付録 A　システムデザイン *

A.1　並列化・高速化

A.1.1　はじめに

今後の計算機の動向として、CPU 単体の演算性能の向上が従来ほどは見込めず、より並列度が増していくこと、メモリの階層構造が複雑化していくこと、B/F 値が低下していくことなどが考えられる。これらのうち、並列度が増していくことに対してはデータ通信をより効率的に行うことが必要である。また、B/F 値の低下及びメモリの階層構造の複雑化を考えると、キャッシュのより効率的な利用といった対策も必要となる。

asuca においては、JMA-NHM で採用している並列化・高速化手法を継承し、さらに改良を行っている。本節ではその取り組み内容について記述する。なお、モデルの並列化についての一般的事項については室井 (1998) で、JMA-NHM の並列化手法については石田・荒波 (2003)、荒波・石田 (2008) でそれぞれ説明されているのであわせてご覧いただきたい。

A.1.2　水平 2 次元分割

メモリを共有しない複数のプロセスによる並列化において、まず予報領域を分割して、それぞれの領域を各プロセスが担当する手法を取る。室井 (1998) が述べている通り、鉛直方向には処理の依存性が強い過程が多いことから、水平方向に 1 次元または 2 次元に分割することが考えられる。JMA-NHM においては、室井 (2000) や石田・荒波 (2003) が述べている通り、当初は y 方向の 1 次元分割を採用していた。しかし、1 次元分割では並列度が増した場合に、各プロセスが担当する領域の大きさが不均等となり効率が低下する。そこで、荒波・石田 (2008) が水平 2 次元分割を導入した。asuca においては現在の JMA-NHM と同じ水平 2 次元分割を採用している。水平分割を行う場合は水平差分の計算等を行うために、隣接するプロセスが予報計算を行っている領域のデータを参照する必要がある。JMA-NHM も asuca もいわゆる「のりしろ」と呼ばれる領域を持っており、隣接するプロセスと適宜通信を行って、のりしろ領域にデータを配置する処理を行っている。2 次元分割の模式図を図 A.1.1 に示す。

A.1.3　入出力専用プロセス

データのディスク入出力は、一般に演算と比較して時間がかかる。そこで、高速化のためには何らかの工夫が必要となる。また、メモリを共有しないプロセス間のデータの分配・集約についても効率的な手法が求められる。

石田・荒波 (2003) ではディスク出力の高速化手法として、分散出力方式（各プロセスが担当する計算領域のデータをそれぞれ独立にディスク出力する方式）と出力専用プロセス方式（計算を行うプロセスが持つ出力データを出力専用プロセスに集約することにより、計算とディスク出力を並列に行う方式）の 2 種類が示されており、asuca においては後者を採用している。分散出力方式を選択した場合、ファイルフォーマットが変わり多数のユーザーが影響を受けるため採用しない [1]。

また、入力においても同様に、分散入力方式と入力専用プロセス方式による並列化が考えられる。出力についてはユーザーの利便性を考慮して分散出力は行っていないが、入力についてはそのような制限は無い。asuca は前処理プログラムが作成したファイルのみを入力とするため、前処理プログラムと asuca 本体を一体管理することにより、ファイルフォーマットの整合性を保ちやすいためである。そこで、現時点では分散入力方式と入力専用プロセス方式の両方を実装している。分散入力であれば、複数ファイルを並行に読み込むことによる時間短縮やデータを読み込んだプロセスからその他のプロセスへデータ送信を行う時間が不要となることにより、高速化につながると期待されるが、ファイル数の増加による入出力時間の増大がありうるため、どちらが高速かはシステムに依存する。そのため、両手法の実装を維持することとしている。なお、入力専用プロセスも出力専用プロセスも複数のプロセスを割り当てることを可能としている [2]。

次に、入力専用プロセスと出力専用プロセスによる処理の設計について説明する。なお、以後は両プロセスをあわせて入出力専用プロセスと呼ぶ。処理に要する時間は一般にディスクの入出力が一番大きく、次いでデータ通信、メモリコピーの順となる。また、ディスクの入出力やデータ通信を効率的に行うためには、なるべく大きな単位にまとめる必要がある。入出力専用プロセス方式の実装では、上記 2 点を念頭におき、計算を担当するプロセス (以下、計算プロセスという) になるべく処理待ちの状態が発生しないようにしている。

データ入力については、計算プロセスが計算している間に、次の計算に用いるデータを先にディスクから読み込んでメモリに保持しておくことが処理の高速化のために有効である。領域モデルとしての asuca の運用においては側面境界値の読み込みが必要であり、緩和領域にレイリーダンピングを適用するために、これ

---

* 石田 純一、 河野 耕平、 荒波 恒平

[1] 出力後にそれぞれ出力したファイルを集約することも考えられるが、そのためのディスク入出力がさらに発生する。特に現業数値予報では、出力したデータをユーザーが利用可能となるまでの時間を短縮する必要があることから、できるだけ余分な入出力を避ける必要がある。

[2] JMA-NHM は荒波・石田 (2008) の時点では入出力専用プロセスは 1 つしかなかったが、現在は asuca と同様に複数のプロセスを割り当てることが可能となっている。

