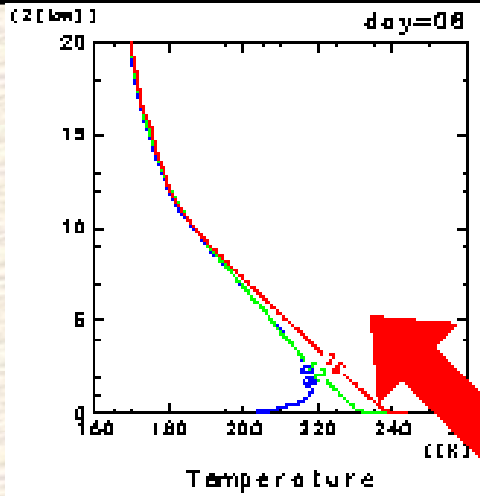
http://www.gfd-dennou.org

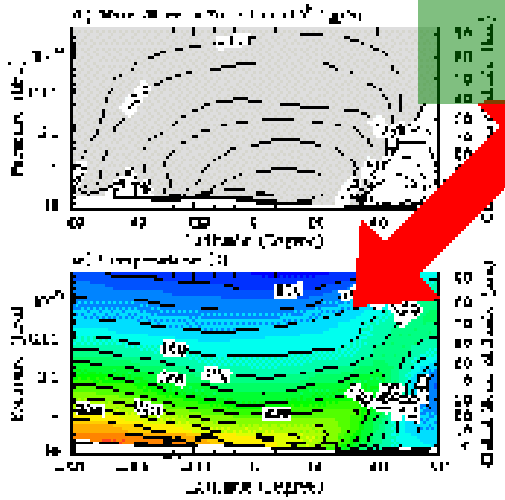
階層的モデル群の開発・整備のための基盤技術開発

○ 森川 靖大 (北大・理)
石渡 正樹 (北大・地球環境)
林 祥介 (神戸大・理)
地球流体電脳倶楽部開発グループ

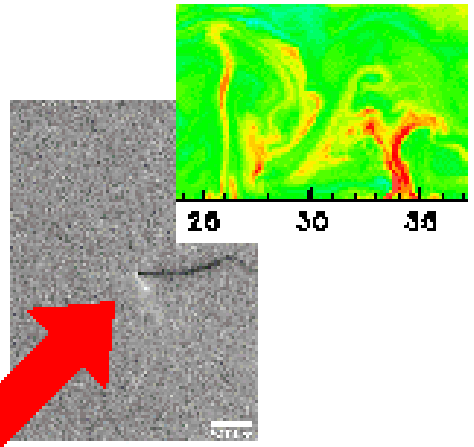
階層的モデル群とは



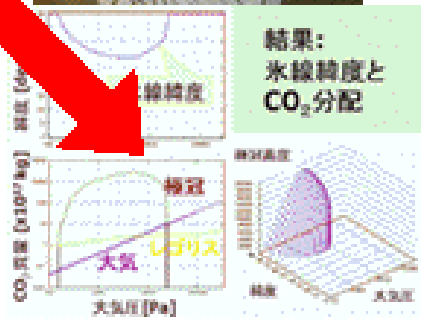
放射対流モデルによる
鉛直大気構造の理解



GCM による大気大循環の研究



雲解像モデルによる
ダストデビルの再現



EBM による極冠の
成長・後退の計算

なぜ階層的モデル群か

- **様々な空間スケールの現象達を物理的に考察し、その流体力学的構造を理解するため**
  - GCM (3 次元)
    - 力学 (静力学) ＋ 放射 (簡単化) ＋ 積雲･地表面過程 (パラメタリゼーション) ＋ …
  - 雲解像モデル (2 次元)
    - 力学 (準圧縮) ＋ 雲微物理･放射･乱流混合過程(パラメタリゼーション)
  - 放射対流モデル (1次元)
    - 放射 ＋ 鉛直対流 (簡単化)
  - エネルギーバランスモデル (1次元)
    - 南北エネルギー輸送 ＋ 放射 (簡単化)
- **簡単化する部分は、他のモデルによってその正当性を保障する必要がある**
  - モデル毎に真面目に扱う部分と簡単化する部分があるから

