Multiprocesorski sistemi (SI4MPS, IR4MPS, MS1MPS)

Laboratorijska vežba 2 - MPI

Cilj laboratorijske vežbe je upoznavanje studenata sa korišćenjem MPI okruženja na platformama Windows i Linux i izrada, prevođenje i debagovanje jednostavnog MPI programa.

Korišćennje MPI okruženja na rtidev5 računaru

Za rad sa MPI bibliotekom na računaru rtidev5 je dostupna OpenMPI implementacija, verzije 1.5.4. Prevođenje i povezivanje programa se može izvršiti pomoću sledeće komandne linije:

- Programski jezik C: mpicc -lm -o dz3z1.exe dz3z1.c
- Programski jezik C++: mpic++ -lm -o dz3z1.exe dz3z1.cpp

Pokretanje programa:

- mpirun -np 4 ./program_name
- mpiexec -np 4 ./program_name

Podešavanje MS MPI okruženja na operativnom sistemu Windows

Za rad sa MPI bibliotekom na Windows platformi može se koristiti Micrsoft MPI implementacija, koja se može preuzeti na adresi http://www.microsoft.com/en-us/download/details.aspx?id=36045. Ova implementacija je zasnovana na MPICH implementaciji i deo je Microsoft HPC pack softvera. Podržava jezike C i Fortran i radi na Win2000/XP/Vista/7 i Win Server 2003. Postoje 32-bit i 64-bit verzija. Jezik C++ nije podržan. Pre instalacije MPI okruženja, potrebno je instalirati odgovarajuću verziju Microsoft Visual C++ Runtime biblioteke (MSVCRT) sa adrese http://www.microsoft.com/en-us/download/details.aspx?id=5555 (32bit) ili http://www.microsoft.com/en-us/download/details.aspx?id=14632 (64bit).

Na nekim računarima firewall može biti podešen tako da sprečava sve ili neke aktivnosti MPI komunikacije. Ako je moguće, podesiti firewall tako da ne sprečava komunikaciju, odnosno da ne obaveštava korisnika o izvršenom sprečavanju. Na laboratorijskim mašinama je firewall trenutno podešen tako da ne ometa suštinski MPI komunikaciju.

Podešavanja okruženja Visual Studio za rad sa MS MPI

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". Neke implementacije zahtevaju da mpi.h zaglavlje bude uključeno pre ulazno/izlaznih zaglavlja (stdio.h, odnosno iostream). Zato je najjednostavnije postaviti mpi.h kao prvo uključeno zaglavlje.

Podešavanje projekta za uspešno prevođenje

C/C++ podešavanja:

1. U sekciji Build->Configuration Manager podesiti 32-bitnu ili 64-bitnu konfiguraciju projekta, u zavisnosti koja verzija MS MPI je instalirana. Ukoliko je instalirana 64-bitna verzija, 64-bitna podešavanja za prevođenje projekta se postavljaju u sekciji Active solution platform->New->x64.
2. Dodati "C:\Program Files\Microsoft HPC Pack 2012\Inc" u General->Additional Include Directories. Putanju podesiti tako da odgovara instalaciji MS MPI.

3. Podesiti Advanced->Compile As na "Compile as C Code (/TC)" ili "Compile as C++ Code (/TP)", zavisno od jezika u kome je izvorni programski kod. Ovo nije neophodno ako izvorni fajlovi imaju ekstenziju .c, odnosno .cpp, pošto za fajlove sa tim ekstenzijama VS podrazumeva da sadrže programski kod na jeziku C, odnosno C++.
4. Po želji, dodati _CRT_SECURE_NO_DEPRECATE u Preprocessor->Preprocessor Definitions radi izbegavanja nepotrebnih upozorenja.

Podešavanja linkera:

1. Dodati "C:\Program Files\Microsoft HPC Pack 2012\Lib" u General->Additional Library Directories.
   Putanju podesiti tako da odgovara instalaciji DeinoMPI.
2. Dodati "msmpi.lib" u Input->Additional Dependencies.

Debbuging paralelnih aplikacija koristeći Visual Studio

Za efikasan debugging, potrebno je koristiti odgovarajući debbugger na odgovarajući način.

- Za sve MPI koje mogu biti korišćene implementacije važi da u okruženju treba ISKLJUČITI opciju Tools->Options->Debugging->Break all processes when one process breaks, da bi kontrola izvršavanja svih procesa radila kako treba.
- Procese je potrebno pokrenuti sa opcijom Start debugging (prečica F5) i postaviti Breakpoint tamo gde je potrebno da procesi budu zaustavljeni.
- Uključiti prozor/karticu Debug->Windows->Processes
- Procese treba kontrolisati ISKLJUČIVO komandama iz prozora Debug->Windows->Processes, NIKAKO uobičajenim komandama za debugging.

Debbuger je pre korišćenja potrebno podesiti za svaki nalog i projekat. Visual Studio drži podešavanja projekta odvojeno od podešavanja za debugging, pa je potrebno za svaku kombinaciju računar/nalog napraviti podešavanja za debugging prilikom prvog rada u datoj kombinaciji. Podešavanja vezana sa sam projekat (C/C++ i Linker) NE TREBA ponovo podešavati, pošto su već snimljena u samim projektima sadržanim u rešenju.

U debugging podešavanjima projekta izvršiti sledeća podešavanja, vodeći računa o ispravnim putanjama:

1. Promeniti Debugger to Launch na "MPI Cluster Debugger".
2. Za MPIRun Command postaviti putanju do MPI izvršnog okruženja (tipično: " C:\Program Files\Microsoft HPC Pack 2012\Bin", ovde su navodnici neophodni).
3. MPIRun Arguments postaviti na "-np X", gde je X broj MPI procesa koje želimo da pokrenemo (obavezno bez navodnika).
4. Za MPIShim Location postaviti putanju na, tipično, "C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\Remote Debugger\x64\mpishim100.exe" (obavezno bez navodnika). MPIShim je Visual Studio komponenta potrebna za paralelni debugging.

