# Multiprocesorski sistemi (SI4MPS, IR4MPS, MS1MPS)

## Laboratorijska vežba 1 - OpenMP

Cilj laboratorijske vežbe je upoznavanje studenata sa korišćenjem OpenMP tehnologije na platformama Windows i Linux i izrada, prevođenje i pokretanje jednostavnog programa koji koristi OpenMP paralelizaciju.

Rad sa OpenMP će biti prikazan na primeru GCC prevodioca koji potpuno podržava ovu tehnologiju. Sledeća tabela prikazuje verzije GCC prevodioca koje podržavaju različite OpenMP standarde.

| OpenMP standard | GCC verzija | Glavna poboljšanja |
|---|---|---|
| 2.5 | 4.2 | Kombinovana C/C++/Fortran specifikacija |
| 3.0 | 4.4 | Podrška za OpenMP poslove (**`tasks`**), paralelizacija ugneždenih petlji |
| 3.1 | 4.7 | Poboljšanja OpenMP poslova, novi redukcioni operatori, izmenjen **`atomic`** konstrukt |
| 4.0 | 4.9 | Podrška za SIMD izvršavanje i akceleratore |

### Korišćenje OpenMP sa GCC prevodiocem na Windows i Linux

OpenMP je dostupan na Windows i Linux operativnim sistemima uz odgovarajući GCC prevodilac. Prilikom prevođenja, potrebno je navesti opciju **`–fopenmp`**, a prevođenje i povezivanje programa se može izvršiti pomoću sledeće komandne linije:

- Programski jezik C: **`gcc -fopenmp -lm -o dz1z1.exe dz1z1.c`**
- Programski jezik C++: **`g++ -fopenmp -lm -o dz1z1.exe dz1z1.cpp`**

Po potrebi, radi povećanja performansi, prevodiocu se može proslediti opcija **`-Ox`**, gde **`x`** označava željeni stepen optimizacije koda. Uobičajeno se koristi nivo optimizacije **`-O3`**.

Na rtidev5 računaru je dostupna verzija 4.6.3 GCC prevodioca. Ona podržava OpenMP standard 3.0 i potrebno je koristiti prilikom izrade zadataka, jer omogućava rad sa poslovima (**`tasks`**). Prevodilac se nalazi u direktorijumu **`/usr/bin/`** i pomoću sistemskih putanja je dostupan na celom sistemu. Verzija prevodioca se uvek može proveriti opcijom --version:

```
gcc --version
```

Na Windows operativnom sistemu, GCC prevodilac je dostupan u okviru Cygwin okruženja (http://www.cygwin.com/) koje simulira Linux komandno okruženje bash. Trenutno dostupna verzija je 4.8.4. Alternativno, može se koristiti GCC implementacija u okviru MinGW-w64 projekta koja donosi GCC verziju 4.8.

Prilikom izvršavanja programa, broj niti se može zadati eksplicitno u okviru programa korišćenjem **`omp_set_num_threads()`** funkcije ili se može podesiti postavljanjem promenljive okruženja **`OMP_NUM_THREADS`**.

Promenljiva se može postaviti eksplicitno za sva naredna izvršavanja OpenMP programa sa naredbom:

```
export OMP_NUM_THREADS=4
```

Alternativno, promenljiva **OMP_NUM_THREADS** se može postavljati za svako izvršavanje programa navođenjem ispred imena programa prilikom pokretanja

```
OMP_NUM_THREADS=4 ./ime_programa
```

## Podešavanja okruženja Visual Studio

OpenMP se može koristiti u okviru okruženja Visual Studio 2012. Međutim, Visual C++ prevodilac podržava samo standard 2.0, pa se stoga ne preporučuje njegovo korišćenje za izradu domaćih zadataka, jer ne podržava koncept poslova.

Pre početka rada, treba podesiti Visual Studio tako da radi sa VC++ podešavanjima. U dijalogu Tools->Import and Export Settings... izabrati "Reset All Settings", na sledećem izabrati "No" i na poslednjem izabrati "Visual C++ Development Settings".

