Wisteria/BDEC-01 利用事例（5）
マルチプログラム連成ライブラリ h3-Open-UTIL/MP (1/2)

荒川隆（CliMTech）
八代尚（国立環境研究所）
中島研吾（東京大学情報基盤センター）

1　はじめに

今回と次回（2022 年 7 月号）の 2 回に分けて複数のシミュレーションプログラムを連成実行し「計算・データ・学習」融合を推進するためのライブラリ h3-Open-UTIL/MP について紹介する。今回は一般的な単一アーキテクチャ環境で複数プログラムを連成するケースについて説明し，次回では h3-Open-UTIL/MP の特徴的な機能であるアンサンブル連成，Python APP 連成，異機種間連成機能などについて紹介する。なお対象読者としては MPI を用いた並列計算には一通り習熟しているが連成計算については未経験のレベルを想定し，今号では煩瑣をいとわず連成計算の基礎的な概念や実行方法も含めて解説してゆく。なお，このような計算方法は「連成解析」「連成計算」「結合計算」など複数の用語が用いられるが本稿では基本的に「連成計算」の語を用い，一部はコミュニティの慣習に従って結合（例：大気海洋結合）を用いることとする。

2　連成計算の基本概念

本節では連成計算の基本的な概念について解説し，次節で MPI 環境における連成計算の実行方法を説明する。既に MPI を用いた連成計算の経験がある読者は本節および次節は飛ばしても差し支えない。

連成計算の基本概念を説明する前に，ひとつの逸話を紹介したい。司馬遼太郎の「街道をゆく」の一篇[1]に，ある僧侶が鈴木大拙の著作「華厳の研究」の翻訳（原著は英語）を読み華厳経の難解な仏教用語である「相即相入」が平易な言葉として表現されていることで経典の内容が腑に落ちる，という経験をしたことが語られている。この際，鈴木大拙が用いた語が interpenetration で，辞書的には相互浸透あるいは相互貫入と説明されている。すなわち世界のすべては相互に影響し合い，何一つとして独立して存在するものはないという意である。これをシミュレーションに置き換えると，単一の事象を再現するだけでは不完全であり関連する様々な事象との相互作用も含めて考える必要がある，という事になろう。連成計算とはまさにこの世界の構造（の極めて不完全なごく一部）を計算機内で実現するための計算手法である。

改めて述べると，連成計算とは複数のシミュレーションプログラムを複合させ相互作用を伴いながら計算してゆく手法である。計算手法として複数の場をひとつの支配方程式で解く強連成と支配方程式を個別に解いて時間ステップ毎に解を交換しながら計算を進める弱連成に大別される。本稿で述べる h3-Open-UTIL/MP は不特定のシミュレーションモデルを対象として弱連成計算を容易に実現するためのソフトウェアである。このようなソフトウェアを一般にカプラと呼ぶ。現代では連成計算は構造や流体，熱解析など様々な分野で実現されており専用の解析ソフトウェアも多くリリースされているが，ここでは連成計算の先駆的な事例のひとつである気象・気候シミュレーションを事例として連成計算の基本概念について解説する。2021 年ノー

ベル物理学賞受賞者に眞鍋淑郎博士が選ばれたことは，本稿の読者にとっても既知のことであろう。真鍋の代表的な業績が大気海洋結合モデルの開発およびそれを用いた気候変動予測シミュレーションである[2,3]。この計算では大気モデルと海洋モデルが相互に海水面温度や風速などの情報（データ）を交換しながら時間積分を進めるようになっている。現代の最先端の気象・気候シミュレーションモデルでも真鍋が開発した基本的なフレームは変わっておらず，例えば日本の代表的なモデルのひとつである MIROC(Model for Interdisciplinary Research on Climate) は大気モデルと海洋モデルの２つのモデルコンポーネントを中心にしてシステムが構成されている[4]。大気モデルと海洋モデルは異なるバイナリとしてコンパイル・リンクされ，MPI 環境下で同時に実行される（具体的な実行方法については後述）。大きなモデルコンポーネントは大気と海洋の２つであるが他にも陸面や河川などの過程がコンポーネント化されており，これらは大気モデルの一サブルーチンとして大気モデルのプログラムに組み込まれている。実際の計算ではこれらの要素モデルが相互に必要な情報を交換しながら時間積分を進めてゆくことになる。この際，個々のモデルコンポーネントはそれが表現している現象に応じた時空間構造を持つため，情報の交換に際しては構造の差違を吸収するための適切な変換を行う必要がある。MIROC の大気海洋モデルを例にすると，大気モデルは緯度経度に沿った規則的な矩形格子を持つのに対して，海洋モデルは極点を陸上に移動させた変則的な矩形格子を持つ。更に，大気モデルが全球をくまなく覆っているのに対して，海洋モデルは陸域を除いた格子のみが意味のある値を持っている。従って，情報の交換に際してはこのような格子の差違を補正する補間計算が必要となる。時間的にも大気モデルと海洋モデルでは代表的なシミュレーションの時間刻み幅（ΔT）が異なっており，大気側のΔT が海洋に比して短いため，大気から海洋への情報は大気の各時間ステップの平均値が渡されるようになっている。これらの状況を模式的に表したのが図１である。図中，Atm. Step は大気モデルの計算，Land Step は陸面モデル，River Step は河川モデル，Ocean Step は海洋モデルの計算を表す。大気モデルと海洋モデルは個別のバイナリとして並列に実行され，各データ交換ステップで前ステップまでの計算結果が相互に送受信される。一方，大気と陸面や河川では同一時間ステップ内で順番にデータが渡され逐次的に計算が実行される。ここで，T_N，T_{N+1}，T_{N+2} は大気モデルと海洋モデルがデータ交換する大きな時間刻み幅を表している。一方，大気モデル(Atm. Step)の中で青い太線で表されるのが大気モデルの細かい時間刻み幅である。大気モデルから海洋モデルへ送られるデータが時間平均値の場合は，細かい時間刻み幅毎の値で平均値が計算される。

ここでは気候シミュレーションを例に連成計算の基本的な概念について説明したが，現代の並列計算環境下における連成計算の基本的な要素は 1)複数のモデルコンポーネントが情報（データ）を交換しながら計算を進める。2)情報交換に際しては時空間構造の差違を補整する計算が実行される。という２点に集約されると考えられる。一方，情報交換や時空間構造の補正の具体的な方法については分野や用途によって異なる対応が要求される。代表的な例として空間構造の補整（補間計算）を考える。大気モデルと海洋モデルを連成して気候のような長期的な現象を計算する場合には，量の保存が厳密に保たれることが要求される（そうでないと海が干上がったりする）。そのために，補間計算には各格子点が代表する面積に応じて値を比例配分する面積重み法が用いられるが[5]，その際に更に問題を複雑にしているのは海陸分布の存在である。海洋モデルが陸域を除いた格子で意味のある値を持つことは前述の通りであるが，大気モデル側の格子が海洋モデル格子の陸と海の境界をまたぐ場合には海陸の割合も考慮した上で

値を適切に配分する必要がある。一方で厳密性を要求されないような空間補間が許容される事例もあり得る。例えば，地震動から構造物の振動を計算するような場合，構造物からの地震動への影響は無視しうるため情報の伝達方向は一方向だけで良く，物理量の厳密な保存は要求されない。そのため，この例では直方体で構成される地震モデルの格子から単純な線形内挿で構造物モデルの格子点値を求めている[6]。