Više detalja o celom procesu za debugging na Visual Studio se može pronaći u dokumentu "Parallel Debugging Using Visual Studio 2005/2008/2010".

Zadatak 1

Paralelizovati program koji ispisuje „Hello World!" korišćenjem MPI tehnologije. Svaki process treba da ispiše pozdravnu poruku, nakon koje će ispisati rang svog procesa i ukupan broj procesa u MPI svetu. Izvorni kod programa je dat u Dodatku A ovog dokumenta [1,N]

Zadatak 2

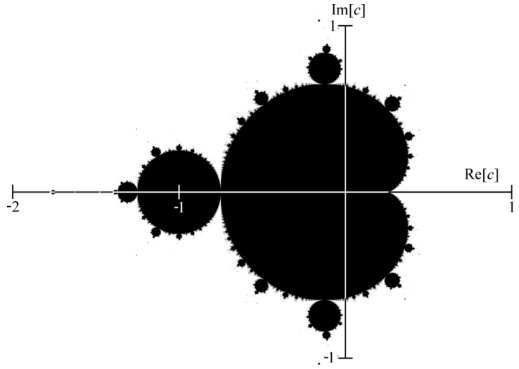
Modifikovati prethodni program, tako da proces sa rangom 0 učitava broj, pa taj broj šalje ostalim procesima, koji ispisuju njegovu neizmenjenu vrednost. Koristiti MPI pozive **Send** i **Receive**. [2,N]

Zadatak 3

Modifikovati prethodni program, tako da proces sa rangom 0 učitava broj, pa taj broj šalje ostalim procesima, koji ispisuju njegovu neizmenjenu vrednost. Koristiti MPI poziv **Bcast**. [1,N]

Zadatak 4

Paralelizovati program koji izračunava površinu koju obuhvata Mandelbrotov skup (fraktal) korišćenjem MPI tehnologije. Proces gospodar (master) treba da učita broj tačaka za obradu, kao i maksimalni broj iteracija za ispitivanje uslova divergencije. Mandelbrotov 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 Mandelbrotovog skupa.



Mandelbrotov skup (crno) u kompleksnoj ravnini

Ne postoji teoretska formula za izračunavanje površine Mandelbrotovog skupa, već se ta vrednost može odrediti simulacijom. Mandelbrotov 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 Mandelbrotovog 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 distribucijom iteracija spoljne petlje po procesima na sledeći način:

1. Obezbediti da proces-gospodar učita podatke o ukupnom broju tačaka za obradu i maksimalnom broju iteracija. Ako broj iteracija na odgovara broju MPI procesa izvršiti zaokruživanje na gore na prvu odgovarajuću vrednost.
2. Proslediti svim procesima broj tačaka za obradu i maksimalan broj iteracija korišćenjem Bcast poziva.
3. Modifikovati soljni **`for`** ciklus tako da koristi dobijeni broj iteracija.
4. Izvršiti sumiranje broja tačaka koje ne zadovoljavaju uslov Mandelbrotovog skupa po procesima korišćenjem **`Reduce`** operacije.
5. Izračunati i ispisati rezultat u procesu-gospodaru.

Zadatak 5

Sastaviti program koji ispisuje ukupan broj parnih i neparnih brojeva u nizu celih brojeva. Proces sa rangom 0 treba da učita niz, raspodeli ga podjednako svim procesima na obradu i ispiše konačan rezultat. Koristiti MPI pozive **`Scatter`** i **`Gather`**. [1,N]

Mali MPI C podsetnik

```
int MPI_Init(int *argc, char ***argv);
int MPI_Finalize();
int MPI_Comm_rank (MPI_Comm comm, int *rank);
int MPI_Comm_size (MPI_Comm comm, int *size);
int MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm);
int MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm, MPI_Status *status);
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
               int root, MPI_Comm comm);
int MPI_Gather(void* sendbuf,int sendcount,MPI_Datatype sendtype, void* recvbuf,
                int recvcount,MPI_Datatype recvtype,int root,MPI_Comm comm);
int MPI_Scatter(void* sendbuf,int sendcount,MPI_Datatype sendtype,void* recvbuf,
                int recvcount,MPI_Datatype recvtype,int root,MPI_Comm comm);
int MPI_Reduce(void* sendbuf,void* recvbuf,int count, MPI_Datatype datatype,
                MPI_Op  op,int root,MPI_Comm comm);
```

Dodatak A – Kostur HelloWorld programa

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

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

        /* Inicijalizujemo MPI */
        ...

        /* Trazimo svoj rang unutar MPI sveta (prvi je 0) */
        ...

        /* Trazimo velicinu MPI sveta */
        ...

        /* Ispisujemo poruku na standardnom izlazu */
        printf("Zdravo svima! Ja sam broj %d od ukupno %d\n", rank, size);

        /* Zavrsavamo MPI */
        ...

        return 0;
}
```

Dodatak B – Mandelbrot skup

```
// area.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct compl {
  double real;
  double imag;
};
int main(){
  int i, j, iter, numoutside = 0;
  int npoints, maxiter;
  double area, error, ztemp;
  struct compl z, c;
  // Should be done by the master only
  printf("Number of points? "); scanf("%d", &npoints);
  printf("Maximum number of iterations? ");  scanf("%d", &maxiter);

  // Calculate & broadcast number of points for every process
  // Broadcast maximum number of iterations

/*  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;
      }
      }
    }
```

```
  }

  // Reduce numoutside variable

/*
 *  Calculate area and error and output the results in the MASTER process
 */

      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);
}
```