# 工学実験実習 V

## —MPI—

学籍番号：16426
5 年 電子情報工学科 24 番
福澤 大地

提出日：2020 年 10 月 19 日

# 1　目的

MPI (Message Passing Interface) を用いて、複数の CPU 間で通信を行いながら、並列計算を行うプログラムを作成する。その上で、並列計算を行うと通常に比べどれほどの高速化を図れるか、並列計算に適したアルゴリズムとはどのようなものなのかなどを検証する。

# 2　実験環境

プログラムの開発、実行を行った環境を表 1 に示す。表 1 と同様の環境のコンピュータ 44 台が同一ネットワーク内に接続されており、公開鍵認証方式でこれらのコンピュータと SSH 通信を行える環境で実験を行った。

表 1　実験環境

| | |
|---|---|
| CPU | Intel Core i5-6600 @ 3.3GHz |
| メモリ | 8GB |
| OS | Ubuntu 14.04 LTS |
| システム | 64bit |
| コンパイラ | GCC 4.8.4 |
| MPI ライブラリ | Open MPI 1.10.2 |

# 3　MPI と Open MPI について

MPI とは、並列計算を行うために標準化された規格である。これを用いることにより、1 個の CPU で行っていた計算を複数の CPU で分散して行えるようになる。

Open MPI [1] は、MPI に準拠したライブラリの 1 つであり、Unix 上で利用できる。MPI のライブラリは他にも MPICH [2] などがあるが、本実験では Open MPI を使用する。

# 4　実行方法

プログラムのコンパルには `mpicc` コマンド、実行には `mpirun` コマンドを使用する。`mpicc` コマンドは `gcc` コマンドと同様の使い方ができ、`-Wall` オプションなどを利用することもできる。`mpirun` コマンドは、`-machinefile` オプションで使用するコンピュータの名前と CPU の数が記述されたファイル名を、`-np` オプションで使用する CPU の数を指定することで、コンパイルしたファイルを実行することができる。

例えば、“com001” ～ “com004” という名前のコンピュータの CPU を 1 つずつ使用する場合は、次のように記述されたテキストファイルを適当なファイル名で保存する。ここでは、ホームディレクトリに “mymachines” というファイル名で保存することとする。

```
com001 cpu=1
com002 cpu=1
com003 cpu=1
com004 cpu=1
```

そして、“program.c” というファイル名のプログラムをコンパイルし、先ほど指定した 4 個の CPU 実行する場合には次のようなコマンドを入力する。

```
$ mpicc program.c
$ mpirun -machinefile ~/mymachines -np 4 ./a.out
```

なお、今回の環境では次のようにエイリアスを設定することにより、-machinefile オプションを省略し実行できるようにしてある。

```
alias mpirun='-machinefile ~/mymachines'
```

## 5 MPI のプログラム

MPI を用いてプログラムを作成する際は、通常のプログラムとは違い、今実行している CPU の数はいくつなのか、自分はどの CPU なのかなどの情報を取得する必要がある。そのため、MPI のプログラムではリスト 1 のように、前処理を行うプログラムを記述する必要がある。なお、プログラムの終了時には、必ず MPI_Finalize 関数を呼び出さなければならない。

リスト 1　MPI のプログラム

```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main(int argc, char **argv) {
5      int nsize;
6      int myrank;
7      int my_name_len;
8      char my_name[MPI_MAX_PROCESSOR_NAME];
9
10     MPI_Init(&argc, &argv);
11     MPI_Comm_size(MPI_COMM_WORLD, &nsize);
12     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
13     MPI_Get_processor_name(my_name, &my_name_len);
14
15     printf("nsize = %d\n", nsize);
16     printf("myrank = %d\n", myrank);
17     printf("my_name_len = %d\n", my_name_len);
18     printf("my_name = %s\n", my_name);
19
20     MPI_Finalize();
21
22     return 0;
23 }
```

リスト 1 のプログラムを 4 個の CPU で実行した結果を、リスト 2 に示す。リスト 2 を見ると、nsize に実行している CPU の数、myrank に自身の番号、my_name に自身のコンピュータ名が入っていることが分かる。実行結果が myrank の順番で表示されていないのは、プログラムが各 CPU 上で同時に実行されているためである。

リスト 2　MPI のプログラムの実行結果

```
$ mpirun -np 4 ./a.out
nsize = 4
```

```
myrank = 1
my_name_len = 6
my_name = ayu002
nsize = 4
myrank = 0
my_name_len = 6
my_name = ayu001
nsize = 4
myrank = 3
my_name_len = 6
my_name = ayu004
nsize = 4
myrank = 2
my_name_len = 6
my_name = ayu003
```

