第 2 回 クラスタゼミ

ゼミ担当者　：　斉藤宏樹，釘井睦和，勝崎俊樹
指導院生　　：　谷村勇輔，児玉憲造，片浦哲平，下坂久司
開催日　　：　2002 年 4 月 25 日

ゼミ内容: 本研究室にあるクラスタ，Cambria System を使用して並列計算を行う．Cabria System の構成を理解し，並列プログラムの実行の仕方を学ぶ．また MPICH と LAM について使用法を学ぶ．

1　Cambria System

Cambria System とは，本研究室にある PC クラスタである．PC クラスタとは，複数の PC をネットワークで接続した並列計算機である．Table 1 に Cambria System の性能を示す．

Table 1　Cambria System の性能

項目	性能
CPU	Pentium　(800MHz) × 256
Memory	256M × 256（計 64GB）
Network	FastEthernet
OS	Debian GNU/Linux2.2

Cambria System の構成については，Fig. 1 のようになり，dna，rna，prot，amin の各スレーブノード 16 台の計 256 台がマスターの Cambria System に接続されている．256 台のスレーブは，16 台に 1 台がハードディスクを備えている．残りの 15 台はディスクレスである．つまり，dna － 101〜116 までの 16 台のうち 1 台がハードディスクを持つ．

ディスクを備えた PC は，Cambria System において/home を mount するだけでよいが，ディスクを備えていない PC はディスクを備えた PC にも mount することになる．

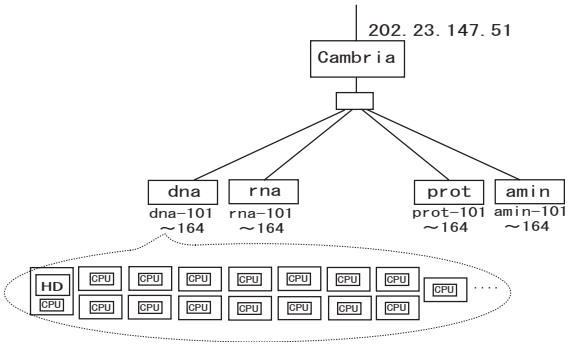

Fig. 1　Cambria System の構成

2　並列計算

MPI を用いたプログラムを使用して，並列計算を行う．PC クラスタ Cambria System を用いて行う．

2.1　π計算のアルゴリズム

並列化を行う対象は π を近似的に求めるプログラムを用いる．π は式 (1) に示す計算式で求めることができる．また，これを逐次的に計算するプログラムは，積分計算が Fig. 2 に示すように行われるので，0 から loop-1 個の区間の積分計算に置き換えられる．この積分計算のループ部分を並列化する．プログラムでは，複数のプロセッサで loop 個の区間の計算を分担するようなアルゴリズムを採用する．（プログラムは後ろに載せている）

$$\pi = \int_0^1 \frac{4}{1+x^2} dx \tag{1}$$

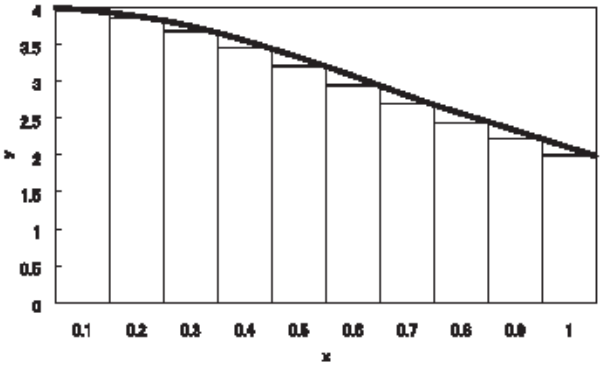

Fig. 2　$y = \frac{4}{1+x^2}$

3　MPICH

MPICH では，ファイルの実行時に様々なオプションをつけてやることで，その動作を制御できる．その方法は以下のようになる．

mpirun（オプションを入れる）（実行プログラム）

3.1 -np

オプションとして -np（数値）を入れることで，プログラムの実行に使用するプロセスの数を設定できる．記入例は以下に示す．

3.2 machinefile

MPI プログラムでは，machinefile をどのように設定するかによって，並列プロセスをどのプロセッサに割り当てるかを決めることができる．このとき，MPICH における machinefile では，1 つのノード名が複数回現れた場合，対応する複数のプロセスを同一ノードに割り当てる．

```
mpirun -np 3 -machinefile name.LINUX a.out
```

このコマンドは，プログラム a.out を実行する際に使用される．name.LINUX は使用するノードを記述するファイルであり，任意のファイル名をつけられる．例えば，server01，server02，server03 という子ノードを利用したい場合，以下のようなファイルを用意する．

```
server01
server02
server03
```

3.3 gdb

gdb はデバッガを行うオプションであり，プログラムを実行しながら，その中で一体何が起こっているのかを調査できる．その機能は細かく分けて 4 つである．

- プログラムを起動し，その実行結果が及ぼすであろう様々な事柄を報告する．
- 指定された状況において，プログラムの実行を停止する．(エラーなど)
- 何が起こっているのかを検査する．そのことによって，プログラムが停止した時，バグの原因を知ることができる．
- プログラムの状態を変更することができる．

