

# MPI 備忘録とか

## ・実行環境

フロントマシン

CPU：Ryzen 7 3700X

GPU：Geforce RTX 3070

メモリ：DDR4-3200 64GB

ドライブ：USB3.2 64GB

バックマシン

CPU：Core i5 3320M

GPU：Intel 内蔵グラフィックス

メモリ：8GB

ドライブ：SSD 128GB

OS：Ubuntu 22.04

基本的には「高校生のためのクラスター入門」に書いてある通りです。そこに書いていないけど必要だったことや、出てきたエラーについてをここに書いておきます。

ちなみに、「MPICH」は「エムピッチ」と読むらしいです。

## ① ダウンロードとインストール

・OS のインストールは、「高校生…」になります。

・MPI を入れる前に

MPI のインストールには、「gcc (C コンパイラ)」、「gfortran (Fortran コンパイラ)」、「make (ビルドツール)」が必要になります。コマンドで

<フロント>

```
sudo apt install build-essential //gcc と make を一緒にインストール
```

```
sudo apt install gfortran
```

を打ちます。ここで、「高校生…」ではインストールコマンドが「yum」ですが、ここでは「apt」となっています。これは、Linux のなかで、Ubuntu などの Debian 系 OS は「apt」、CentOS などの RedHat 系は「yum」となっています。

「高校生…」にある通り、ダウンロードして、展開して、ビルドして…とやってもいいですが、コマンドから直接インストールすることもできます。というかこっちのほうが簡単です。

<フロント>

```
sudo apt install mpich
```

```
sudo apt install mpich-devel
```

## ② MPI がインストールされているか確認

まず、フロント 1 台だけで MPI を動かしてみます。いきなり並列環境を作ると、エラーが出たときにネットワークによるものなのか MPI によるものなのかわからないからです。

```
<プログラム④: sample1.c>
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv){
    int rank; //ランク (自分が何プロセス目か)
    int proc; //全プロセス数
    int namelen = 10;
    char *name[namelen]; //プロセスを行うマシンの名前

    MPI_Init(&argc, &argv); //MPI の初期化
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); //ランクの取得
    MPI_Comm_size(MPI_COMM_WORLD, &proc); //全プロセス数の取得
    MPI_Get_processor_name(name, &namelen); //名前の取得

    printf("Hello, World! From process %d of %d, %s\n", rank, proc, name);
    MPI_Finalize(); //MPI を終了

    return 0;
}
```

以下のコマンドでコンパイルします。パスやファイル名は適宜変更してください。

<フロント>

`cd <ファイルパス> //sample1.c を保存した場所 ~/program/MPI など`

`mpicc -o sample1 sample1.c`

すると、なんか色々出てくるかもしれませんが、「error」と書いていなければ（「warning」だけなら）たぶん大丈夫です。コンパイルに成功すると、該当フォルダに **sample1** という実行ファイルができます。