何が問題か

- 計算結果の比較が面倒
  - 各モデルから出力されるデータ形式が異なっていると、計算結果の比較のために、それぞれのデータの構造を知り、それらに合わせた可視化･解析ツールを用意しなければならないので面倒
- モデルが何を計算しているか知るのが面倒
  - プログラムが複雑なため、読みやすいコーディングを行わないと、元々何が計算されているのか読み解くのが面倒
  - 複数のモデル群で書き方が異なっていると、モデルの内容(何を計算しているか) を比較する上で面倒
- モデルを構成する個々のプログラムの共有が面倒
  - プログラム構造がモデル毎に大きく異なっていると、結局モデルのほとんどを読み解いて理解する必要があるため面倒

どのようになるのが望ましいのか

- 計算結果を簡単に比較するために
  - データ自身が、データが何であるか知っているようなデータ構造を扱える (入出力する) モデル群が欲しい
- モデルが何を計算しているか簡単に知りたために
  - ソースコードの書式が揃ったモデル群が欲しい
    - ソースコードを読むことでどのような方程式系を解いているか簡単に理解したい
    - データ I/Oなどに関する煩雑なコードは読みたくない
- モデルを構成するプログラムを簡単に共有するために
  - モデルの細部を理解せずとも、各パーツの着脱が可能なモデル群が欲しい
    - モデル群でプログラム構造を揃っていて欲しい
    - モデルの解説文書 (各ルーチンの使用方法や概要) が整備されていて欲しい

方針

- 使う人　作る人の視点で
  - 使いやすいモデル自体の開発とともに、作る人がいかに簡単に使いやすいモデルを作れるか (= 開発手法自体) も考える
- FORTRAN77 時代の AGCM5 を参考にする
  - AGCM5：プログラミング書法に工夫
- プログラミング言語には Fortran 90/95 を使用
  - 新しい機能を積極的に活用
    - モジュール、総称名称、構造体、配列関数、etc...
  - 実行効率もそこまで犠牲にしない (速くする事は望まない)
    - 「実行効率の低下 ≦ モデル設定・ソースコード変更作業のコストダウン」程度を目指す
- 開発・整備支援にはスクリプト言語を積極的に利用
  - 現在、オブジェクト指向スクリプト言語 Ruby を使用

現在継続中の試み

- 可搬性や自己記述性に優れたデータ構造
  - gtool4 netCDF 規約の策定
- データ I/Oなどに関する煩雑なコードの隠蔽
  - データ入出力ライブラリ gt4f90io の開発・整備
- ソースコードを見ただけで何を計算しているか簡単に分かるための工夫
  - ソースコードを元の支配方程式の数学的表現に似せる
    - 配列を返すユーザ定義関数の活用
    - 関数と変数の命名規則の策定
- モデルの細部を理解せずに各パーツの着脱
  - モジュール機能を活かしたプログラム構造の設計
- モデルの解説文書の作成を容易に
  - オブジェクト指向スクリプト言語 Ruby のドキュメント自動生成ライブラリ RDoc によるモデル解説文書の自動生成

現在継続中の試み

- 可搬性や自己記述性に優れたデータ構造
  - gtool4 netCDF 規約の策定
- データ I/Oなどに関する煩雑なコードの隠蔽
  - データ入出力ライブラリ gt4f90io の開発・整備
- ソースコードを見ただけで何を計算しているか簡単に分かるための工夫
  - ソースコードを元の支配方程式の数学的表現に似せる
    - 配列を返すユーザ定義関数の活用
    - 関数と変数の命名規則の策定
- モデルの細部を理解せずに各パーツの着脱
  - モジュール機能を活かしたプログラム構造の設計
- モデルの解説文書の作成を容易に
  - オブジェクト指向スクリプト言語 Ruby のドキュメント自動生成ライブラリ RDoc によるモデル解説文書の自動生成

データ構造：背景

- モデルごとにデータ構造が違うと大変
  - モデル間での結果の比較が面倒
- 我々が扱いやすいデータ構造とは？
  - 可搬性に優れる
    - スパコン上でもパソコン上でも同様に取り扱える
  - 自己記述性に優れる
    - データが何者であるか、データ自身が知っている
  - 様々な次元のデータを取り扱える
    - 1 次元モデルのデータも 3 次元モデルのデータも

