačavanja opterećenja po procesorima. **Algoritam u kojemu postoji stalan broj od najviše 7 zadataka (radnika) nije prihvatljiv!**

3.1. Predaja zadaće

Do datuma određenog na stranicama predmeta potrebmo je predati komprimirane (zip format) izvorne kodove rješenja domaće zadaće i dokumentaciju u pdf formatu.

Dokumentacija se sastoji od sljedećih djelova:

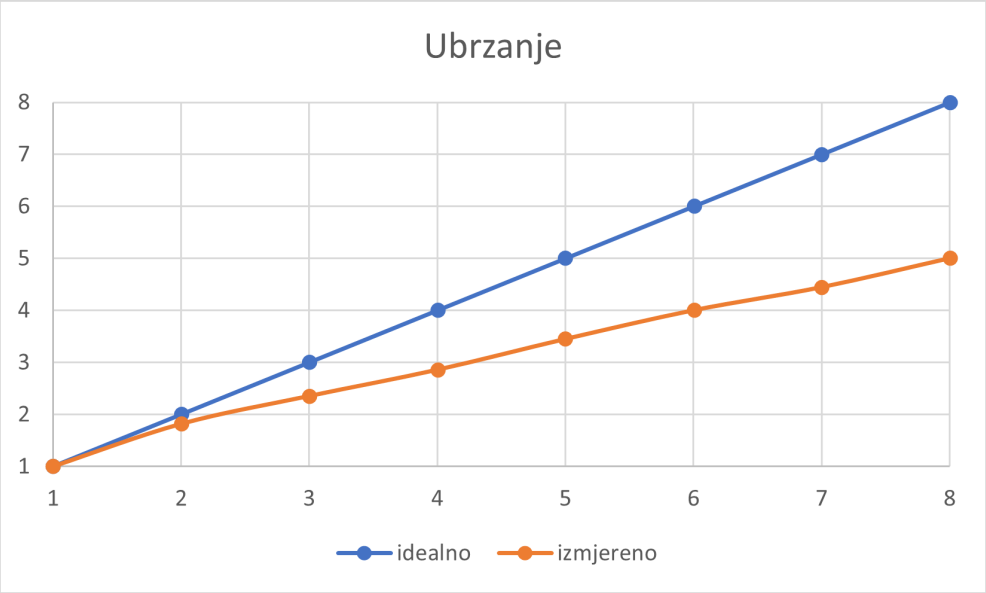
- opis četiri faze razvoja: podjela, komunikacija, aglomeracija i pridruživanje u vašem rješenju
- empirijski utvrditi *ubrzanje i učinkovitost* algoritma (definicija na predavanju) mjerenjem trajanja programa (trajanje jednog poteza računala na samom početku igre) na broju procesora $P = 1, .., 8$ (minimalno). Mjerenje je potrebno

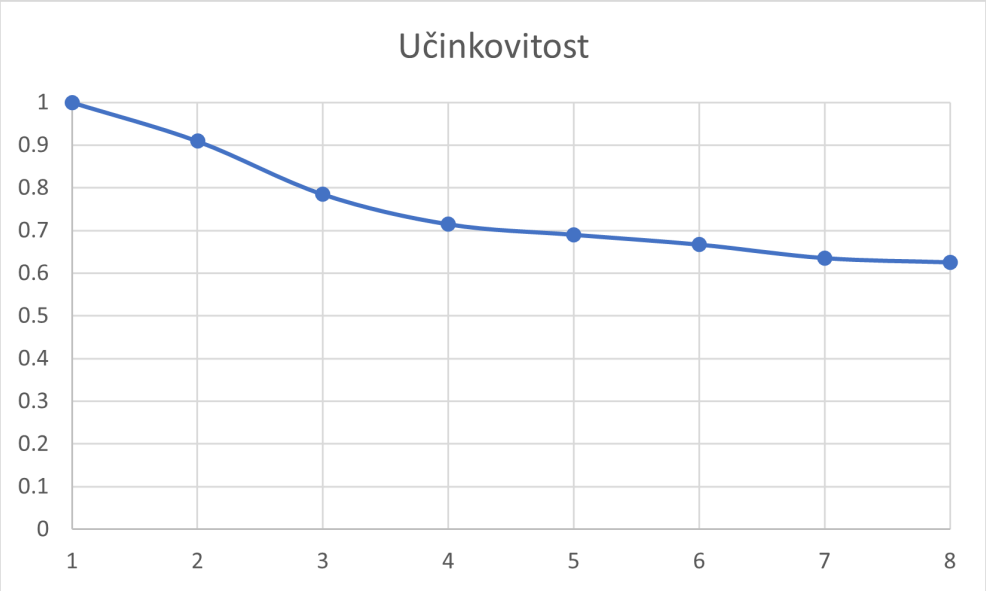
obavljati na početku igre kada su uvjeti jednaki, a rezultate pripremiti u elektroničkom obliku, grafički i tablično. Mjerenja treba provesti tako da je najmanje mjereno trajanje (za 8 ili najveći broj procesora) reda veličine barem nekoliko sekundi (odredite potrebnu dubinu pretraživanja). Za mjerenje je potrebno 8 procesora odnosno 8 jezgara (*hyperthreading* se u pravilu ne broji!). **Dobivene rezultate potrebno je komentirati (obrazložiti).**

Primjerice, ako su rezultati mjerenja sljedeći:

broj procesora (P)	1	2	3	4	5	6	7	8
trajanje (s)	20	11	8.5	7	5.8	5	4.5	4

prikaz ubrzanja i učinkovitosti izgledao bi ovako:




Učinkovitost
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8

4. Literatura

[1] K. M. Chandy i J. Misra. The drinking philosophers problem. *ACM Trans. Program. Lang. Syst.*, 6(4):632–646, Listopad 1984. ISSN 0164-0925. doi: 10.1145/1780.1804. URL http://www.fer.unizg.hr/_download/repository/Philosophers.pdf.

[2] Adrian Colyer. The drinking philosophers problem, Jun 2015. URL https://blog.acolyer.org/2015/06/11/the-drinking-philosophers-problem/.