本稿で解説するカプラ h3-Open-UTIL/MP は特定の分野に特化することなく様々な用途に用いられることを目標としており，上記のような計算方法の差違や更には格子構造の差違に対して柔軟に対応されることが求められ，かつ実現している。そのための仕組みについては第 4 節以降で詳述する。

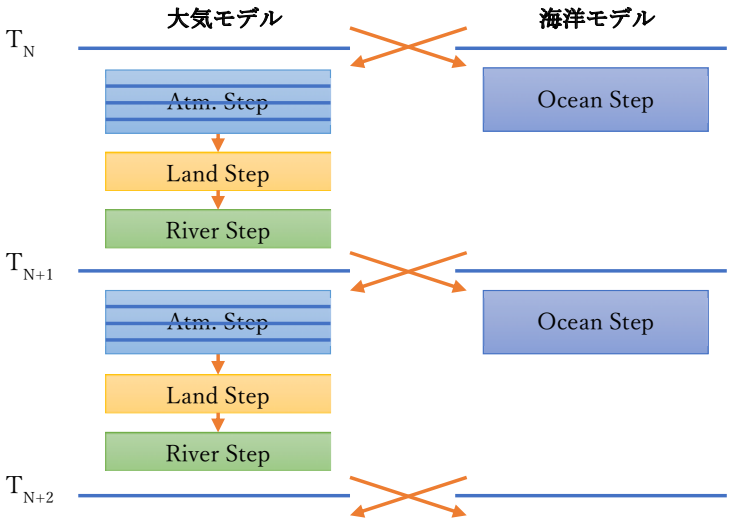

図 1　MIROC における大気モデル・海洋モデル他の連成方法

3　MPI 環境における連成計算

本節では MPI 環境における連成計算の方法について説明する。前提として，連成対象となる個々のモデルコンポーネントは既に MPI 並列化されているものとし，複数のモデルコンポーネントを連成する場合の基本的な概念や手続きについて解説する。

MPI で複数のプログラムを実行する方法には 2 通りの方法がある。ひとつは Master/Worker 方式であり，この場合，マスタとなるプロセスが子プロセスを生成する MPI コマンド(mpi_comm_spawn)をコールする。例えばカプラが独立したプロセスとして最初に起動され，そこから連成対象となるモデルプロセスを生成する場合である。もうひとつは連成対象となるプログラムを一度に起動する方式である。具体的には下の例のように mpiexec（もしくは類似）コマンドに対して複数のプログラムを引数として与えるようにする。

```
mpiexec -np 8 app1.exe : -np 4 app2.exe
```

これら 2 通りの方式では起動されるプログラムの MPI 情報が異なる。簡単なテストプログラム

で具体例を示す。まず Master/Worker 方式のテストプログラムを図 2 に示す。Master 側で MPI サブルーチン mpi_comm_spawn をコールし worker プログラムを起動している。３番目の引数が起動されるプロセス数で，例では５プロセスの worker が起動される。次いでサブルーチン mpi_comm_rank をコールして MPI のデフォルトコミュニケータである MPI_COMM_WORLD を引数として自プロセスのランク番号を取得し出力している。

```
program master
   use mpi
   implicit none
   integer :: my_rank
   integer :: inter_comm
   integer :: ierror

   call mpi_init(ierror)

   call mpi_comm_spawn("./worker", MPI_ARGV_NULL, 5, &
         MPI_INFO_NULL, 0, MPI_COMM_WORLD, &
         inter_comm, MPI_ERRCODES_IGNORE, ierror)

   call mpi_comm_rank(MPI_COMM_WORLD, my_rank, ierror)

   write(*,*) "my_rank = ", my_rank

   call mpi_finalize(ierror)

end program maste
```

```
program worker
   use mpi
   implicit none
   integer :: my_rank
   integer :: ierror

   call mpi_init(ierror)

   call mpi_comm_rank(MPI_COMM_WORLD, my_rank, ierror)

   write(*,*) "my_rank = ", my_rank

   call mpi_finalize(ierror)

end program worker
```

図 2　連成計算テストプログラム（上図：Master，下図：Worker）

このプログラムを mpiexec -n 2 ./master コマンドで実行した場合の結果が図 3 左図である。一方，worker プログラムを worker1 と worker2 として mpiexec -n 2 ./worker1 : -n 5 ./worker2 コマンドで実行した場合の結果が図 3 右図である。結果からわかるように Master/Worker 方式の実行では起動されるプログラム毎に独立した MPI_COMM_WORLD が割り当てられ，その中でランク番号が取得されているのに対し，複数のプログラムを同時に起動する場合はすべてのプロセスで MPI_COMM_WORLD が共有されている。

master_rank =	0
master_rank =	1
worker_rank =	0
worker_rank =	1
worker_rank =	2
worker_rank =	3
worker_rank =	4

worker_rank =	2
worker_rank =	5
worker_rank =	6
worker_rank =	4
worker_rank =	0
worker_rank =	3
worker_rank =	1

図 3 連成計算テスト結果（左図：Master/Worker 方式，右図：同時起動方式）

上の例で示したように Master/Worker 方式の場合はプログラム間でコミュニケータが共有されない。そのためプログラム間の通信を行う際には mpi_comm_spawn の 7 番目の引数で MPI から返されるプログラム間コミュニケータ inter_comm を用いることになる。一方，同時起動方式では MPI_COMM_WORLD が共有されるためプログラム間通信は MPI_COMM_WORLD を用いれば良いが，一方で各プログラム内の通信を混乱なく実行するためには MPI サブルーチン mpi_comm_split を用いてコミュニケータを分割し，プログラム毎に分割されたコミュニケータを用いるようにする必要がある。これらの状況を簡単に示したのが図 4 である。両者の手法には一長一短があるが，Master/Worker 方式の場合，複数の Worker が起動される状況では Worker 間の通信手段が限られる（ないわけではないが相応の手続きを要求される）という問題がある。従って Mater/Worker 方式で複数のモデルコンポーネントを連成する場合には，図に示すように Master を経由して通信を行うのが一般的である。この際にカプラプログラムを Master とすることで補間計算やデータ通信の管理などをモデルとは別のプロセスで処理できるようになる。このことは連成に際しての計算性能向上（実行時間短縮）要因となり得るが，モデルプロセスに比して計算負荷が軽いカプラに専用プロセスを割り当てることは，全体としては計算効率の低下に繋がる懸念がある。一方，同時に起動する場合はカプラに個別のプロセスを割り当てることなく，モデルプロセス同士で直接データ交換することが可能である。この場合カプラはライブラリとしてモデルプロセスにリンクされ実行されることになる。h3-Open-UTIL/MP はモデルプロセスを同時に起動するタイプのカプラである。以下では，この形式での実行におけるデータ通信と補間計算について解説する。図 4 では暗黙の前提としてモデルの各プロセスは相手モデルの各プロセスと直接データ交換を行う Local To Local 通信を行うとして描かれている。相互に複数のプロセスで実行されるモデル間でデータを送受信する方法として，より単純なのは 1)送信側の代表プロセスに送信側各プロセスのデータを集約，2)送信側の代表プロセスから受信側の代表プロセスへ集約したデータを送受信，3)受信側の代表プロセスから受信側各プロセスへデータを分配，という方法である。この際，送信側の代表プロセスもしくは受信側の代表プロセスで全領域のデータを対象として補間計算を行う。この方法は実装が容易だがデータ通信量，プロセスあたりの演算量ともに Local To Local の場合に比して大きくなり，性能上のボトルネックが生じる可能性がある。加えて，大規模並列計算ではメモリ容量の観点から全領域のデータをひとつのプロセスで保持すること自体が不可能な場合もあり得るため，単純な手法では適用できる条件が限られる。一方，Local To Local 通信方法は性能面では優位であるが，適切な格子点データを適切なプロセス間で送受信するためのアルゴリズムが煩雑で実装が難しい場合が(もちろん難易度は双方の格子の形状や対応関係に依存するが)ある。カプラの存在意義のひとつはこの点にあり，本稿で紹介する h3-Open-UTIL/MP においても単純な API の呼び出しだけで Local To Local 通信を実現できるよう