NetCDF

- **データ形式としてnetCDF (Network Common Data Format)**
  - http://www.unidata.ucar.edu/software/netcdf/
  - **米国の** Unidata **で開発**
  - **高い可搬性**
    - **スパコン上でもパソコン上でも変換無しに取り扱い可能**
  - **自己記述的なファイル形式**
    - **データファイルはメータデータと数値データから成る**
    - **メタデータの例: 変数の記述的名称、単位など. 履歴、作成者など**
  - **多次元データの取り扱いが可能**
    - **次元の数を制限しない**
- **規約が必要**
  - **メタデータの記述の仕方のとりきめ**
  - **気象業界で広く使われつつある規約は**CF**規約**
    - CF**規約**: http://www.cfconventions.org/
    - CF**規約の特徴** – standard names
      - **気象業界でよく使われる物理量に関して標準となる名前を規定**
      - **例**: **南北風速**: northward_wind, **地表面気圧** surface_air_pressure

gtool4 netCDF 規約

- 我々が取り扱う上で必要最低限な情報を必須メタデータとして規定
  - タイトル、データがどのように作られたか、履歴
  - 変数の記述的名称、単位
- 互換性
  - 元々は COARDS, CMS 規約互換として作成
  - 現在は CF規約互換 (調整中)
    - CF 規約には「必須」メタデータと規定されるものがほとんど無いためそのままでは使いづらい (「推奨」は山ほど)
    - standard name は積極的に使用

gtool4 netCDF 規約まとめ

- http://www.gfd-dennou.org/library/gtool4
- モデル群から得られる結果を簡単に比較するためのデータ構造
- netCDF データ形式により高い可搬性と自己記述性
- 最低限必要なメタデータを規定
- CF 規約との互換性 (調整中)

現在継続中の試み

- 可搬性や自己記述性に優れたデータ構造
  - gtool4 netCDF 規約の策定
- データ I/Oなどに関する煩雑なコードの隠蔽
  - データ入出力ライブラリ gt4f90io の開発・整備
- ソースコードを見ただけで何を計算しているか簡単に分かるための工夫
  - ソースコードを元の支配方程式の数学的表現に似せる
    - 配列を返すユーザ定義関数の活用
    - 関数と変数の命名規則の策定
- モデルの細部を理解せずに各パーツの着脱
  - モジュール機能を活かしたプログラム構造の設計
- モデルの解説文書の作成を容易に
  - オブジェクト指向スクリプト言語 Ruby のドキュメント自動生成ライブラリ RDoc によるモデル解説文書の自動生成

モデルのデータI/O：背景

- 自己記述的データの入出力(I/O)に関する問題
  - ソースコードが煩雑になる
    - ファイルID や 変数ID の管理
  - ソースコードの書き方が揃わない
    - 同じ形式のファイル出力に対して様々な書き方がある

gt4f90io ライブラリ

- **gtool4 規約に基づく Fortran90 netCDF I/O ライブラリ**
  - gtool4 netCDF **規約に基づくデータを入出力するためのライブラリ**
  - Fortran 90/95 **で書かれたプログラムで使用**
  - **データ入出力のための簡潔なインターフェース (サブルーチン群) を提供**
  - **その他、汎用なライブラリも同梱**
    - **文字列処理、デバッグ支援、CPU時間計測、メッセージ出力、コマンドライン引数の処理、テストプログラム作成支援** etc...

gt4_history

- gt4f90io ライブラリに含まれるデータI/O用モジュール
- 最低限 5 つのサブルーチンでデータの入出力が可能

- HistoryCreate(file, title, ...)
  - 初期設定
    - 出力ファイル名、タイトル、...、次元変数名、次元サイズ、...
- HistoryAddVariable(varname, dims, ...)
  - 変数定義
    - 変数名、依存次元名、...
- HistoryPut(varname, value, ...)
  - 変数出力
    - 変数名、出力値、...
- HistoryClose
  - 終了処理
- HistoryGet(file, varname, ...)
  - 変数入力
    - ファイル名、変数名、...