<フロント>

`ls`

`sample1.c sample1 //表示結果`

つづいて、以下のコマンドで実行します。

<フロント>

`mpirun -np 4 ./sample1`

-np の後に続く 4 という数字は、**sample1** を並列実行するプロセスの数です。上の例では、**sample1** を

4 並列で実行することを表します。

そうすると、たぶんこんな感じに出力されると思います。

Hello, World! From process 0 of 4, node01

Hello, World! From process 2 of 4, node01

Hello, World! From process 1 of 4, node01

Hello, World! From process 3 of 4, node01

※ 1 個目の数字（1 から 4）は順番が変わることがあります。

こう表示されれば、MPI のインストールは成功しています。

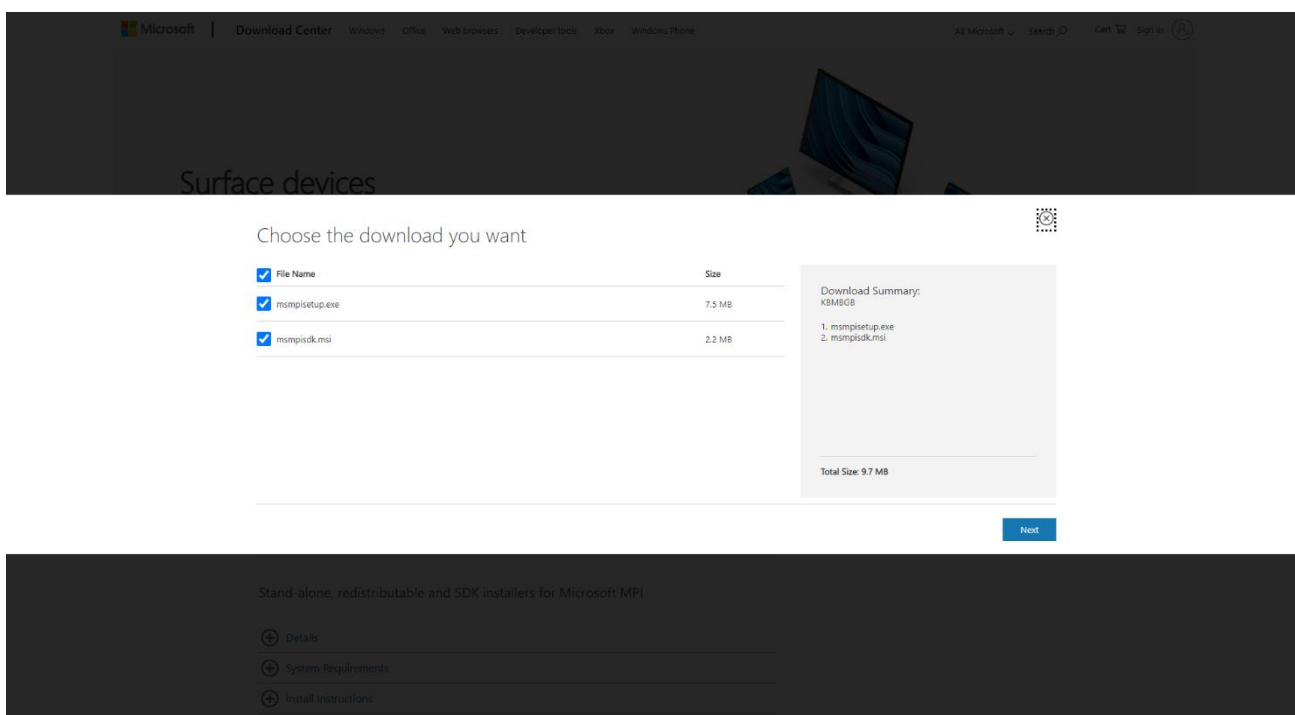
## ◎Windows で MPI を使う

MPI のプログラムを書く時などは Windows のほうが都合がいいこともあるので、MPI の Windows へのダウンロード方法も書いておきます。

下のサイトの「MS-MPI のダウンロード」というところから、MS-MPI をダウンロードします。

<https://learn.microsoft.com/ja-jp/message-passing-interface/microsoft-mpi>

ダウンロードに移って、赤の「Download」をクリックすると、下のような画面になります。



2 つにチェックを入れて、Next を押すと、ダウンロードが始まります。

ダウンロードした 2 つのインストーラーを実行して、インストールします。

デフォルトのインストール場所は、

**mpiexec.exe**（実行ファイル）：C:\Program Files\Microsoft MPI\Bin

**mpi.h**（ヘッダーファイル）：C:\Program Files(x86)\Microsoft SDKs\MPI\Include

**msmpi.lib**（ライブラリ）：C:\Program Files(x86)\Microsoft SDKs\MPI\Lib\x64

です。

それでは、プログラムを実行します。

まず、gcc などの C コンパイラをダウンロードしておき、2 ページの sample1.c を、以下のコマンドでコンパイルします。Windows では、コマンドを打つときはコマンドプロンプトか Windows PowerShell を使います。

```
gcc sample1.c -o sample1.exe -I" C:\Program Files(x86)\Microsoft  
SDKs\MPI\Include" -L" C:\Program Files(x86)\Microsoft SDKs\MPI\Lib\x64"
```

-o は出力先を指定するオプション、-I がインクルードするヘッダーファイル (sample1.c の 3 行目で mpi.h をインクルードしている) のパスを指定するオプション、-L がライブラリファイルを指定するオプションです。

コンパイルに成功したら、sample1.exe という実行ファイルができるはずです。そうしたら、以下のコマンドで実行します。

```
mpiexec -np 4 sample1.exe
```

Linux でやったときと同じく、-np オプションの数値は並列実行するプロセス数です。成功したら、3 ページにあったものと同じような結果が出力されると思います。