に設計・実装されている。

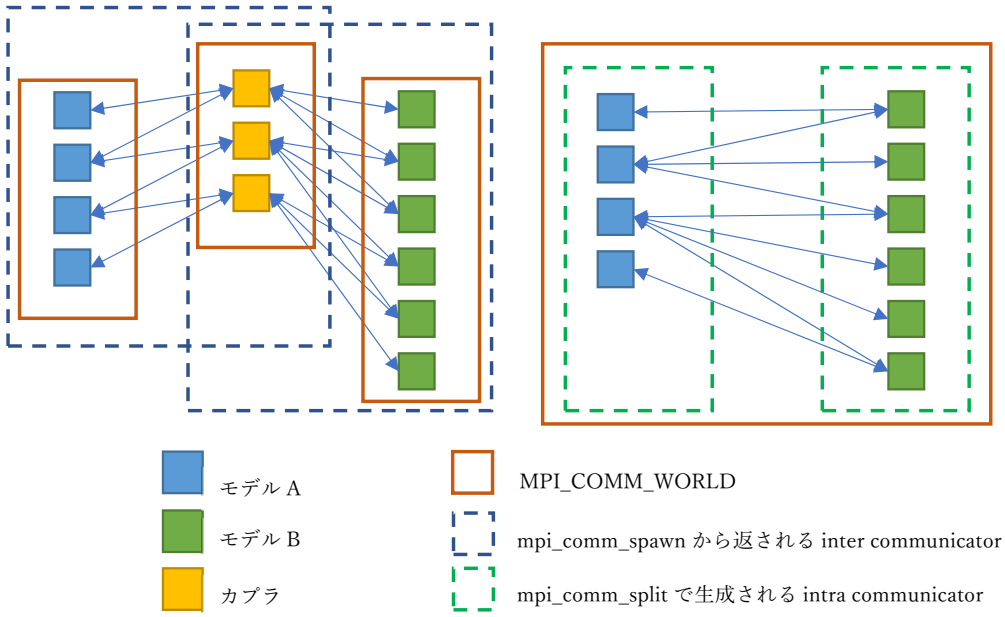

図 4 プログラムの起動方法とコミュニケータの関係（左図：Master/Worker 方式，右図：同時起動方式）

次に格子変換機能について説明する。異なる格子形状のモデル間で情報を交換するには受信側の格子点値を送信側の複数の格子点の値から内挿する必要がある。この内挿計算には前項で述べたように，面積で比例配分する方法や線形内挿する方法など複数の手法がある。しかしながらこれらの手法の違いが影響するのは内挿時の格子点の選択と係数に対してであり，計算アルゴリズムとしては同等である。すなわち受信側のある格子点値 R は送信側の複数(N 個)の格子点値 S(i)と係数 C(i)から次の式のように計算される。

$$R = \sum_{i=1}^{N} S(i) * C(i)$$

すべての受信側格子点に対して送信側の格子点番号とそれに対応する係数を列挙したものをここでは補間テーブルと呼ぶ。ここで各モデルの格子形状が時間的に変わらないとすると，補間テーブルも時間変化しないため事前に計算しておくことが可能である。この補間テーブルおよびモデルの各プロセスが担当する格子の格子点番号の情報が与えられれば，モデルの各プロセスは 1)自身が保持する各格子点値を相手モデルのどのプロセスに送信すれば良いか，2)相手モデルのどのプロセスから何番の格子点値を受信すれば良いか，が決定でき適切な Local To Local のデータ交換と各プロセスでの領域毎の補間計算が可能になる。h3-Open-UTIL/MP においては，これらの情報を入力とすることによって各モデルの格子形状そのものには依存しないで連成計算を実行することを可能としている。

4　h3-Open-UTIL/MP

4.1 h3-Open-UTIL

h3-Open-UTIL はスーパーコンピューティングニュース前号記事[7]で解説された革新的ソフトウェア基盤「h3-Open-BDEC」[8,9]を構成するソフトウェアユニットのひとつである。h3-Open-UTIL ユニットの目的は Wisteria/BDEC-01 のようなヘテロジニアスなシステム上で，「計算・データ・学習(S+D+L)」融合を容易に実現するための環境を提供することである[10,11,12]。h3-Open-UTIL だけでなく h3-Open-BDEC の各ユニットを有機的に協調させ利用することによって，シミュレーションノード群で計算科学シミュレーションコードを実行し，データ・学習ノード群では外部から取り込んだ観測データや，機械学習による推論等に基づきパラメータを最適化し，更に計算を実施するというサイクルを容易に実現することができ，またパラメータ最適化によって計算時間を全体として短縮できることが期待される。

4.2 h3-Open-UTIL/MP の構造

h3-Open-UTIL/MP は連成計算を容易に実現するためのソフトウェア(カプラ)である。対象となるのは MPI を用いて領域分割により並列化されたソフトウェア群であり，今のところ Fortran と Python の API が提供されている。h3-Open-UTIL/MP のプログラム構造を図 5 に示す。図は Fortran プログラムと Python プログラムを連成する例である。h3-Open-UITL/MP は最下層に連成ライブラリ Jcup を用いている[13]。Jcup は連成されるモデルコンポーネントの管理や MPI ルーチンコールなどの基本的な機能を提供する。その上に h3-Open-UTIL/MP のプログラムが構築される。h3-Open-UTIL/MP のプログラムは Python の API を除いて Fortran の module 群で構成されており，Fortran プログラムから用いる場合は API のモジュールを use して API サブルーチンをコールする。Python に関連したプログラムについては次号の記事で説明する予定である。

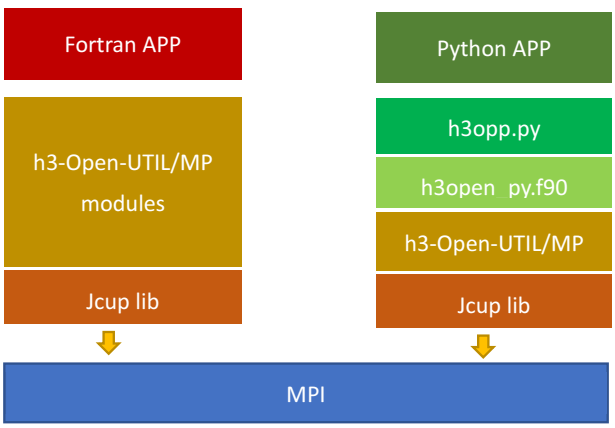

図 5　h3-Open-UTIL/MP のプログラム構造

4.3 h3-Open-UTIL/MP の機能

前項で述べたように連成計算はモデルコンポーネント間で情報(データ)を交換しながら計算を進めることであり，情報の交換に際しては時空間構造の差違を補正する必要がある。h3-Open-UTIL/MP は MPI でデータ並列化されたモデルコンポーネントに対して連成計算を可能とするソフトウェアである。制約条件としては 1)データ交換の時間間隔はデータ毎に一定間隔でなければならない。2)モデルコンポーネントの格子は一意に番号づけられ相対的な位置が時間変化しな