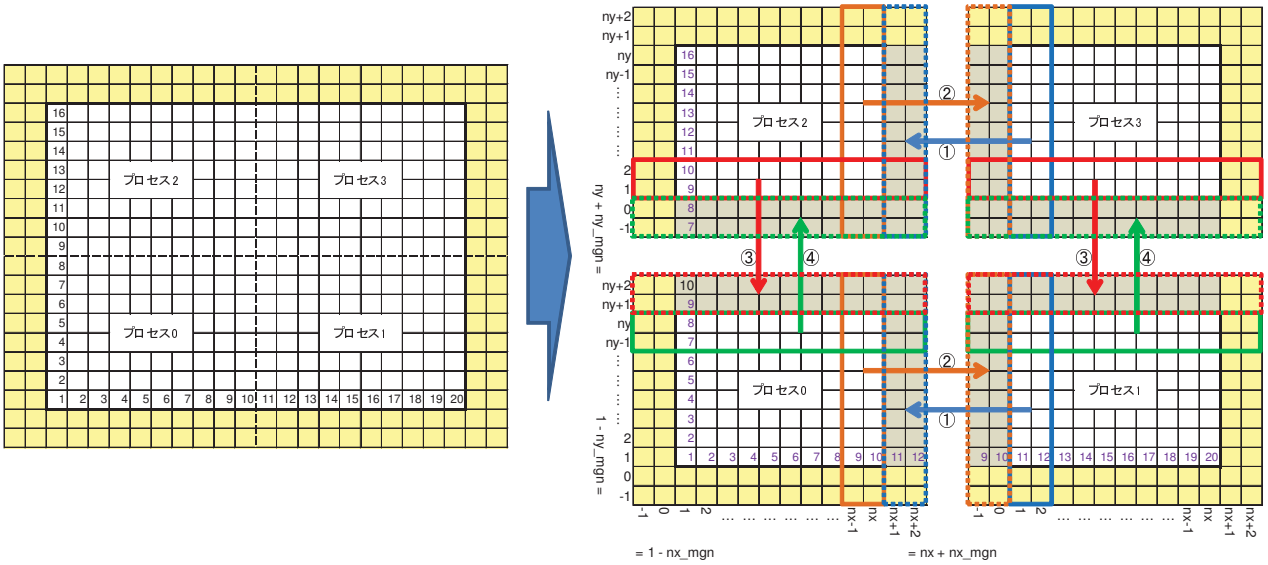

図 A.1.1　asuca における 2 次元分割の模式図。左の図は予報領域全体（格子数は 20 × 16）の図。黒い太線が計算領域と外部領域の境界。白色のセルが計算領域を、薄黄色のセルが外部領域を表す。また、点線は水平 2 次元分割 (2 × 2) を行った場合の各プロセスが担当する計算領域の境界を表す。図中の数字は、予報領域全体の配列を考えた場合のインデックスを表す。右の図は、予報領域を 4 プロセスに分割した図。この場合、各プロセスが分担する計算領域の大きさは 10 × 8 となる。灰色の領域はのりしろ領域（のりしろの幅は 2）を表す。図中の数字は、予報領域全体の配列のインデックスを、枠外の数字は分担した計算領域における配列のインデックスを表す。図中の矢印はのりしろ領域のデータ通信を表す。同じ色の矢印で表されるデータ通信は分割数が増えても同時に行うことが可能である。

に必要な入力データを読み込む時間は無視できない[3]。そこで、計算プロセスが時間積分を行っている間に入出力専用プロセスがデータをあらかじめ読み込んでおき、各計算プロセスへの送信準備を行う。これにより、時間積分が進んで側面境界値を更新する段階で入出力専用プロセスから直ちにデータを受信して計算を再開することができる。ここで、あらかじめ読み込む必要があるデータ量は計算の設定（側面境界値で言えば、緩和領域の大きさや更新頻度）に依存する一方、読み込めるデータ量にはシステムによる制限（メモリ量等）があるため、複数のプロセスを入出力専用としている。

計算プロセスが出力データのディスクへの書き出しを待たずに計算を進めるためには、出力される変数が予報計算の途中で上書きされる可能性があることに注意する必要がある。そこで、計算プロセスが出力のタイミングになったときに、自分が持つバッファに出力するデータをコピーしてから非同期の通信処理を起動し、後続計算を速やかに再開する。入出力専用プロセスは全予報領域のデータを格納できるバッファを準備し、計算プロセスの予報計算と並行してデータを受信・出力する。既に述べた通り、ディスクへの出力はデータ通信より時間がかかるために、多くのデータをバッファに貯めておく必要がある。ここで必要となるバッファの大きさもシステムによる制限を受けることから、複数のプロセスを入出力専用に利用できるようにしている。なお、出力する変数毎にバッファを用意しているため、ディスクへの出力間隔が短くなりすぎると出力が終了する前に次の出力する変数でバッファが上書きされる可能性がある。そこで、出力の場合は予報対象時刻毎に出力データを集めるプロセスを切り替えている。図 A.1.2 に入出力専用プロセスの模式図を示した。