# 6 課題 1

## 6.1 課題内容

コマンドライン引数から数値 $X$ を受け取り、$1 \sim X$ までの和を $N$ 台の CPU で求めるプログラムを作成する。$X$ と $N$ は任意の自然数とする。

## 6.2 プログラムリスト

課題 1 のプログラムを、リスト 3 に示す。

リスト 3　課題 1 のプログラム

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <mpi.h>
4
5  typedef long long int ll;
6
7  int main(int argc, char **argv) {
8      int nsize;
9      int myrank;
10     int my_name_len;
11     char my_name[MPI_MAX_PROCESSOR_NAME];
12
13     ll i;
14     ll num;
15     ll sum = 0;
16     ll ans, max, min;
17
18     double start_t;
19     double finish_t;
20
21     MPI_Init(&argc, &argv);
22     MPI_Comm_size(MPI_COMM_WORLD, &nsize);
```

```
23      MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
24      MPI_Get_processor_name(my_name, &my_name_len);
25
26      if (argc <= 1) {
27          if (myrank == 0)
28              printf("引数が与えられていません。\n");
29
30          MPI_Finalize();
31          return 0;
32      }
33
34      // 引数を整数型に変換
35      num = atoll(argv[1]);
36
37      if (num <= 0) {
38          if (myrank == 0)
39              printf("引数の値が小さすぎます。\n");
40
41          MPI_Finalize();
42          return 0;
43      }
44
45      if (num > 400000000011) {
46          if (myrank == 0)
47              printf("引数の値が大きすぎます。\n");
48
49          MPI_Finalize();
50          return 0;
51      }
52
53      // 処理開始時間を取得
54      start_t = MPI_Wtime();
55
56      // 演算
57      for (i = 1 + myrank; i <= num; i += nsize) {
58          sum += i;
59      }
60
61      // 集計
62      MPI_Reduce(&sum, &ans, 1, MPI_INTEGER8, MPI_SUM, 0, MPI_COMM_WORLD);
63      MPI_Reduce(&sum, &max, 1, MPI_INTEGER8, MPI_MAX, 0, MPI_COMM_WORLD);
64      MPI_Reduce(&sum, &min, 1, MPI_INTEGER8, MPI_MIN, 0, MPI_COMM_WORLD);
65
66      // 処理終了時間を取得
67      finish_t = MPI_Wtime();
68
69      // 結果表示
70      if (myrank == 0) {
71          printf("answer: %lld\n", ans);
72          printf("max: %lld\n", max);
73          printf("min: %lld\n", min);
74          printf("time: %.10f\n", finish_t - start_t);
75      }
76
77      MPI_Finalize();
78
```