い。の 2 つである。逆に言えば，この要求を満たすモデルや計算条件であればどのようなモデルでも適用可能である。なおデータ交換の時間間隔は一定である必要があるが，その間の積分時間ステップは変化しても構わない。この様子を模式的に示したのが図 6 である。図の例では 3 つのモデルコンポーネントがデータを交換しており，モデル A のデータ交換間隔の間に 3 ステップと 4 ステップの時間ステップを計算している。時間構造の補正として h3-Open-UTIL/MP にはデータの時間平均値を計算する機能がある。平均値計算のアルゴリズムは単純に各時間ステップの ΔT で比例配分する方法であり，式で表すと次のようになる。ここで Dmean は時間平均値，D(t) は時間ステップ t の値， ΔT(t)は時間ステップ t での ΔT，Tex はデータ交換間隔である。先述のようにデータ交換間隔 Tex は一定値でなければならず，またモデルの時間ステップの刻みと一致している，すなわち Tex=Σ ΔT(t)でなければならない。なお平均値を交換するか瞬間値を交換するかは設定で切り替えることができる。設定方法については後述する。

$$Dmean = \sum_{t=1}^{N} D(t) * \Delta T(t)/Tex$$

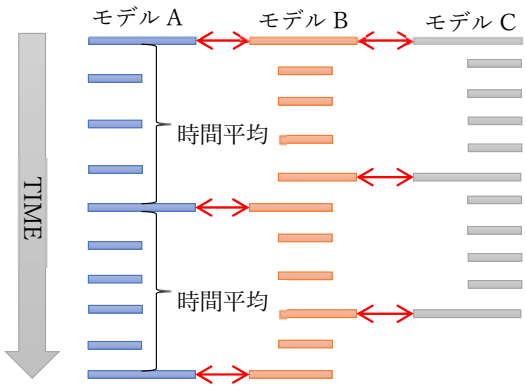

図 6 時間ステップとデータ交換の関係

時間積分ループ中でのデータ交換に関わるサブルーチンコールの様子を図 7 に示す。図で青の四角は時間積分ループの開始と終了，グレーの四角はモデル計算コードを表す。オレンジの四角が h3-Open-UTIL/MP の API ルーチンコールである。時間積分ループ中で連成計算に関わるルーチンコールは h3ou_set_time, h3ou_put_data, h3ou_get_data の 3 種類のみである。h3ou_set_time はモデル時刻をカプラに教えるためのルーチンで，時間積分ループの冒頭付近で呼ばれることが期待されている。h3ou_put_data は相手モデルに送信するデータをカプラに与えるサブルーチン，h3ou_get_data は相手モデルから受信したデータを取得するためのルーチンで，これらのサブルーチンは h3ou_set_time が呼ばれた後であればプログラムの任意の箇所でコールできる。データ送受信や補間計算が実際に行われるのはサブルーチン h3ou_set_time の内部である。モデル A からモデル B へデータが送受信される際のデータの流れを図 8 に示す。図中，青枠の四角は利用者がコールするルーチンを表す。緑枠とオレンジ枠はカプラ内部での動作である。緑とオレンジの 2 種類の枠の相違については次号で解説する。赤枠はカプラ内部で保持されるデータを表す。h3ou_put_data でカプラに与えられたデータはカプラ内部のデータバッファに保持される。

h3ou_set_time がコールされた時点で，当該データがその時刻の送受信対象データかどうかが判定され，送受信データであればバッファから取り出し(get_from_buffer)相手モデルのプロセス順に応じてデータを並び替える（rearrange_send_data)。送信と受信には MPI_Isend, MPI_Irecv が用いられる。モデル B で受信されたデータは再度並び替えられ(rearrange_recv_data)，補間計算サブルーチン interpolate_data に渡される。補間計算で得られたデータはカプラのデータバッファに保持され(put_to_buffer)，API サブルーチン h3ou_get_data がコールされた時点でバッファから取り出され呼び出し側に渡される。