[3] 第 5 章で述べた通り、LFM としての利用においては、予報領域が 1581 × 1301 であるのに対して、緩和領域の幅は 90 格子を考えている。これは、全予報領域の約 23%の領域となる。

A.1.4　通信関数の使い方

プロセス間のデータ通信は、時間積分の計算においては各プロセスと東西南北に隣接するプロセスとのデータ交換を基本とし、入力データの分配や出力データの集約を別途行う。これらの通信を行うライブラリは JMA-NHM にも同等の機能が組み込まれているが、asuca では計算効率や開発効率を念頭においた機能向上を行っている。

(1)　のりしろの交換

asuca と JMA-NHM において、のりしろの交換は時間積分で行われるデータ通信は大きな時間を要する。プログラミングにおいては、のりしろの交換が必要となる変数に対して、適切なタイミングでデータ通信のサブルーチンを呼ぶが、そのタイミングを誤ると適切でないデータがのりしろに配置される。この場合、適切な値とわずかに異なる値（例えば、時間積分で 1 ステップずれた値など）が配置されることがあり、バグに気づきにくい。JMA-NHM の開発においては、このようなバグを生じさせずに効率的にデータ通信するために多大な開発労力を割いてきた。そこで、開発効率

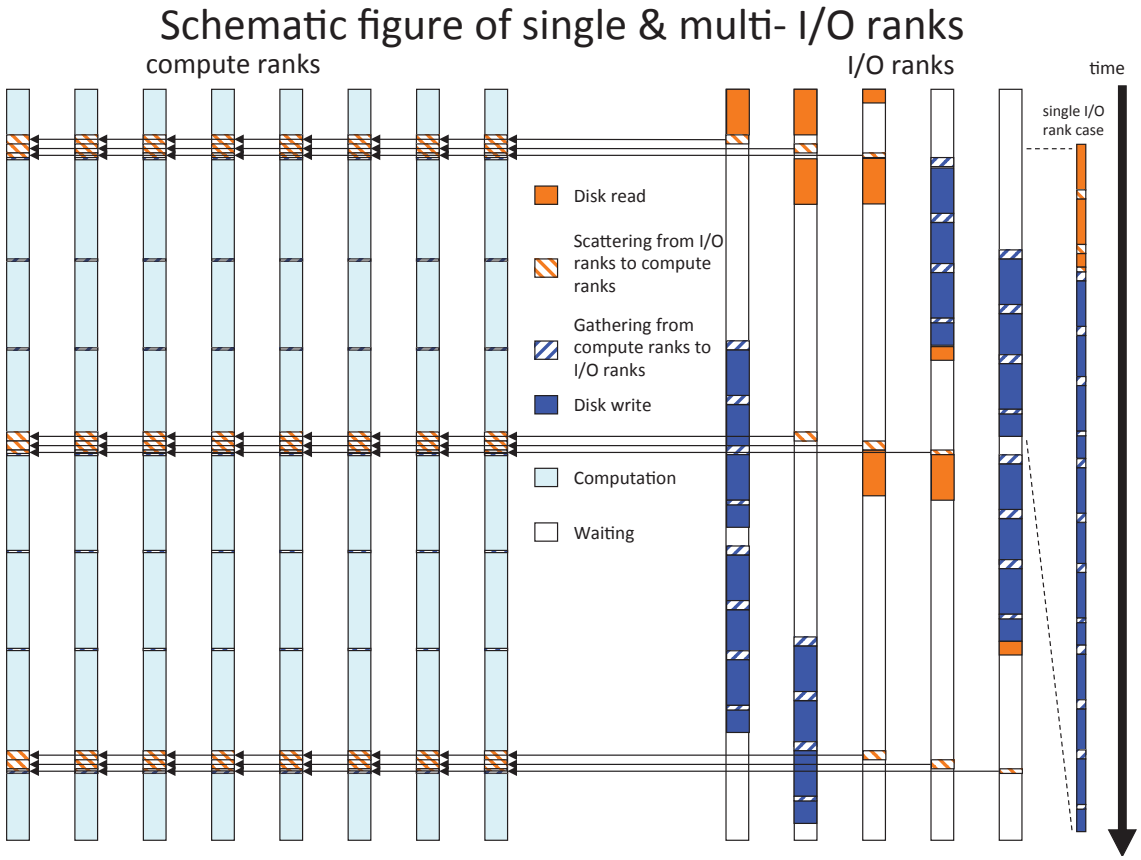

図 A.1.2　入出力専用プロセスの模式図。時間は上から下へ進んでいく。compute ranks と I/O ranks がそれぞれ計算プロセスと入出力専用プロセスを表す。この図においては出力と入力（側面境界値の更新）を同時に行うこととしている。ディスクの入出力に要する時間は入出力専用プロセスのオレンジ及び青の塗りつぶし領域に、データ通信及びメモリコピーの時間はオレンジと青の網掛け領域に示されている。計算プロセスはデータ通信やメモリコピーの時間だけで済むようにしている。

