ISSN 0286-7419

TOHOKU UNIVERSITY

東北大学
サイバーサイエンスセンター

大規模科学計算システム広報

SENAC

Vol.45 No.4 2012−10

Supercomputing System
Cyberscience Center
Tohoku University
www.ss.isc.tohoku.ac.jp

大規模科学計算システム関連案内

＜大規模科学計算システム関連業務は、サイバーサイエンスセンター本館内の情報部情報基盤課が担当しています。＞

http://www.ss.isc.tohoku.ac.jp/

階	係・室名	電話番号(内線)* e-mail	主なサービス内容	サービス時間 平 日
一階	共同利用支援係 （受　付）	022-795-3406(3406) FAX:022-795-6099 uketuke@isc.tohoku.ac.jp	各種申請、講習会、利用相談、広報、センター業務全般に関する質問や要望の受付	8:30～17:15
	利用相談室	022-795-6153(6153) sodan05@isc.tohoku.ac.jp 相談員不在時 022-795-3406(3406)	計算機利用全般に関する相談 大判プリンタ、利用者端末等の利用	8:30～17:15 8:30～21:00
	利用者談話室	(3444)	各センター広報の閲覧	8:30～21:00
	展 示 室（分散コンピュータ博物館）		歴代の大型計算機等の展示	9:00～17:00
三階	庶務係	022-795-3407(3407) syomu@isc.tohoku.ac.jp	庶務に関すること	8:30～17:15
	会計係	022-795-3405(3405) kaikei@isc.tohoku.ac.jp	会計に関すること、負担金の請求に関すること	8:30～17:15
	共同研究支援係	022-795-6252(6252) rs-sec@isc.tohoku.ac.jp	共同研究、計算機システムに関すること	8:30～17:15
	共同利用支援係	022-795-6251(6251) uketuke@isc.tohoku.ac.jp	ライブラリ、アプリケーションに関すること	8:30～17:15
	ネットワーク係	022-795-6253(6253) net-sec@isc.tohoku.ac.jp	ネットワークに関すること	8:30～17:15
四階	研究開発部	022-795-6095(6095)		
五階	端末機室	(3445)	PC 端末機(X 端末)	8:30～21:00

＊（　）内は東北大学内のみの内線電話番号です。青葉山・川内地区以外からは頭に 92 を加えます。

本誌の名前「SENAC」の由来

昭和 33 年に東北地区の最初の電子計算機として、東北大学電気通信研究所において完成されたパラメトロン式計算機の名前で SENAC-1(SENdai Automatic Computer-1)からとって命名された。

［共同研究成果］

光によって引き起こされる化学反応の量子力学に基づく理論解析

— 非断熱遷移を引き起こす円錐交差の取り扱いについて —

山崎 馨・菅野 学・河野 裕彦
東北大学大学院理学研究科化学専攻

分子に光が照射されて生成する電子が高いエネルギーを持った電子励起状態においては，一般に，複数の電子状態のポテンシャルエネルギー曲面(PES)が接近する．とくに，ポテンシャル面が交差してできる円錐交差の領域では，原子核がある電子状態から別の電子状態のポテンシャル面上を動くようになる非断熱遷移が起き，化学反応の機構や速度を大きく変化させる．そこで本稿では，電子励起状態を経由する化学反応の理論解析の鍵となる円錐交差の理論的取り扱いについて概説する．特に円錐交差の分子構造が正しく求められているかを，円錐交差における幾何位相効果に基づいて確認する方法について実際の計算例を交えて議論する．

1. 序論

分子に近赤外から可視・紫外領域の光を照射すると，分子内の電子が高いエネルギーを持った電子励起状態が生成する．電子励起状態においては，化学反応中の原子核の運動を決定づける断熱ポテンシャルエネルギー曲面(PES)の形状は，電子基底状態とは異なった物となる．このため，適切な波長・波形・強度を持った光を用いて特定の電子励起状態を選択的に生成することで，電子基底状態では起こりにくい化学反応を，別の反応機構によって効率良く進行させることが可能である[1,2]．この様に，分子への光照射が引き金になって起こる反応を光化学反応という．

光化学反応では，分子が電子励起状態において構造変化をする過程で，関連する電子状態間のポテンシャルエネルギー差が非常に狭い領域に到達することがある．この様な領域では，電子と原子核との相互作用（振電相互作用または非断熱相互作用）が他の領域に比べて強い．このため，電子エネルギーと原子核のエネルギーの間に交換が起こり，電子状態が光の吸収・発光をともなわないで変化する．この原子核の動きによって誘起される過程を非断熱遷移という（これに対して，原子核がゆっくり動く極限の過程は断熱過程と呼ばれ，原子核は同一の断熱電子状態上を動く）[2,3]．非断熱遷移は電子状態間のエネルギー間隔が狭くなるほど起こりやすくなり，図 1 に示す PES の交差点である円錐交差の分子構造は最も非断熱遷移が起こる確率が高い構造であるといえる．円錐交差における非断熱遷移は，多くの光化学反応においてその反応経路を分岐させ，電子基底状態とは異なった反応機構をとる原因となっている[4]．このため，円錐交差の構造を探索してそのエネルギーや電子状態間の非断熱結合の強さを求めることは，光化学反応の反応機構を理論的に解析する上での中心的な課題の一つとなっている[5]．そこで本稿では，筆者らの最近の研究[6]を例に，非断熱遷移を引き起こす円錐交差の理論的取り扱いおよびその役割について概説する．

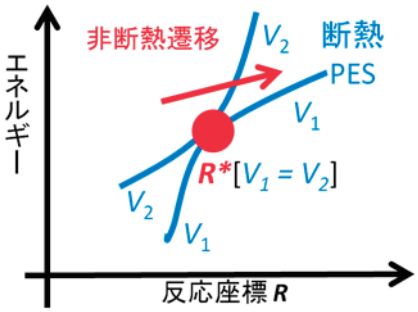


図 1：円錐交差と非断熱遷移の概念図．断熱ポテンシャルエネルギー曲線 V_1 と V_2 の値が等しくなる円錐交差 $\boldsymbol{R}^*$ において V_1 と V_2 の間の非断熱相互作用が非常に強くなり，分子は光を出さずに V_2 から V_1 へと非断熱遷移する．

2. 円錐交差の確認法：幾何学的位相効果の利用

円錐交差の構造探索は，MOLPRO[7]などの電子励起状態における量子化学計算を得意とする計算パッケージに実装されている構造最適化ルーチン[8,9]を用いて実行することが可能である．また近年，大野・前田らによって開発された化学反応経路自動探索プログラム GRRM[1][10]により，電子励起状態の安定な分子構造を出発点として円錐交差を系統的に自動探索することも可能になってきた[11]．これらのプログラムによって求められた構造が円錐交差の構造に正しく収束しているかどうかは，全電子の確率的空間分布を特徴付ける関数である断熱電子波動関数$\{\Psi_n\}$の円錐交差近傍における符号変化（幾何学的位相あるいはベリー位相効果）に着目することで確認することができる[8,9]. 本節では幾何学的位相効果の確認による円錐交差の同定法について概説する．

2.1 円錐交差における非断熱結合ベクトルと幾何学的位相効果

まず，円錐交差が核配置 $\boldsymbol{R}^*$に存在し，2 つの断熱電子波動関数$\{\Psi_1(\boldsymbol{r},\boldsymbol{R}), \Psi_2(\boldsymbol{r},\boldsymbol{R})\}$で特徴付けられる 2 準位の M 原子分子系を考える（M は任意の整数）．ここで，$\boldsymbol{r}$ と $\boldsymbol{R}$ はそれぞれ電子と原子核の座標ベクトルである．この時，これらの断熱電子波動関数が，電子がどの分子軌道に収まっているかを示す電子配置を特徴付ける N 個の透熱電子波動関数の組$\{\psi_i(\boldsymbol{r},\boldsymbol{R}); i = 1,2,\ldots N\}$ を用いて[9]

$$\Psi_n(\boldsymbol{r},\boldsymbol{R}) = \sum_{i=1}^{N} C_{n,i}(\boldsymbol{R})\psi_i(\boldsymbol{r},\boldsymbol{R}) \tag{1}$$

と展開できるとする．ここで $C_{n,i}(\boldsymbol{R})$ は $\psi_i(\boldsymbol{r},\boldsymbol{R})$の展開係数であり，$|C_{n,i}(\boldsymbol{R})|^2$ が $\psi_i(\boldsymbol{r},\boldsymbol{R})$の重みを表す．なお，分子が円錐交差 $\boldsymbol{R}^*$で非断熱遷移を起こすとき，その確率は $\boldsymbol{R}^*$ 近傍のポテンシャルエネルギー曲面(PES) の形状と 2 つの状態の非断熱結合強度を特徴付ける 2 つの $3 \times M$ 次元($3M$ 次元) のベクトル（非断熱結合ベクトル）によって決まる．1 つ目は，円錐交差に関与する 2 つの PES の傾きの差を表す $\boldsymbol{g}$ ベクトルである．2 つ目は，円錐交差に関与する 2 状態の非断熱結合強度を表す $\boldsymbol{h}$ ベクトルである. 円錐交差近傍において 2 つの透熱電子波動関数$\{\psi_a(\boldsymbol{r},\boldsymbol{R}), \psi_b(\boldsymbol{r},\boldsymbol{R})\}$ が$\{\Psi_1(\boldsymbol{r},\boldsymbol{R}), \Psi_2(\boldsymbol{r},\boldsymbol{R})\}$の主要な成分になっているとき，$\boldsymbol{g}$ ベクトルと $\boldsymbol{h}$ ベクトルはそれぞれ次の様に書くことができる．

$$\boldsymbol{g}(\boldsymbol{R}) = \frac{1}{2}\left[\langle\psi_a|\nabla_R H(\boldsymbol{r},\boldsymbol{R})|\psi_a\rangle - \langle\psi_b|\nabla_R H(\boldsymbol{r},\boldsymbol{R})|\psi_b\rangle\right] \tag{2}$$

$$\boldsymbol{h}(\boldsymbol{R}) = \langle\psi_a|\nabla_R H(\boldsymbol{r},\boldsymbol{R})|\psi_b\rangle \tag{3}$$

ここで，$H(\boldsymbol{r},\boldsymbol{R})$と $\nabla_{\boldsymbol{R}}$ はそれぞれ，系の全エネルギーを表すハミルトニアン演算子と $3M$ 次元の全ての原子座標についての微分を行う微分演算子である（$\nabla_{\boldsymbol{R}}H(\boldsymbol{r},\boldsymbol{R})$ はグラジエント）．また，(2)式と(3)式では全電子座標についての積分を行っているため，$\boldsymbol{g}$ ベクトルと $\boldsymbol{h}$ ベクトルが核座標 $\boldsymbol{R}$ のみに依存する $3M$ 次元のベクトルになっていることに注意されたい．

さて，円錐交差 $\boldsymbol{R}^*$ 近傍で $\boldsymbol{R}^*$ を含む閉ループを考え，それに沿って得られる断熱電子波動関数の符号を調べると，閉ループを 1 周した際にその符号が反転することが知られている[5,8,9]．このように，量子力学においては，ハミルトニアンがゆっくり断熱的に変化して元に戻っても，対応する波動関数の位相が元に戻らない場合があり，この効果は幾何学的位相効果とよばれている．$\boldsymbol{R}^*$が円錐交差の場合もその一例である．$\boldsymbol{R}^*$ を含む閉ループを定義する際は，次の様に規格直交基

---

[1] GRRM に関しては，東北大学サイバーサイエンスセンターで利用可能である．詳細については，http://www.ss.isc.tohoku.ac.jp/application/grrm11.html　を参照のこと．

底 （$\hat{\boldsymbol{x}}$,$\hat{\boldsymbol{y}}$） を構造最適化の過程で得られる $\boldsymbol{R}^*$ における $\boldsymbol{g}$ ベクトルと $\boldsymbol{h}$ ベクトルから作ると便利である[8,9].

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{g}_\perp}{\|\boldsymbol{g}_\perp\|} \tag{4}$$

$$\hat{\boldsymbol{y}} = \frac{\boldsymbol{h}}{\|\boldsymbol{h}\|} \tag{5}$$

ここで $\boldsymbol{g}_\perp$ は $\boldsymbol{h}$ ベクトルに垂直になるように変換した $\boldsymbol{g}$ ベクトルであり, 以下の式で定義される.

$$\boldsymbol{g}_\perp = \boldsymbol{g} - (\hat{\boldsymbol{y}} \cdot \boldsymbol{g})\,\hat{\boldsymbol{y}} \tag{6}$$

$\hat{\boldsymbol{x}}$ と $\hat{\boldsymbol{y}}$ を用いることで，円錐交差 $\boldsymbol{R}^*$ を中心とする半径 ρ の円状ループ

$$\begin{cases} \boldsymbol{x}(\rho, \theta) = \hat{\boldsymbol{x}}\rho\cos\theta \\ \boldsymbol{y}(\rho, \theta) = \hat{\boldsymbol{y}}\rho\sin\theta \end{cases} \tag{7}$$

を定義することができる．このループに沿って断熱電子波動関数$\{\Psi_n\}$の符号を調べると，2π 回転した際にその符号が反転する，すなわち $\Psi_n(\rho, \theta) = -\Psi_n(\rho, \theta + 2\pi)$ なる関係が成立する．

2.2 実際に計算するときの注意点

幾何学的位相効果を利用した円錐交差の確認を行う際には，以下の 3 点に気をつける必要がある[2].

第 1 に, 量子化学計算パッケージで主に使用される単位系は原子単位系と呼ばれるものであり, 化学や物性物理の研究において日常的に使われる単位とは異なっていることに注意が必要である. 原子単位系においてはエネルギーの単位は Hartree, 長さの単位は Bohr で表され, 1 Hartree ~ 27.211 eV, 1 Bohr ~ 0.529 Å である．非断熱結合ベクトルは力の次元を持っているため，その各成分の値は Hartree/Bohr で出力される．このため，非断熱結合ベクトルから規格直交基底（$\hat{\boldsymbol{x}}$,$\hat{\boldsymbol{y}}$）を作る場合，そのノルムは 1 Hartree/Bohr となる．一方，構造最適化後の構造は量子化学計算パッケージ利用者の便宜を図って Å 単位で出力されることが多いので, ループ上の構造を求める際には適宜単位換算を行う必要がある[3].