Najpre je potrebno napraviti novi Visual Studio projekat, tipa "Win32 Console Application" (File->New Project->Visual C++ Project->General->Empty Project). Zatim dodati jedan novi izvorni fajl. U C/C++ podešavanjima je potrebno u sekciji Languages podes podesiti OpenMP support opciju na Yes (/openmp). Visual Studio je sada podešen za prevođenje OpenMP programa bez ikakvih dodatnih podešavanja.

Prilikom izvršavanja programa, broj niti se može zadati eksplicitno u okviru programa korišćenjem **omp_set_num_threads()** funkcije ili se može podesiti postavljanjem promenljive okruženja **OMP_NUM_THREADS**. U okviru Debugging sekcije podešavanja projekta, u polje Environment je potrebno upisati **OMP_NUM_THREADS=broj_niti**.

## Debbuging multithreaded aplikacija koristeći Visual Studio

Proces debagovanja se vrši kroz prozor Threads do koga se može doći klikom na Debug->Windows->Threads kada je aktivna debug sesija. U datom prozoru je moguće izabrati određenu nit za izvršavanje (Switch To Thread opcija), moguće je zamrznuti izvršavanje ostalih niti (opcija Freeze). Takođe, moguće je svakoj niti dati ime i označiti je odgovarajućom bojom ili zastavicom, što olakšava praćenje izvršavanja.

Više detalja o celom procesu za debugging na Visual Studio se može pronaći u dokumentu na adresi http://msdn.microsoft.com/en-us/library/ms164746%28v=VS.90%29.aspx.

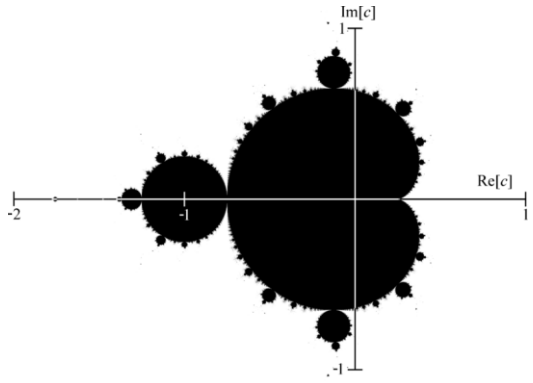
## Zadatak 1

Sastaviti program koji ispisuje „Hello World!" korišćenjem OpenMP tehnologije. Svaka nit treba da ispiše pozdravnu poruku i svoj jedinstveni identifikacioni broj. Za dohvatanja identifikacionog broja, koristiti int **omp_get_thread_num()** funkciju.

Izvorni kod programa je dat u Dodatku A ovog dokumenta.

## Zadatak 2

Paralelizovati program koji izračunava površinu koju obuhvata Mandlebrotov skup (fraktal) korišćenjem OpenMP tehnologije. Mandlebrotov skup je skup kompleksnih brojeva *c* za koji iteracija $z = z^2 + c$ ne divergira, kada se zada početni uslov $z = c$. Da bi se približno ustanovilo da li se neka tačka *c* nalazi u skupu, izvršava se konačan broj iteracija. Ukoliko je uslov $|z| > 2$ ispunjen, onda se smatra da se tačka nalazi izvan Mandlebrotovog skupa.




Mandlebrotov skup (crno) u kompleksnoj ravnini

Ne postoji teoretska formula za izračunavanje površine Mandlebrotovog skupa, već se ta vrednost može odrediti simulacijom. Mandlebrotov skup je simetričan, pa je dovoljno izračunati površinu gornje polovine. Površina se može izračunati generisanjem mreže tačaka u prozoru u kompleksnoj ravni. Prozor obuhvata tačke [-2.0, 0.5] po *x* osi i [0, 1.25] po *y* osi. Nakon generisanja, svaka tačka se iterira korišćenjem zadate formule konačan broj puta (npr. 2000 puta). Ukoliko je nakon iteriranja uslov $|z| > 2$ ispunjen, onda se smatra da se tačka nalazi izvan Mandlebrotovog skupa. Procena površine koju obuhvata skup se na kraju može izvršiti određivanjem odnosa broja tačaka koji se nalazi u skupu i ukupnog broja generisanih tačaka.

