Rspberry Pi の並列処理

森谷　祐太　柴田　大暉

１．研究概要

Raspberry Pi を２台以上並列接続し、一つの仕事を分散処理させる。

複雑で実行に時間のかかるプログラム(円周率の正確な計算等)を MPI(Message Passing Interface)を使用し、複数の CPU で一つのタスクを並列に処理させる。スーパーコンピューターやグリッドコンピューティングの基礎について理解する。

２．研究の具体的内容

(１)並列処理について

MPI(Message Passing Interface)とは、CPU が情報をやり取りするための通信規格である。

並列処理とは一つのタスク（仕事）を処理する CPU のコア数分に分割し、それぞれのタスクをコアに割り当て処理させることをいう。並列処理を行うと一つのコアが行う処理が減るので、結果的に全体の処理が早くなる。並列化の簡単な仕組みを図１に示す。

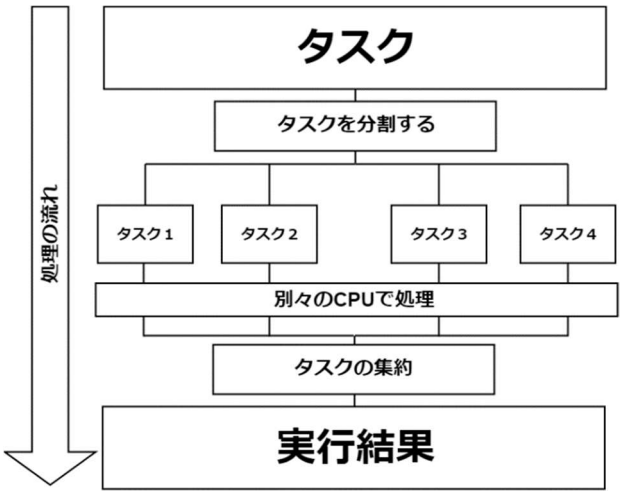

図１　並列処理の仕組み

(１) 使用機器

・Raspberry Pi Model B+

CPU Broadcom BCM2837B0, Cortex-A53 (ARMv8) SoC 1.4Ghz

メモリ　１GB

・LAN ハブ

CentreCOM FS708TXL

100Mbps　８ポート

・USB 急速充電器

Anker PowerPort 10

DC５V/12A(2.4A Max per port)

(２)研究の流れ

図２に研究の流れを示す。

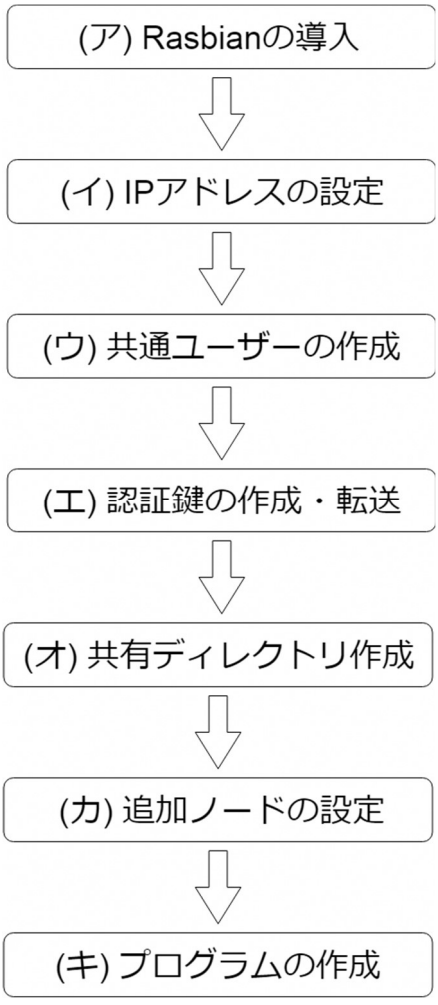

図２　研究のチャート

(３)研究の詳細

ア Raspbian の導入

Raspberry Pi を使用するために、https://www.raspberrypi.org/downloads/raspbian/ から OS を導入した。使用した OS は Raspbian Stretch Lite である。