第 2 に，閉ループ（$\boldsymbol{x}, \boldsymbol{y}$）に沿って展開係数$\{C_{n,i}(\boldsymbol{R})\}$の値を調べていく際には，その値が連続的に変化していくことを確認する必要がある．これは，断熱電子波動関数$\{\Psi_n\}$の符号が反転しても分子の総エネルギーは変化しないため，$\{C_{n,i}(\boldsymbol{R})\}$の値の連続性を考慮しないで多くの量子化学計算パッケージが$\{\Psi_n\}$の符号を決めていることに起因する． $\{C_{n,i}(\boldsymbol{R})\}$ の符号が角度 θ において急に反転した場合には，次の様な対応をとると良い．なお，符号が反転する前の角度を θ’ とする．

(1) Jmol 等の可視化ソフトウエアを使って電子の確率的な空間分布を特徴付ける関数である分子軌道を角度 θ’と θ において可視化し，その位相（符号）が反転していないか確認する．反転していた場合はその軌道が関与する電子配置の係数$\{C_{n,i}(\boldsymbol{R})\}$の符号を反転させて補正する.

---

[2] ここでは MOLPRO を使用して計算を行う場合について特に述べるが, 他の量子化学計算パッケージ(例えば GAMESS や MOLCAS)を使用する場合でも要点は同じである.

[3] 閉ループ（$\boldsymbol{x}, \boldsymbol{y}$）に沿って展開係数$\{C_{n,i}(\boldsymbol{R})\}$の値を調べる場合には, Bash や Perl などのスクリプト言語を使って自動的に閉ループ $(\boldsymbol{x}, \boldsymbol{y})$ 上の分子構造を作成して計算を実行するようにすると効率が良い. なお，MOLPRO においては，ユーザーが任意のベクトル型配列や do ループをインプットファイル内に作成して計算を自動化させることができるので，この機能を使うのも有効である.

(2) 分子軌道の位相が反転していない場合，$\{\Psi_n\}$全体の符号が反転している可能性が高い．そこで，$\{C_{n,i}(\boldsymbol{R})\}$ の符号が反転する前後の$\{\Psi_n\}$の重なり積分

$$\langle\Psi_n(\rho,\theta')|\Psi_n(\rho,\theta)\rangle\approx\sum_{i=1}^{N}C^{*}_{n,i}(\rho,\theta')C_{n,i}(\rho,\theta) \tag{8}$$

を計算する．ここで $C^{*}_{n,i}$ は $C_{n,i}$ の複素共役を表す．重なり積分がほぼ −1 になっていれば θ' から θ で$\{\Psi_n\}$の符号が反転していることが確認される．この場合，全電子配置の係数$\{C_{n,i}(\boldsymbol{R})\}$の符号を反転させて補正する．

以上の確認作業を円滑に行うためには，すべての電子配置$\{\psi_i(\boldsymbol{r},\boldsymbol{R})\}$に関してその展開係数$\{C_{n,i}(\boldsymbol{R})\}$を出力するように，量子化学計算パッケージの出力オプションを設定するとともに，分子軌道の出力形式を正準軌道[4]に指定するとよい．また，計算の際に 1 つ前の θ における断熱電子波動関数を初期値として読み込ませると，$\{\Psi_n\}$の不連続な変化を少なくするとともに，計算効率の向上を図ることができる[5]．

第 3 に，分子中の 1 つ 1 つの電子の運動がそれぞれ独立ではなく，1 つの電子が他の電子に近づくとお互いに避け合うように相関運動して安定化する「電子相関」の効果をいかに取り込むかである．電子励起状態においては，(1) 式に示すようにいくつもの電子配置が系全体の断熱電子波動関数に寄与している場合が多い．これらの電子配置の中でそのエネルギー準位が近接しているもの同士の相互作用による安定化効果を「静的電子相関」効果[12]，それ以外の安定化効果を「動的電子相関」効果と便宜上呼び分けることにする．

円錐交差においては 2 つの断熱電子状態間のエネルギー差がゼロであるため，これらの断熱電子状態を記述する波動関数$\{\Psi_1(\boldsymbol{r},\boldsymbol{R}),\Psi_2(\boldsymbol{r},\boldsymbol{R})\}$に寄与する電子配置$\{\psi_i(\boldsymbol{r},\boldsymbol{R})\}$のエネルギー準位も非常に近接したものになる．このため，静的電子相関が非常に大きくなり，その効率の良い取り込みが円錐交差の構造を精度良く計算するための出発点となる．このため，静的電子相関に主要な寄与を示すと考えられる幾つかの電子配置の重ね合わせ(多配置展開)で断熱電子波動関数$\{\Psi_n\}$を近似する多配置 Self consistent field (SCF)法[13,14]が頻繁に用いられる．多配置 SCF 法は静的電子相関を重点的に取り込むため，動的電子相関の考慮は不十分になる傾向がある[12,14]．この動的電子相関の欠如を改善し円錐交差のエネルギーを定量的に算出するために，多配置 SCF 法によって得られた円錐交差の構造における断熱電子波動関数とエネルギーを 0 次近似とするエネルギー補正がしばしば行われる[6,15]．代表的な補正法として，動的電子相関の根源である多配置展開に漏れた電子配置の寄与を摂動展開してその 2 次の項までを考慮する 2 次の多参照摂動法[16-18]などがある．一般に，多配置 SCF 法で求めた円錐交差の構造において多配置摂動法などを用いたエネルギー補正を行うと，円錐交差を構成する 2 つの断熱電子状態のエネルギー差がゼロではなくなる．これは動的電子相関の効果により円錐交差の分子構造が変化する[19]ことに起因する．このため，多配置 SCF 法によって得られた円錐交差のエネルギー補正を行う場合には，次の 2 点について最低限確認する必要がある．

(1) 補正後の 2 状態のエネルギーの平均が多配置 SCF 法によって得られた円錐交差のエネルギーと十分に近いこと．

(2) 2 つの断熱電子状態のエネルギー差が十分に小さいこと．

---

[4] 正準軌道は幾つかある分子軌道の形式のひとつである．詳細については文献[20]を参照のこと．

[5] MOLPRO や Gaussian などの量子化学計算パッケージは，計算結果をバイナリファイルとして保存しておく機能を備えている場合が多い．それらから波動関数を読み込ませればよい．

3. 計算例：ナノカーボンの光誘起構造欠陥生成に関与する円錐交差

本節では，グラフェンやカーボンナノチューブなどのナノカーボンにおける主要な構造欠陥生成機構の一つである Stone-Wales 転位(SW 転位)の電子励起状態経由の反応経路に関与する円錐交差[6]を例に，円錐交差の構造最適化と幾何学的位相効果の確認について述べる．

SW 転位とは図 2 に示すようにナノカーボンが持つ 4 つの六員環の中心の C1＝C2 二重結合が段階的に 90 度回転して五員環と七員環を 2 つずつ形成する反応である．この反応を制御することはナノカーボンのバンドギャップ調節につながるので，筆者らは，図 2 に示すように，C-C 結合の開裂と形成が段階的に起きる多段階反応経路について調べた． SW 転位を再現できる最小の分子であるピレン(図 2 左端)をモデルとし，とくに，電子基底(S_0) 状態及び第 1・第 2 一重項電子励起（S_1・S_2）状態における反応経路を探索した．

本研究では，幾つかの電子状態のエネルギーの加重平均を最小化する多状態の多配置 SCF 法である State-averaged complete active space SCF 法(SA-CASSCF 法)[21,22]を用いて反応物（ピレン，図 2 左端）と生成物（アズピレン，図 2 右端），及び図 3 に示す中間体や 2 つの円錐交差の分子構造を求めた．また，SA-CASSCF 法の波動関数を 0 次近似とする 2 次の多参照摂動法である multistate complete active space self-consistent field second-order perturbation theory (MS-CASPT2)法[23]を用いて動的電子相関の効果を取り込み，求めた構造における分子のエネルギーを補正した．SA-CASSCF 法において考慮する電子配置としては，最安定電子配置とこの配置において電子が収容されている分子軌道 (被占有軌道) のうちエネルギーが高い順に 3 個と，電子が収容されていない分子軌道(空軌道)のうちエネルギーが低い者から順に 3 個の計 6 軌道（これが活性空間 active space である）の中に含まれる 6 つの電子が取り得る全ての励起配置を考慮した（[6,6]-SA-CASSCF 法）．なお，以上の計算では S_0 から S_2 までの 3 状態を考慮し，ガウス型関数の線形結合（ガウス型基底関数 6-31G(2d,p)）を用いて分子軌道の形を表現した．以上の計算には量子化学計算パッケージ MOLPRO2010.1[7]を使用した．

図 4 にピレンにおける多段階 SW 転位(図 2)の反応経路に沿った PES と見いだした円錐交差を示す．この反応機構において，円錐交差が重要な役割を果たしていることがわかる。つまり，図

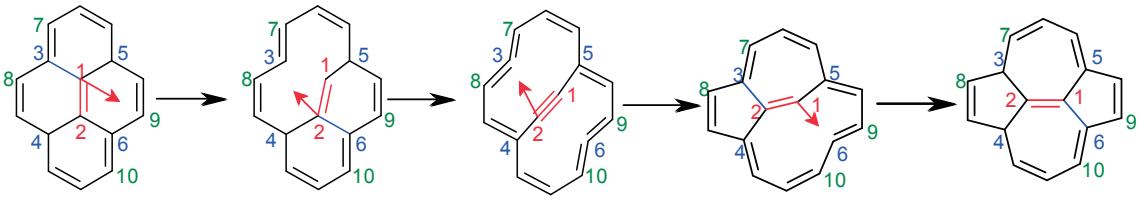
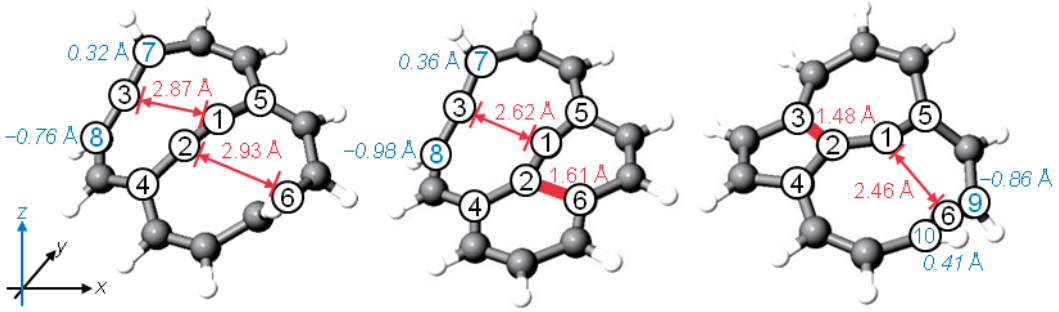


図 2: グラフェンやカーボンナノチューブにおける Stone-Wales 転位の反応機構の多段階反応経路．赤い矢印が原子の動きを表す．なお，反応サイトだけを抜き出していることに注意．



(a) 中間体　　(b) 円錐交差 1　　(c) 円錐交差 2

図 3: ピレンにおける非断熱 Stone-Wales 転位に関与する(a)S_0 状態の中間体および (b), (c) S_1/S_0 状態間の 2 つの円錐交差の構造．$z = 0$ に位置する分子平面からのずれを斜体字で示す．

3b に示す円錐交差 1 はピレンと図 3a に示す中間体の間に位置し，図 3c に示す円錐交差 2 は中間体とアズピレンの間に位置することがわかる．光励起により S_1 状態から反応が開始した場合，円錐交差 1 で S_0 状態に非断熱遷移すると S_0 状態から反応が始まった場合と同じ反応障壁(9.54 eV)で反応が進行する．一方，円錐交差 1 で非断熱遷移が起きず，S_1 状態上で反応が進行した場合，分子は 14.56 eV もの反応障壁を越えなくてはならない．このため，この反応経路はエネルギー的に圧倒的に不利である．また図 5 で示された円錐交差 1 の $\boldsymbol{R}_1^*$ における非断熱結合ベクトル$\{\boldsymbol{g}(\boldsymbol{R}_1^*), \boldsymbol{h}(\boldsymbol{R}_1^*)\}$を見ると，どちらの非断熱結合ベクトルも SW 転位の反応座標である C1=C2 結合の回転方向に大きな成分を持つことがわかる．これは SW 転位が進行すると，円錐交差 1 における非断熱遷移も起こりやすいことを示している．以上より，S_1 状態から始まる SW 転位では，円錐交差 1 において S_1 状態から S_0 状態へ非断熱遷移する反応経路の方が S_1 状態に分子が留まったまま反応する経路より優勢だと考えられる．

この節の最後に，得られた交差点が真の円錐交差点か，つまり，幾何学的位相効果の調べ方について説明する．まず，円錐交差 1 の $\boldsymbol{R}_1^*$ におけるこれら 2 つの非断熱結合ベクトル$\{\boldsymbol{g}(\boldsymbol{R}_1^*), \boldsymbol{h}(\boldsymbol{R}_1^*)\}$から(4)-(6)式を用いて規格直交基底$(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$を作り，円形ループ $(\rho = 0.05$ Bohr) を $\boldsymbol{R}_1^*$を中心として(7)式にしたがって構築した．円錐交差 1 と 2 において大きく寄与する電子配置(図 6)の SA-CASSCF 法における係数$\{C_{n,i}\}$を図 7 に示す．ここでケット$|\ldots\rangle$内の数字はすべての電子配置を生み出す活性空間内の 6 つの分子軌道の電子占有数を表している．例えば図 6 に示すように $|222000\rangle$ は電子が最高占有軌道 (HOMO)占有軌道以下におさまった電子基底配置を表し，$|221100\rangle$