違うデータ型も
同サブルーチンで対応

gt4_history 使用例


```
program sample
  use gt4_history                                          ! モジュールの使用を宣言
  [型宣言] ......
  call HistoryGet( file='init.nc' ... )                    ! 初期値入力

  call HistoryCreate( &                                    ! ヒストリー作成
       file='sample.nc', title='gt4_history',    &         !    ・ファイル名、タイトル
       ..., dims=(/'x','t'/), dimsizes=(/30,0/), &         !    ・次元変数、次元サイズ
       .......)
  call HistoryAddVariable( &                               ! 変数定義
       varname='temp', dims=(/'x','t'/), .... )            !    ・変数名、依存次元、..

  [時間積分ループ]
         :
       call HistoryPut(varname= 'temp', value=temp) ! 変数の出力
         :
  [時間積分ループ 終わり]

  call HistoryClose                                        ! 終了の処理
  stop
end program sample
```

gt4f90io まとめ

- http://www.gfd-dennou.org/library/gtool4
- gtool4 netCDF 規約に基づくデータを入出力するための Fortran 90/95 プログラム用ライブラリ
- モデル群のデータI/Oに使用
  - モデル側のデータI/O関連のソースコードが簡潔になり、書き方が揃う

現在継続中の試み

- 可搬性や自己記述性に優れたデータ構造
  - gtool4 netCDF 規約の策定
- データ I/Oなどに関する煩雑なコードの隠蔽
  - データ入出力ライブラリ gt4f90io の開発・整備
- ソースコードを見ただけで何を計算しているか簡単に分かるための工夫
  - ソースコードを元の支配方程式の数学的表現に似せる
    - 配列を返すユーザ定義関数の活用
    - 関数と変数の命名規則の策定
- モデルの細部を理解せずに各パーツの着脱
  - モジュール機能を活かしたプログラム構造の設計
- モデルの解説文書の作成を容易に
  - オブジェクト指向スクリプト言語 Ruby のドキュメント自動生成ライブラリ RDoc によるモデル解説文書の自動生成

コーディングルール：背景

- 問題点
  - プログラムが複雑なため、読みやすいコーディングを行わないと、元々何が計算されているのか読み解くのが面倒
  - 複数のモデル群で書き方が異なっていると、モデルの内容 (何を計算しているか) を比較する上で面倒
- 解決策
  - プログラム構造を階層化し、離散化 (スペクトル法など) 部分は下部のライブラリに隠蔽
  - 複数のモデル群で利用可能なプログラム書法を決め、その書法でソースコードを記述

例題

- **GCM で使用される渦度方程式 (多少簡略化)**

$$\frac{\partial \zeta(t)}{\partial t} = \frac{1}{a\cos\varphi}\left(\frac{\partial V_A(t)}{\partial \lambda} - \frac{\partial (v\cos\varphi) U_A(t)}{\partial \varphi}\right)$$

$$\zeta(t+\Delta t) = \zeta(t-\Delta t) + 2\Delta t \times \frac{\partial \zeta(t)}{\partial t}$$

λ ：経度
φ ：緯度
σ ：圧力/地表面圧力
t ：時刻
a ：惑星半径
f ：コリオリパラメータ

$\zeta\ (\lambda, \varphi, \sigma, t)$ ：渦度
$U_A\ (\lambda, \varphi, \sigma, t) = (\zeta + f)\ v \cos\varphi$
$V_A\ (\lambda, \varphi, \sigma, t) = -(\zeta + f)\ u \cos\varphi$
$u\quad (\lambda, \varphi, \sigma, t)$ ：東西風速
$v\quad (\lambda, \varphi, \sigma, t)$ ：南北風速

AGCM5 (F77) でのプログラム例


```
      CALL SMTV2S
     I      ( MMAX , IMAX , IDIM, JMAX, JDIM, KMAX,
     I        GTUA , GTVA ,
     O        WTDIV, WTVOR,
     W        WORK,
     I        IT, T, IP, P, QUSDER, R, ML)

      CALL SMTG2S
     I      ( MMAX , IMAX, IDIM, JMAX, JDIM, KMAX,
     I        GBVOR,
     O        WBVOR,
     W        WORK,
     I        IT, T, IP, P, QGS )

      DO 7100  K = 1 , KMAX
         DO 7100  NM = 1 , NMDIM
            WAVOR( NM,K ) =  WBVOR( NM,K ) + WTVOR( NM,K )*DELT2
7100 CONTINUE

      CALL SMTS2G
     I      ( MMAX, IMAX, IDIM, JMAX, JDIM, KMAX,
     I        WAVOR,
     O        GAVOR,
     W        WORK,
     I        IT, T, IP, P, QGS )
```

$$\frac{\partial \zeta(t)}{\partial t} = \frac{1}{a\cos\varphi}\left(\frac{\partial VA(t)}{\partial \lambda} - \frac{\partial (v\cos\varphi)UA(t)}{\partial \varphi}\right)$$

$$\zeta(t+\Delta t) = \zeta(t-\Delta t) + 2\Delta t \times \frac{\partial \zeta(t)}{\partial t}$$