```
79      return 0;
80  }
```

## 6.3 プログラムの説明

### 6.3.1 エラーチェック

26 ～ 47 行目では、与えられた引数が正しいものであるのかチェックを行っている。

本プログラムでは計算を long long int 型で行っている。long long int 型は 64 ビットであるため、表せる値の最大値は、$2^{63}-1$ である。最終的な計算結果がこの範囲に収まっている必要があるので、入力として許容できる最大値を $n$ とすると、式 (1) のようにして求められる。

$$\begin{aligned} \sum_{k=1}^{n} k &= 2^{63}-1 \\ \frac{1}{2}n(n+1) &= 2^{63}-1 \\ \frac{1}{2}n^2+\frac{1}{2}n-2^{63}+1 &= 0 \\ n &\simeq \pm 4.3\times 10^9 \end{aligned} \tag{1}$$

式 (1) より、$n$ が $4\times 10^9$ 以内であれば確実にオーバーフローが起こることはないため、これより大きい値が入力された際はエラーとしてプログラムを終了している。また、コマンドライン引数が与えられていなかった場合や、入力された値が 0 以下であった場合も同様にエラーとしてプログラムを終了している。

### 6.3.2 演算

演算は 51 ～ 53 行目で行っており、myrank+1 から始め、nsize 間隔で数字を足している。例えば、4 個の CPU で実行した場合には、各 CPU が担当する数字は次のようになる。このようにすることで、各 CPU で担当する数字の個数と合計のばらつきを少なくしている。

CPU 0　1, 5, 9, 13, 17, ...
CPU 1　2, 6, 10, 14, 18, ...
CPU 2　3, 7, 11, 15, 19, ...
CPU 3　4, 8, 12, 16, 20, ...

### 6.3.3 集計

各 CPU で行った計算結果の集計は、55 行目の MPI_Reduce 関数で行っている。このような記述を行うことで、全ての CPU の sum の合計を、ans に代入することができる。

第 4 パラメータには演算の種類を指定することができ、56 行目の MPI_MAX では最大値、57 行目の MPI_MIN では最小値を取得することができる。

### 6.3.4 処理時間の計測

MPI には、過去のある地点からの経過時間を取得する MPI_Wtime 関数が用意されている。この関数を処理の開始時と終了時に呼び出し、その差分を取ることで、処理に掛かった時間を計測することができる。本プログラムでは、49 行目で開始時間、59 行目で終了時間を取得し、65 行目でその差分を表示している。

## 6.4　実行結果

### 6.4.1　1 ～ 54321 の和

引数に 54321 を指定し、4 個の CPU で実行した場合の結果をリスト**??**に示す。

### 6.4.2　大きな数を指定した場合

入力できる最大値である、$4 \times 10^9$ を引数に指定した場合の結果をリスト**??**に示す。なお、4 個の CPU で実行した。

### 6.4.3　エラーチェック

引数を指定しなかった場合の結果をリスト**??**、0 以下の数を指定した場合の結果をリスト**??**、$4 \times 10^9$ を上回る数を指定した場合の結果をリスト**??**に示す。

## 6.5　考察

# 7　課題 2

## 7.1　課題内容

課題 1 で作成したプログラムを用いて、並列計算の効果を測定する。CPU の数 $N$ を変えながら処理時間を測定し、その結果をグラフにして考察する。

## 7.2　実行結果

CPU の数 $N$ を 1~30 個に増やしながら処理時間を測定した。全ての CPU が作業が行われていない状態で測定を行い、それぞれ 10 回の平均値を取った。

それぞれの CPU の個数についての処理時間についてまとめたものを表 2 に、グラフにプロットしたものを図 1 に示す。

表 2　CPU の個数と処理時間

| CPU 数 [個] | 処理時間 [ms] | CPU1 個の処理時間に対する短縮率 |
|---:|---:|---:|
| 1 | 7663 | 1.00 |
| 2 | 3842 | 1.99 |
| 3 | 2572 | 2.98 |
| 4 | 1929 | 3.97 |
| 5 | 1544 | 4.96 |
| 6 | 1289 | 5.94 |
| 7 | 1113 | 6.89 |
| 8 | 971 | 7.89 |
| 9 | 862 | 8.89 |
| 10 | 780 | 9.83 |
| 11 | 711 | 10.78 |
| 12 | 652 | 11.76 |
| 13 | 613 | 12.49 |
| 14 | 576 | 13.29 |
| 15 | 536 | 14.29 |
| 16 | 492 | 15.58 |
| 17 | 466 | 16.46 |
| 18 | 440 | 17.42 |
| 19 | 415 | 18.48 |
| 20 | 397 | 19.29 |
| 21 | 391 | 19.61 |
| 22 | 370 | 20.71 |
| 23 | 346 | 22.17 |
| 24 | 328 | 23.36 |
| 25 | 319 | 24.05 |
| 26 | 309 | 24.83 |
| 27 | 300 | 25.51 |
| 28 | 296 | 25.91 |
| 29 | 282 | 27.19 |
| 30 | 266 | 28.77 |

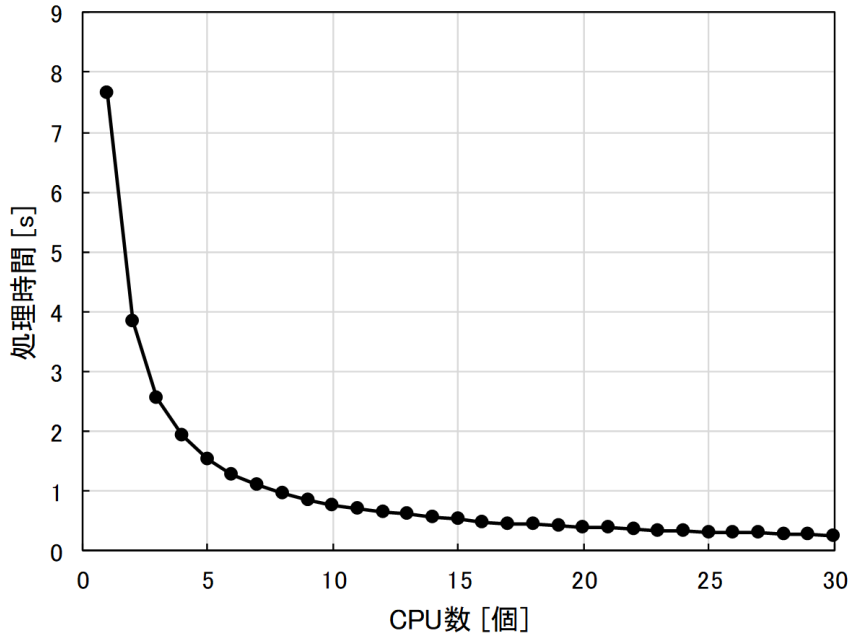


図 1　CPU の個数と処理時間の関係

### 7.3　考察

図 1 を見ると、反比例のグラフとなっている。また、表 2 の CPU1 個の処理時間に対する短縮率より、おおよそ CPU の数の分だけ処理時間の短縮がされていることが分かる。このことから、課題 1 のような、ほとんど通信を行わない単純な並列計算を行うと、投入した CPU の数の分だけ効率化が行えると言える。

## 8　課題 3

### 8.1　課題内容

$N$ 個の CPU でモンテカルロシミュレーションを並列処理するプログラムを作成する。乱数の種は CPU ごとに異なるようにする。

### 8.2　プログラムリスト

課題 3 のプログラムを、リスト 4 に示す。

リスト 4　課題 3 のプログラム

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <math.h>
5  #include <mpi.h>
6
7  typedef long long int ll;
8  typedef long double ld;
9
```

```
10 int main(int argc, char **argv) {
11     int nsize;
12     int myrank;
13     int my_name_len;
14     char my_name[MPI_MAX_PROCESSOR_NAME];
15
16     unsigned int seed;
17     ll i;
18     ll n;
19     ll cnt = 0;
20     ll sum = 0;
21
22     double start_t;
23     double finish_t;
24
25     MPI_Init(&argc, &argv);
26     MPI_Comm_size(MPI_COMM_WORLD, &nsize);
27     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
28     MPI_Get_processor_name(my_name, &my_name_len);
29
30     if (argc <= 1) {
31         if (myrank == 0)
32             printf("引数が与えられていません。\n");
33
34         MPI_Finalize();
35         return 0;
36     }
37
38     // 引数を整数型に変換
39     n = atoll(argv[1]);
40
41     if (n <= 0) {
42         if (myrank == 0)
43             printf("引数の値が小さすぎます。\n");
44
45         MPI_Finalize();
46         return 0;
47     }
48
49     // 処理開始時間を取得
50     start_t = MPI_Wtime();
51
52     // シード値決定
53     srandom(time(NULL));
54     for (i = 0; i < myrank; i++)
55         random();
56
57     seed = random();
58     srandom(seed);
59
60     printf("CPU %02d seed: %u\n", myrank, seed);
61
62     // シミュレーション
63     for (i = 0; i < n / nsize; i++) {
64         ld x = (random() / ((ld)RAND_MAX + 1) + myrank) / nsize;
65         ld y = random() / ((ld)RAND_MAX + 1);
```

```
66
67          if (x * x + y * y < 1)
68              cnt++;
69      }
70
71      // 計算結果の統合
72      MPI_Reduce(&cnt, &sum, 1, MPI_INTEGER8, MPI_SUM, 0, MPI_COMM_WORLD);
73
74      printf("CPU %02d count: %lld\n", myrank, cnt);
75
76      if (myrank == 0) {
77          // 端数の処理
78          for (i = 0; i < n % nsize; i++) {
79              ld x = random() / ((ld)RAND_MAX + 1);
80              ld y = random() / ((ld)RAND_MAX + 1);
81
82              if (x * x + y * y < 1)
83                  sum++;
84          }
85
86          ld ans = (ld)4.0 * sum / n;
87
88          // 処理終了時間を取得
89          finish_t = MPI_Wtime();
90
91          printf("answer: %.20Lf\n", ans);
92          printf("error: %.20Lf\n", ans - M_PI);
93          printf("time: %.10f s\n", finish_t - start_t);
94      }
95
96      MPI_Finalize();
97
98      return 0;
99  }
```