を高めるためバグが混入しにくいデータ通信の仕組みを開発中である。

時間積分等により担当する領域の変数を更新したとき、その変数が水平差分の計算等に用いられる場合は隣接プロセスにデータを送信する必要がある。データを送られる側の視点から見ると、水平差分等の水平方向に隣接するデータを参照して計算を行う場合、その前に隣接プロセスからデータを受信してのりしろ領域に正しいデータを配置しておく必要がある。データ通信は変数の更新とのりしろ領域の参照との間であれば任意のタイミングで行えばよいが、タイミングを誤ると適切でないのりしろ領域のデータを使って計算を行ってしまうこととなる。このとき、

- 担当する領域の変数を更新した後に必ずデータ通信を行う。
- 水平方向に隣接するデータを参照する前に必ずデータ通信を行う。

のいずれかのルールに従っていれば、のりしろ領域のデータが適切であることは保証される。しかし、前者の場合、変数の更新後にさらに別の処理で同じ変数が更新されると結果的に余計な通信が発生する一方、後者の場合は変数が更新されないにも関わらず、複数箇所で参照されると余計な通信が発生する。さらに、効率よくデータ通信を行うためには、複数の変数の通信をまとめる必要があり (荒波・石田 2008)、そのためには単純に上記のルールで対応できない。

そこで、更新された変数のリスト（以下、更新リストと言う）と更新リストに登録されていてかつ参照される変数、すなわち通信が必要な変数のリスト（以下、通信リストと言う）を準備し、通信リストに登録されている変数に対してのみ通信を行うこととする。この場合、上記のルールを少し変更して、

- 担当する領域の変数を更新した後に必ず更新リストに変数を登録する。
- 水平方向に隣接するデータを参照する前に、必ず更新リストに登録されているかチェックして、登録されていれば通信リストに変数を登録する。その後、通信リストに登録されている変数の通信を行う。

とすることにより、開発者にとっては変更箇所がほぼ一意に定まるために機械的にコーディングすることが可能となることに加え、不要な通信を行わずに高速化

を図ることが期待できる。ただし、これだけでは、変数をまとめて通信することができない。そこで、任意のタイミングで更新リストにある任意の変数を通信リストに登録して、今後通信が必要となる変数をあらかじめ通信しておくようにしている。

以上を踏まえて、現在の asuca では、以下の機能を持つサブルーチンを準備している。

- 担当する領域の変数を更新した後に更新リストに変数を登録する (mpi_halo_post_calc)。
- 水平方向に隣接するデータを参照する前に、「これからのりしろを使う」と宣言する (mpi_halo_pre_refer)。更新リストに変数が登録されていれば、通信リストに変数を登録し、更新リストから変数を抹消する。
- 実際に通信リストに登録されている変数全ての通信を行うサブルーチン (mpi_halo_comm_all) を呼ぶ。mpi_halo_pre_refer の後に呼ぶことが必須。通信リストに登録された変数を全て通信し、通信した変数を通信リストから削除する。

mpi_halo_post_calc と mpi_halo_pre_refer は上記のルールを実現するために更新リスト及び通信リストを書き換えるためのサブルーチンであり、mpi_halo_comm_all は全ての通信を行うためのサブルーチンである。

これらのサブルーチンの使い方と実際の通信のタイミング、及びリストに含まれる変数に関する模式図を図 A.1.3 に示した。ここで、サブルーチン modify_xxx と refer_xxx が数値予報モデルにおける処理であり、それぞれ時間積分等の変数を更新する処理と水平差分の計算等の変数を参照する処理を表している。変数毎に見ると余計な通信を行っていないことがわかる。また、任意のタイミングで任意の変数を通信するためのサブルーチンとして、図 A.1.4 に示したラッパーサブルーチン mpi_halo_run を準備している。図 A.1.3 の例で言えば、modify_a_3 と refer_b_1 との間で mpi_halo_run(a, b) として呼び出すことにより、変数 a, b の通信をまとめて行い、通信コストを減らすことができる。効率を追求するためには、サブルーチン mpi_halo_run を適切なタイミング・引数で呼び出す必要があり、コーディングの際はその見極めが重要である。

ただし、現在の asuca では、これらのサブルーチンはプログラム上は動作可能となっているものの、処理が若干煩雑なこと、上記のルールを徹底したことによる更新リストや通信リストの変更による処理の負荷について未確認であること、及び通信が必要なのりしろの幅が異なりうること（後述）への対応が終わっていないなどの理由により、mpi_halo_post_calc と mpi_halo_pre_refer の使い分けはせずに、JMA-NHM と同様にのりしろの交換が必要な変数の更新後にまとめて通信を行うようにしている。今後、mpi_halo_pre_refer については、参照が必要な変数の通信が必要な場合には通信を行うように変更するとともに、リストの変更に係る処理の負荷を確認し、高速化を図る等の対応を行った上で、上記ルールの徹底を図りたいと考えている。

(2) データの読み込みと各プロセスへのデータの分配

2/3 次元データのディスクからの読み込みと各プロセスへのデータの分配には mpi_comm_read_scatter_run というサブルーチンを準備している。これは、以下のように複数のサブルーチンのラッパーとなっている。

```
subroutine mpi_comm_read_scatter_run
  call read_to_buffer
  call scatter_one_record
  call scatter_buf2array
  call scatter_post
end subroutine mpi_comm_read_scatter_run
```