導入後は、sudo apt update というコマンドで OS 等のアップデートを行う。

イ IP アドレスの設定

並列処理を行うにあたって、ノードの IP アドレスが変動してはいけないので固定 IP アドレスを設定する。

学校の LAN に接続して Tera Term から操作できるようにするために、空き IP アドレスの範囲である 192.168.20.240~192.168.20.243 に設定し、IP アドレスとホスト名を紐づけるために hosts ファイルを編集する(図３)。

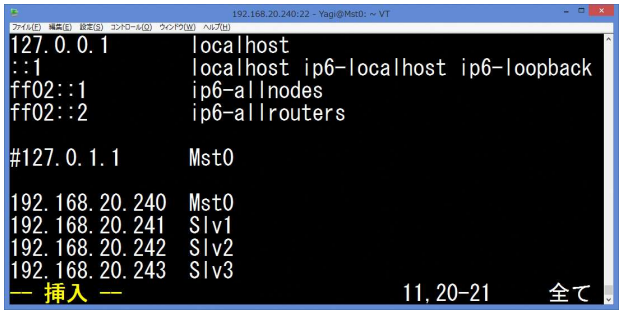

図３　編集した hosts ファイル

ウ 共通ユーザ ID の作成

すべてのノードに同じ名前のユーザ ID を作成する。共通ユーザ ID を作成することですべてのノードが１つにまとまり、認証鍵を使用することでシームレスに通信できるようになる。(エ)の作業を行うにはこの作業が必要となる。

エ 認証鍵の作成・転送

通常、ノード間の通信には、パスワードが必要となるがパスワードなしで相互通信させるため、マスターノードで共通鍵を作成(図4)し、スレーブノードに転送する。そうすることで、パスワードなしでの相互通信が可能となる。

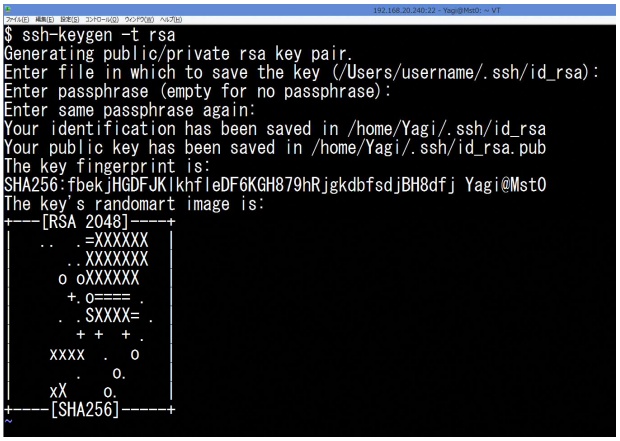

図４　鍵の作成

オ 共有ディレクトリの作成

複数台で同時に同じプログラムを実行するには、すべてのノードがプログラムの入っているディレクトリに接続できる状態でなければならない。そのためにマスターノードに共有のディレクトリを作成する。

カ 追加ノードの設定

計算するノードを増やすためには、ア～オの作業をもう一度繰り返す必要はない。既にノードに接続しているマイクロ SD カードの内容を新しい SD カードにコピーし、IP アドレスとホスト名を変更すればよい。