### 8.3 プログラムの説明

### 8.4 実行結果

### 8.5 考察

## 9 課題 4

### 9.1 課題内容

以下の処理を実行するプログラムを作成する。

1. 整数配列 a

   = {3, 1, 4, 1, 5, 9} を CPU 0 で定義する。
2. CPU1 ～ 9 のそれぞれで適当な乱数 R を 1 個ずつ発生させる。
3. a

を 9 台の CPU に `MPI_Send` で送信し、受信側では `a`

のそれぞれの要素に手順 2 で発生させた $R$ を加えた配列 `b`

を作る。

4. 9 台の CPU からそれぞれが持っている `R` と `b` を CPU 0 に送り返す。
5. CPU 0 で CPU 番号とともに送り返されてきた `R` と `b` を表示する。

### 9.2 プログラムリスト

### 9.3 プログラムの説明

### 9.4 実行結果

### 9.5 考察

## 10 課題 5

### 10.1 課題内容

### 10.2 プログラムリスト

### 10.3 プログラムの説明

### 10.4 実行結果

### 10.5 考察

## 11 課題 6

### 11.1 課題内容

### 11.2 プログラムリスト

### 11.3 プログラムの説明

### 11.4 実行結果

### 11.5 考察

## 参考文献

[1] OpenMPI: Open Source High Performance Computing, `https://www.open-mpi.org/`
[2] MPICH | High-Performance Portable MPI, `https://www.mpich.org/`