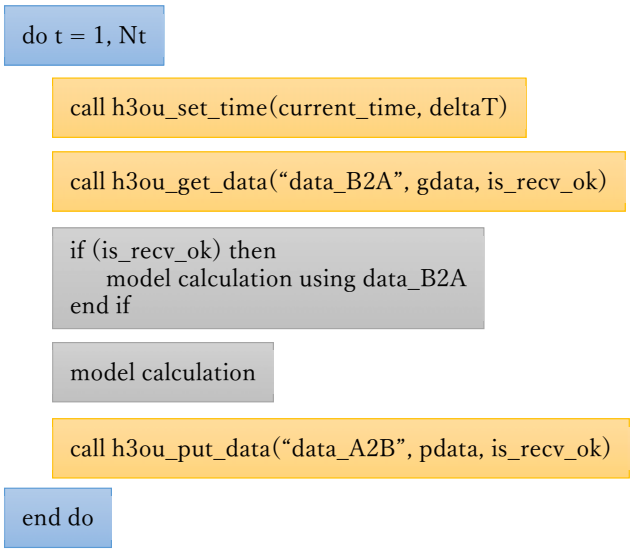

図 7 時間積分ループ中でのカプラ API コールの模式図

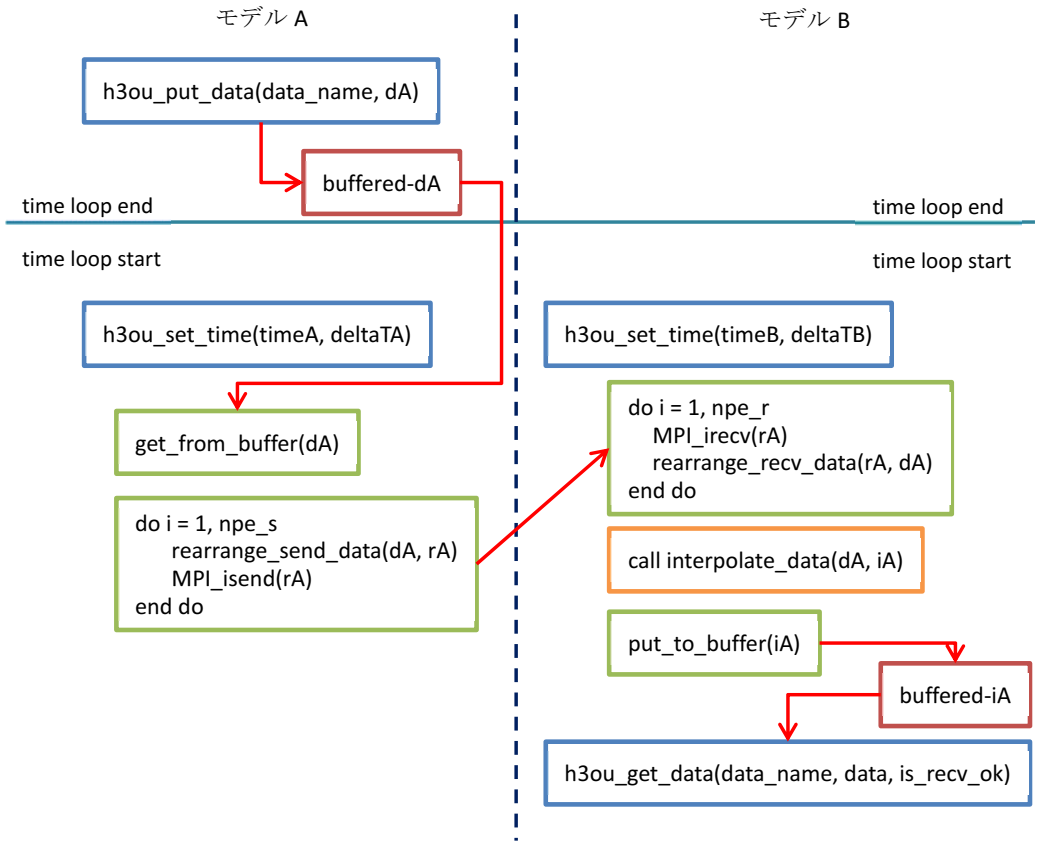

図 8 カプラ内部での送受信データの流れ

（青枠は利用者コールルーチン，緑とオレンジ枠はカプラの内部動作を表す）

ここまで述べたのは h3-Open-UTIL/MP の基本的な機能であり，カプラとして特に新規性のあるものではない。しかしながら h3-Open-UTIL/MP は他のカプラにはない独自の機能を備えている。それらは，

1) アンサンブル連成機能
2) Python アプケーション連成機能
3) 異機種間連成機能

である。これら独自機能については次号の記事で詳細を説明する予定である。以下の節では h3-Open-UTIL/MP の基本的な使用方法について具体例に基づきながら説明してゆく。

5 h3-Open-UTIL/MP の使用方法

この節では具体的な事例に則り h3-Open-UTIL/MP の基本機能の使用方法について説明する。対象としているマシンは Wisteria/BDEC-01 の Odyssey である。サンプルコードおよび実行シェルスクリプトは/work/share/h3-Open-UTIL-MP/src/sample/h3ou/test1（ディレクトリ名は変更になる可能性があります。確定版は次号記事でお知らせいたします）以下の src および run ディレ

クトリにあるので適宜参照されたい。このサンプルでは app1, app2, app3 の 3 つのアプリケーション（モデル）が相互にデータを交換しながら時間積分を進める構成になっている。状況を単純にするため，モデルの格子は同一で積分時間間隔やステップ数も同一としている。

5.1 設定ファイルの準備

h3-Open-UTIL/MP はカプラの動作と交換データの詳細情報を設定するために設定ファイルを用いる。テスト計算プログラムで用いられている設定ファイルを図 9 に示す。セクション h3ou_coupling はカプラ全体の動作を規定するセクションで，今のところ設定可能なのはログファイルの出力レベルである。log_level は”SILENT”,”WISPER”,”LOUD”の３通りが設定可能であり SILENT はログを出力せず LOUD は詳細なログ情報を出力する。LOUD モードは大量のログを出力するためテストやデバッグ時に使用するモードである。debug_mode はログを標準エラー出力に出力するかどうかのフラグである。セクション h3ou_var は２通りの記述形式が定められている。ひとつはデータ交換するモデル名と格子名のセットであり，comp_put, comp_get, grid_put, grid_get のそれぞれに送信モデル名，受信モデル名，送信格子名，受信格子名を与える。この組み合わせに対して以下に記述される交換変数の設定が適用される。h3ou_var セクションのもう一つの記述形式は交換データの設定に関するものである。var_put は送信側データ名，var_get は受信側データ名である。grid_intpl_tag は補間計算時にデータを識別するためのタグで，特に必要ない場合は１を与える。なお，このこのタグを参照する事例については次号で解説する予定である。intvl はデータ交換間隔で単位は秒である。lag は連成計算が並列に行われるか逐次的に行われるかを設定するフラグで，一般的な並列実行の場合は-1 を与える。layer は当該データの鉛直層数である。後述する格子番号設定サブルーチンでも鉛直層数を与えるが，これは格子が取り得る最大の鉛直層数であり，データ毎にはこれを超えない任意の層数を設定可能である。flag はデータを時間平均するかどうかの設定で時間平均する場合は”AVR”を，瞬間値を交換する場合は”SNP”を設定する。

```
# log_level : "SILENT" or "WHISPER" or "LOUD", default = "SILENT"
# debug_mode : .true. or .false., default = .false.
&h3ou_coupling
  log_level = "SILENT"
  debug_mode   = .false.
&end

&h3ou_var   comp_put = "app2",   comp_get = "app1",
            grid_put = "app2_grid", grid_get = "app1_grid", /
&h3ou_var   var_put   = "app2_to_app1",   var_get   = "app2_to_app1", grid_intpl_tag = 1, intvl=720,   lag=-1,
layer=40, flag='AVR' /

&h3ou_var   comp_put = "app1"         ,   comp_get = "app2",
            grid_put = "app1_grid" ,   grid_get = "app2_grid", /
&h3ou_var   var_put   = "app1_to_app2",   var_get   = "app1_to_app2", grid_intpl_tag = 1, intvl=720,   lag=-1,
layer = 40, flag='AVR' /

&h3ou_var   comp_put = "app3"         ,   comp_get = "app1",
            grid_put = "app3_grid" ,   grid_get = "app1_grid", /
&h3ou_var   var_put   = "app3_to_app1"       ,   var_get   = "app3_to_app1"       , grid_intpl_tag = 1, intvl=720,
lag=-1, layer=40, flag='AVR' /

&h3ou_var   comp_put = "app1"         ,   comp_get = "app3",
            grid_put = "app1_grid" ,   grid_get = "app3_grid", /
&h3ou_var   var_put   = "app1_to_app3"       ,   var_get   = "app1_to_app3"       , grid_intpl_tag = 1, intvl=720,
lag=-1,   layer = 40, flag='AVR' /

&h3ou_var   comp_put = "app3"         ,   comp_get = "app2",
            grid_put = "app3_grid" ,   grid_get = "app2_grid", /
&h3ou_var   var_put   = "app3_to_app2"       ,   var_get   = "app3_to_app2"       , grid_intpl_tag = 1, intvl=720,
lag=-1, layer=40, flag='AVR' /

&h3ou_var   comp_put = "app2"         ,   comp_get = "app3",
            grid_put = "app2_grid" ,   grid_get = "app3_grid", /
&h3ou_var   var_put   = "app2_to_app3"       ,   var_get   = "app2_to_app3"       , grid_intpl_tag = 1, intvl=720,
lag=-1,   layer = 40, flag='AVR' /
~
```

図 9 h3-Open-UTIL/MP の設定ファイル例

5.2 API ルーチンコールの概要

h3-Open-UTIL/MP の API を連成対象モデルに組み込む際の手続きは，1)カプラの初期化，2)格子の設定，3)補間テーブルの設定，4)初期時刻の設定，5)第 0 ステップデータの put，6)現在時刻と ΔT の設定，7)データの get，8)データの put，9)連成の終了，の 9 ステップに分けられる。このうち 1)から 4)までが初期設定，6)7)8)が時間積分ループ内の手続きである。これらの手順を図 10 に示す。図では複数回呼び出し可能あるいは設定に従い複数回コールする必要のあるルーチンを水色で，一度だけコールされるルーチンをピンクで示している。なお時間積分ループ中のルーチンコールについては時間ステップ毎に 1 度だけコールするか複数回コール可能かで色分けされている。各ルーチンの引数などについては次節で説明される。初期化部分でコールしている h3ou_get_mpi_parameter は図 4 右図の緑の破線で表されているモデル毎のコミュニケータおよびこのコミュニケータのグループやサイズ，プロセスのランク番号を取得するサブルーチン