- **工夫**
  - 変数命名規則
  - プログラム書法
- **可読性低下の原因**
  - サブルーチンコール
  - 配列サイズ情報
  - WORK 領域
  - 6 文字制限

階層的モデル群でのプログラム例

- 数式に近い形でソースコードを記述可能に

$$\frac{\partial \zeta(t)}{\partial t} = \frac{1}{a\cos\varphi}\left(\frac{\partial V_A(t)}{\partial \lambda} - \frac{\partial (v\cos\varphi) U_A(t)}{\partial \varphi}\right)$$

$$\zeta(t+\Delta t) = \zeta(t-\Delta t) + 2\Delta t \times \frac{\partial \zeta(t)}{\partial t}$$

```
wz_DVorDtN =                                                  &
       & wa_Div_xya_xya( xyz_VaN ,  - xyz_UaN )  &
       & / Rplanet
xyz_VorA    =                                                 &
       &  xya_wa( wa_xya( xyz_VorB )                          &
       &           + 2. * DelTime * wz_DVorDtN )
```

配列演算関数 (F90/95) の活用

- Fortran 90/95 では配列の演算を一行で記述可能
  - このような、配列を返す関数をユーザも定義可能

```
! FORTRAN 77 style
      REAL*8    A(10,10), B(10,10)
      INTEGER   I, J
  ...
      DO 1000 J=1,10
          DO 1000 I=1,10
              B(I,J) = EXP( A(I,J) )
 1000 CONTINUE
```

```
! Fortran 90/95 style
real(8) :: a(10,10), b(10,10)
...
b = exp(a)
```

変数・関数命名規則

- **変数**
  - (次元の接頭詞)_(物理的意味)(時間方向添字) のように書く
    - 経度 (1 次元) x_Lon
    - 温度 (3 次元, 格子点) 時刻 $t-\Delta t$ xyz_TempB
    - 渦度 (1 次元, スペクトル) 時刻 $t+\Delta t$ w_VorA
- **配列関数**
  - (出力型)_(機能)_(入力型) のように書く
    - スペクトル変換 w_Vor=w_xy(xy_Vor)
    - 逆変換 xy_Vor=xy_w(w_Vor)
    - 発散 w_Div=w_Div_xy_xy(xy_U, xy_V)
  - 引数の型を間違えることが少ない

