MPIを用いた並列数値シミュレーション

名古屋大学 宇宙地球環境研究所 堀田英之

1. MPIとは
2. 環境設定
3. Hello World
4. 1対1通信
5. 集団通信
6. 実際の例(移流方程式)

MPIとは

Message Passing Interface (MPI) とは、並列コンピューティングを利用するための標準化された規格である。実装自体を指すこともある。wikipediaより。

今回は、MPIの基礎を(fortran)で学ぶことにより、並列実行可能なコードを実装できるようになることを目指す。

前提(知識)

- UNIX・Linux
- エディタ(Emacs・vi・VS code...)
- fortranの基礎構文
- fortranとMPIを設定ずみの環境(`solar0*`が使えれば問題ない)
- pythonで`matplotlib`と`numpy`が使える(可視化のため。移流方程式までやりたい場合)
- 移流方程式の解法(簡単に復習する)

環境設定など

CIDAS (ISEE, Nagoya-U)

CIDASシステムの`solar0*`で実行することを想定しているために特別な環境設定は必要ない。

ローカルマシンに環境構築する場合

Mac (Homebrew)

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" # Homebrew
brew install gcc # gfortran
brew install openmpi # OpenMPI
```

Linux (Ubuntu)/ WSLを用いれば、Windowsでも多分

```
sudo apt-get update
sudo apt-get install gfortran # gfortran
sudo apt-get install openmpi-doc openmpi-bin libopenmpi-dev # OpenMPI
```

Hello world: code

`src/00_hello_world/main.F90` 以下にプログラムあり

MPIを使う場合は

```
use mpi
```

としてモジュールを宣言する。以下でMPIの初期設定

```
call mpi_init(merr)
```

MPIのプロセス数(`npe`)やMPIのランク(`myrank`)を呼び出すには以下

```
call mpi_comm_rank(mpi_comm_world, myrank , merr)
call mpi_comm_size(mpi_comm_world, npe    , merr)
```

最後にはMPIの終了設定をする。

```
call mpi_finalize(merr)
```

Hello world: compile

コンパイルは `mpif90` を用いる

```
mpif90 main.f90
```

実行は `mpiexec` 。以下のような書式

```
mpiexec -n np ./a.out
```

`np` はプロセス数である。8プロセスを立ち上げたい場合は

```
mpiexec -n 8 ./a.out
```

などとする。

1対1通信

MPIでは、プロセスごとに違うメモリ空間を持っており、明示的にプロセス間のデータのやり取りをする必要がある。ここではプロセスごとの1対1通信を解説する。通信にはブロッキング通信/ノンブロッキング通信がある。

ブロッキング

送受信側で送信/受信バッファを解放しても良いタイミング(一般には送受信が完了したタイミング)になるまで送信関数・受信関数から復帰しない。処理の順番を間違えるとデッドロック(お互いに情報を待つ)が起こる可能性がある。今回はこちらは説明しない

ノンブロッキング

送信処理・受信処理を開始する宣言のみで、送信関数・受信関数から復帰。データの同期は`mpi_wait`関数などでユーザーが保証する必要がある。今回はこちらを説明する。

ノンブロッキング 1対1通信 (1/n)

`myrank = 0` から `myrank = 1` へノンブロッキングに変数 `a` を `mpi_isend/mpi_irecv` 関数を用いて送受信するプログラムを `src/01_1on1/main.F90` に配置した。

`mpi_isend` 関数

自分の `myrank` から指定した `myrank` へデータを送信する関数。書式は以下

```
mpi_isend(buff,count,datatype,dest,tag,comm,mreq,merr)
```

引数	型	入手力	意味
`buff`	任意	入力	送信する変数、配列も可
`count`	`integer`	入力	要素の個数。配列なら要素数。スカラーならば1。
`datatype`	`integer`	入力	送信するデータの型。MPIによる定義(後述)
`dest`	`integer`	入力	送信先の `myrank`
`tag`	`integer`	入力	メッセージタグ。 `mpi_irecv` で同じものを使う
`comm`	`integer`	入力	コミュニケータ。 `mpi_comm_world`
`mreq`	`integer`	出力	通信識別子。サイズは `mpi_isend` 呼び出す回数
`merr`	`integer`	出力	エラーコード

`mpi_irecv` 関数

自分の `myrank` から指定した `myrank` へデータを受信する関数。書式は以下

```
mpi_isend(buff,count,datatype,orgn,tag,comm,mreq,merr)
```

引数	型	入手力	意味
`buff`	任意	入力	送信する変数、配列も可
`count`	`integer`	入力	要素の個数。配列なら要素数。スカラーならば1。
`datatype`	`integer`	入力	送信するデータの型。MPIによる定義(後述)
`orgn`	`integer`	入力	送信元の `myrank`
`tag`	`integer`	入力	メッセージタグ。 `mpi_irecv` で同じものを使う
`comm`	`integer`	入力	コミュニケータ。 `mpi_comm_world`
`mreq`	`integer`	出力	通信識別子。サイズは `mpi_isend` 呼び出す回数
`merr`	`integer`	出力	エラーコード

`mpi_wait` 関数

`mpi_isend/mpi_irecv` を実行した後は、``通信が終わるまで待機する。