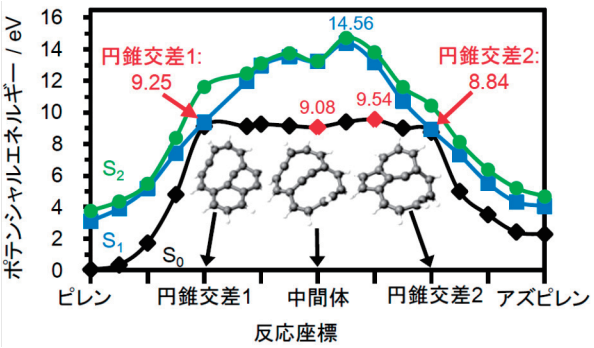


図 4：ピレンの Stone-Wales 転位における多段階機構のポテンシャルエネルギー曲線．図中に示したエネルギー値は MS-CASPT2 法で求めたものである．なお，円錐交差におけるエネルギーは S_1 状態と S_0 状態のエネルギーの平均を示していることに注意．S_1 状態から反応が開始した場合，円錐交差 1 で S_0 状態に非断熱遷移すると S_0 状態から反応が始まった場合と同じ反応障壁(9.54 eV)で反応が進行する．

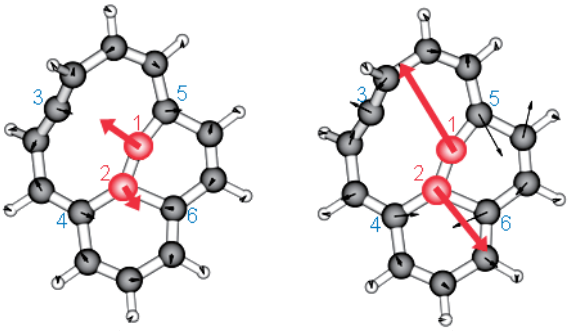


図 5: 円錐交差の $\boldsymbol{R}_1^*$ における非断熱結合ベクトル：(a) $\boldsymbol{h}$ ベクトル，(b) $\boldsymbol{g}$ ベクトル．どちらの非断熱結合ベクトルも Stone-Wales 転位の反応座標である C1=C2 結合の回転方向に大きな成分を持つことがわかる．

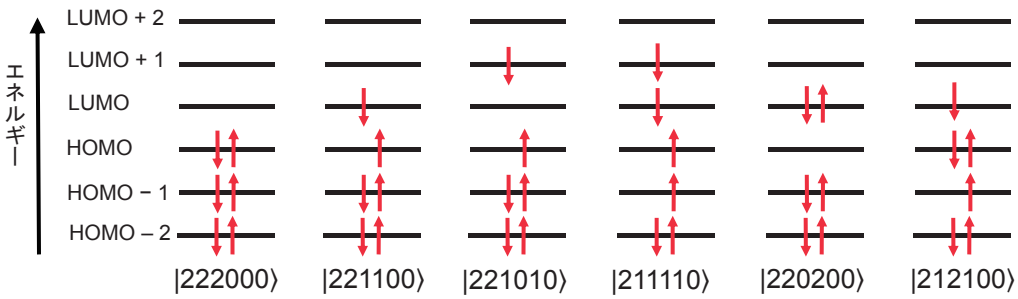


図 6: 円錐交差 1･2 において大きな寄与を示す電子配置．赤い矢印が電子 1 個を表す．

はHOMO から最低空軌道 (LUMO) への1電子励起配置を表す. 図7aの左パネルに示すように, 円錐交差 1 を中心とする円形ループを一周することで, S_0 状態における|221100⟩ の係数が−0.66 から 0.66 に変化していることが分かる. 同じように, 図 7a の右パネルに示す|222000⟩ の係数も −0.94 から 0.94 へ変化している. 同様の手続きで, 円錐交差 2(図 7b)においても, S_0 状態と S_1 状態の 4 つの主配置係数の符号反転を確認した. このことから, 構造最適化された円錐交差 1・2 は S_0 状態と S_1 状態の PES が交差して形成される円錐交差であることが確かめられた.

また, SA-CASSCF 法で求めた円錐交差 1 と 2 のエネルギーはそれぞれ 9.51 eV と 8.75 eV であった. 一方, MS-CASPT2 法で動的電子相関の効果を取り込んで補正した円錐交差 1 における S_0 状態と S_1 状態のエネルギーは 9.12 eV と 9.38 eV であり, その平均エネルギーは 9.25 eV (図 4)であった. 同様に, 円錐交差 2 における MS-CASPT2 法で求めた S_0 状態と S_1 状態のエネルギーは 8.75 eV と 8.92 eV であり, その平均エネルギーは 8.84 eV (図 4)であった. これら 2 つの円錐交差における S_1 状態と S_0 状態の MS-CASPT2 法によるエネルギーの平均は SA-CASSCF 法によるエネルギーと非常に近い. また MS-CASPT2 法で求められた S_1 状態と S_0 状態のエネルギー差は円錐交差 1 で 0.26 eV, 円錐交差 2 で 0.18 eV と十分に小さい. このことから, 今回 SA-CASCCF 法で求められた円錐交差 1 と 2 の構造は MS-CASPT2 法で求めた場合の構造と定性的には大きな差が無いと考えられる.

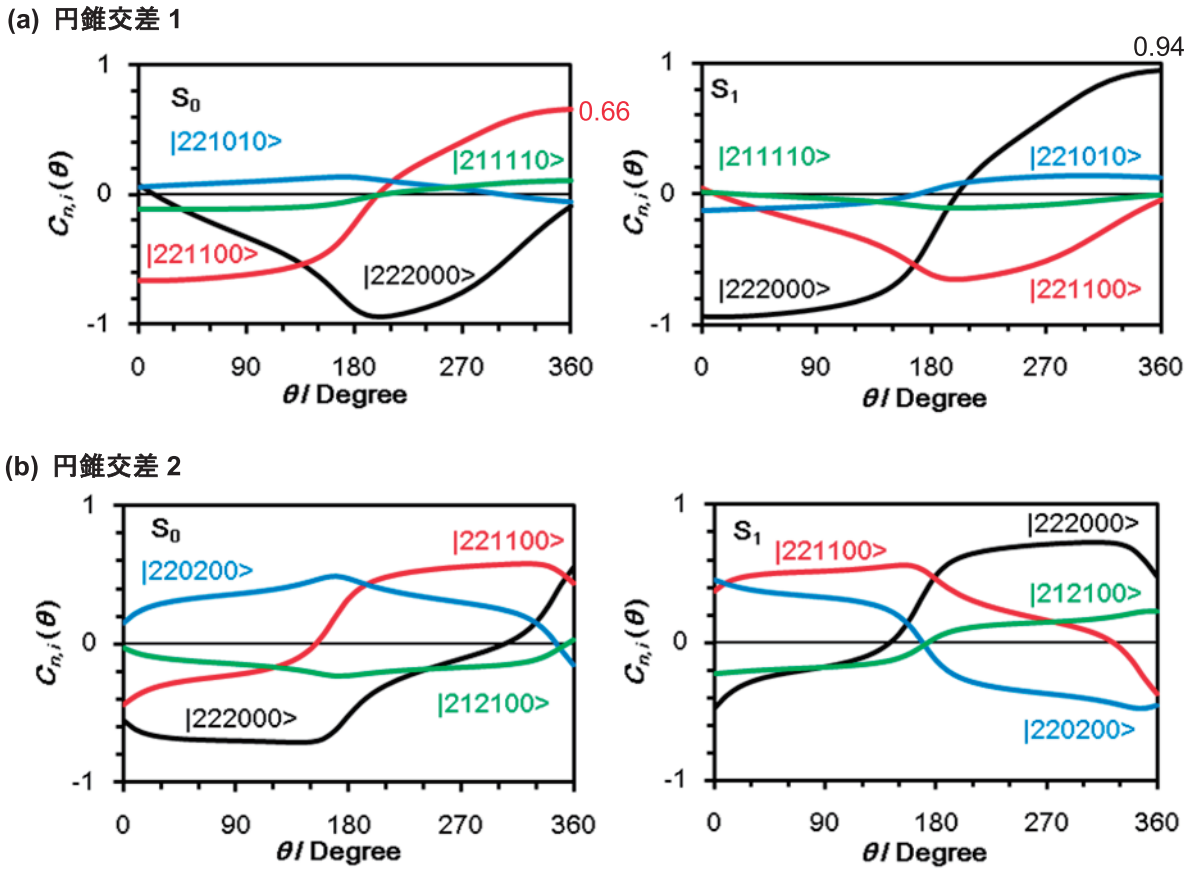


図 7 : S_1/S_0 断熱状態間の円錐交差 1・2 における幾何学的位相効果. SA-CASSCF 法の各円錐交差における 4 つの主配置$\{\psi_i;\ i = 1,2,3,4\}$の係数$\{C_{n,i}(\boldsymbol{R});\ i = 1,2,3,4\}$を閉ループの角度 θ の関数として示す. ここで, 円錐交差 1 では$\{\psi_1, \psi_2, \psi_3, \psi_4\} = \{|222000\rangle, |221100\rangle, |221010\rangle, |211110\rangle\}$, 円錐交差 2 では$\{\psi_1, \psi_2, \psi_3, \psi_4\} = \{|222000\rangle, |221100\rangle, |220200\rangle, |212100\rangle\}$である. なお, $\rho = 0.05$ Bohr を計算に用いた.

4. 結論

本稿では, 光化学反応の理論解析の鍵となる円錐交差の役割と理論的取り扱いについて概説した. 円錐交差における非断熱遷移により，電子基底状態とは異なった機構で化学反応を進行させることができる．円錐交差の構造が正しく求められているかは，円錐交差における幾何学的位相効果を確認することとで確認できる. また, 動的電子相関によって円錐交差の構造は変化しうるので, 慎重な取り扱いが必要である.

謝辞

本稿に掲載した結果の一部はサイバーサイエンスセンターの並列コンピュータを用いて得られた．また, MOLPRO の並列化とその運用に当たっては, 山下毅氏（共同利用支援係）から多大な技術的支援をいただきました. これらの支援・協力にこの場を借りて感謝いたします.

参考文献

[1] J. E. McMurry “Organic Chemistry” 8th ed., Brooks Cole (2011).
[2] 中村 宏樹「化学反応動力学」朝倉書店 (2004).
[3] H. Nakamura, “Nonadiabatic Transition: Concepts, Basic Theories and Applications”, 2nd ed.;World Scientific (2012).
[4] B. G. Levine and T. J. Martinez “Isomerization through Conical Intersections” *Annu. Rev. Phys. Chem.* **58**, 613–634 (2007).
[5] W. Domcke, D. R. Yarkony, and H. Köppel Eds. “Conical Intersections: Theory, Computation and Experiment” *Advanced Series in Physical Chemistry* Vol. 17, World Scientific (2011).
[6] K. Yamazaki, N. Niitsu, K. Nakamura, M. Kanno, and H. Kono “Electronic Excited State Paths of Stone-Wales Rearrangement in Pyrene: Roles of Conical Intersections” *J. Phys. Chem. A*, in press, doi: 10.1021/jp306894x (2012).
[7] H.-J. Werner *et al.*, MOLPRO, version 2010.1, a package of *ab initio* programs, see http://www.molpro.net
[8] D. R. Yarkony “Conical Intersections: Diabolical and Often Misunderstood” *Acc. Chem. Res.* **31**, 511–518 (1998).
[9] D. R Yarkony, “Nonadiabatic Quantum Chemistry—Past, Present, and Future” *Chem. Rev.* **112**, 481-498 (2012).
[10] 最近の日本語総説 : 前田 理, 大野 公一 「化学反応経路の自動探索」*Mol. Sci.* 5, A0042 (2011).
[11] 最近の総説 : S. Maeda, K. Ohno, and K. Morokuma “Exprloring Multiple Potential energy Surfaces: Photochemistry of Smal Carbonyl Compounds” *Adv. Phys. Chem.* in press, doi:10.1155/2012/268124 (2012).
[12] 常田 貴夫 「密度汎関数法の基礎」講談社 (2012).
[13] 多配置 SCF 法の入門的な解説に関しては，山崎 馨，河野 裕彦 「量子化学計算パッケージ MOLPRO の電子励起状態計算への応用と並列化」*SENAC* **44**, *2*, 33-39 (2011), 及びその参考文献を参照.
[14] B. O. Roos, “Multiconfigurational quantum chemistry” in *Theory and Applications of Computational Chemistry: The First Forty Years,* 725-764*,* Elsevier (2005).
[15] W. M. I. Hassan, W. C. Chung, N. Shimakura, S. Koseki, H. Kono, and Y. Fujimura “Ultrafast radiationless transition pathways through conical intersections in photo-excited 9H-adenine” *Phys. Chem. Chem. Phys.* **12**, 5317-5328 (2010).
[16] K. Andersson, P.-Å. Malmqvist, B. O. Roos, A. J. Sadlej, and K. Wolinski. “Second-order perturbation theory with a CASSCF reference function” *J. Phys. Chem.* **94**, 5483(1990).
[17] K. Andersson, P.-Å. Malmqvist, and B. O. Roos. “Second-order perturbation theory with a complete

active space self-consistent field reference function" *J. Chem. Phys.* **96**, 1218 (1992).
[18] K. Hirao "Multireference Møller-Plesset method" *Chem. Phys. Lett.* **190**, 374 (1992).
[19] S.Yamazaki and T. Taketsugu "Nonradiative deactivation mechanisms of uracil, thymine, and 5-fluorouracil: a comparative ab initio study" *J. Phys. Chem. A* **116**, 491–503 (2012).
[20] 藤永 茂 「分子軌道法」岩波書店 (1980).
[21] H.-J. Werner and P. J. Knowles "A second order multiconfiguration SCF procedure with optimum convergence" *J. Chem. Phys.* **82**, 5053 (1985).
[22] P. J. Knowles and H.-J. Werner "An efficient second-order MC SCF method for long configuration expansions" *Chem. Phys. Lett.* **115**, 259-267 (1985).
[23] H.-J. Werner "Third-order multireference perturbation theory The CASPT3 method" *Mol. Phys.* **89**, 645-661 (1996).