ソースコードを数学的表現に近づける試み まとめ

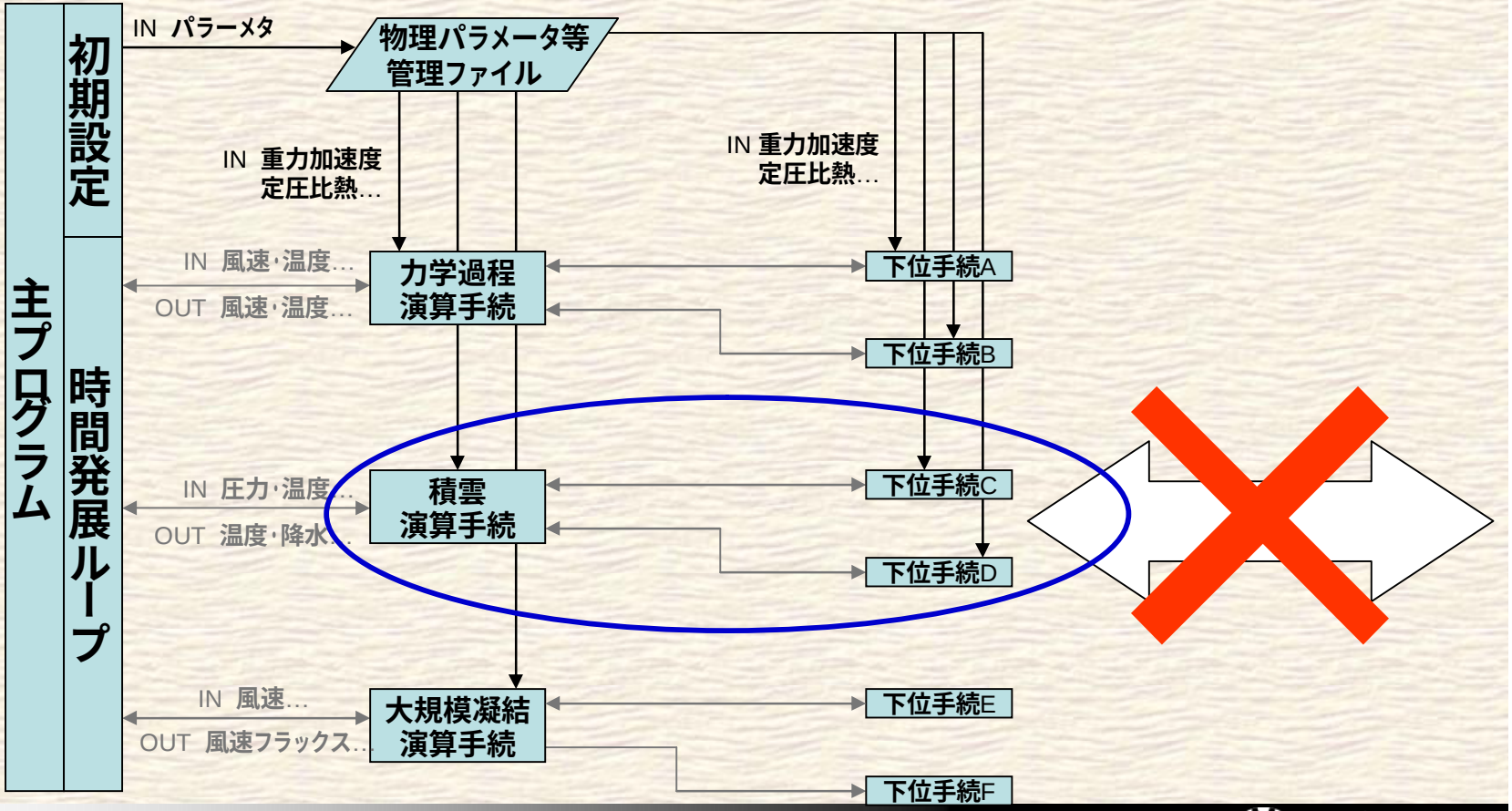
- http://www.gfd-dennou.org/library/dcmodel
  - dcmodel プログラミングガイドライン
- 配列を返すユーザ定義関数の活用
- 変数・関数命名規則
- FORTRAN77 の場合に比べ、ソースコードをより数式に近い形で記述できる
- モデル群が何を計算しているかを理解しやすくなる事が期待できる
- 課題
  - 関数利用によってどれくらい実行速度が落ちるか？？
  - 並列化対応の際にも読みやすさは維持できるか？？

現在継続中の試み

- 可搬性や自己記述性に優れたデータ構造
  - gtool4 netCDF 規約の策定
- データ I/Oなどに関する煩雑なコードの隠蔽
  - データ入出力ライブラリ gt4f90io の開発・整備
- ソースコードを見ただけで何を計算しているか簡単に分かるための工夫
  - ソースコードを元の支配方程式の数学的表現に似せる
    - 配列を返すユーザ定義関数の活用
    - 関数と変数の命名規則の策定
- モデルの細部を理解せずに各パーツの着脱
  - モジュール機能を活かしたプログラム構造の設計
- モデルの解説文書の作成を容易に
  - オブジェクト指向スクリプト言語 Ruby のドキュメント自動生成ライブラリ RDoc によるモデル解説文書の自動生成