```
call mpi_wait(mreq, mstatus,merr)
```

引数	型	入手力	意味
`mreq`	`integer`	入出力	通信識別子。サイズは `mpi_isend` 呼び出す回数
`mstatus`	`integer`	出力	状況オブジェクト配列。サイズは `mpi_status_size`
`merr`	`integer`	出力	エラーコード

複数回 `mpi_isend/mpi_irecv` を行った場合は `mpi_waitall` でまとめて待機させることもできる(省略)。

MPIの `datatype`

MPI関数の定義する型は多岐にわたるが、よく使うものだけここに示す

MPI `datatype`	fortran `type`	意味
`mpi_integer`	`integer`	整数
`mpi_real`	`real`	単精度実数
`mpi_double_precision`	`double precision`	倍精度実数
`mpi_complex`	`complex`	(通常は)単精度複素数
`mpi_logical`	`logical`	ブール値。`true`か`false`

ユーザーが型を定義することも可能。`mpi_type_create_subarray`など。メモリ上不連続なデータをひとまとめにして送りたい場合に有用(省略)

集団通信

多くのプロセスと同時に関連して通信するのが集団通信。多数回1対1通信をしても実現できるが、MPIの集団通信は最適化してあるので、多数のプロセスが関わる場合はこちらを使うようにする。

- `mpi_bcast` 関数を用いて `myrank = 0` の `a` というデータを全プロセスに送信するプログラムを `src/02_collective/main01.F90`
- `mpi_allreduce` 関数を用いて各プロセスの `myrank` を合計するプログラムを `scr/02_collective/main02.F90`

においた。

`mpi_bcast` 関数

あるプロセスにあるデータを(コミュニケータの)全プロセスに配布する関数。書式は以下

```
call mpi_bcast(buff,count,datatype,orgn,comm,merr)
```

引数	型	入手力	意味
`buff`	任意	入力	送信する変数、配列も可
`count`	`integer`	入力	要素の個数。配列なら要素数。スカラーならば1。
`datatype`	`integer`	入力	送信するデータの型。MPIによる定義(後述)
`orgn`	`integer`	入力	送信元の `myrank`
`comm`	`integer`	入力	コミュニケータ。 `mpi_comm_world`
`merr`	`integer`	出力	エラーコード

`mpi_allreduce` 関数

(コミュニケータの)全プロセスからデータを集め、何らかの操作(合計、最大、最小)を実施する関数。CFL条件で求めた `dt` の最小の値を計算する際などに使う。

```
call mpi_allreduce(buff_send,buff_recv,count,datatype,op,comm,merr)
```

引数	型	入手力	意味
`buff_send`	任意	入力	送信する変数、配列も可
`buff_recv`	任意	出力	受信する変数、配列も可
`count`	`integer`	入力	要素の個数。配列なら要素数。スカラーならば1。
`datatype`	`integer`	入力	送信するデータの型。MPIによる定義(後述)
`op`	`integer`	入力	演算の種類
`comm`	`integer`	入力	コミュニケータ。`mpi_comm_world`
`merr`	`integer`	出力	エラーコード

`mpi_allreduce` の `op`

- `mpi_max`: 最大
- `mpi_min`: 最小
- `mpi_sum`: 総和
- `mpi_prod`: 積
- `mpi_land`: 論理AND

移流方程式の数値解法

$$\frac{\partial Q}{\partial t} = -c\frac{\partial Q}{\partial x}$$

という移流方程式を解くことを考える。移流方程式の解法はそれだけで研究対象クラスであるが、ここではLax-Friedrich法を用いる。
この手法では、方程式を以下のように離散化する

$$Q_i^{n+1} = \frac{Q_{i+1}^n + Q_{i-1}^n}{2} - \frac{c\Delta t}{2\Delta x}\left(Q_{i+1}^n - Q_{i-1}^n\right)$$

となる。これをMPIを用いた並列計算を解くことを考える。今回は周期境界条件を使う。CFL条件は

$$\Delta t \leq \frac{\Delta x}{c}$$

となり、時間・空間一定だが、練習のために `mpi_allreduce` で求める。

計算の流れ

実際のプログラムは `src/03_advection/` 以下に配置

(フローチャートを書きたかったが簡単ではなさそうだったので)

1. 変数宣言
2. 座標設定
3. 初期条件
4. 時間発展
5. 境界条件->MPI通信

の順で行っている。プログラムを見ながら説明する。

MPIを用いた時の出力

MPIを用いた場合は、大きく分けて

1. 各プロセスごとに別々のファイルに書き込む

- 実装が簡単
- 富岳ではこちらが推奨
- ファイル数が多くなる

2. 各プロセスから一つのファイルに書き込む(MPI-IO)

- 慣れていないと実装が複雑(に感じる)
- 富岳では多プロセス(>1000)では動かないorとても遅い
- ファイル数は少なくなる

と長所・短所がそれぞれある。今回は「1. 各プロセスごとに別々のファイルに書き込む」を説明する。後者は`mpi_file_write`などで検索してみると良い(それか生成AIに効くと良い)