［共同研究成果］

海底圧力観測データから海底鉛直変位を検出するための海洋モデルの開発

稲津大祐*・日野亮太・藤本博己
東北大学大学院理学研究科　地震・噴火予知研究観測センター
*現在　独立行政法人　防災科学技術研究所

東北大学サイバーサイエンスセンターのスーパーコンピュータシステム SX-9 を利用し、現場海底圧力観測データから cm スケールの海底鉛直変位を検出するために有効な全球海洋シミュレーションモデルを開発してきた。開発した海洋モデルは、2011 年東北地方太平洋沖地震の発生直前の約２日間にゆっくりと進行した数 cm の海底鉛直変位を、現場海底圧力観測データから明瞭に検出することに奏功した。

１．はじめに

印加される圧力によって周波数が変化する水晶振動子を組み込んだセンサーを搭載した自律型ゲージ[1]を用いることで、海底において圧力を連続的に観測できる。この海底圧力の観測方法は、海洋物理学および測地学といった地球物理学的観測に用いられてきた[2, 3]。得られた海底圧力観測データ（$p_B(t)$）には、諸々の地球物理学・非地球物理学的シグナルが含まれる。

$$p_B(t) = p_{ref} + \Delta p_B(t),$$

$$\Delta p_B(t) = \Delta p_C(t) + \Delta p_T(t) + \Delta p_O(t) + \Delta p_A(t) + \Delta p_D(t) + \varepsilon(t).$$

ここで $\Delta p_B(t)$は水深に相当する圧力 p_{ref}からの圧力偏差である。添え字の C、T、O、A、D はそれぞれ、海底鉛直変位、潮汐、非潮汐海洋変動、大気圧、測器ドリフトを表す。$\varepsilon(t)$はそれらからの残差である。我々はこれらのうち地殻変動に関係する海底鉛直変位のシグナルを抽出したい。特に、超巨大地震に関係し得るマグニチュード７クラスの地震を反映する cm スケールの海底鉛直変位を正確に抽出したい[4]。地殻変動は、地震時にほぼ瞬間的に変位するものと、非地震性のゆっくりとしたものに分けられる。特に、ゆっくりとした地殻変動を抽出するためには、変動の時間スケールが重なる海洋変動を正確に見積もり、除去する必要がある。

我々はこの目的のために、東北大学サイバーサイエンスセンターのスーパーコンピュータシステム SX-9 を利用し、海洋起源の海底圧力変動を現実的に表現できる海洋シミュレーションモデルを開発してきた[5]。本稿では、開発した海洋モデルについて簡単に記述し、そのモデルを現場海底圧力データの解析に適用することで、2011 年東北地方太平洋沖地震に関係するゆっくりとした数 cm の海底鉛直変位を明瞭に抽出できたことを報告する。海底ケーブル等を利用した海底圧力リアルタイムモニタリングシステムへの適用についても言及しておく。本稿の技術的詳細については学会誌で公表済である[6]。

２．海洋モデル開発と SX-9 利用

東北大学地震・噴火予知研究観測センターでは、これまで想定されてきた宮城沖地震[7]に関係する海底の鉛直変位を直接観測するために、宮城沖で現場海底圧力観測を行ってきた[4]。宮城沖に代表される日本海溝だけでなく、海溝型巨大地震が発生するその他の世界のプレート沈み込み帯においても同様な観測を行うことは重要であるため、それらへの適用も意識し、モデル開発は全球海洋を対象とした。この海洋モデルは実海洋を一層海洋で近似するものである[8, 9]。モデル海洋は気象再解析データ（気圧・海上風）によって駆動される。モデルの精度は、我々の宮城沖の観測データを含む、全球の深海（水深 1000m 以深）の現場海底圧力データによって評価した[6]。

ベースとなるシミュレーション（以下、Control Run と呼ぶ）について説明する。計算の空間解像度は東西・南北方向ともに 1/12°であり、全格子数は 4320×2160 である。東北大学サイバーサイエンスセンターの SX-9 システムの利用による、ノード内の 16CPU を用いる自動並列処理（ジョブクラス p16 : 1.6TFLOPS）による計算で、海洋変動の１年分のランが約３日で完了できる。これは Intel 社の Core2Duo プロセッサ搭載のパソコン（15GFLOPS）での計算と比べ、実性能で約 200 倍であった。MPI チューニングを施すことで、自動並列処理 p16 の場合と比べ、さらに約３倍の計算速度が達成可能となった。ベクトル化率は 99.68%であった。

SX-9 を利用することで、空間解像度が 1/30°の計算まで現実的に行うことができた。その結果、解像度を向上させればさせるほどモデル精度が向上するわけではなく、解像度が 1/12°のときに最もモデル精度が高くなることがわかった。この問題について、風によってモデル一層海洋に入力されるエネルギーが、高解像度で表現される急峻な海底地形によって非現実的に強く消散してしまう、という一層海洋モデルによる近似の限界が要因の一つと考えられる[6]。

このように、本研究のモデルの枠組みにおいては、Control Run が最も現実的な海底圧力変動を推定・予測できることがわかった（図１）。Control Run は気象庁の気象再解析データ JRA-25[10] で駆動されている。現在、JRA-25 は現時点から約２日後にはデータ取得可能なシステムになっている。それに従い、現時点から約２日遅れで海洋モデルも更新可能となっている。

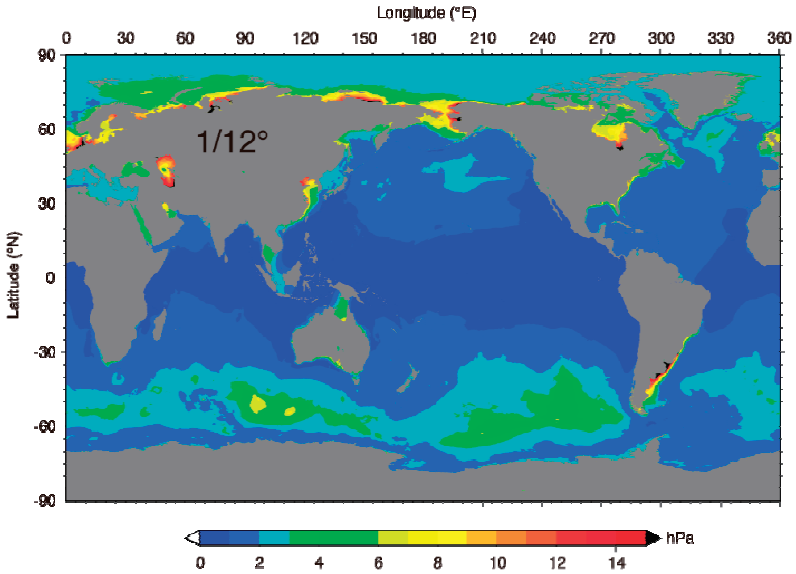


図１：Control Run による海底圧力変動の RMS 振幅。

３．現場海底圧力データへの適用　—2011 年東北地方太平洋沖地震—

Control Run は現場海底圧力データの非潮汐海洋変動成分を RMS 振幅で約 18%減少させることができる[6]。東北大学による宮城沖の観測データでは、非潮汐海洋変動成分の RMS 振幅を 2.1hPa から 1.5hPa（海底圧力の 1hPa の変動は、海底または海面の 1cm の変位に相当する）にまで減少させることができる[5]。

宮城沖の観測を継続している最中、2011 年 3 月 11 日に東日本大震災を引き起こした 2011 年東北地方太平洋沖地震（以下、東北沖地震）が、この観測網で囲まれる領域を破壊の開始点として発生した。本震は非常に巨大（マグニチュード９）であり、宮城沖を主な破壊域として数十 m の断層すべりが生じた[11]。このすべりは日本海溝に近いほど大きく、鉛直方向で数 m の海底変位となり、この鉛直変位が災害の主要因となる、沿岸域で 10m を優に超える津波を引き起こしたと考えられている[12]。東北大学の海底圧力観測網でこうした鉛直変位が直接観測された[13]。この m スケールの海底鉛直変位は、海洋変動や測器ドリフトと比べ桁違いに大きく、何の補正もなしにはっきりと認識できた（図２）。

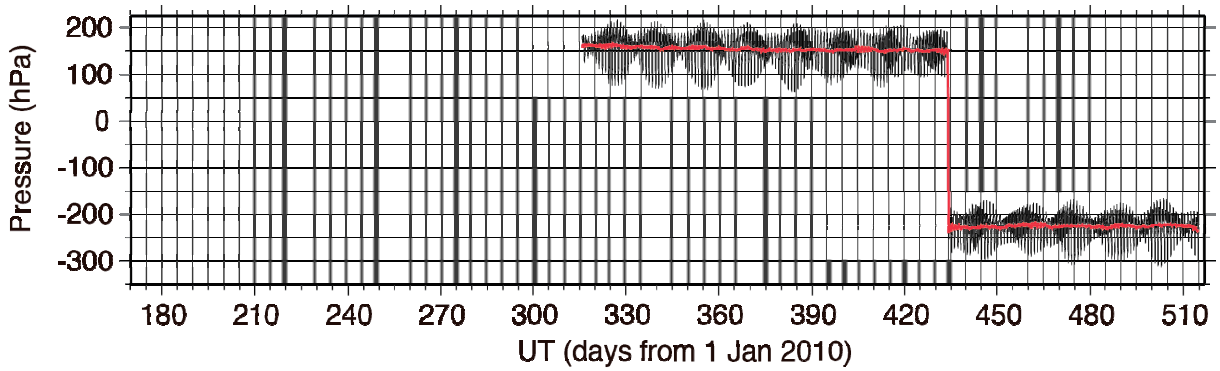


図２：宮城沖のある観測点（GJT3：図３参照）の海底圧力偏差。黒線が生時系列で、赤線が潮汐だけを補正した時系列である。434 日目（2011 年 3 月 11 日）に約 400 hPa の圧力減少（約 400 cm の海底隆起）が確認できる。

一方で、本震発生の２日前に、東北沖地震の前震と考えられているマグニチュード７の地震が発生し、その後活発になった地震活動が本震につながったと考えらえている[14, 15]。我々は、東北沖地震に関係する海底鉛直変位を精密に検出するために、宮城沖の現場海底圧力データから、潮汐成分に加え、本研究で開発した海洋モデルによって推定される海洋起源の海底圧力変動を差し引く補正を施した（図３）。その結果、海洋変動補正後の現場海底圧力データ（図３c）では、特に観測点 GJT3 において、本震までの２日間に約 5 cm のゆっくりとした海底の隆起（圧力減少）が明瞭となった。

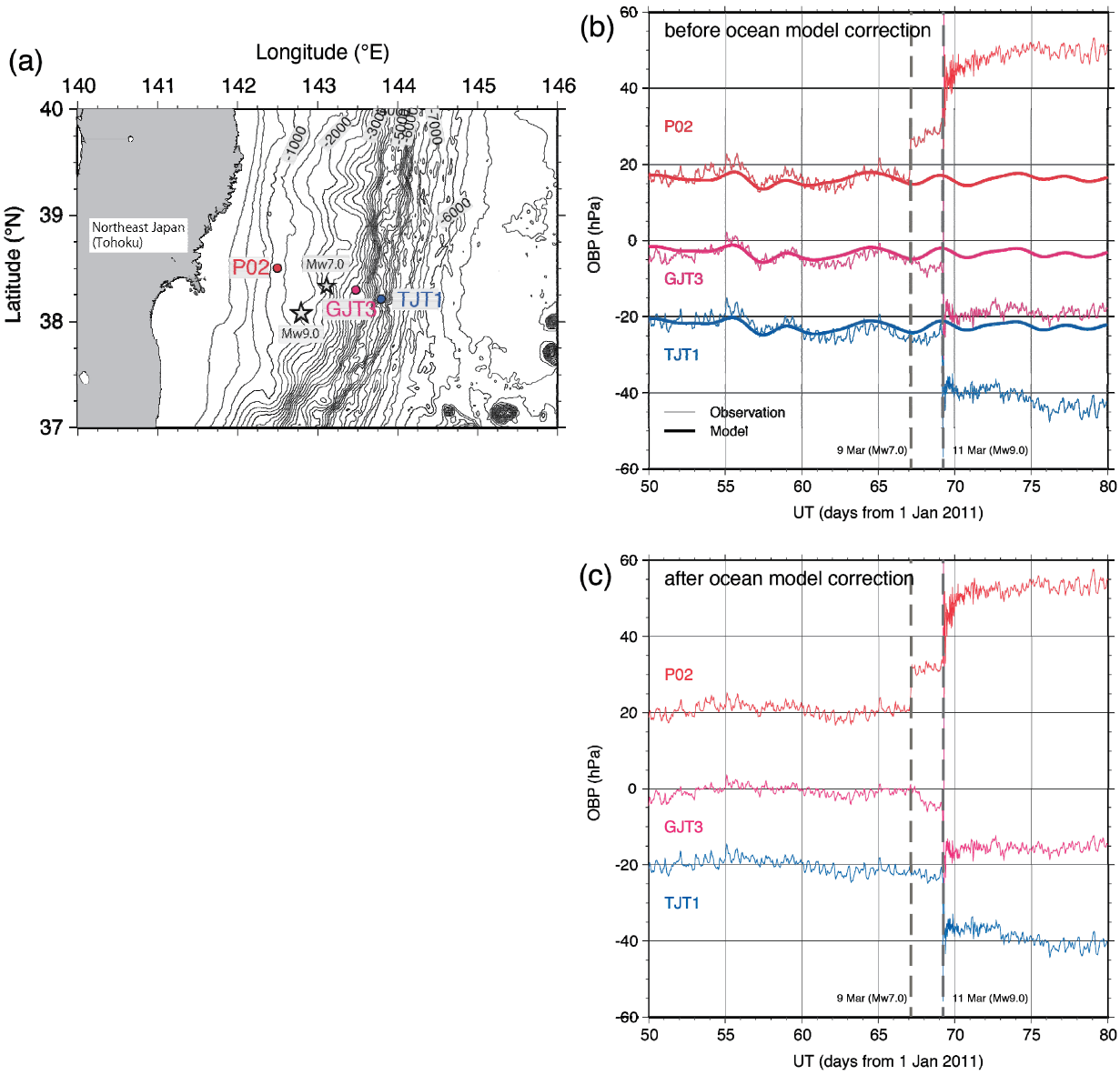


図３：(a) 震源位置を含む海底圧力観測点配置と (b, c) 本震時（69 日目）を含む 30 日間の海底圧力偏差時系列。(b) では、潮汐だけを補正した時系列に、モデルで推定される海洋変動成分を重ねたものを表示した。(c) では、潮汐に加え、推定された海洋変動成分を差し引いた時系列を表示した。(b, c) において、本震時の数百 hPa の変位は枠に入るよう縮めて表示した。

