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、複数のコンピュータ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企画である。これを用いることにより、1 台のコンピュータで行っていた計算を複数台のコンピュータ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コンピュータ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、“main.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 台のコンピュータで実行する場合には次のようなコマンドを入力する。なお、先頭の$はプロンプトを表す。

```
$ mpicc main.c -o main
$ mpirun -machinefile ~/mymachines -np 4 main
```

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、今実行しているコンピュータの数はいくつなのか、自分はどのコンピュータなのかなど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 台のコンピュータで実行した結果を、リスト 2 に示す。リスト 2 を見ると、nsize に実行しているコンピュータの数、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、プログラムが各コンピュータ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

```
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         printf("引数が与えられていません。\n");
28
```

```
29          MPI_Finalize();
30          return 0;
31      }
32
33      num = atoll(argv[1]);
34
35      if (num <= 0) {
36          printf("引数の値が不正です。\n");
37
38          MPI_Finalize();
39          return 0;
40      }
41
42      if (num > 400000000011) {
43          printf("引数の値が大きすぎます。\n");
44
45          MPI_Finalize();
46          return 0;
47      }
48
49      start_t = MPI_Wtime();
50
51      for (i = 1 + myrank; i <= num; i += nsize) {
52          sum += i;
53      }
54
55      MPI_Reduce(&sum, &ans, 1, MPI_INTEGER8, MPI_SUM, 0, MPI_COMM_WORLD);
56      MPI_Reduce(&sum, &max, 1, MPI_INTEGER8, MPI_MAX, 0, MPI_COMM_WORLD);
57      MPI_Reduce(&sum, &min, 1, MPI_INTEGER8, MPI_MIN, 0, MPI_COMM_WORLD);
58
59      finish_t = MPI_Wtime();
60
61      if (myrank == 0) {
62          printf("answer: %lld\n", ans);
63          printf("max: %lld\n", max);
64          printf("min: %lld\n", min);
65          printf("time: %f\n", finish_t - start_t);
66      }
67
68      MPI_Finalize();
69
70      return 0;
71  }
```

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 4×10^9 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 計算

計算は 51 ～ 53 行目で行っている。`myrank`+1 から始め、`nsize` 間隔で数字を足していく。例えば、4 台のコンピュータで実行した場合には、各コンピュータ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、各コンピュータ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。

コンピュータ 0　1, 5, 9, 13, 17, ...
コンピュータ 1　2, 6, 10, 14, 18, ...
コンピュータ 2　3, 7, 11, 15, 19, ...
コンピュータ 3　4, 8, 12, 16, 20, ...

6.3.3 集計

各コンピュータ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、全てのコンピュータの `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` という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。

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/`