である。連成対象プログラムが MPI 並列化されている場合，モデルがこれまで用いているコミュニケータはこのルーチンから得られたコミュニケータに置き換える必要がある。

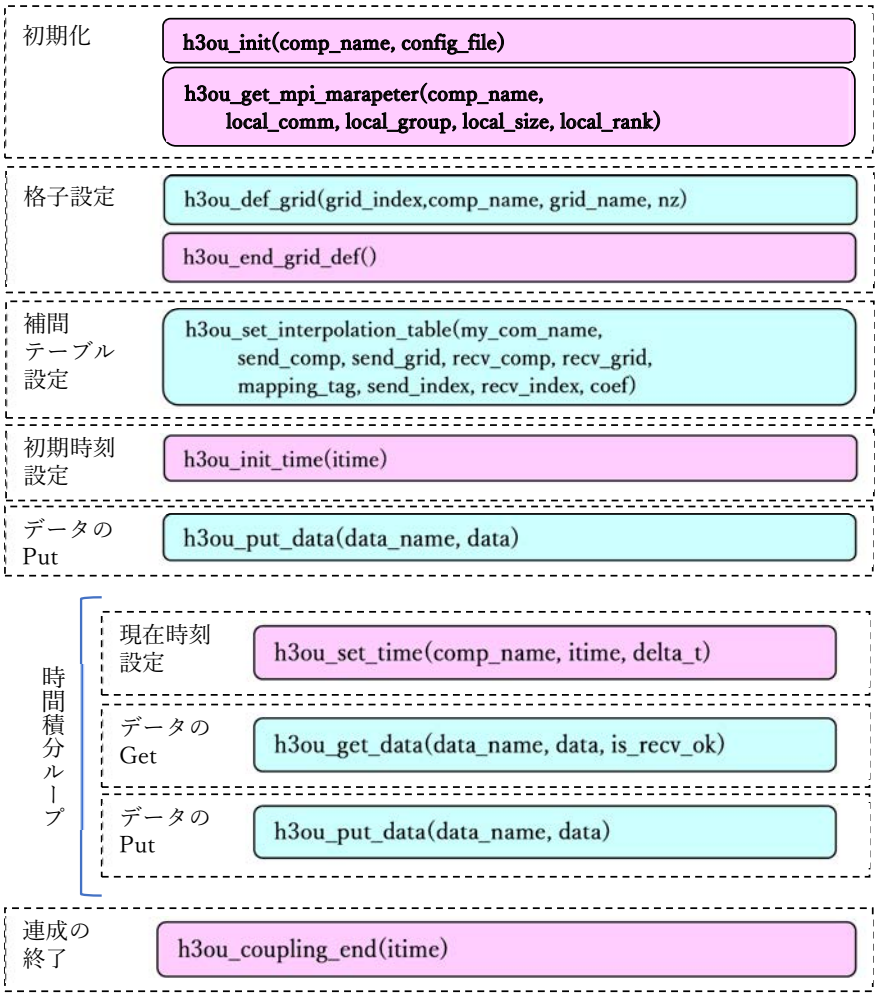

図 10 カプラ API ルーチンコールの概要

サンプルプログラムのメイン部分を図 11 に示す。9 行目のサブルーチン init_common 中で h3ou_init と h3ou_get_mpi_parameter がコールされカプラの初期化と MPI 情報の取得を行っている。次いで 11 行目の cal_and_def_grid でプロセス毎の格子点番号を計算し h3ou_def_grid でカプラに渡した上で，h3ou_end_grid_def をコールしている。13 行目から 24 行目までが補間テーブルの設定に関わるルーチンコールである。ここでは app2→app1，app1→app2，app3→app1，app1 → app3 の順序で補間テーブルを計算し補間テーブルの設定ルーチン h3ou_set_interpolation_table をコールしている。26 行目で h3ou_init_time をコールした後，28 行，29 行めで第 0 ステップのデータをカプラに渡している。時間積分最初のステップの h3ou_set_time で設定ファイルに記述されたすべてのデータが交換されるため，第 0 ステップの h3ou_put_data は自モデルが送信すべきすべてのデータについてコールする必要がある。時間積分ループ内の h3ou_get_data， h3ou_put_data はまとめて記述されているが，実際には必要に応

じてプログラムのどの箇所からでも順不同にコール可能である。最後に 43 行目で h3ou_coupling_end をコールし連成を終了している。

```
 1  program app1
 2    use mpi
 3    use common
 4    implicit none
 5    integer :: int_array(1)
 6    integer :: ierror
 7    integer :: i
 8
 9    call init_common(APP1_NAME, 1)
10
11    call cal_and_def_grid()
12
13    call cal_and_set_map(APP1_NAME, &
14                         APP2_NAME, APP2_GRID, APP2_SIZE, &
15                         APP1_NAME,    APP1_GRID,    APP1_SIZE)
16    call cal_and_set_map(APP1_NAME, &
17                         APP1_NAME,    APP1_GRID,    APP1_SIZE, &
18                         APP2_NAME, APP2_GRID, APP2_SIZE)
19    call cal_and_set_map(APP1_NAME, &
20                         APP3_NAME, APP3_GRID, APP3_SIZE, &
21                         APP1_NAME,    APP1_GRID,    APP1_SIZE)
22    call cal_and_set_map(APP1_NAME, &
23                         APP1_NAME,    APP1_GRID,    APP1_SIZE, &
24                         APP3_NAME, APP3_GRID, APP3_SIZE)
25
26    call init_time(time_array)
27
28    call put_data_2d("app1_to_app2", 1, i)
29    call put_data_2d("app1_to_app3", 1, i)
30
31    do i = 1, APP1_STEP
32      call set_time(APP1_NAME, time_array, APP1_DELTA_T)
33      call get_data_2d("app2_to_app1")
34      call get_data_2d("app3_to_app1")
35
36      call sleep(1)
37
38      call put_data_2d("app1_to_app2", 1, i)
39      call put_data_2d("app1_to_app3", 1, i)
40
41    end do
42
43    call finalize_common()
44
45  end program app1
```

図 11 サンプルプログラムのメイン部分

6　h3-Open-UTIL/MP の API

この節では h3-Open-UTIL/MP の API について一般的な連成に用いられる主立ったものを説明する。

6.1 初期設定 API

表 1 に初期設定に関わる h3-Open-UTIL/MP の API を示す。基本的な連成に用いられるのは h3ou_init と h3ou_get_mpi_parameter の 2 つである。

表 1. 初期設定用 API

ルーチン名	概要
h3ou_init	**カプラの初期化**
h3ou_get_mpi_parameter	**ローカルな MPI 情報を取得する**

これらのルーチンの引数と意味は下のようになる。

(1) カプラ初期化 API

```
subroutine h3ou_init(comp_name, config_file_name, num_of_ensemble)
  character(len=*), intent(IN)   :: comp_name        ! モデルコンポーネント名
  character(len=*), intent(IN)   :: config_file_name ! 設定ファイル名
  integer, intent(IN), optional  :: num_of_ensemble  ! アンサンブル数
```

- カプラの初期化プロセスを実行する。すべてのプロセスが必ず最初に呼ぶ必要がある。カプラ内部でモデルコンポーネント名に従って MPI コミュニケータを生成する。num_of_ensemble は optional 引数であり，次号で説明するアンサンブル結合の場合のみ意味を持つ。