前震や本震といった地震時の瞬間的な変位は、ゆっくりとした海洋変動と変動の時間スケールが異なるため、その特定・検出は比較的容易であるが、前震から本震にかけて見られたゆっくりとした変位は、海洋変動成分の補正無しには明確に指摘することはできない（図３b, c）。このように、本共同研究で開発してきた海洋モデルの適用によって、海底のゆっくりとした鉛直変位が実際に検出できる一例を示すことができた。現在、東北沖地震の発生に至る過程[15]や発生後進展する余効変動[16]について、精力的に研究が行われてきているが、震源近傍で取得されたデータを用いることで、より詳細な調査が可能となるだろう。

４．まとめと展望

本稿では、東北大学サイバーサイエンスセンターとの共同研究で開発した海洋モデルとその利用例について報告した。この海洋モデルは、現場海底圧力観測データから cm スケールのゆっくりとした海底鉛直変位を検出するために開発された。結果として、2011 年東北地方太平洋沖地震の発生２日前から進行した数 cm のゆっくりとした海底鉛直変位を明瞭に検出できた。

巨大地震および津波発生ポテンシャルのある領域は、基本的に陸から遠い海溝に近い海底にある。こうした海底の変動や津波をリアルタイムでモニタリングするための海底観測網は、これまでも主に日本沖合で構築されてきた[17]が、海洋研究開発機構の DONET[18, 19]や NEPTUNE-Canada[20]などの最新の大規模な海底ケーブルシステムでも実用的に運用されるようになった。この他のリアルタイムモニタリングシステムとして、海上ブイを経由し海底圧力データを転送する NOAA の DART[21]、および、港湾空港研究所の NOWPHAS[22]も、より信頼度の高い早期津波警報の観点で重要な役割を果たしている。東日本大震災を受け、今後、防災科学技術研究所が主導となり、日本海構沿いでより広範囲な海底観測網が構築される予定である（図４）[23, 24]。こうした海底圧力観測網は、早期津波警報のための沖合津波観測を主目的とするが、本稿で示したように観測点直下の海底鉛直変位も計測している。我々が開発してきた海洋モデルは全球海洋をカバーしており、これらのオンラインデータへの利用が可能である。

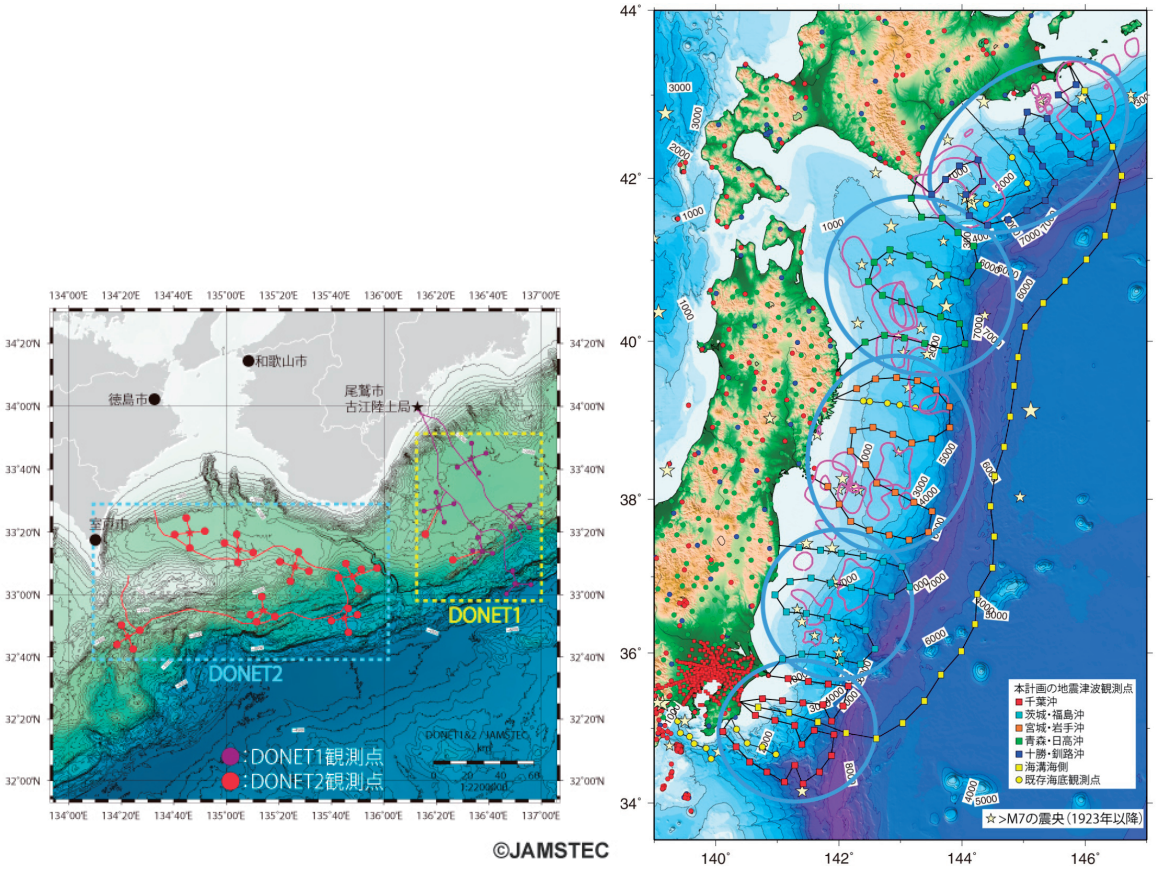


図４：海底ケーブルによる海底観測網の例。左図は海洋研究開発機構の DONET と DONET2[18]、右図は防災科学技術研究所で設置予定の海底ケーブルシステム[24]をそれぞれ示す。

前述したように、現在の気象再解析データの公開タイミングの制限で、オンラインモニタリングデータに対し、約２日遅れで海洋変動の補正が可能となっている。気象予報データを利用しモデルを駆動すれば、まさに現時点での海洋変動の予測が行われ、リアルタイムデータへの利用まで現実的になる。たとえば、気象庁の全球数値気象予報モデル GPV（GSM）は、最長 192 時間後の予報まで利用可能であり、十分実用的と期待できる[25]。今後、開発してきたモデルをオンラインモニタリングデータに即座に適用できるシステムに組み込む必要がある。将来の海溝型巨大地震に関係し地殻が変形するならば、それをできるだけ早い段階で検知できるようになることを期待する。

謝辞

本研究は文部科学省の委託研究である「東海・東南海・南海地震の連動性評価研究」の一環として行われた。また、東北大学サイバーサイエンスセンターとの共同研究「海海底鉛直地殻変動検出のための数 km スケールを解像する全球海底圧力モデリング」により多くの数値実験を行うことができた。一部の計算は海洋研究開発機構の地球シミュレータの SX-9 によっても行った。シミュレーションコードのチューニングにあたり、同センターのテクニカルアシスタントの沢田雅洋氏と山下毅氏から多大な協力を受けた。ここに謝意を示す。

参考文献

[1] Eble, M. C., F. I. González, D. M. Mattens and H. B. Milburn, Instrumentation, field operations, and data processing for PMEL deep ocean bottom pressure measurements. *NOAA Tech. Memo. ERL PMEL-89*, 71 pp., 1989.

[2] Fujimoto, H., M. Mochizuki, K. Mitsuzawa, T. Tamaki and T. Sato, Ocean bottom pressure variations in the southeastern Pacific following the 1997–98 El Niño event. *Geophys. Res. Lett.*, **30**, 1456, doi:10.1029/2002GL016677, 2003.

[3] Donohue, K. A., D. R.Watts, K. L. Tracey, A. D. Greene and M. Kennelly, Mapping circulation in the Kuroshio Extension with an array of current and pressure recording inverted echo sounders. *J. Atmos. Oceanic Technol.*, **27**, 507–527, doi:10.1175/2009JTECHO686.1, 2010.

[4] Hino, R, S. Ii, T. Iinuma and H. Fujimoto, Continuous long-term seafloor pressure observation for detecting slow-slip interpolate events in Miyagi-Oki on the landward Japan Trench slope. *J. Disast. Res.*, **4**, 72–82, 2009.

[5] 稲津大祐・日野亮太・藤本博己，大気擾乱によって駆動される短周期全球順圧海洋モデルの解像度依存性．*SENAC*，**44**，41–48，2011．

[6] Inazu, D., R. Hino and H. Fujimoto, A global barotropic ocean model driven by synoptic atmospheric disturbances for detecting seafloor vertical displacements from in situ ocean bottom pressure measurements. *Mar. Geophys. Res.*, **33**, 127–148, doi:10.1007/s11001-012-9151-7, 2012.

[7] 地震調査研究推進本部：http://www.jishin.go.jp/main/choukihyoka/ichiran_past/ichiran20110111.pdf

[8] Kim, C.-H. and J.-H. Yoon, Modeling of the wind-driven circulation in the Japan Sea using a reduced gravity model. *J. Oceanogr.*, **52**, 359–373, doi:10.1007/BF02235930, 1996.

[9] Hirose, N. and J.-H. Yoon, Barotropic response to the wind in the Japan Sea. *Proc. 4th CREAMS Works.*, 39–43, 1996.

[10] Onogi, K. and Coauthors, The JRA-25 reanalysis. *J. Meteor. Soc. Japan*, **85**, 369–432, doi:10.2151/jmsj.85.369, 2007.

[11] 鷺谷威，2011 年東北地方太平洋沖地震―3.11 から 1 年で何が分かったのか―．*日本地球惑星科学連合ニュースレター JGL*，8，2，2012．

[12] Fujii, Y., K. Satake, S. Sakai, M. Shinohara and T. Kanazawa, Tsunami source of the 2011 off the Pacific coast of Tohoku Earthquake. *Earth Planets Space*, **63**, 815–820, doi:10.5047/eps.2011.06.010, 2011.

[13] Ito, Y. and Coauthors, Frontal wedge deformation near the source region of the 2011 Tohoku-Oki earthquake. *Geophys. Res. Lett.*, **38**, L00G05, doi:10.1029/2011GL048355, 2011.

[14] Ando, R. and K. Imanishi, Possibility of Mw 9.0 mainshock triggered by diffusional propagation of after-slip from Mw 7.3 foreshock. *Earth Planets Space*, **63**, 767–771, doi:10.5047/eps.2011.05.016, 2011.

[15] Kato, A. and Coauthors, Propagation of slow slip leading up to the 2011 Mw 9.0 Tohoku-Oki earthquake. *Science*, **335**, 705–708, doi:10.1126/science.1215141, 2012.

[16] Ozawa, S. and Coauthors, Coseismic and postseismic slip of the 2011 magnitude-9 Tohoku-Oki earthquake. *Nature*, **475**, 373–377, doi:10.1038/nature10227, 2011.

[17] Hirata, K. and Coauthors, Integration of seafloor geodetic observation and offshore tsunami observation - toward researches on tsunami forecast. *Proc. 21st Ocean Eng. Symp.*, OES21-181, 2009.

[18] DONET: http://www.jamstec.go.jp/donet/e/

[19] Lubick, N., Earthquakes from the ocean: Danger zones. *Nature*, **476**, 391–392, doi:10.1038/476391a, 2011.

[20] NEPTUNE-Canada: http://www.neptunecanada.ca/

[21] DART: http://nctr.pmel.noaa.gov/Dart/

[22] NOWPHAS: http://www.mlit.go.jp/kowan/nowphas/

[23] Monastersky, R., Tsunami forecasting: The next wave. *Nature*, **483**, 144–146, doi:10.1038/483144a, 2012.