<エラー対策>

Q 用語“mpiexec”は、コマンドパレット……として認識されませんでした。名前が正しく…  
といわれたら

A mpiexec.exe のパスが環境変数に通っていないと思われます。「コントロールパネル」を開き、「環境変数」で検索して「システム環境変数の編集」を開いてください。出てきたウィンドウの下の方の表にある「Path」をクリックし、**C:\Program Files\Microsoft MPI\Bin** のパスを追加します。そして、コマンドプロンプトを再起動すれば行けると思います。環境変数の追加方法はいろんなサイトに記載されているのでそれを参考にしてください。

### ③ ネットワークの構築

並列処理を複数のマシンで行うため、ネットワーク環境を作ります。

まず、フロントマシンとバックマシンを、LAN ケーブルでスイッチングハブにつなぎます。ここでのハブは 1 番安いやつでいいです。

#### 3-1 IP アドレスの設定

まず IP アドレスを設定します。以下の設定はフロント・バック共通です。

ターミナルを開き、以下のコマンドを打ちます。

```
ip -a
```

すると、こんなかんじに表示されると思います。

```

1: lo: <LOOPBACK>...
    link/loopback ...
    inet ...
    ...
2: ens4s0: ...
    Link/ether ...
    inet ...

```

ここで、下線を引いた「ens4s0」という文字列を覚えておきます。（マシンによって異なります）

Ubuntu で固定 IP アドレスを設定するには、/etc/netplan/\*\*-cloud-init.yaml（\*\*は数字 2 桁）を編集します。ここで、netplan が読み込むこの yaml ファイルには、アルファベット順で参照し、最後に参照したものが優先されるという特徴があります。そのため、たとえば 10-cloud-init.yaml と 20-cloud-init.yaml というファイルがあったとすると、後者の設定のほうが優先されます。

今回は、デフォルトの設定も残すため、\*\*-cloud-init.yaml をコピーして編集します。ここでは、\*\*が 01 という数値であったとします。

```

cd /etc/netplan
cp 01-cloud-init.yaml 10-cloud-init.yaml //ファイルのコピー
vi 12-cloud-init.yaml または sudoedit 12-cloud-init.yaml

```

個人的に **sudoedit** のほうが使いやすくおすすめです。

そうしたら、以下のように書き込みます。

```

network:
  ethernets:
    ens4s0 //上で覚えた文字列
      dhcp4: false
      dhcp6: false
      addresses: [192.168.1.30/24] //末尾の 30 は 2~254 までの任意の数
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8]
  version: 2 //これはたぶん最初から書いてある

```

ちなみに、4 行目と 5 行目の「DHCP」は、IP アドレスを自動で割り振る仕組みです。ここでは自分で IP アドレスを設定するのでオフにしておきます。

終わったら、以下のコマンドで設定を更新します。

```
sudo netplan apply
```

これで、IP アドレスが変わっているはずです。ip a コマンドで調べてみてください。

ここで、フロントとバックがちゃんとつながっているか確認します。以下のコマンドを打ってみてください。

```
ping 192.168.1.32 //さっき設定した、自分のではない IP アドレス
```

1 秒に 1 行出力されるので、適当なところで **Ctrl + C** を押します。そうすると、以下のように表示されます。

```
PING 192.168.1.32 ..... 56 data bytes
64 bytes from 192.168.1.32 ..... : ..... time=0.501 ms
64 bytes from 192.168.1.32 ..... : ..... time=0.499 ms
64 bytes from 192.168.1.32 ..... : ..... time=0.481 ms
64 bytes from 192.168.1.32 ..... : ..... time=0.519 ms
64 bytes from 192.168.1.32 ..... : ..... time=0.522 ms
64 bytes from 192.168.1.32 ..... : ..... time=0.514 ms

--192.168.1.32 ping statistics--
6 packets transmitted, 6 received 0% packet loss, time 3.036ms
.....
```

ここで注目するのは、下線を引いた 0% packet loss というところです。これが 0% ならば、通信は成功しています。一方、通信に失敗している場合は、100% loss と表示されます。

<通信に失敗する場合>

上から順に試してみてください。

- ・ ping で指定した IP アドレスは正しいですか？
- ・ sudo netplan apply をしましたか？
- ・ LAN ケーブルはちゃんと刺さっていますか？ハブの電源は入っていますか？
- ・ ここまでできなかつたら、3-1 の最初に戻って、もう 1 回 IP アドレスを設定しなおしましょう。
- ・ それでもできなければ、設定する IP アドレスを変えてみてください。