Izvorni kod programa je dat u Dodatku B ovog dokumenta. Paralelizacija se može izvršiti na sledeći način:

1. Započeti **`parallel`** region pre početka glavne petlje.
2. Obezbediti ispravno deklarisanje svih deljenih, privatnih i redukcionih promenljivih korišćenjem **`shared`**, **`private`** i **`reduction`** odredbi. Dodavanje odredbe **`default(none)`** može pomoći u izbegavanju grešaka u ovom koraku.
3. Distribuirati iteracije spoljne petlje po nitima, tako da svaka nit dobije jednak broj tačaka na obradu. Koristiti funkciju **`omp_get_thread_num()`**.

## Zadatak 2

Izmeriti i ispisati vreme izvršavanja sekvencijalnog i OpenMP programa korišćenjem **`omp_get_wtime()`** funkcije. Da bi se izmerilo vreme izvršavanja dela programa, potrebno je izmeriti vreme u početnoj i krajnjoj tački i odrediti njihovu razliku. Eksperimentisati sa alatom **`time`** iz Linux bash komandog okruženja za merenje vremena celog programa.

## Zadatak 4

Rešiti Zadatak 2 korišćenjem direktiva za podelu posla (*worksharing* direktiva). Paralelizacija se može izvršiti na sledeći način:

1. Započeti **`parallel`** region pre početka glavne petlje.
2. U okviru **`parallel`** regiona, spolju petlju paralelizovati korišćenjem **`for`** direktive. Direktivu **`for`** je moguće zadati u istoj liniji kao i **`parallel`** direktivu ili se može zadati u posebnom redu.
3. Obezbediti ispravno deklarisanje svih deljenih, privatnih privatnih i redukcionih promenljivih korišćenjem **`shared`**, **`private`** i **`reduction`** odredbi. Dodavanje odredbe **`default(none)`** može pomoći u izbegavanju grešaka u ovom koraku.
4. Dodati **`schedule`** odredbu i eksperimentisati sa različitim načinima za raspoređivanje iteracija petlje, kao što su **`static`**, ciklični **`static`**, **`dynamic`**, **`guided`** i slično.

## Zadatak 5

Rešiti Zadatak 2 korišćenjem poslova (*tasks*). Obratiti pažnju da se svaka tačka može obraditi nezavisno od ostalih tačaka i iskoristiti tu činjenicu za generisanje poslova. Takođe, obratiti pažnju na deklarisanje promenljivih i njihov opseg vidljivosti.

## Mali OpenMP podsetnik

```
int omp_get_thread_num(void);
int omp_get_num_threads(void);
int omp_get_num_procs();
double omp_get_wtime(void);
double omp_get_wtick(void);
```

## Dodatak A – HelloWorld program

```
// HelloWorld.c

#include <stdio.h>

#define NUM_THREADS 3

int main(int argc, char *argv[]){

    printf("Zdravo svete! \n");

    return 0;
}
```

## Dodatak B – Mandlebrot skup

```
// area.c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

# define NPOINTS 2000
# define MAXITER 2000

struct complex{
  double real;
  double imag;
};

int main(){
  int i, j, iter, numoutside = 0;
  double area, error, ztemp;
  struct complex z, c;

/*
 *     Outer loops run over npoints, initialise z=c
 *
 *     Inner loop has the iteration z=z*z+c, and threshold test
 */

  for (i=0; i<NPOINTS; i++) {
    for (j=0; j<NPOINTS; j++) {
      c.real = -2.0+2.5*(double)(i)/(double)(NPOINTS)+1.0e-7;
      c.imag = 1.125*(double)(j)/(double)(NPOINTS)+1.0e-7;
      z=c;
      for (iter=0; iter<MAXITER; iter++){
        ztemp=(z.real*z.real)-(z.imag*z.imag)+c.real;
        z.imag=z.real*z.imag*2+c.imag;
        z.real=ztemp;
        if ((z.real*z.real+z.imag*z.imag)>4.0e0) {
          numoutside++;
          break;
        }
      }
    }
  }

/*
 *  Calculate area and error and output the results
 */

      area=2.0*2.5*1.125*(double(NPOINTS*NPOINTS-
numoutside)/(double)(NPOINTS*NPOINTS);
      error=area/(double)NPOINTS;

      printf("Area of Mandlebrot set = %12.8f +/- %12.8f\n",area,error);
}
```