[24] 防災科学技術研究所，*防災科研ニュース*，**176**，pp. 24, 2012：http://www.bosai.go.jp/activity_general/pdf/k_news176.pdf

[25] 気象業務支援センター，全球数値予報モデル GPV（GSM）：http://www.jmbsc.or.jp/hp/online/f-online0a.html

［共同研究成果］

MPI による数値タービンの大規模並列計算手法の開発

笹尾泰洋 [1], 山本悟 [2], 三宅哲 [2], 岡崎健志 [2]
[1] 帝京大学理工学部航空宇宙工学科笹尾研究室
[2] 東北大学大学院情報科学研究科山本・古澤研究室

東北大学大学院情報科学研究科山本・古澤研究室にて開発された，蒸気タービン内部流動解析プログラムである「数値タービン」コードについて，NEC の協力を得て，移動境界問題に対応した MPI による大規模並列計算手法の開発を行った．13 流路-13 並列における並列化率は 99.2%，27 流路-27 並列における並列化率は 97.3%を達成した．並列計算の逐次計算に対する加速率は，13 並列において 11.3，27 並列において 16.0 となった．

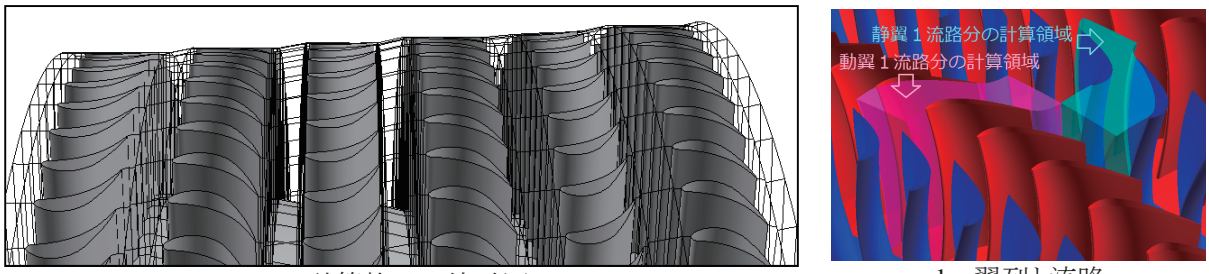
1. 緒言

蒸気タービンは，高温高圧の水蒸気から運動エネルギーを取り出すためのターボ機械の一種である．蒸気タービンの多くは，図 1 に示すような翼列と呼ばれる周方向と回転軸方向に配置された翼の集合によって構成されており，特に，周方向には周期性の強い流れ場が形成されることが知られている．そこで，ターボ機械に関する多くの解析的研究においては，翼列全体を解析対象とするのではなく，周期境界条件を活用し，周方向の一部分を対象とした流動解析が行われている．

しかし，現実の蒸気タービンにおいては，入口より供給される蒸気流量は周方向に対して不均一であり，加えて，蒸気タービンの下流に配置される排気室の形状や，失速といった様々な要因によって，周方向に著しく不均一な流れ場が形成されていることが知られている．そこで，蒸気タービンの更なる性能向上を目指し，全周を対象とした大規模流動解析による流れ場の把握に対する関心が高まっている．

蒸気タービンの大まかな流路形状は，周期的に配置された翼列によって決定される．よって，翼列間の 1 流路(ブロック)を 1 プロセッサ毎に割り当てて計算することを考えた場合，ほぼ完全な負荷分散が可能であり，MPI による並列計算においては極めて高い台数効果が期待できる．我々の研究グループにおける既往の研究では，周方向 1 流路，軸方向 6 流路から構成される計 6 流路の計算格子に周期境界条件を適用し，MPI による並列計算を行うことで，90%以上の並列化率が達成されている．しかしながら，周方向複数流路において並列計算を行う場合，動翼(図 1b の赤い翼列)は静翼(同じく青)に対して回転運動しているため，移動境界面におけるデータの送受信先は時々刻々と変化することになる．よって，ターボ機械の大規模並列計算において高い加速率と並列化率を達成するためには，移動境界におけるデータの送受信をいかに高速に行うかが課題となる．

そこで，本研究では NEC の協力を得て，MPI を用いた移動境界に適用できる数値タービンの並列化手法の開発に取り組んだ．移動境界面に属するブロック間においては，MPI_BCAST 関数を用いて，1 流路対多流路間でのデータ通信アルゴリズムを作成し，その性能と問題点について評価した．



a. 計算格子の俯瞰図　　b. 翼列と流路

図 1. 計算格子の全体像と 1 プロセッサが担当する計算領域の比較

2. 数値タービンの計算手法

2.1 基礎方程式

数値タービンにおいては，凝縮を伴う圧縮性流れの基礎方程式として，蒸気の相変化を考慮した蒸気の質量保存式，運動量保存式，エネルギー保存式，液滴の質量保存式，液滴の数密度保存式，乱流運動エネルギーおよびその比散逸率からなる式を解く．本研究で取り扱う気液二相流は液滴の質量分率が

十分に小さい均質流を仮定する($\beta < 0.1$). 湿り蒸気の状態方程式および音速の式は石坂らにより定式化された式より算出する[1]. 凝縮による液滴の質量生成率は古典凝縮論に基づき, 凝縮核生成と液滴の成長による質量増加の和で表される. 本研究ではさらに, 液滴の成長を液滴の数密度を関数にした式で近似する[1]. 凝縮核生成率は Frenkel[2]の式より, 液滴の成長率は, Gyarmathy[3]のモデルより算出した. 数値解法として, 空間差分には Roe の流束差分離法[4]および 4 次精度 Compact MUSCL TVD スキーム[5]を用いた. 粘性項には 2 次精度中心差分を用い, 乱流モデルには SST モデル[6]を用いた. 時間積分には LU-SGS 法[7]を用いた.

2.2 数値タービンのデータ構造と並列化における領域分割

ターボ機械の翼列は, 回転機械の物理的な制約上, 周方向に対して相似性の強い構造を採用している場合が殆どである. 数値タービンにおいては, 図 2b に示すように, 翼と翼に囲まれた空間(流路)を1つの計算領域(ブロック)と定義し, 計算対象となる翼枚数に助走区間を加えた数だけブロックを確保する. 表 1 は逐次計算と並列計算における配列構造とループ構造の概要である. 逐次計算においては, 1 つの 3 次元配列(I,J,K)を I 方向にブロック数と等しい数だけ分割し, 各ブロックの計算領域として割り当てている. ループ構造からも明らかなように, 基礎方程式はブロック毎に閉じており, 隣接するブロックとのデータのやり取りは陽的に処理される. よって, 並列計算を行う場合においても, 計算領域をブロック単位で分割し, 各 RANK において個別に管理・計算を行う限りにおいては, 逐次計算と並列計算の間に解の不一致は生じない. なお, 並列計算におけるデータ通信は, 境界条件に関するサブルーチンが CALL される段階で行う.

過去の数値タービンの並列化に関する研究においては, 1 ブロックを複数領域に分割し, OpenMP を用いた並列計算した例について報告している. 山本らは, 領域分割計算においても陰解法の依存関係を考慮するために, パイプライン法を用いた LU-SGS 法の並列化手法を提案した[8][9]. しかし, 本研究においては, 移動境界における並列化アルゴリズムの複雑化を避けるために, 領域分割計算は行わないものとした.

表 1. 逐次計算と並列計算における配列構造の違い

1 nodeが配列全体を保持	1プロセッサが1ブロック分の配列を保持
	
DO L=1, N DO I=IS(L), IT(L) DO J=JS, JT DO K=KS, KT WW(I, J, K) = W(I, J, K) CONTINUE	LRANK0=MYRANK DO I=IS(LRANK0), IT(LRANK0) DO J=JS, JT DO K=KS, KT WW(I, J, K) = W(I, J, K) CONTINUE
a. 逐次計算の場合	b. MPI による並列計算の場合

2.3 数値タービンの境界条件

表 2 は数値タービンにおいて考慮されている, 主な境界条件の一覧である. ターボ機械の形状は様々であり, また考慮する現象や機械的構造も大きく異なるため, 表に挙げる境界条件の他にも適宜追加される境界条件が存在するが, ここでは割愛する.

これらの代表的な5つの境界条件は, 他ブロックとのデータの送受信の有無によって, 2つに大別できる. 先ず, 他ブロックとのデータ交換を必要としない種類の境界条件について述べる. 流入-流出境界条件は, 蒸気タービンの入口と出口に適用される境界条件である. 圧縮性流れの数値解析では, 計算領域の入口において全温と全圧を, 出口において静圧を境界条件として固定する手法が一般的に用いられており, 数値タービンにおいても同様の境界条件が適用されている. 図 2a は, 静翼面を青, 動翼面を赤, 翼端壁

を緑, シーリング部前後のキャビティ入口を黄色と水色に着色した図である. 壁面境界条件は, 図 2a の赤, 青および緑に着色された境界面に適用され, 個体壁面上に展開された計算格子が保有すべき値は, 計算条件に基づいた代数的な計算や, ブロック内部の計算格子からの外挿によって与える. 壁面境界条件は, 各ブロック内において完結しているため, 他ブロックとのデータの送受信は必要としない. 一方, 蒸気タービンの内部においては, 蒸気の一部は翼列を通過せず, 翼列と構造体の間に施されたシーリングを通過する. この様な流れは一般に漏れ流れ(Leak Flow)と呼ばれており, 図 2a における, 吸込境界(水色)や吐出境界(黄色)において近似的に考慮される. シーリングを通過する蒸気は翼に対して仕事をしないため, 漏れ流れを計算に取り入れることは, 蒸気タービンの性能を正しく評価する上で重要である. 上流のブロックにおいて発生した漏れ流れは, シーリングを通過し下流のブロックへと流入するため, 本来であれば, 漏れに関与するブロック間において流量やエンタルピ等に関するデータの送受信が必要となる. ただし, 本研究においては, 漏れ量を主流流量の 1.0%と定義し, 代数的に漏れ量を算出することで, 本境界条件をデータの送受信の対象外とした. 漏れ流れを考慮しない解析においては, 漏れ流れ境界面には壁面境界条件やすべり境界条件が適用されることが多い.

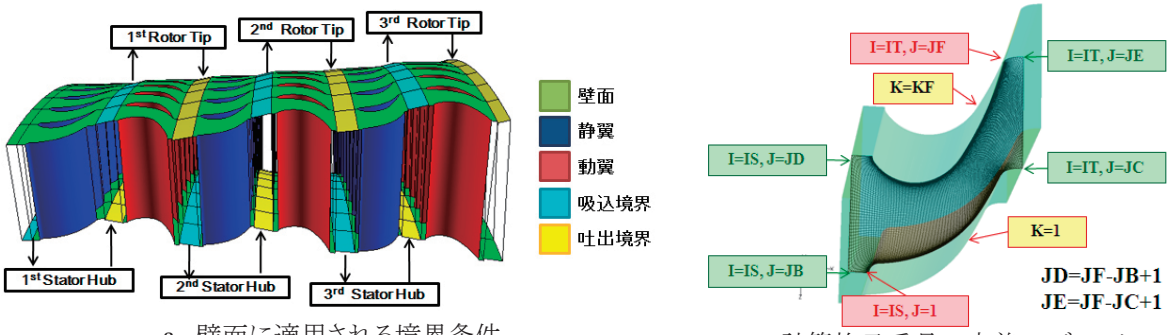
次に, 他ブロックとのデータの交換を必要とする境界条件について述べる. 図 2b は, 計算格子番号の定義の概略図であり, 数値タービンにおいては, 軸方向格子番号を I, 周方向格子番号を J, 半径方向格子番号を K と定義している. 周期境界条件は, 図 2c, 2d に示すように, 周方向に隣り合うブロックとの間に適用される境界条件であり, 周期境界面から流出した蒸気は, 隣接する周期境界面に流入するというのが基本原則である. 数値タービンにおいては, 周期境界面上の計算格子は, 隣接するブロックの計算格子と空間的に重なり合うように作成されている. よって, それぞれのブロックの計算を担当しているプロセッサを調べ, MPI_SENDRECV を用いてシフト通信を行うことで, 周期境界条件に必要なデータの送受信は完了する. 周期境界条件は JK 面に適用される境界条件である.

移動境界条件は, 流れ方向に隣接するブロック間に適用される境界条件であり, 図 2e において着色された円盤状の領域が適用範囲となる. 動翼は常に旋回しているため, 隣接するブロックは時々刻々と変化する点が, 周期境界条件と大きく異なる点である. また, 静翼列と動翼列の翼枚数比(周方向に並んだ翼の数の比)は, 蒸気タービン毎に固有の値を持ち, さらに, 静動翼列の各段落によっても変化する. すなわち, データ送受信の対象となるブロックを一意に決定できない点が, 並列化を行う上で課題となる. 移動境界条件においては, 移動境界面上に存在する格子点の情報を配列に列挙し, 対面する移動境界面上の格子点に対して相対位置を評価した上で, 情報の交換が行われる. しかしながら, 移動境界面上の全ての格子点について検索を行うと演算量が膨大となる. そこで, 移動境界面上の計算格子は, 回転軸に対して同心円状に配置され, K 方向の格子番号が同じであれば, 回転軸からの距離も等しくなるように設計されている. よって, 周方向の位置に関してのみ検索を行えば, 情報の交換対象となる隣接格子点を容易に知ることができる.

周期境界に適用するデータの送受方法に関しては次節において詳細に述べる.