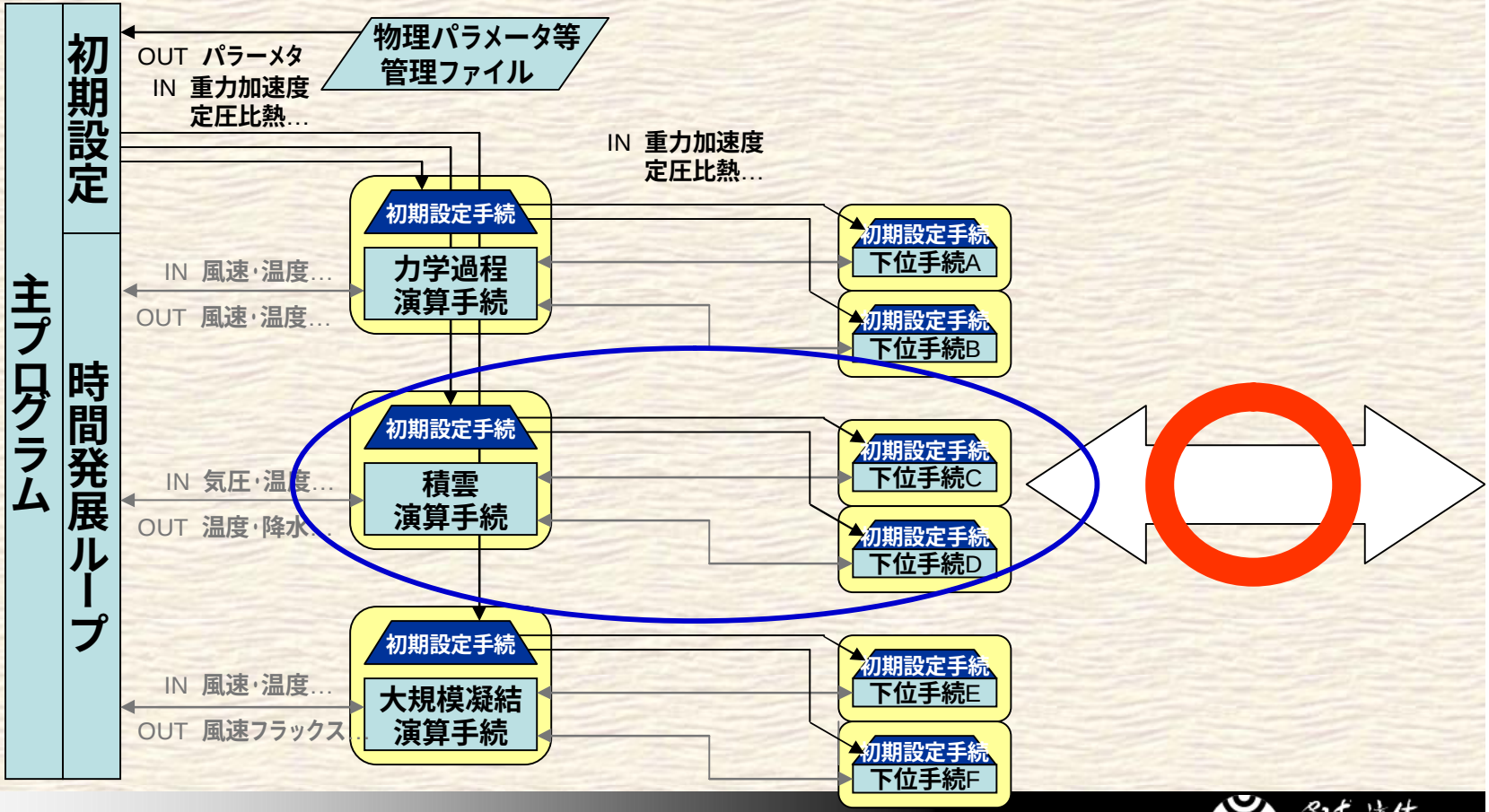
AGCM5 (F77) のプログラム構造

- 個々の演算で必要なパラメータを1つのファイルで集中管理する方法の問題点
  - モデルの一部を交換にはソースコード解読が必要



階層的モデル群での試み

- 個々の演算で必要なパラメータはモジュール毎に保持
  - 各モジュールで初期設定手続を用意し、その手続でパラメータを設定
  - スキーム交換に必要な情報が**初期設定手続と演算手続の引数**として集約



積雲パラメタリゼーションの実装例

- 複数の積雲パラメタリゼーションの実装
  - 異なるスキーム毎に異なるモジュール
    - 対流調節スキーム: phy_cumulus_adjust
    - Kuo スキーム: phy_cumulus_kuo
  - それぞれのスキームの使い方は同じになるよう実装
    - 総称手続きを用い、初期設定手続と演算手続の名称はそれぞれ Create および Cumulus
      - Create には重力加速度、気体定数、定圧比熱、...
      - Cumulus には気圧、温度、比熱、降水量、...
- スキーム交換に必要な情報が集約
  - 初期設定: 重力加速度、気体定数、定圧比熱、...
  - 演算: 気圧、温度、比熱、降水量、...