実際に gdb によるデバッグを行う場合は，オプションとして -gdb を加えてやればよい．その記入例を以下に示す．

```
mpirun -np 3 -gdb -machinefile name.LINUX a.out
```

3.4 nolocal

nolocal は，ローカルマシンをノードとして動かさないようにするためのオプションである．つまり，各ノードにプログラムを分配するマシンでのプログラムの実行を禁止している．その記入例を以下に示す．

```
mpirun -np 3 -nolocal -machinefile name.LINUX a.out
```

4 LAM

LAM では，ファイルの実行時に様々なオプションをつけてやることで，その動作を制御できる．その方法は以下のようになる．

```
mpirun（オプションを入れる）（実行プログラム）
```

4.1 -O

複数のマシンが均質である場合，メッセージ送信時にデータ変換をしないことで高速化を実現する．その記入例を以下に示す．

```
mpirun -np 3 -O a.out
```

4.2 -toff

プログラム中で MPI_LTrace_on を指示したときのみ trace generation[1]を実行するため，処理速度を高速化できる．その記入例を以下に示す．

```
mpirun -np 3 -toff a.out
```

5 実行手順

MPICH と LAM の実行プログラムが Cambria System では利用できる．両方の実行手順を示していく．

5.1 MPICH の場合

以下のようにコマンドを入力し，PATH を通す．

```
export PATH=/usr/lib/mpich/bin:$PATH
```

次に，以下のようにコマンドを入力し，コンパイルする．

```
mpicc プログラム名.c -o 実行ファイル名
```

利用するマシンを設定するため，以下のように vi や emacs で，利用するマシン名を一つずつ縦に入力し，名前を付けて拡張子「LINUX」として保存する．

[1]長時間アプリケーションを動作させたときに発生する不要なデータ群がたまるのを防ぐために，断続的にまとまった演算を送りつづける動作

```
dna-101
dna-102
   :
```

ホームの下のディレクトリに，同様に利用するマシン名を書き，名前を付けて拡張子「.rhosts」として保存する．

```
dna-101
dna-102
   :
```

並列計算を実行する．

```
mpirun -np プロセス数 -machinefile 設定した名前.LINUX 実行ファイル名
```

5.2 LAM の場合

ホームの下のディレクトリのファイル「.bashrc」に次の PATH を追加する．vi や emacs で「.bashrc」を開いて，次の PATH を追加し保存する．

```
export PATH=/usr/lib/lam/bin:$PATH
```

利用するマシンを設定するため，以下のように vi や emacs で，利用するマシン名を一つずつ縦に入力し，名前を付けて拡張子「LINUX」として保存する．

```
dna-101
dna-102
   :
```

ホームの下のディレクトリに，同様に利用するマシン名を書き，名前を付けて拡張子「.rhosts」として保存する．

```
dna-101
dna-102
   :
```

正常に動作するのか確認する．

```
recon -v 設定した名前.LINUX
```

デーモンを立ち上げる．

```
lamboot -v 設定した名前.LINUX
```

並列計算を実行する．

```
mpirun -np プロセス数 -machinefile 設定した名前.LINUX 実行ファイル名
```

デーモンを消しておく．

```
wipe -v ホストファイル
```

6 ソースリスト

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int i,loop;
    double width,x,pai=0.0;
    loop = atoi(argv[1]);
    width = 1.0 / loop;
    for(i=0;i<loop;i++){
        x = (i + 0.5) * width;
        pai += 4.0 / (1.0 +x *x);
    }
    pai = pai / loop;
    printf("PAI = %f¥n",pai);
    return 0;
}
```

Fig. 3　逐次プログラム

7 参考文献

- ADVenture Cluster
  http://www.clubsse.com/advc/information/mpiinit/mpiinit.html
- ADVenture Cluster
  http://www.clubsse.com/advc/information/mpich/debug.html
- Gregor User's Manual
  http://202.23.147.51/gregor-mpi.htm
- MPI による並列プログラミングの基礎
  http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/PDF/chapter02.pdf

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f( double a ){ return (4.0 / (1.0 + a * a)); }
int main( int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264 3;
    double mypi, pi, h, sum, x;
    double startwtime, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);
    fprintf(stderr,"Process %d on %s¥n",myid, processor_name);
    n =0;
    while (!done){
        if (myid == 0){
            /*
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
            */
            if (n==0) n=100; else n=0;
                startwtime = MPI_Wtime();
            }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
            done = 1;
        else{
                h = 1.0 / (double) n;
                sum = 0.0;
                for (i = myid + 1; i <= n; i += numprocs){
                    x = h * ((double)i - 0.5);
                    sum += f(x);
                }
            mypi = h * sum;
            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
    MPI_COMM_WORLD);
                if (myid == 0){
                    printf("pi is approximately %.16f, Error is %.16f¥",pi,
fabs(pi - PI25DT));
                    endwtime = MPI_Wtime();
                    printf("wall clock time = %f¥n",endwtime-startwtime);
                }
        }
        MPI_Finalize();
        return 0;
    }
}
```

Fig. 4　並列プログラム