表 2. 数値タービンにおいて考慮される主な境界条件

他ブロックとデータを交換しない境界条件	**1. 流入-流出境界条件(JK 面)**: 蒸気タービンに流入する気体の状態や, 入口-出口における圧力比などを設定する.
	2. 壁面境界条件(IK 面): 翼面および内周壁, 外周壁など壁面上における流体の状態を決定する.
	3. 漏れ流れ境界条件(IJ 面): 翼端漏れ流れを代数的に近似計算する.
他ブロックとデータを交換する境界条件	**4. 周期境界条件(JK 面)**: 周方向に隣り合うブロック間に適用する境界条件. 周方向のブロックの相対位置は変化しないため, 常に同じブロックがデータの送受信対象となる.
	5. 移動境界条件(JK 面): 軸方向(流れ方向)に隣り合うブロック間に適用する境界条件. 動翼列は静翼列に対して相対運動しているため, 周方向複数流路を考慮する場合には, データの送受信対象が時々刻々と変化する.



a. 壁面に適用される境界条件　　b. 計算格子番号の定義(1 ブロック)

c. 周期境界条件(翼列間)　　d. 周期境界条件(助走区間)　　e. 移動境界条件

図 2. 数値タービンにおいて考慮される主な境界条件の概略図

2.4 移動境界におけるデータの送受信について

動翼列は周方向に絶えず移動し，翼枚数比は蒸気タービンや段落毎に固有の値を持つため，移動境界においてデータの送受信対象となるブロックは一意に決定することはできない．そこで，図 3 に示すように，周方向に並ぶブロック(静翼列，動翼列および助走区間)をコミュニケータにおける1グループと定義し，移動境界処理を行うプロセッサ間の通信を限定することを考える．

例えば，上流側のブロックから下流側のブロックに対しデータ通信を行う場合，上流側の 1 プロセスと下流側の全プロセスとを 1 つのコミュニケータグループと定義する(図 3 右図)．次に上流側の 1 プロセスから下流側の他プロセスへ MPI_BCAST(Broad Cast)通信を行う．これによって，任意のプロセスが保有する移動境界面上の情報は，移動境界面を共有する対面側の全プロセスに対して送信される．一方，下流側から上流側へデータ通信を行う場合も，下流側の 1 プロセスから上流側の全プロセスに対し MPI_BCAST を用いた通信を行う．よって，上流側の流路数を M，下流側の流路数を N とした場合，1 つの移動境界面における MPI_BCAST は，M×N 回だけ CALL されることになる．

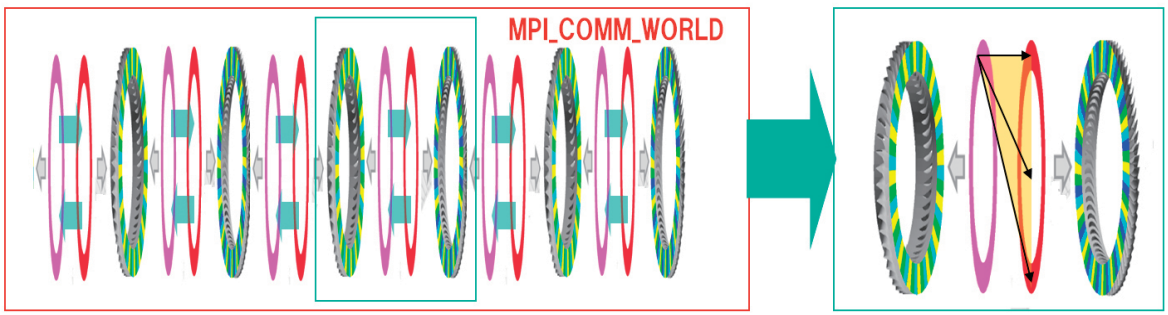


図 3. 移動境界面におけるコミュニケータのグループ分けの概略図

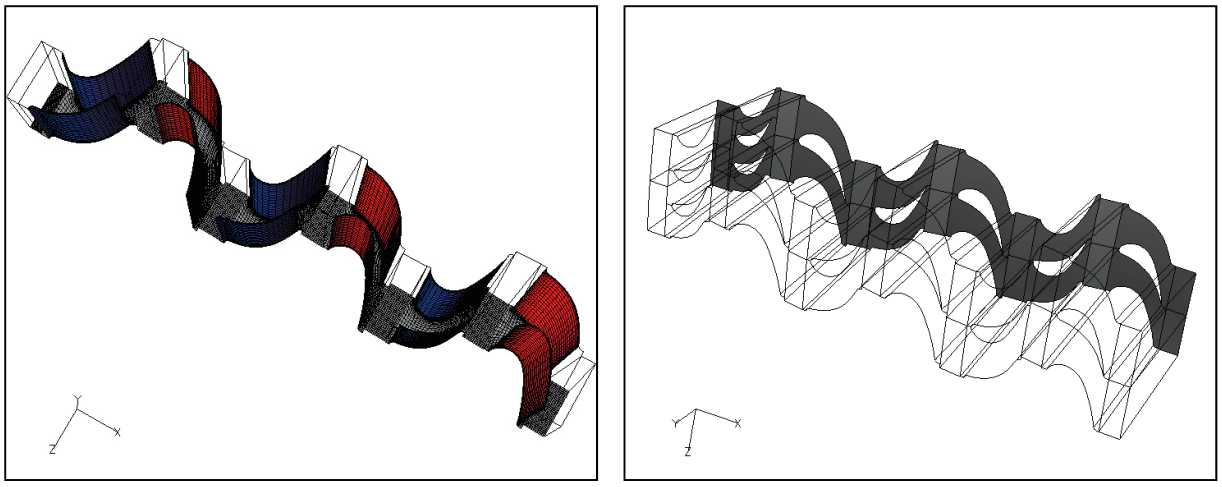
3. 計算結果

数値タービンの MPI による並列化は，逐次計算の場合とバイナリレベルで計算結果が一致することを確認しながら，段階的に行った．図 4 は本研究に用いた計算格子の俯瞰図である．図 4a は，静動翼列 3 段 6 流路と助走区間 7 流路の計 13 ブロックから構成される計算格子であり，図 4b は，初段静翼列のみ周方向 3 ブロック，他流路は周方向 2 ブロックの計 27 ブロックから構成される計算格子である．翼列間流路における格子点数が 31x31x31 点であるのに対し，助走区間の格子点数は 16x31x31 点であるため，並列計算を行う場合には，RANK 間にロード・インバランスが発生する．本共同研究における計算は，全て東北大学サイバーサイエンスセンターSX-9 上にて行った．13 ブロックを対象とした計算においては 1 node (1, 4, 7, 13 CPU)を用い，27 ブロックにおいては，1 node (1, 8, 16CPU)および 2 node (27 CPU)を用いて計算を行った．

表 3 は並列数に対する加速率の比較である．加速率は，逐次計算に必要とされた計算時間を，並列計算に必要とされた計算時間で除した値である．13 流路-13 並列の場合における加速率は 11.9，並列化率は 99.2%に達したのに対し，27 流路-27 並列の場合においては，加速率は 16.0，並列化率は 97.3%となった．また，13 流路，27 流路ともに，1 プロセッサが複数流路の計算を行った場合には，加速率の著しい悪化が見られた．

表 4 は 27 流路に対して並列計算を行った場合の，各サブルーチンのロードバランスと加速率の比較である．オレンジ色に着色されたサブルーチンは，MPI によるデータ通信を含むサブルーチンである．Explicit，Implicit2，Result，Viscous といった，計算負荷の高いサブルーチンについては，プロセッサ数に応じた高い加速率が得られたものの，データ通信に関するサブルーチンは，ほとんど加速率の向上が得られていないことがわかる．これは，移動境界における BCAST 通信に多くの時間を要したためである．

図 5 は 27 流路について並列計算した場合における，通信時間が計算時間に占める割合の比較である．全プロセッサが 2 ブロック以上の計算を行う 8 並列の場合と，各プロセッサが 1 ブロックずつ計算を行う 27 並列においては，データ通信が全計算時間に占める割合は 10%に満たないことがわかる．一方，プロセッサ間のロード・インバランスが顕著に表れる 16 並列においては，データ通信に要した時間が IDLE 時間(データ通信待ち時間)も含めて全体の 6 割以上にも及び，データ通信が処理速度のボトルネックとなっていることがわかる．以上より，BCAST 通信を頻繁に行う本並列化手法においては，プロセッサ間のロードバランスが可能な限り均一になるように，計算領域を分配することが，極めて重要であるといえる．



a. 単流路 13 ブロック　　b. 複数流路 27 ブロック

図 4. 計算格子

表 3. 並列数による加速率の比較

	逐次計算(1 並列)	4 並列	7 並列	13 並列
13 流路	1.0	3.2	6.3	11.9
	逐次計算(1 並列)	8 並列	16 並列	27 並列
27 流路	1.0	5.8	6.6	16.0

表 4. ランク 0 が要した各サブルーチンの計算時間(CPU-time [sec])と加速率の比較

	並列数 (27 流路)							
	1		8		16		27	
subroutine	[sec]	加速率	[sec]	加速率	[sec]	加速率	[sec]	加速率
MAIN_LOOP	184.37	(1.0)	31.92	(5.8)	27.76	(6.6)	11.54	(16.0)
explicit	94.29	(1.0)	15.30	(6.2)	6.76	(13.9)	6.67	(14.1)
implicit2	49.62	(1.0)	5.27	(9.4)	1.56	(31.7)	1.54	(32.2)
result	13.56	(1.0)	1.52	(8.9)	0.51	(26.4)	0.51	(26.7)
viscous	9.27	(1.0)	1.06	(8.7)	0.35	(26.6)	0.35	(26.6)
tbmdl2	8.32	(1.0)	0.93	(9.0)	0.31	(26.9)	0.31	(26.8)
mpisub_read_grid	8.04	(1.0)	5.89	(1.4)	7.35	(1.1)	5.84	(1.4)
mpisub_comm_bundperi	0.00	(0.0)	0.88	(0.0)	16.21	(0.0)	0.03	(0.0)
bundslid	7.99	(1.0)	0.64	(12.5)	0.21	(37.7)	0.21	(38.0)
main	0.59	(1.0)	0.55	(1.1)	0.52	(1.1)	0.54	(1.1)
mpisub_output_restart	0.33	(1.0)	0.34	(1.0)	0.31	(1.1)	0.29	(1.1)
bundwall	0.19	(1.0)	0.02	(8.1)	0.01	(20.7)	0.01	(20.7)
bundperi	0.12	(1.0)	0.03	(4.6)	0.01	(9.7)	0.01	(10.5)
init	0.06	(1.0)	0.01	(7.4)	0.00	(14.8)	0.00	(14.8)
metric	0.04	(1.0)	0.01	(8.0)	0.00	(20.0)	0.00	(20.0)
initmp	0.04	(1.0)	0.03	(1.2)	0.01	(2.6)	0.01	(2.6)
boundlhs	0.04	(1.0)	0.00	(8.8)	0.00	(17.5)	0.00	(17.5)
boundrhs	0.04	(1.0)	0.00	(8.8)	0.00	(17.5)	0.00	(17.5)
mpisub_read_figure	0.03	(1.0)	0.03	(1.0)	0.02	(1.6)	0.02	(1.7)
bundstrm	0.02	(1.0)	0.01	(1.4)	0.01	(2.9)	0.01	(2.9)
mpisub_final	0.02	(1.0)	0.14	(0.1)	0.24	(0.1)	0.39	(0.1)
mpisub_init	0.02	(1.0)	0.10	(0.2)	0.15	(0.1)	1.11	(0.0)
mpisub_comm_bundslid_theta	0.01	(1.0)	5.08	(0.0)	0.64	(0.0)	0.57	(0.0)
outigs	0.01	(1.0)	0.00	(6.5)	0.00	(13.0)	0.00	(13.0)
mpisub_read_init	0.01	(1.0)	0.01	(0.9)	0.01	(0.9)	0.01	(0.9)
mpisub.allocate_distributed_arrays	0.01	(1.0)	0.01	(1.4)	0.01	(0.8)	0.01	(1.1)
mpisub_comm_bundslid	0.01	(1.0)	0.01	(0.8)	0.01	(1.1)	0.01	(0.8)
bundmain	0.01	(1.0)	0.01	(1.0)	0.01	(1.0)	0.01	(1.0)

※オレンジ:データ通信を含むサブルーチン, ピンク:ファイル入出力を含むサブルーチン

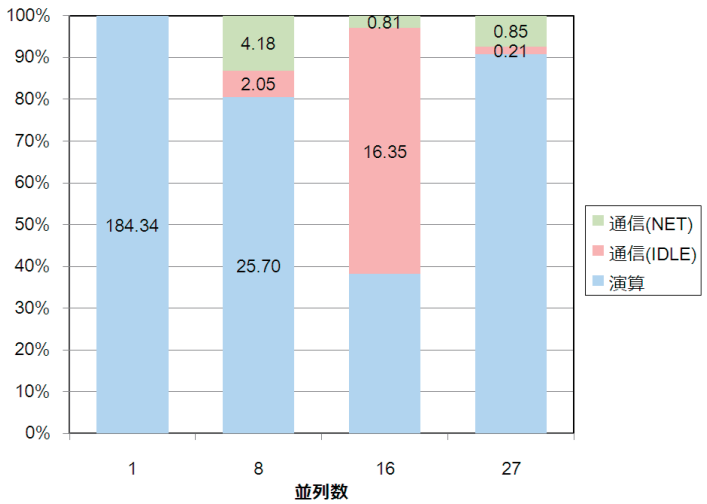


図 5. 各並列数において通信時間が計算時間に占める割合(27 流路)

4. 結言

MPI を用いて数値タービンの並列化を行い，加速率と並列化率を評価した．移動境界問題の並列化に関しては，移動境界面に属する1つの RANK から，その他の RANK に対する BCAST を用いて実装した．13 流路-13 並列の場合における加速率は 11.9，並列化率は 99.2%，27 流路-27 並列の場合においては，加速率は 16.0，並列化率は 97.3%を達成した．

謝辞

本研究における数値タービンの並列化は，NEC および東北大学サイバーサイエンスセンターの全面的な技術協力の元，行いました．ご協力頂きました団体ならびに皆様に対し，厚く御礼申し上げます．

参考文献

[1] Ishizaka, K., Ikohagi, T. and Daiguji, H., A High-Resolution Numerical Method for Transonic Non-Equilibrium Condensation Flow through a Steam Turbine Cascade, Proc. of the 6th ISCFD, 1, 479-484, 1995.
[2] Frenkel, J., Kinetic Theory of Liquids, Dover, 1955.
[3] Gyarmathy, G., Zur Wachstumsgeschwindigkeit kleiner Flussigkeitstropfen in einer ubersattigten Atmosphare, Zeitschrift fur angewandte Mathematik und Physik, 14-3, 280-293, 1963.
[4] Roe, P.L., Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes, Journal of Computational Physics, 43, 357-372, 1981.
[5] Yamamoto, S. and Daiguji H., Higher-Order-Accurate Upwind Schemes for Solving the Compressible Euler and Navier-Stokes Equations, Computers and Fluids, 22-2/3, 259-270, 1993.
[6] Menter, F.R., Two-equation Eddy-viscosity Turbulence Models for Engineering Applications, AIAA Journal, 32-8, 1598-1605, 1994.
[7] Yoon, S. and Jameson, A. Lower-upper Symmetric-Gauss-Seidel method for the Euler and Navier-Stokes equations, AIAA Journal, 26, 1025-1026, 1988.
[8] Satoru Yamamoto, Yasuhiro Sasao, Shoichiro Sato and Kentaro Sano, Parallel-Implicit Computation of Three-dimensional Multistage Stator-Rotor Cascade Flows with Condensation, 18th AIAA CFD Conference-Miami, AIAA Paper 2007-4460, June, 2007.
[9] Satoru Yamamoto, Yasuhiro Sasao, Kentaro Sano, Hiroshi Satsuki, Kouichi Ishizaka and Hiroharu Ooyama, Parallel Computation of Condensate Flows through 2-D and 3-D Multistage Turbine Cascades, Int. Gas Turbine Congress 2007(IGTC'07)-Tokyo, Dec.2007, CD-ROM..