また、それぞれの関数は

```
read_to_buffer      :読み込みバッファの確保、
                     データの読み込み
scatter_one_record:通信 (受信用バッファへの格納)
scatter_buf2array :受信用バッファから
                     ユーザー配列へのコピー
scatter_post        :受信用バッファの開放
```

であり、ユーザーはこれらを個別に利用することもできる。

(3) 各プロセスからのデータの集約

全領域の総和計算や全データを集約してファイル出力などを行うためには、1 つのプロセスにデータを集約する必要がある。ファイル出力においては、出力を行う複数プロセスの制御（入出力専用プロセス方式を用いる場合）、といった独自の処理が必要なため、一般データ用のサブルーチンとは別に出力専用のサブルーチンを準備している。

一般のデータの集約

単にデータを集約する場合（たとえば、全領域のデータを集めて和をとる場合など）は、サブルーチン mpi_comm_gather を用いる。このサブルーチンを呼び出すと、計算プロセスにおいて集約したい配列が通信バッファに格納され、出力専用プロセスに送信される。次いで、出力専用プロセスに集約された配列は集約用の配列に格納される。集約したプロセスではこの集約用の配列を参照することにより、全領域の総和計算等を実行できる。このような通信パターンは非常に通信時間を要するため、なるべく避けるべきである。現在のところ、この方法はログ出力のための領域平均・最大・最小の計算と並列化した前処理のみで利用しており、asuca 本体の時間積分計算では、総和計算等の全領域のデータを集約する処理は不要としている。

```
call modify_a_1                  ! 変数 a の更新
call mpi_halo_post_calc(a, ...) ! 更新リストは変数 a、通信リストは無しとなる。
call modify_b_1                  ! 変数 b の更新
call mpi_halo_post_calc(b, ...) ! 更新リストは変数 a, b、通信リストは無しとなる。
call modify_a_2                  ! 変数 a の更新
call mpi_halo_post_calc(a, ...) ! 更新リストは変数 a, b、通信リストは無しとなる。
...
call mpi_halo_pre_refer(a, ...) ! 更新リストは変数 b、通信リストは変数 a となる。
call mpi_halo_comm_all           ! 変数 a の通信を行い、更新リストは変数 b、通信リストは無しとなる。
call refer_a_1                   ! 変数 a ののりしろ領域の参照
call mpi_halo_pre_refer(a, ...) ! 更新リストは変数 b、通信リストは無し（リストは変わらない）となる。
call mpi_halo_comm_all           ! 通信リストに登録された変数はないので何もしない。
call refer_a_2                   ! 変数 a ののりしろ領域の参照
...
call modify_a_3                  ! 変数 a の更新
call mpi_halo_post_calc(a, ...) ! 更新リストは変数 a, b、通信リストは無し。
...
call mpi_halo_pre_refer(b, ...) ! 更新リストは変数 a、通信リストが変数 b となる。
call mpi_halo_comm_all           ! 変数 b の通信を行い、更新リストは変数 a、通信リストは無しとなる。
call refer_b_1                   ! 変数 b ののりしろ領域の参照
...
call mpi_halo_pre_refer(a, ...) ! 更新リストは無し、通信リストは変数 a となる。
call mpi_halo_comm_all           ! 変数 a の通信を行い、更新リスト・通信リストともに無しとなる。
call refer_a_3                   ! 変数 a ののりしろ領域の参照
```

図 A.1.3 通信サブルーチンの使い方の例とその挙動についての模式図。サブルーチン `modify_xxx` が変数を更新する処理、`refer_xxx` が変数を参照する処理を表す。`mpi_halo_post_calc` と `mpi_halo_pre_refer` はそれぞれ `modify_xxx` と `refer_xxx` と 1 対 1 に対応している。適切なタイミングで通信を行っており、余計な通信を行っていない。

```
subroutine mpi_halo_run(a, b, ..., y, z)
...
call mpi_halo_post_calc(a, ...) ! 更新リストは変数 a、通信リストは無しとなる。
call mpi_halo_pre_refer(a, ...) ! 更新リストは変数無し、通信リストは変数 a となる。
call mpi_halo_post_calc(b, ...) ! 更新リストは変数 b、通信リストは変数 a となる。
call mpi_halo_pre_refer(b, ...) ! 更新リストは変数無し、通信リストは変数 a, b となる。
...
call mpi_halo_post_calc(z, ...) ! 更新リストは変数 z、通信リストは変数 a-y となる。
call mpi_halo_pre_refer(z, ...) ! 更新リストは変数無し、通信リストは変数 a-z となる。
call mpi_halo_comm_all           ! 変数 a-z の通信を行い、更新リスト、通信リストの変数は無しとなる。
```

図 A.1.4 任意の場所で a から z の変数の通信を行う通信サブルーチンのラッパー。新しく作成した通信サブルーチンを用いている。1 変数用、2 変数用…とまとめて通信したい変数の数に応じてサブルーチンを準備すれば、任意の場所で任意の変数の通信を行うことが可能になる。なお、実装では引数は全て `optional` 属性とすることにより、引数として渡した変数のみを通信するようにして、一つのサブルーチンにまとめている。