キ プログラムの作成

私達が C 言語の実習で作ったようなプログラムは並列処理用に作られていないので複数のノードを使って並列で実行することはできない。それを解決するため、MPI(Message Passing Interface)を使用してプログラムを作成した。プログラムは円周率を計算するものを作成した(図５)。

```
#include<mpi.h>
#include<math.h>
#include<stdio.h>

int main(int argc, char* argv[])
{
        int total_iter;
        int n = 60000,rank,length,numprocs,i,ret=0;
        double sum,sum0,rank_integral,A;
        char hostname[MPI_MAX_PROCESSOR_NAME];

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Get_processor_name(hostname, &length);

        if(rank == 0)
        {
                printf("¥n");
                printf("##################################################");
                printf("¥n¥n");
                printf("Master node name; %s¥n", hostname);
                printf("¥n");
                //printf("Enter the number of intervals:¥n");
                //printf("¥n");
                //scanf("%d",&n);
                printf("¥n");

        }

        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

        for(total_iter = 1; total_iter < n; total_iter++)
        {
                sum0 = 0.0;
                //printf("loop %d ¥n", total_iter);
                for(i = rank + 1; i < total_iter; i += numprocs)
                {
                        A = 1.0/(double)pow(i,2);
                        sum0 += A;
                }

                rank_integral = sum0;

                MPI_Reduce(&rank_integral, &sum, 1, MPI_DOUBLE,MPI_SUM, 0, MPI_COMM_WORLD);

        }

        if(rank == 0)
        {
                printf("¥n¥n");
                printf("*** Number of processes: %d¥n",numprocs);
                printf("¥n¥n");
                printf("        Calculated pi = %.30f  <ーー計算された値¥n", sqrt(6*sum));
                printf("                 M_PI = %.30f  <ーー真値¥n", M_PI);
                printf("      Relative Error = %.30f  <ーー誤差¥n", fabs(sqrt(6*sum)-M_PI));
                printf("¥n");
        }

        MPI_Finalize();

        return 0;
```

図５　作成したプログラムの例

（４）完成

メタルラックに Raspberry Pi 等を設置し、配線した。コードが長いものばかりだったので、見た目をよくするためにコードマネジメントを徹底して行った(写真１)。

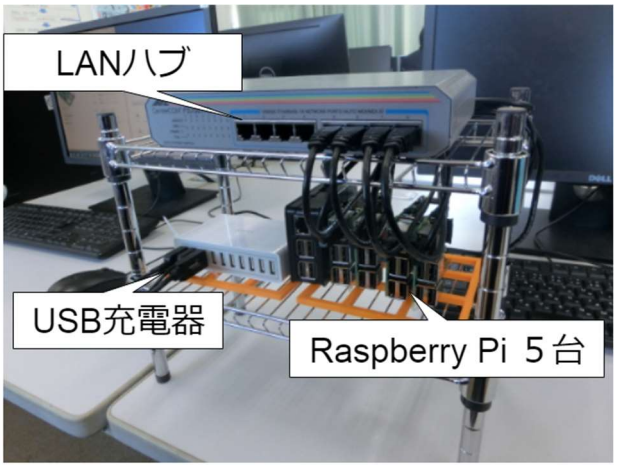

写真１　完成したラズパイクラスタ

３．研究のまとめ

図５で示したプログラムをコア１つで動作させた場合と、２コアで動作させた場合、12コアで動作させた場合を比較した。１コアでは約４分２７秒(図６)かかった処理が、２コアでは約１分３９秒(図７)、１２コアでは約３４秒(図８)で終了した。

このことから、１コアから２コアでは約２．５倍、１コアから１２コアでは約８倍はやくなったことが分かる。コアが１２倍になったのに、なぜ速さが１２倍にならないかというと、タスクの分割や集約など、計算以外のことにリソースが使用されるためである。

