Parallel Programming: Message-Passing-Interface

MPI-Einführung

Voraussetzungen im Source-Code für ein MPI Programm:

- mpi.h includen
- Die Kommandozeilenparameter des Programms müssen an *MPI_Init* übergeben werden (die Parameter werden automatisch von dem Run-Script *mpirun* zugewiesen). *MPI_Init* muß vor allen anderen MPI-Funktionen aufgerufen werden:

```
MPI_Init(&argc,&argv);
```

- Nach der letzten MPI-Funktion muß der Befehl *MPI_Finalize* aufgerufen werden, um der MPI-Umgebung das Programmende mitzuteilen. Nach dem Befehl *MPI_Finalize* darf keine andere MPI-Funktion mehr aufgerufen werden:

```
MPI_Finalize();
```

- Jeder Prozess kann mittels MPI-Funktionen feststellen, wieviele Prozesse es insgesamt gibt (*MPI_Comm_size*) und seine eindeutige Nummer ermitteln (*MPI_Comm_rank*):

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

- Der Standard-Kommunikator ist *MPI_COMM_WORLD*. Es ist jederzeit möglich, neue Kommunikatoren zu erstellen und verschiedene Topologien auf diese Kommunikatoren zu mappen.

```
//////////////////////////////////////////////////////////////////////
// A very short MPI test routine to show the involved hosts
//////////////////////////////////////////////////////////////////////
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  int size,rank;
  char hostname[50];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank); //How many processes?
  MPI_Comm_size(MPI_COMM_WORLD, &size); //Who am I?
  gethostname (hostname, 50);
   printf ("Hello World! I'm number %2d of %2d running on host %s\n",
rank, size, hostname);
  MPI_Finalize();
  return 0;
}
```

Message-Passing-Interface

Kompilieren/Starten eines MPI-Programmes

Einlesen des Source-Scriptes für MPI:

```
pcatoll03 # source /opt0/Administration/common/mpi_path
Setting up MPI-enviroment for Linux
```

Kompilieren des Programmes (anstatt von gcc):

```
pcatoll03 # mpicc -Wall prog1.c -o prog1
```

Starten des Programmes:

```
pcatoll03 # mpirun -np 4 prog1
Hello World! I'm number  0 of  4 running on host pcatoll03
Hello World! I'm number  2 of  4 running on host pcatoll02
Hello World! I'm number  1 of  4 running on host pcatoll01
Hello World! I'm number  3 of  4 running on host pcatoll01
```

Beachte:

- Angabe der Anzahl Prozesse mit dem Parameter *'-np <Anzahl>'*
- Reihenfolge der Ausgabe!
- Ohne Angabe eines *machinefiles* wird das Standard-File verwendet
- MPI basiert auf rsh, einloggen ohne Angabe des Passworts ist mit *.rhosts* möglich

Beispiel für *.rhosts*:

```
+ <user>   # Für User <user> Zugang von allen Rechnern aus erlauben
```

oder

```
<host> <+|-> <user> # Für Rechner <host> User <user> zulassen/sperren
```

Beispiel für ein *Machinefile*:

```
pcatoll04
pcatoll06
```

Aufruf mit individuellen *Machinefile*:

```
pcatoll03 # mpirun -np 4 -machinefile machinefile prog1
Hello World! I'm number  0 of  4 running on host pcatoll03
Hello World! I'm number  2 of  4 running on host pcatoll06
Hello World! I'm number  1 of  4 running on host pcatoll04
Hello World! I'm number  3 of  4 running on host pcatoll04
```

Beachte:

- Der Node auf dem das Programm gestartet wird, ist immer mit beteiligt (hier: pcatoll03) und gleichzeitig der Root-Prozess (id=0).
- Die restlichen benötigten Nodes (hier 3) werden aus dem machinefile gelesen, gegebenfalls werden auf einem Rechner mehrere Prozesse gestartet.
- So kann man z.B. SMP-Rechner besonders gut ausnutzen, indem man in dem machinefile den entsprechenden Node-Name mehrmals einträgt.

Message-Passing-Interface

Beispiel für eine Kommunikation mit MPI

```
//////////////////////////////////////////////////////////////////////
//// A very short MPI test routine to test message passing
//////////////////////////////////////////////////////////////////////
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
  int i, size, rank, signal;
  MPI_Status status;
  double starttime, endtime;

  //Initialisation of MPI
  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);


  //Master: send to every proc and wait for answer
  if (rank == 0) {
    signal = 666;
    starttime=MPI_Wtime();

    for (i=1; i<size; i++) {
      MPI_Send(&signal, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
      printf ("%d: Sent to %d\n", rank, i);
      MPI_Recv (&signal, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &status);
      printf ("%d: Received from %d\n", rank, i);
    }

    endtime=MPI_Wtime();
    printf ("%d: Time passed: %f\n", rank, endtime-starttime);
  }

 //Clients: wait for message from master and send back
  else {
     MPI_Recv (&signal, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
&status);
    printf ("%d: Received from %d\n", rank, status.MPI_SOURCE);

    MPI_Send(&signal, 1, MPI_INT, status.MPI_SOURCE, 0, MPI_COMM_WORLD);
    printf ("%d: Sent to %d\n", rank, status.MPI_SOURCE);
  }

  //Deinitialisation of MPI
  MPI_Finalize();
  return 0;
}
```