出力用データの集約

出力用のデータを集約する場合は、サブルーチン `output_gather_pre` を用いる。このサブルーチンが呼ばれると、計算プロセスにおいて、出力用の変数を格納するモジュール変数にデータが格納され、その後 `output_gather_comm_and_out` を呼ぶことにより、出力専用プロセスへの通信およびファイルへの書き込みが行われる。

A.1.5 OpenMP の利用

asuca は OpenMP による指示行を用いてプロセス内の並列化（メモリ共有並列）を行っている。JMA-NHM においては特に指示行等は用いずにコンパイラによる自動並列化を用いていたが、さらなる高速化を図ることがその目的である。

現在のところ、物理過程は鉛直 1 次元とみなすことができ、物理過程ライブラリを実装に用いていることは第 4.1 節で既に述べた。そのため、物理過程の呼び出しは水平方向のループが外側となり、コーディングは図 A.1.5 のようになる。OpenMP による並列化においては、プロセス内でそれぞれ処理を行うスレッドの起動・終了の時間を減らすためにも、最外側で並列化を適用することが望ましい。物理過程ライブラリが鉛直 1 次元で実装されていることから、OpenMP と組み合わせることにより最外側で並列化を適用することが可能となっている。なお、雲物理過程とそれ以外の物理過程はそれぞれまとめて並列化を行っている [4] 。

[4] 雲物理過程が分かれる理由は第 4.2.2 項で述べた通り計算安定性確保のためである。当然のことながら計算効率より計算安定性が優先される。

```
!$OMP PARALLEL DO
do j = 1, ny
do i = 1, nx
  call A ! 物理過程ライブラリのサブルーチン
  call B ! 物理過程ライブラリのサブルーチン
  call C ! 物理過程ライブラリのサブルーチン
  ...
end do
end do
!$OMP END PARALLEL DO
```

図 A.1.5 物理過程ライブラリを用いて OpenMP による並列化を行うコードの例。

```
do k = 1, nz
do j = 1, ny
do i = 1, nx
  a(i,j,k) = ...
end do
end do
end do
```

図 A.1.6 a(i,j,k) とインデックスを用いる場合の三重ループのコーディングの例。

A.1.6 配列のとりかた: kij 方式

(1) コーディング

asuca のコーディングの特徴の一つに配列の次元の取り方がある。Fortran の配列を考えると、多くのモデルでは a(i, j, k) や a(i, k) のように水平方向の格子数を表すインデックス（ここでは i と j）を鉛直層数を表すインデックス（ここでは k）より内側に配置することが多い。その場合、配列の全ての要素にわたるループはメモリアクセスを連続にしようとすると、通常は図 A.1.6 のようになる。気象モデルの特徴として、多くの場合に鉛直層の数よりも東西もしくは南北方向の格子数が多いことや物理過程を中心に鉛直方向に依存性がある処理が水平方向に依存性がある処理よりも多いことがあげられる。鉛直方向に依存性がある場合は k のループでは並列化が適用できず、i または j のループで並列化を適用することとなり、スレッドの起動・終了の時間を要してしまう。そこで、図 A.1.7 といったコーディングスタイルをとることにより、鉛直方向に依存性があっても外側で並列化でき、かつ外側のループ長が大きいため並列化効率も高くなる。また、格子点やカラムに含まれる変数を使って大量の計算を行う場合は、図 A.1.8 のように ループを融合することにより、必要な変数が全てキャッシュにのったまま、大量の計算を行える可能性がある。

以上のことを考慮して、asuca では a(k, i, j) というようにインデックスを用いる。なお、x, y, z 方向

```
!$OMP PARALLEL DO
do j = 1, ny
do i = 1, nx
do k = 1, nz
  a(k,i,j) = ...
end do
end do
end do
!$OMP END PARALLEL DO
```

図 A.1.7 a(k,i,j) とインデックスを用いる場合の三重ループのコーディングの例。

```
!$OMP PARALLEL DO
do j = 1, ny
do i = 1, nx
  ...
  do k = 1, nz
    a(k,i,j) = ...
    b(k,i,j) = ...
    c(k,i,j) = ...
  end do
  ...
end do
end do
!$OMP END PARALLEL DO
```

図 A.1.8 大量の三重ループを融合したコーディングの例。

のループのインデックスとして、一般的に i, j, k が用いられることから a(i, j, k) のような利用方式を「ijk 方式」と呼ぶのに対し、asuca のような方式を「kij 方式」と呼んでいる。

(2) ファイルのフォーマット

上記で述べた「kij 方式」とは、あくまでもモデル本体や前処理のコーディングスタイルとして用いているものであり、入出力ファイルにはユーザーの利便性を考慮して他のモデルと同様にしている（上述の呼び方では「ijk 方式」）。

asuca の処理ではモデル本体や前処理の全てにおいて、3 次元データの入力で読み込む「ijk 方式」のデータをプログラムの内部処理である「kij 方式」に変更する必要があり、またデータの出力の際は、逆に「kij 方式」の配列を「ijk 方式」に変更する必要がある。