### 3-2 SSH の設定

次に、SSH 通信の設定をします。

ちなみに、SSH というのはコンピューターどうしをリモートアクセス（有線・無線にかかわらず）する際に通信を暗号化してやり取りするためのプロトコルです。Secure Shell の略です。昔は、リモートアクセスには TELNET というプロトコルを使っていたようなのですが、TELNET は通信がすべて平文で行われており、セキュリティ上問題があることから、現在では SSH 通信が主流となっています。

次に、ssh 関連のパッケージをインストールします。

<フロント>

```
sudo apt install openssh-server
sudo apt install openssh-clients
```

<バック>

```
sudo apt install openssh-server
```

そうしたら、SSH 関連の設定を変えます。/etc/ssh/sshd\_config を開きます。

<フロント・バック>

```
cd /etc/ssh
```

```
sudoedit sshd_config
```

以下の部分を変えます。

```
#PubkeyAuthentication yes
```

```
→PubkeyAuthentication yes （#を削除）
```

編集が終わったら、sshd を再起動し、設定を反映させます。

<フロント・バック>

```
sudo systemctl restart sshd
```

これで SSH の初期設定が完了しました。

### 3-3 SSH 通信

では、SSH を使って通信してみます。

SSH の暗号化には、パスワード認証方式と公開鍵認証方式の 2 つがありますが、MPI で通信する際にいちいちパスワードを入力してられないので、公開鍵方式を使います。公開鍵暗号に関しては、結構複雑なので、いろいろ読んで頑張って理解してください。

まず、公開鍵を作ります。SSH 通信では、~/ssh ディレクトリ（~は Desktop とか Documents などがある、いわゆるホームディレクトリ）を使うので、それを作っておきます。

<フロント>

```
cd ~
```

```
mkdir .ssh
```

```
ssh-keygen -t rsa -N "" -f ~/.ssh/key1_rsa
```

ssh-keygen が鍵を生成するコマンドです。-t オプションは暗号の方式を決めるものです。ここでは RSA 暗号を採用しています。-N はパスフレーズを指定するものです。パスフレーズとは、パスワードの強い版みたいなやつです（「ワード」じゃなく「フレーズ」なので長い）。上述のとおりパスワードなんていないのでパスフレーズなしを指定しています。-f オプションは鍵を生成する場所を指定するものです。「高校生…」にある方式と違って、ここではオプションで一括指定しています。どちらでもいいです。

終わったら、ls コマンドで鍵が生成されているか見てみましょう。key1\_rsa と key1\_rsa.pub というのがあはずです。

次に、作った公開鍵をバックマシンに渡します。

<バック>

```
cd ~
```

```
mkdir .ssh
```

<フロント>

```
ssh-copy-id -i ~/.ssh/key1_rsa user@192.168.1.32
```

ここで、**user** は 6 ページの下のところ宛先のマシンにあるユーザー名（管理者アカウントで OK）を、**192.168.1.32** は送信先のバックマシンの IP アドレスを入力してください。IP アドレスのところはホスト名（PC 名、たとえば学校から貸与されたものなら SF11223344 とか コマンド **hostname** で調べられる）

これが終わったら、鍵交換に成功しているはずです。コマンドラインにも「**”ssh” ”192.168.1.32”**」を試してみてください！」みたいな内容が表示されていると思います。ということで、つないでみます。

<フロント>

```
ssh user@192.168.1.32
```

これで、パスワード入力が必要でなくつなげられたら成功です。「**Last login:** なんかかんとか」みたいに表示されると思います。

ちなみに、生成する鍵の名前を **id\_rsa** にすると **-i** オプションをつけずに済みます。

<パスワードが必要になる場合>

- ・バックマシンを見て、鍵が渡されているか確認（**~/.ssh** に **authorized\_key** があればよい）
- ・そもそもフロントマシンにリモートアクセスしていないか（リモートアクセスした状態でのパスワードなしリモートアクセスはできないっぽい）
- ・鍵を作り直してみる

### 3-4 NFS の設定

NFS は、Network File System の略で、あるマシンにあるファイルを、同じネットワーク上にあるほかのマシンからもアクセスできるようにするシステム（プロトコル）です。Google ドライブをパソコンと同期させると Windows のエクスプローラーからも Google ドライブのフォルダが見えるようになる、みたいな感じだと思います。

