数値計算演習
07.12.05
--並列環境(mpich)の使い方--

- 課題
  - 【課題1】並列環境MPICHを使って，台形近似にもとづいてpiの値を求めよ．
  - 【課題2(レポート：提出は来週以降でよい)】時間のかかる計算(例えば，mandelbrot)を並列環境に移植し，計算時間をもとめよ．また，以下について考察せよ．
    - 【１】台形近似で-np 4としても計算速度が上がらないのはなぜか．
    - 【２】台形近似の刻み幅を細かくしていくと，精度が落ちるのはなぜか．どうすれば回避できるか．
    - 【３】どのような問題に対して，並列化の効率がいいのか．
- cpi
  - 先ずはmpichについているサンプルが動くかを確認する．
  - bob1 /home/bob> /usr/local/mpich/bin/mpirun -np 2 /usr/local/mpich/examples/cpi
    - と打ち込んでみて，
  - Process 0 on bob1.site
    Process 1 on bob2.site
    pi is approximately 3.1416009869231241, Error is 0.0000083333333309
    wall clock time = 0.000000
    - となれば成功．Process 0 と1で並列計算されている．
- hello.c
  - 次に，もっとも簡単なプログラムである"Hello world!\n"を並列化する．MPIではprocess間でmessageをやり取りして，計算を進める．以下のプログラムでは，my_rank!=0なら子プロセスとなり，親(dest=0)に挨拶(message)を送る．親(my_rank==0)は挨拶を子プロセスの数(p)だけ受け取って，表示する．
  - ```
    #include <stdio.h>
    #include <string.h>
    #include "mpi.h"

    main (int argc, char* argv[]){
        int my_rank,p,source,dest,tag=0;
        char message[100];
        MPI_Status status;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
        MPI_Comm_size(MPI_COMM_WORLD,&p);

        if (my_rank!=0){
            sprintf(message, "Greetings from process %d!",my_rank);
            dest = 0;
            MPI_Send(message, strlen(message)+1, MPI_CHAR, dest, tag,
    MPI_COMM_WORLD);
        } else {
            for (source=1;source<p;source++){
                MPI_Recv(message, 100, MPI_CHAR, source, tag,
    MPI_COMM_WORLD,&status);
                printf("%s\n",message);
            }
        }
        MPI_Finalize();
    }
    ```
- Makefileを/usr/local/mpich/shareからコピーする．
  - EXECS = hello
    をtargetのファイル名に書き換える．
  - make
    すればコンパイラーやライブラリを自動的に指定して，compileできるはず．
- 台形公式にしたがって4/(1+x^2)の面積を求める．
  - ```
    #include <math.h>
    ```

```
#include <stdio.h>

double f1(double x);

int main(void){
    int i,n;
    double x=0,dx;
    scanf("%d",&n);
    dx=1.0/n;
    double S=(f1(0)+f1(1))/2.0;
    for (i=1;i<n;i++){
        x+=dx;
        S+=f1(x);
    }
    printf("%10d %20.15f\n",n,S*dx);
}

double f1(double x){
    return 4.0/(1+x*x);
}
```

▼ 並列計算でPiを求めるコード.

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

  double f1(double x);
  double trapezoid(double a, double b, int n);

  main (int argc, char* argv[]){
      int my_rank,p,source,dest,tag=0,n;
      double sum,sum1;
      double local_a, local_b;
      double startwtime=0.0, endwtime;
      MPI_Status status;

      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
      MPI_Comm_size(MPI_COMM_WORLD,&p);

      if (my_rank!=0){ //分割数の入力と通知
          MPI_Recv(&n, 1, MPI_INT, source, tag, MPI_COMM_WORLD,&status);
      } else {
          printf("Input dividing number:");
          scanf("%d",&n);
          n = n/p;
          printf("Dividing number per process:%10d\n",n);
          for (dest=1;dest<p;dest++){
              MPI_Send(&n, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
          }
      }

      local_a=1.0/p*my_rank; //区間の割り出し
      local_b=1.0/p*(my_rank+1);

      startwtime = MPI_Wtime();
      if (my_rank!=0){
          sum=trapezoid(local_a,local_b,n); //子プロセスによる積分
          dest = 0;
          MPI_Send(&sum, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
      } else {
          sum1=trapezoid(local_a,local_b,n); //親プロセスによる積分と和
          for (source=1;source<p;source++){
  ```
  ```

```
                MPI_Recv(&sum, 1, MPI_DOUBLE, source, tag,
        MPI_COMM_WORLD,&status);
        //          printf("Process %3d:%20.15f\n",source,sum);
                    sum1+=sum;
                }
                printf("Total:%20.15f\n",sum1);
                endwtime = MPI_Wtime();
                printf("wall clock time = %10.7f\n",endwtime-startwtime);
            }
            MPI_Finalize();
        }

        double trapezoid(double a, double b, int n){
            int i;
            double x=a,dx;
            dx=(b-a)/n;
            double S=(f1(a)+f1(b))/2.0;
            for (i=1;i<n;i++){
                x+=dx;
                S+=f1(x);
            }
        //    printf("%10d %20.15f\n",n,S*dx);
            return S*dx;
        }

        double f1(double x){
            return 4.0/(1+x*x);
        }
```

▼ mpirunの結果

- ```
  bob1 bob/mpich> /usr/local/mpich/bin/mpirun -np 2 para_tra
  Input dividing number:1000000
  Dividing number per process:    500000
  Total:   3.141592653583196
  wall clock time =  0.0273438
  ```
  ```