A.2 新規にコードを組み込む際の考え方

A.2.1 初期化部と時間積分ループ部の区別

asuca では、変数の初期値の設定といった、計算の最初のみに行う処理（初期化）と、時間積分ループの中で繰り返し計算が行われる処理とをコーディング上、

明確に区別している。初期化の処理は、時間積分ループに入る前にサブルーチン xxxx_ini により行う。これにより、モデルの初期化部分と時間積分ループが明確に区別されて見通しがよくなる。

A.2.2 診断量の計算

物理過程や力学過程で必要な診断量をそれぞれの処理で計算するのは、計算コストやメモリ利用の面から見て不適当である。そこで、時間積分やルンゲクッタ法の仮の時間積分により予報変数の値が更新されたタイミングで、時間変化率の計算に必要な変数をまとめて診断し、各処理でそれらを利用する、という設計にしている。

A.3 変数

A.3.1 命名規則

asuca で使う変数の大まかな命名規則は以下の通りである。なお、格子配置については第 2.2.1 項を参照のこと。

- _x：x 方向を示す。また、x 方向に半格子ずれたポイントを表すところもある。y, z も同様。
- _xi :計算空間における x 方向を示す。y, z も同様。
- _f：セル中心のレベル（フルレベル）における値
- _h：w が定義されるレベル（ハーフレベル）における値
- tend_：時間変化率
- 先頭の r：密度 (ρ) がかかっているもの
- _v：体積 ($1/J$) がかかっているもの
- ptb_：基本場からの偏差
- _l：ルンゲクッタループの外の f^t
- _s：ショートタイムステップ中の f^t
- _a：アジャストメントに使われる f^t
- _rk_s：ショートタイムステップ中の f^*, f^{**}
- _rk：ルンゲクッタループ (長い方) の f^*, f^{**}

A.3.2 予報変数

asuca で用いる予報変数の一覧を表 A.3.1 に示す。なお、rqa_v でまとめて確保される変数は、rqa_v : id_XX のように示す。予報変数ののりしろの幅は、移流の精度から決まる nx_mgn, ny_mgn [5] で与える。

A.3.3 診断変数

次に、診断変数の一覧を表 A.3.2 に示す。これらは物理過程や力学過程の処理における入力として利用され、複数の処理で共用されることが前提となっている。表 A.3.2 には、変数ののりしろの幅と目的を示している。のりしろの幅は、移流のフラックスの評価、内挿など目的によって異なるためである。

[5] nx_mgn, ny_mgn は 2 以上とする。

表 A.3.1 予報変数。ただし、概要説明においてヤコビアンは省略している。

変数名	概要	式
mom_x_v	x 方向の運動量	$\rho u/J$
mom_y_v	y 方向の運動量	$\rho v/J$
mom_z_v	y 方向の運動量	$\rho w/J$
dens_ptb_v	密度の偏差	ρ'/J
rmpt_ptb_v	$\rho\theta_m$ の偏差	$(\rho\theta_m)'/J$
rqa_v : id_qv	水蒸気の密度	$\rho q_v/J$
rqa_v : id_qc	雲水の密度	$\rho q_c/J$
rqa_v : id_qr	雨の密度	$\rho q_r/J$
rqa_v : id_qi	雲氷の密度	$\rho q_i/J$
rqa_v : id_qs	雪の密度	$\rho q_s/J$
rqa_v : id_qg	霰の密度	$\rho q_g/J$
rqa_v : id_qke	2× 乱流エネルギー	$\rho\overline{u_i'u_i'}/J$
rqa_v : id_tsq	液水温位の揺らぎの自己相関	$\rho\overline{\theta_l'^2}/J$
rqa_v : id_qsq	総水混合比の揺らぎの自己相関	$\rho\overline{q_w'^2}/J$
rqa_v : id_cov	$\overline{\theta_l'}$ と $\overline{q_w'}$ の相関	$\rho\overline{\theta_l'q_w'}/J$

A.3.4 変数ののりしろについて

以下では、変数ののりしろについて補足する。前述のとおり、予報変数ののりしろとして、幅 nx_mgn, ny_mgn の格子を確保する。また、診断変数はその目的に応じてのりしろが異なる。のりしろの値は、予報変数についてはのりしろを交換し、診断変数についてはのりしろを含めた範囲で予報変数から診断する方法を基本とする。なお、ひとつの変数について、場面に応じて必要なのりしろが異なることもある。例えば、x 方向の風 (vel_x) は、移流のフラックス評価のためにはのりしろ幅 nx_mgn が必要であるが、内挿処理のためにはのりしろ幅 1 の範囲のデータを通信すれば良い。このような場合、のりしろ幅は最大の大きさ（この例では nx_mgn）を取っておき、必要となる場所において、通信を行うのりしろ幅を指定するようにしている。以下では、のりしろの扱いを分類したものを示す。

(1) 予報変数

$\rho u/J$, $\rho v/J$, $\rho w/J$, ρ'/J, $(\rho\theta_m)'/J$, $\rho q_x/J$

- のりしろの幅: nx_mgn, ny_mgn で与える。
- 東、西、南、北と隣接しているプロセスとのりしろを通信する。
- 北東、北西、南東、南西のプロセスとも通信を行う必要がある [6]。

[6] 例えば x 方向に半格子ずれた点での y 方向の気圧傾度力を考える際に、x, y の両方向に半格子ずれた点での気圧が必要となる。予報変数からのりしろ領域を含めて気圧を診断するため、予報変数を通信しておく。ただし、移流のフラックス評価に使う場合、この通信は必須ではない（ただし、現状は通信している）。

表 A.3.2 診断変数。 のりしろの幅の mgn は nx_mgn, ny_mgn で与えることを示す.