まずインストールします。ここらへんは OS によって異なります。

<フロント>

```
sudo apt install nfs-kernel-server
```

<バック>

```
sudo apt install nfs-common
```

次に、**NFS** として公開するディレクトリを **/mpi** とします。

NFS として外部のマシンからのアクセスを許可する設定をします。

<フロント>

```
cd /etc
```

```
sudoedit exports
```

これに、以下を書き込みます。

```
/mpi 192.168.1.32(rw,no_root_squash)
```

構文は、指定されている IP アドレスですが、**192.168.1.0/24** のように **CIDR** 表記で書くこともできます。この場合、**192.168.1.1 ~ 192.168.1.254** までを許可するということと同義です。



引数の **rw** は、読み書き (**Read, Write**) 可能であることを示します。

できたら、**NFS** サーバを再起動します。

<フロント>

```
sudo systemctl enable nfs-blkmap.service --now
```

そのあとは、「高校生…」にあるようにバックの **/etc/fstab** を編集すれば、**NFS** が有効化されるはずです。フロントの共有ディレクトリにファイルを追加したり編集したりして、ちゃんと同期できているか確認してください。

今後、プログラムファイルはそのディレクトリに入れることになります。

——私はこのやり方でできませんでした——

マウントしたら、.....**not parmitted** みたいな表示が出ました。

### 3-4 番外編 公式ドキュメントを読もう

上に書いてある方法で 2 回やって 2 回ともダメだったのでおとなしく公式ドキュメントを見ることにします。

URL はこれ→ <https://ubuntu.com/server/docs/service-nfs>

(つながらないときは **wayback machine** をつかうとできます)

以下、””内はすべて上サイトからの引用です。

#### ・ introduction

“**NFS allows a system to share directries and files with other over a network**”

そのままです。その下には **NFS** を使えばディスク容量を節約できるよーみたいなことが書いてあります。

#### ・ インストール

サーバー向け (フロントマシン) のインストールコマンドは、

```
sudo apt install nfs-kernel-server
```

のようです。

また、**NFS** を開始するには

```
sudo systemctl start nfs-kernel-server.sevice
```

だそうです。上にある **sudo systemctl enable nfs-blkmap.service** は、このコマンドに書き換えたほうがよさそうです。

#### ・ 設定

先ほどと同じように、**/etc/exports** を編集します。

```
/mpi 192.168.1.32(rw,no_root_squash,no_subtree_rock)
```

ここで、**no\_subtrr\_rock** はセキュリティ認証をするかしないかの設定だそうです。そんなものあったら **MPI** が通らないので **no** を設定します。

適用するには次のコマンドを打ちます。

```
sudo exportfs -a
```

次はクライアント（バック）の設定です。

先ほどと同じように、**nfs-common** をインストールしたら、コマンドからマウントします。

```
sudo mount 192.168.1.30:/mpi /mpi
```

このとき、何も表示が出ずにコマンドが終わったら、次のコマンドで **NFS** の疎通確認をします。

```
df -h
```

これで、最下行に **192.168.1.30:/mpi** と表示されていたら成功です。あとは、**fstab** の設定もすれば終了です。

#### ④ MPI を動かす

いよいよ動かす、その前に、1つインストールするものがあります。

<バック>

```
sudo apt install mpich
```

MPICH のディレクトリも NFS でアクセスしてしまえばバックにインストールする必要はありません。

「高校生…」でやっていることはこれです。ただ、私の環境では NFS がうまくいっているか微妙な感じだったので、個別インストールすることにしました。

1 台でやった時と同じように mpicc でコンパイルします。

```
mpicc sample1.c -o a.out
```

次に、ホストファイルを書きます。これは、どのマシンで何プロセス動かすかを決めたファイルのことです。共有ディレクトリ内で作成し、編集します。

<フロント>

```
cd /mpi
```

```
sudo touch hosts
```

```
sudoedit hosts
```

<ホストファイル書式>

IP アドレス:プロセス数 user=ユーザー名

IP アドレス:プロセス数 user=ユーザー名

IP アドレス:プロセス数 user=ユーザー名

IP アドレス:プロセス数 user=ユーザー名

...