(2) MPI 情報取得 API

```
subroutine h3ou_get_mpi_parameter(comp_name, comm, group, size, rank)
  character(len=*), intent(IN)  :: comp_name  ! モデルコンポーネント名
  integer, intent(OUT)          :: local_comm ! Local なコミュニケータ
  integer, intent(OUT)          :: group      ! Local な MPI グループ
  integer, intent(OUT)          :: size       ! local_comm に所属するプロセス数
  integer, intent(OUT)          :: rank       ! local_comm 内のランク番号
```

- カプラ内部で生成された Local な MPI 情報を取得する。local_comm がモデル内部で使用されるコミュニケータである。group, size, rank はモデルローカルなグループ ID, ローカルなプロセス数, ローカルな自プロセスのランク番号である。

6.2 格子設定 API

表 2 に格子設定用 API を示す。h3ou_def_grid は個々の格子の格子点番号を設定する。h3-Open-UTIL/MP はひとつのモデルコンポーネントに対して複数の格子が設定可能であり，h3ou_def_grid も格子の数に対応して複数回呼び出し可能である。h3ou_end_grid_def は格子設定の終了をカプラに通知する。

表 2. 格子設定用 API

ルーチン名	概要
h3ou_def_grid	**格子点番号の設定**
h3ou_end_grid_def	**格子点番号設定終了**

これらのルーチンの引数と説明は下記の通りである。

(1) 格子設定 API

```
subroutine h3ou_def_grid(grid_index, comp_name, grid_name, num_of_layer)
  integer, intent(IN)           :: grid_index(:)   ! 各プロセスの格子点番号
  character(len=*), intent(IN)  :: comp_name       ! モデルコンポーネント名
```

```
character(len=*), intent(IN) :: grid_name        ! 格子名
integer, intent(IN)          :: num_of_ensemble  ! 鉛直格子数
```

- 格子の格子点番号を設定する。このサブルーチンはデータ交換に関わる格子の数だけ複数回コールしなければならない。grid_index は各プロセスが担当する格子のグローバルな格子点番号である。番号は自然数でなければならない。格子点番号は連続している必要はなく番号に空きがあることは許される。ただし番号の重複は許されない。カプラは番号の重複をチェックしないため注意が必要である。comp_name と grid_name は格子が所属するモデル名と格子名である。nz は補間計算が水平 2 次元で行われる際に，3 次元データの鉛直格子の数を与える。この場合，格子点番号 grid_index は水平格子の分だけを与えれば良い。逆に 3 次元空間で補間計算が必要な場合は 3 次元分の大きさの grid_index を与え nz=1 とすればよい。

(2) 格子設定終了通知 API

```
subroutine h3ou_end_grid_def()
```

- 格子情報設定の終了をカプラに通知する。引数はなく h3ou_def_grid のコールの後に一度だけ呼び出す。このサブルーチンの内部で交換データの設定など複数の初期設定が実行される。

6.3 補間テーブル設定 API

表 3 に補間テーブル設定 API を示す。個々の引数については次に説明する。重要なのは，このサブルーチンがモデルコンポーネント間で各種情報の通信を行っているということである。そのため，送信側モデルと受信側モデルでルーチンコールが正しく対応している必要がある。呼び出しの順序を誤り対応関係が崩れると通信がデッドロックするなどの不具合が生じるため注意しなければならない。

表 3. 補間テーブル設定用 API

ルーチン名	概要
h3ou_set_interpolation_table	**補間テーブルの設定**

このルーチンの引数と説明は下記の通りである。

(1) 補間テーブル設定 API

```
subroutine h3ou_set_interpolation_table(my_name,
                                        send_comp, send_grid,
                                        recv_comp, recv_grid,
                                        mapping_tag,
                                        send_index, recv_index, coef)
  character(len=*), intent(IN)       :: my_name       ! 自モデル名
  character(len=*), intent(IN)       :: send_comp     ! 送信モデル名
  character(len=*), intent(IN)       :: send_grid     ! 送信格子名
  character(len=*), intent(IN)       :: recv_comp     ! 受信モデル名
  character(len=*), intent(IN)       :: recv_grid     ! 受信格子名
  integer, intent(IN)                :: mapping_tag   ! 補間テーブル識別タグ
  integer, optional, intent(IN)      :: send_index(:) ! 送信側格子点番号
  integer, optional, intent(IN)      :: recv_index(:) ! 受信側格子点番号
  real(kind=8), optional, intent(IN) :: coef(:)       ! 補間係数
```

- 補間テーブルを設定する。前述のようにこのサブルーチンは送信側と受信側で正しく

対応している必要がある。6 番目の引数 mapping_tag は同じモデルと格子の組み合わせで複数の補間テーブルを使い分ける際の識別子である。複数の補間テーブルを設定しない場合は 1 を与える。格子の格子点番号を設定する。send_index, recv_index, coef が補間テーブルである。配列の大きさは 3 つの引数で同一でなければならない。これら 3 引数は optional となっており，送信側か受信側のいずれかで与えれば良い。また引数が意味を持つのはルート（0 番）プロセスのみであるためメモリを節約する場合は 0 番以外は大きさ 1 の配列を与えるようにするとよい。

6.4 時刻設定 API

表 4 に時間設定 API を示す。h3ou_init_time は積分の初期時刻を設定する。このサブルーチンは時間積分開始前に一度だけコールされる。h3ou_set_time は時間積分ループ中で毎ステップコールされる（しなければならない）。第 4 節で述べたように h3ou_set_time の中でデータ交換や補間計算など連成に関わる多くの処理がなされる。

表 4. 時間設定 API

ルーチン名	概要
h3ou_init_time	**初期時刻の設定**
h3ou_set_time	**現在時刻とΔT の設定**

これらのルーチンの引数と説明は下記の通りである。

(1) 初期時刻設定 API

```
subroutine h3ou_init_time(time_array)
  integer, intent(IN) :: time_array(6) ! 年月日時分秒の配列
```

- h3ou_init_time は積分初期時刻を設定する。引数 time_array は年月日時分秒を表す大きさ 6 の整数配列である。このサブルーチンは時間積分開始前に 1 回だけコールされる。

(2) 現在時刻と ΔT 設定 API

```
subroutine h3ou_set_time(comp_name, time_array, delta_t)
  character(len=*), intent(IN) :: comp_name     ! モデル名
  integer, intent(IN)          :: time_array(6) ! 現在時刻の配列
  integer, intent(IN)          :: delta_t       ! ΔT
```

- h3ou_set_time は時間積分ループの中で現在時刻と ΔT を設定する。comp_name は当該モデル名，time_array は現在時刻を表す年月日時分秒の配列，delta_t はそのステップの ΔT である。カプラ内部では delta_t の積算時間（秒）でデータ交換判定などを行っている[1]。従ってこのサブルーチンは基本的に（データ交換を行わないステップであっても）毎時間ステップコールしなければならない。

6.5 データ提供取得 API

表 5 にデータ提供取得 API を示す。これらのサブルーチンは時間積分ループ中で h3ou_set_time

[1] カプラ内部での時間管理は delta_t の積算値（秒単位）が用いられており第２引数の年月日時分秒配列はログ出力などでのみ参照されるようになっている。これはユリウス暦とグレゴリウス暦の違いなどカレンダ計算に関する面倒な問題を回避するためである。