変数名	概要	式	のりしろ幅：のりしろの目的
vel_x	x 方向の風	u	mgn:フラックス評価
vel_y	y 方向の風	v	mgn:フラックス評価
vel_z	z 方向の風	w	mgn:フラックス評価
mom_xi_v	x 方向の運動量（計算空間）	$\rho U/J$	1:移流速度
mom_yi_v	y 方向の運動量（計算空間）	$\rho V/J$	1:移流速度
mom_zi_v	z 方向の運動量（計算空間）	$\rho W/J$	1:移流速度
dens_ptb	密度の偏差	ρ'	mgn:フラックス評価、 u 等の算出
mptmp	θ_m	θ_m	mgn:フラックス評価
pres_ptb	気圧の偏差	p'	1:圧力の水平勾配
exner	エクスナー関数	π	1:圧力の水平勾配
ptemp	温位	θ	mgn:水平拡散
temp	気温	T	mgn:観測演算子
qa : id_qv	水蒸気の密度と全密度の比	q_v	mgn:フラックス評価
qa : id_qc	雲水の密度と全密度の比	q_c	mgn:フラックス評価
qa : id_qr	雨の密度と全密度の比	q_r	mgn:フラックス評価
qa : id_qi	雲氷の密度と全密度の比	q_i	mgn:フラックス評価
qa : id_qs	雪の密度と全密度の比	q_s	mgn:フラックス評価
qa : id_qg	霰の密度と全密度の比	q_g	mgn:フラックス評価
qa : id_qke	2× 乱流エネルギー	$\overline{u_i'u_i'}$	mgn:フラックス評価
qa : id_tsq	液水温位の揺らぎの自己相関	$\overline{\theta_l'^2}$	mgn:フラックス評価
qa : id_qsq	総水混合比の揺らぎの自己相関	$\overline{q_w'^2}$	mgn:フラックス評価
qa : id_cov	$\overline{\theta_l'}$ と $\overline{q_w'}$ の相関	$\overline{\theta_l'q_w'}$	mgn:フラックス評価

- ネストの場合は外から与える。
- 周期境界条件の場合は反対側から与える。

(2) 移流のフラックス評価に使う変数

u, v, w, θ_m, q_a, ρ'

- のりしろの幅: nx_mgn, ny_mgn で与える。
- nx_mgn, ny_mgn の範囲で予報変数から診断する。ただし、 u, v は ρ' と水平方向の定義ポイントが異なることから全ての範囲では正しい値にはならないため診断後に通信が必要になる。
- この目的においては、北東、北西、南東、南西のプロセスと通信を行う必要はない。

(3) 水平拡散する可能性のある変数

u, v, w, θ_m, θ, q_v

- 移流のフラックス評価に使う変数と同様の属性（のりしろ幅は水平拡散の次数によるが、現状は特に区別しない）。

(4) 移流速度

$\rho U/J$, $\rho V/J$, $\rho W/J$ （移流項を評価する際、この変数を移流速度として使っている。）

- のりしろの幅:1
- のりしろの値は予報変数から診断（通信はしない）。

(5) 圧力の水平勾配

p', π

- のりしろの幅:1
- のりしろの値は予報変数から診断（通信はしない）。

(6) 観測演算子

観測地点への内挿処理（4 点内挿）のため、観測演算子の計算にのりしろの幅 1 が必要となる。気温、海面気圧、地表面気圧、地上物理診断量及びその診断に必要な変数が対象となる。

- のりしろの幅:1
- のりしろ幅 1 を含む範囲で診断する。
- 計算範囲にのりしろを含まない変数に対しては、計算後にのりしろを通信する必要がある。

(7) 時間変化率

時間変化率を表す変数はのりしろを持たない。例外として、あるポイントで求めた時間変化率を半格子ずれたポイントに内挿するためにのりしろ幅 1 を持たせる場合がある。

(8) モニタ変数

モニタのために配列を確保する変数については、のりしろは不要である。

参考文献

荒波恒平, 石田純一, 2008: 並列化と高速化. 数値予報課報告・別冊第 54 号, 気象庁予報部, 58–65.

石田純一, 荒波恒平, 2003: 並列化. 数値予報課報告・別冊第 49 号, 気象庁予報部, 107–111.

室井ちあし, 1998: 非静力モデルの開発. 数値予報課報告・別冊第 44 号, 気象庁予報部, 25–41.

室井ちあし, 2000: 非静力学メソモデル・飛行場予報モデルの開発と将来展望. 数値予報課報告・別冊第 47 号, 気象庁予報部, 114–121.