<ホストファイル例>

```
192.168.1.30:2 user=front
```

```
192.168.1.32:3 user=backnode01
```

```
192.168.1.33:3 user=backnode11
```

ここで、ユーザー名は、各マシンに渡した鍵を持っているユーザーの名前を入れてください。フロントマシンも含め同一のユーザー名の場合は、ここを省略することが可能です。プロセス数は、そのマシンで実行したいプロセスの数です。この指定では、上から順にプロセス **0**、プロセス **1**、プロセス **2**、となるようです。実際に実行されるプロセス数は **mpirun** のオプションとして指定されます。そのオプションの値が **hosts** に書かれたプロセス数より少なければ該当プロセスまでが割り当て（例えば、上に書い

た **hosts** で 4 プロセスを実行すると、プロセス **0,1** が **user=front** で、プロセス **2,3** が **user=backnode01**、**user=backnode11** はおそらく仕事をしない) られます。また、オプション地のほうが多ければ、**hosts** に書かれた数を超過したプロセスは各マシンに均等に割り振られるようです。長々と書きましたが、ようやく実行です。

<フロント>

```
mpirun -f hosts -np 8 ./a.out
```

これで、以下のように表示されると思います。

```
Hello, World! From process 2 of 8, back01
Hello, World! From process 4 of 8, back01
Hello, World! From process 5 of 8, back02
Hello, World! From process 7 of 8, back02
Hello, World! From process 6 of 8, back02
Hello, World! From process 3 of 8, back01
Hello, World! From process 0 of 8, frontmachine
Hello, World! From process 1 of 8, frontmachine
```

表示されたら成功です。

◎**ssh\_askpass** と表示される場合

意味：**ssh** 通信でパスワード要求が発生しています。

原因 1：**hosts** に鍵を渡したユーザー名が記載されていない（ユーザー名を書かないと、おそらくシステムはフロントマシン内のユーザー名で探しに行く。例えば、上の **hosts** ではフロントのユーザー名が **front**、バックが **backnode01**、**backnode11** となっているが、ユーザー名を書かないとシステムはバックマシン内の **front** という名前のユーザーにアクセスを求める。当然そんなユーザーはいないので接続は成立しない。）

私はこれで詰まってました。

原因 2：鍵が正しく渡されていない

**ssh IP アドレス コマンド**で、パスワード入力なしで接続できるかどうか確かめてください。

・ 付録

#### ① 用語解説

特にネットワーク関連とか、専門用語が多くなってしまうので、解説を入れておきます。

・ 参考サイト

公式ドキュメントを見るのが一番ですが、それだけを見てもわからないところが多いので、ある程度説明しているサイトを載せておきます。

追記 でもやっぱり最後は公式ドキュメントに助けられました。

MPI の環境構築や基本コマンドまとめ

<https://qiita.com/kkk627/items/49c9c35301465f6780fa>

ssh-copy-id で公開鍵を渡す

<https://qiita.com/kentarosasaki/items/aa319e735a0b9660f1f0>

CentOS 7 で NFS サーバを構築してみた

[https://qiita.com/tanuki\\_mujina/items/5c706b2eab6e7eed71fd](https://qiita.com/tanuki_mujina/items/5c706b2eab6e7eed71fd)

MPICH2 のインストール

<https://www.nslabs.jp/mpi-setup.rhtml>

MPI 超入門（C 言語編）

<https://www.cc.u-tokyo.ac.jp/events/lectures/13/MPIprog.pdf>

NFS サーバの構築手順（Ubuntu 20.4 編）

[https://changineer.info/vmware/hypervisor/vmware\\_ubuntu\\_nfs.html](https://changineer.info/vmware/hypervisor/vmware_ubuntu_nfs.html)

【Ubuntu】固定 IP アドレスの設定（無線 LAN、Wi-Fi 対応）：netplan

<https://office54.net/iot/linux/ubuntu-ipaddress-netplan>

MPICH 公式ドキュメント

インストールガイド

<https://www.mpich.org/static/downloads/4.0.3/mpich-4.0.3-installguide.pdf>

ユーザーガイド

<https://www.mpich.org/static/downloads/4.0.3/mpich-4.0.3-userguide.pdf>

公式 Wiki

<https://github.com/pmodels/mpich/blob/main/doc/wiki/Index.md>

Wiki の実行コマンドに関するヘルプ

[https://github.com/pmodels/mpich/blob/main/doc/wiki/how\\_to/Using\\_the\\_Hydra\\_Process\\_Manager.md](https://github.com/pmodels/mpich/blob/main/doc/wiki/how_to/Using_the_Hydra_Process_Manager.md)

補足

- ・間違っているところや質問等あったら言ってください
- ・数日で作ったものなのでもっといいやり方があると思います