がコールされた後なら任意の箇所でコール可能である。また第一ステップで交換するデータをカプラに与えるため，時間積分ループ前に交換される全データについて h3ou_put_data をコールしておく必要がある。また時間平均データについては時間ステップ毎のΔT で比例配分しているため h3ou_put_data については毎ステップコールしなければならない。

表 5. データ提供取得 API

ルーチン名	概要
h3ou_init_time	**初期時刻の設定**
h3ou_set_time	**現在時刻とΔT の設定**

これらのルーチンの引数と説明は下記の通りである。

(1) データ提供 API

```
subroutine h3ou_put_data(data_name, data)
  character(len=*), intent(IN) :: data_name            ! 送信データ名
  real(kind=8), intent(IN)     :: data(:) or data(:,:) ! 送信データ
```

- h3ou_put_data はカプラに対して送信データを与える。引数 data_name は送信データ名である。送信データはデータが鉛直層を持たない場合は一次元配列，鉛直層を持つ場合は 2 次元配列を与える。なお一次元目の配列の大きさは h3ou_def_grid の引数 grid_index の大きさと同じでなければならない。

(2) データ取得 API

```
subroutine h3ou_get_data(data_name, data, is_recv_ok)
  character(len=*), intent(IN) :: data_name            ! 受信データ名
  real(kind=8), intent(INOUT)  :: data(:) or data(:,:) ! 受信データ
  logical, intent(OUT)         :: is_recv_ok           ! データ受信フラグ
```

- h3ou_get_data はカプラから受信データを取得する。送信用サブルーチンと同様，data の配列は鉛直層の有無で 1 次元もしくは 2 次元となる。is_recv_ok は当該時間ステップがデータ交換のタイミングかどうかを返す。データ交換のタイミングでない場合は is_recv_ok は.false.が返り data の値は不定となる。従って受信データを使う場合は必ず is_recv_ok を参照し，真となるタイミングでのみ data の値を参照するようにしなければならない。

6.6 終了処理 API

表 6 に終了処理を示す。このサブルーチンは時間積分ループ終了後，プログラム終了前に一度だけコールされる必要がある。

表 6. 終了処理 API

ルーチン名	概要
h3ou_coupling_end	**連成の終了**

このルーチンの引数と説明は下記の通りである。

(3) 終了処理 API

```
subroutine h3ou_coupling_end(itime, is_call_finalize)
  integer, intent(IN), optional :: itime(:)         ! 積分終了時刻
  logical, intent(IN), optional :: is_call_finalize ! MPI 終了フラグ
```

- h3ou_coupling_end はカプラに対して連成の終了を通知する。引数 itime は積分終了時刻で年月日時分秒を表す大きさ６の整数配列である。is_call_finalize はこのサブルーチン内で MPI の終了サブルーチン MPI_finalize を呼び出すかどうかのフラグである。

7 まとめ

本稿では Wisteria/BDEC-01 利用事例として汎用連成ソフトウェア h3-Open-UTIL/MP の基本機能について解説した。本稿の解説に従って h3-Open-UTIL/MP の API を組み込む事で基本的な連成機能は実現可能である。本文で述べたように h3-Open-UTIL/MP にはアンサンブル連成や Python アプリケーション連成[14], h3-Open-SYS/WaitIO[7]と協調した異機種間連成，など従来のカプラにはない機能が実装されている。これらについては次号で紹介する。

参考文献

[1] 司馬遼太郎, 街道をゆく 24「近江散歩，奈良散歩」, 朝日文庫, 249-260

[2] Manabe, S. and Bryan, K., 1969, Climate Calculations with a Combined Ocean-Atmosphere Model, Journal of the Atmospheric Sciences 26, 786-789, https://doi.org/10.1175/1520-0469(1969)026%3C0786:CCWACO%3E2.0.CO;2

[3] Stouffer, R. J., Manabe, S. and Bryan, K., 1989. Interhemispheric asymmetry in climate response to a gradual increase of atmospheric CO2. Nature 342, 660–662.

[4] Tatebe, H., Ogura, T., Nitta, T., Komuro, Y., Ogochi, K., Takemura, T., Sudo, K., Sekiguchi, M., Abe, M., Saito, F., Chikira, M., Watanabe, S., Mori, M., Hirota, N., Kawatani, Y., Mochizuki, T., Yoshimura, K., Takata, K., O'ishi, R., Yamazaki, D., Suzuki, T., Kurogi, M., Kataoka, T., Watanabe, M., and Kimoto, M.: Description and basic evaluation of simulated mean state, internal variability, and climate sensitivity in MIROC6, Geosci. Model Dev., 12, 2727–2765, https://doi.org/10.5194/gmd-12-2727-2019, 2019.

[5] Jones, P.W., 1999, First- and Second-Order Conservative Remapping Schemes for Grids in Spherical Coordinates, Monthly Weather Review 127, 2204-2210.

[6] Matsumoto, M., Arakawa, T., Kitayama, T., Mori, F., Okuda, H., Furumura, T., Nakajima, K., 2015, Multi-Scale Coupling Simulation of Seismic Waves and Building Vibrations using ppOpen-HPC, Procedia Computer Science 51, 1514-1523, doi: 10.1016/j.procs.2015.05.341

[7] 住元真司, 坂口吉生, 松葉浩也, 中島研吾, Wisteria/BDEC-01 利用事例(3) データ受け渡しライブラリ h3-Open-SYS/WaitIO(1/2), スーパーコンピューティングニュース Vol.24 No.2, https://www.cc.u-tokyo.ac.jp/public/VOL24/No2/10_202203Wisteria-1.pdf

[8] h3-Open-BDEC：https://h3-open-bdec.cc.u-tokyo.ac.jp/

[9] Iwashita, T, Nakajima, K., Shimokawabe, T., Nagao, H., Ogita, T., Katagiri, T., Yashiro, H., Matsuba, H., h3-Open-BDEC: Innovative Software Platform for Scientific Computing in the Exascale Era by Integrations of (Simulation + Data + Learning), Project Poster, ISC-HPC 2020

[10] Wisteria/BDEC-01（「計算・データ・学習」融合スーパーコンピュータシステム）:https://www.cc.utokyo.ac.jp/supercomputer/wisteria

[11] 中島研吾, 塙敏博, 下川辺隆史, 伊田明弘, 芝隼人, 三木洋平, 星野哲也, 有間 英志, 河合直

聡，坂本龍一，岩下武史，八代尚，長尾大道，松葉浩也，荻田武史，片桐孝洋，古村孝志，鶴岡弘，市村強，藤田航平，近藤正章，「計算・データ・学習」融合スーパーコンピュータシステム「Wisteria/BDEC- 01」の概要，情報処理学会研究報告（2020-HPC-179-01），2021

[12] 塙 敏博，中島 研吾，下川辺隆史，芝隼人，三木洋平，星野哲也，河合直聡，似鳥啓吾，今村俊幸，工藤周平，中尾 昌広，「計算・データ・学習」融合スーパーコンピュータシステム「Wisteria/BDEC- 01」の性能評価，情報処理学会研究報告（2021-HPC-180-22），2021

[13] Arakawa, T., Inoue, T., Yashiro, H. et al. Coupling library Jcup3: its philosophy and application. Prog Earth Planet Sci 7, 6 (2020). https://doi.org/10.1186/s40645-019-0320-z

[14] Arakawa, T., Yashiro, H., Nakajima, K., Development of a coupler h3-Open-UTIL/MP, ACM Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region(HPC Asia 2022), 2022, https://doi.org/10.1145/3492805.3492809