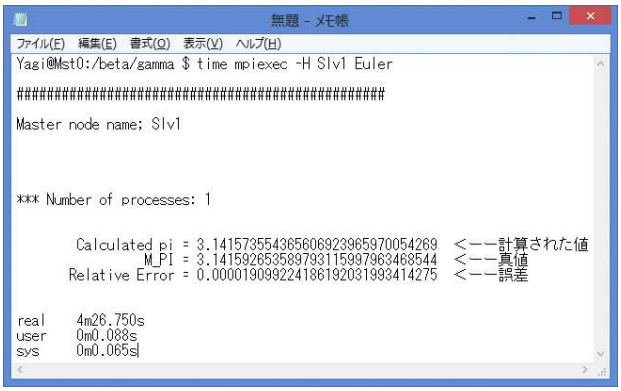

図６　１コア時の実行結果

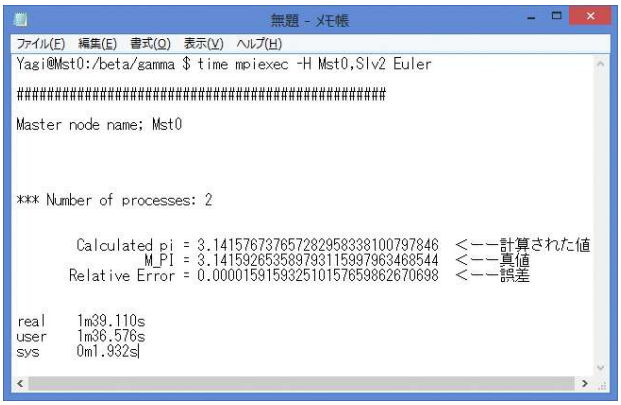

図７　２コア時の実行結果

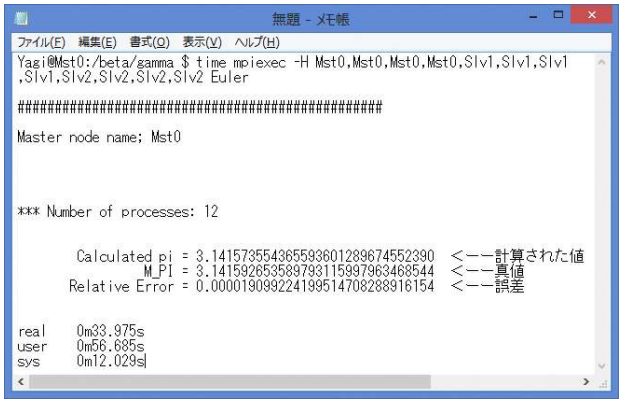

図７　12 コア時の実行結果

当初は Raspberry Pi で並列化をする前に、Linux の Ubuntu をインストールしたノートパソコンで並列接続し、プログラミングをして並列接続化する基礎などを学ぼうとしていた。しかし、Ubuntu をインストールしていたパソコンのうち、ヘッドノードに設定していたパソコンの GUI の調子が悪くノートパソコンの並列化を断念した。Raspberry Pi の並列化は、資料がインターネット上にあまりなかったが本などを活用することで完成させることができた。

今回の研究で、行ったことは基礎中の基礎だが校外学習で見学したスーパーコンピューター「京」も似たような仕組みで動いているということが分かった。

４．感想

(森谷)途中、作業が詰まってしまうことが多く、本当に完成するのか疑問に思ってしまうことが何度もあった。しかし、ノートパソコンの調子が悪かったり、ラズパイが５台中２台故障してしまったりいろいろあったが完成させることができて本当によかったと思う。プログラミングや配線はとても楽しく、今後も似たようなことをしていきたいと思った。課題研究のテーマを決めるときにあまり見通しを立てずに立ててしまったので、完成までにとてもたくさんの時間とお金がかかってしまった。最初にもっと計画を立てておけばこんなことにはならなかったので、作業の計画はとても大切なことだということを学ぶことができた。

(柴田)最初はすぐ完成するのではないかと思っていたが OS のバージョンによって作業方法が違ったりして計画通りに作業を進めることができなかった。このことから余裕をもって計画を立てることが大切だと改めて分かった。ノートパソコンの不調やラズパイの故障などいろいろな出来事があり、とても時間とお金がかかってしまったが最終的には性能を向上させる事ができて良かった。この研究で学んだことを活かしてもっとたくさんのラズパイを並列化させたり、ノートパソコンの並列化にも再挑戦したい。

５．参考文献

・Raspberry Pi でスーパーコンピューターをつくろう

Carlos R.Morrison　(著)
齋藤　哲哉 (訳)

・ 計算工学ナビ　RaspberryPi クラスタ製作記　第０回～第４回
http://www.cenav.org/raspi1/