プログラム構造の設計 まとめ

- 各モジュールで初期設定手続を用意し、その手続でパラメータを設定
- モデル群の間でプログラムのパーツを交換する際、知る必要があるのは、初期設定手続と演算手続の引数のみ

- 課題
  - 実装しているのは開発中の 3 次元モデル DCPAM バージョン4
  - 階層モデル群で使いまわせるシステムかどうかはこれから検討

現在継続中の試み

- 可搬性や自己記述性に優れたデータ構造
  - gtool4 netCDF 規約の策定
- データ I/Oなどに関する煩雑なコードの隠蔽
  - データ入出力ライブラリ gt4f90io の開発・整備
- ソースコードを見ただけで何を計算しているか簡単に分かるための工夫
  - ソースコードを元の支配方程式の数学的表現に似せる
    - 配列を返すユーザ定義関数の活用
    - 関数と変数の命名規則の策定
- モデルの細部を理解せずに各パーツの着脱
  - モジュール機能を活かしたプログラム構造の設計
- モデルの解説文書の作成を容易に
  - オブジェクト指向スクリプト言語 Ruby のドキュメント自動生成ライブラリ RDoc によるモデル解説文書の自動生成

解説文書の重要性と問題点

- ここで言う解説文書とは
  - サブルーチンなどの使い方を記した文書
    - 内容：名称、必要な引数、引数の型、概要、etc...
- 解説文書の利便性
  - ソースコードを読まずにプログラムの交換が可能に
  - 他人とプログラムの共有を行う際にも必須
- 解説文書の維持・更新のコスト高
  - モデルを構成するプログラムは頻繁に交換・変更されることを想定
  - セットとなる解説文書も同時に手動で維持・更新するのは面倒
    - ソースコードの書き換えだけで十分大変
    - ソースコードの編集と似て非なる作業を再度行うのは苦痛

RDoc とは

- オブジェクト指向スクリプト言語 Ruby で書かれた、ソースコードからドキュメントを自動生成するライブラリ (Ruby の標準ライブラリの1つ)
- Fortran 90/95 の解析も可能
  - ソースコード解析機構とマニュアル生成機構が分離しているため、他の言語で書かれたソースコードも解析可能
  - 標準で C および Fortran95 用の解析機構が付属
  - しかし、以前は解析機能が不十分で実用に耐えなかった

RDoc Fortran 90/95 強化版

- Fortran 90/95 の解析機能を強化
  - サブルーチンや関数の引数、総称名称なども解析
  - MathML の利用により数式も表現可能

```
module phy_cumulus_adjust
  != 積雲パラメタリゼーション:
  !  対流調節スキーム
  !== Prodedures list
  ! Create        :: 初期設定
  ! Calculation   :: 演算
                :
contains
                :
 subroutine PhyCumulusAdjustCreate( &
    & phy_cum_ad, &
    & Grav, RAir, Cp, ... )
                :
 end subroutine PhyCumulusAdjustCreate
                :
end phy_cumulus_adjust
```

RDoc によるドキュメント自動生成 まとめ

- 森川 他, 2007, 天気 Vol 54 No 2.
- http://www.gfd-dennou.org/library/dcmodel
  - RDoc Fortran 90/95 解析機能強化版
- 解説文書が低コストで定常的に更新
- ソースコードを読むことなくプログラムの交換を可能に

まとめ：階層的モデル群の開発・整備のための基盤技術開発

- 可搬性や自己記述性に優れたデータ構造 gtool4 netCDF 規約の策定
  - CF 規約へ対応中
- データ入出力ライブラリ gt4f90io による煩雑なソースコードの隠蔽
  - できた
- 数式と近い形でコードを記述できるようにする
  - コーディングルールを策定中
    - 関数利用によってどれくらい実行速度が落ちるか？？
    - 並列化対応の際にも読みやすさは維持できるか？？
- モジュール機能を活かしたプログラム構造の設計によるプログラムの可変性向上
  - 3 次元モデルへ導入、試行中
    - 他のモデルとのプログラム共有はうまくいくか？？
- RDoc によるモデル解説文書の自動生成
  - できた