Fortran スマートプログラミング

— 第２回　サブルーチンと入出力 —

田口　俊弘
摂南大学理工学部電気電子工学科

前回はFortranの基本的な文法とエラーの出にくいプログラムの書き方について説明しました。ここまでの知識を使うだけで，原理的にはどんなプログラムを作ることも可能です．プログラムとは計算機に仕事をさせるための手順ですから，それほどバラエティはありません．動作の繰り返しを書くためのdo文や，条件分岐のif文が使いこなせれば，計算機の能力のほとんどを使いこなすことができます．

しかし長いプログラムを書くときは，メンテナンスのしやすさを考慮して書かなければなりません．プログラムが短ければ全体を見ながらチェックや修正ができますが，長くなるとチェックに時間がかかり，見落としによる修正ミスが起こる可能性も高くなります．見落としを防ぐには何度も見直す必要があり，チェック時間はさらに増大します．

このメンテナンスを容易にする手法の一つが“サブルーチン”です．サブルーチンとはメインプログラムと同じレベルの閉じたプログラムのことで，必要に応じて他のプログラムから呼び出してその機能を使います．長いプログラムをサブルーチンに分割し，それらをビルディングブロックとして全体を構成すれば，プログラム作成や修正の作業が楽になります．今回はこのサブルーチンの作り方と使い方を説明します．

プログラムは計算をさせるだけでは意味がありません．結果を出力して初めて完結します．これまでは，もっとも単純なprint文でとりあえず画面に表示する方法を使ってきましたが，今回は好みに応じて数値の表示形式を変える方法や，ファイルに保存する方法について説明します．さらにデータを入力することでプログラムを変更せずに動作を変える方法についても説明します．

1.　サブルーチン

1.1　サブルーチンの利用目的

“ルーチン”とは，動作開始点と終了点が定義されている閉じたプログラム手順を示す用語です．プログラムを起動したときに最初に動作するのはメインプログラムですが，これを“メインルーチン”ともいいます．“サブルーチン”は，メインプログラムと同様に動作開始点と終了点を持っていますが，メインプログラムと異なり，単独で動作することはできません．必ず，他のルーチンに動作開始命令を記述して，必要に応じて実行させる必要があります．一つのプログラムにメインプログラムは一つですが，サブルーチンは名前が異なれば複数存在してもかまいません．また，いくつかのルーチンごとにファイルを作成して別々にコンパイルし，必要に応じて結合（リンク）させることも可能です．

サブルーチンを利用する目的は主として３つあります．

(1) 複数の場所で同じ計算手順を使用するため
(2) 方程式の解法や行列式の計算などのような定型処理をするため
(3) 長いプログラムを分割してメンテナンスを容易にするため

(1)がサブルーチン本来の使い方です．ある一連の手順を複数の場所で使うとき，それぞれの場所に同じプログラムを書くと，プログラムが長くなるし，修正の際に何カ所も同じ修正をしなければなりません．このとき，その手順をサブルーチンに置き換えれば，プログラムは短くなるし，チェック作業も短縮されます．

(2)は(1)を発展させたものです.複雑な数学公式や汎用性のある数値計算アルゴリズムをプログラム中に直接記述するときは，プログラム中で宣言された変数や配列を使って書きます．このため，同じ計算手順を別のプログラムで使いたくてもそのまま使えるとは限りません．そこで計算手順をサブルーチンにして，計算に必要な数値を与える部分と計算結果の数値を受け取る部分だけが外部から見えるようにしておきます．そうすれば，受け渡し部を書き換えるだけで，複雑な計算手順を色々なプログラムから利用することができます．このようにブラックボックス化したサブルーチンは，他の人に提供したり，他の人からもらって利用することも可能で，そのような汎用性のあるサブルーチンを集めたものが，“ライブラリ”です.

さて,計算機シミュレーションのように多数の解析手順を複合したプログラムを書く場合には,(3)の目的が重要になります．様々な物理過程の計算や，入出力に関する作業など，ある程度まとまった内容ごとにサブルーチンにするのです.文章でいえば,章や節に分割するようなものです.このため，メインプログラムには，それぞれのサブルーチンを呼び出す文と，必要ならそれらを繰り返すための do 文だけ書いておきます．シミュレーションプログラムは，サブルーチンという部品を使って組み立てるものだと考えればいいでしょう.

サブルーチンに分割しておけば，プログラムのメンテナンスが楽になります．これは，それぞれのルーチンが独立しているので，プログラムを修正したときの影響や，エラーが発生する原因がルーチン内に限定されるからです.

1.2　サブルーチンの宣言と呼び出し

サブルーチンは subroutine 文で開始を宣言し，end subroutine 文で終了します．すなわち，次のような構造にします.

```
      subroutine subr1
        implicit none
        real a,b
        integer i
          .......
          .......
      end subroutine subr1
```

先頭の subroutine 文で，subroutine の後に指定した文字列（この例では subr1）を“サブルーチン名”といいます．また，最後の end subroutine文にもサブルーチン名を指定します．見てわかるように，メインプログラムと同じ構造です．サブルーチン内部のプログラムの書き方も基本的にメインプログラムと同じで，subroutine 文の次に implicit noneを書き，非実行文を上方に集約して，その後に実行文を書きます．サブルーチンはそれ自体で閉じているので，実行文中で使う変数や配列は，基本的にその内部で宣言しなければなりません．ただし，異なるルーチン間で共用可能な変数を別途用意することは可能です．これについては，1.7節で説明します.

上の例では，subroutine 文にサブルーチン名しかありませんが，これを“引数なしサブルーチン”といいます．これに対して，サブルーチン名の後に，かっこで囲んだ変数リストを付加することができ，これを“引数ありサブルーチン”といいます．以下に例を示します.

```
      subroutine subr2(x,m,y,n)
        implicit none
        real x,y,a,b,z(10)
        integer m,n,i,k
          .......
          .......
      end subroutine subr2
```

リスト中の変数を“引数”といいます．この例では，x,m,y,n が引数です．引数は，サブルーチンとそのサブルーチンを使うルーチンの間で数値を受け渡すのに使います．引数もサブルーチン内部の変数なので，必要な型に応じた宣言をしなければなりません.

サブルーチンを動作させてその機能を使うときは，call 文を使ってそのサブルーチンを指名します．このため，サブルーチンを指名することを，“サブルーチンを呼び出す”とか“コールする”といいます．call 文は以下のような形式です．

```
call サブルーチン名                                    ！引数なしサブルーチン用
call サブルーチン名(数値または変数のリスト)    ！引数ありサブルーチン用
```

例えば，先ほど出てきた，引数なしと引数ありの二つのサブルーチンを使うときは，それぞれ，次の(1)と(2)のようにコールします．

```
  real z
  integer m
  call subr1          ......................(1)
  m = 21
  call subr2(10.0,100,z,m*5+1)          ..... (2)
```

前回説明したように，計算式は計算結果の数値を動作命令に与えるので，call 文の引数には，(2)の一番右のように計算式を与えることもできます．

サブルーチンは単独では動作できないので，メインプログラムが別途必要です．例えば，次のようなセットを構成して初めて一つのプログラムが完成します．

```
program stest1
   implicit none
   real x,y
   x = 5.0
   y = 100.0
   call subr(x,y,10)
   print *,x,y
end program stest1

subroutine subr(x,y,n)
   implicit none
   real x,y
   integer n
   x = n
   y = y*x
end subroutine subr
```

ここでは，メインプログラムを先に，サブルーチンを後に書きましたが，逆でもかまいません．サブルーチンが複数存在する場合も，ルーチンを記述する順番は実行結果とは無関係です．サブルーチンの中から別のサブルーチンをコールすることも可能です．

上記のプログラム実行の流れを図１に示します．

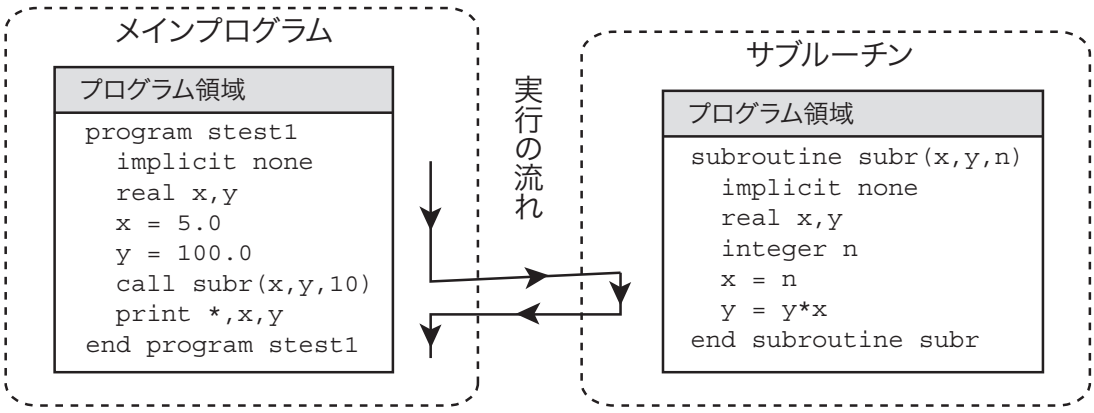


図１．サブルーチンの呼び出しと実行の流れ

図のように，メインプログラムの call 文でサブルーチンを指名すると，サブルーチンの一番最初の実行文に動作が移り，サブルーチン内部の動作が完了すると，コールしたメインプログラムに戻って，その call 文の次の文から実行を継続します.

条件に応じて，途中でサブルーチンの処理を打ち切って戻るときには return 文を用います.

```
subroutine subr(x,y,m,n)
  implicit none
  real x,y
  integer m,n
    .......
  if (m<0) return
    .......
end subroutine subr
```

この例では，m＜0 のときにサブルーチンの処理が終了し，コールしたルーチンに戻ります．ここで return 文の代わりに stop 文を用いれば，プログラム自体が終了します.

1.3　サブルーチン内部の変数と引数

サブルーチン内部で宣言した変数や配列は，メインプログラムや他のサブルーチンから独立しています．すなわち，同じ名前を使っても全く別の変数です．例えば，

```
program stest1
  implicit none
  real x,y
  x = 10.0
  y = 30.0
  call subr
end program stest1

subroutine subr
  implicit none
  real x,y
  print *,x,y
end subroutine subr
```

と書いても，サブルーチン中の print 文によって出力される x と y はメインプログラムで代入した 10.0 と 30.0 ではなく，全く無関係な数字です．逆に言えば，他のルーチンでの宣言を気にせずに変数や配列の名前を決めることができます．このルーチン内部でのみ有効な変数を“ローカル変数”といいます.

上記のプログラムを期待通り働かせるために，コール側ルーチンの数値をサブルーチンに伝えるのが引数です．例えば，上記のプログラムを以下のように書き直せば，サブルーチン中の print 文で出力される x と y は，メインプログラムと同じ 10.0 と 30.0 になります.

```
program stest2
  implicit none
  real x,y
  x = 10.0
  y = 30.0
  call subr(x,y)
end program stest2

subroutine subr(x,y)
  implicit none
  real x,y
  print *,x,y
end subroutine subr
```

引数は，数値の受け渡しをするための窓口に過ぎないので， call 側の引数と subroutine 側の

引数の変数名を同じにする必要はありません．次のサブルーチンに置きかえても全く同じ動作をします．

```
      subroutine subr(a,b)
        implicit none
        real a,b
        print *,a,b
      end subroutine subr
```

外部からの影響を受けたり，外部に影響を与える，という性質を持つことを除けば，引数もローカル変数です．すなわち，コールしたルーチンからその詳細は見えません．見えるのは，引数という窓口の並びだけです．このため，引数ありサブルーチンを使うときに重要なポイントは，

(1) 引数の数
(2) 対応する引数の型

が call 文と subroutine 文とで一致していなければならないことです．例えば，

```
      program stest3
        implicit none
        real z
        integer n
        z = 200.0
        n = 21
        call subr(10.0,z**2,100,n*5+1)
      end program stest3

      subroutine subr(x,y,m,n)
        implicit none
        real x,y
        integer m,n
        print *,x,y,m,n
      end subroutine subr
```

というプログラムでは，表１のような対応になっています．

表１．サブルーチンの引数対応

call 文	subroutine 文	数値型
10.0	x	実数
z**2	y	実数
100	m	整数
n*5+1	n	整数

ここで注意すべきなのは，第 1 引数の x は実数型なので実定数 10.0 を与え，第 3 引数の m は整数型なので整定数 100 を与えていることです．もし，第 1 引数に同じ意味だろうと思って 10 という整定数を与えると，実行時エラーになります．

サブルーチン内部の実行文で，引数に数値を代入すると，call 文の対応する引数に与えた変数にその数値が代入されます．例えば，

```
      program stest4
        implicit none
        real x,y,p
        x = 10.0
        y = 30.0
        call subr(x+y,20.0,p)
        print *,x,y,p
      end program stest4
```

```
subroutine subr(x,y,z)
  implicit none
  real x,y,z
  z = x*y
end subroutine subr
```

と書くと，サブルーチン中の x はコール側の x+y，すなわち 40.0 であり，サブルーチン中の y はコール側の 20.0 なので，サブルーチン中の z には x*y の計算結果である 800.0 が代入されます．このとき，z が引数なので，call 文の対応する位置にある変数 p に 800.0 が代入され，call 文の次の print 文では x，y，p として，10.0,30.0,800.0 が出力されます．

このようにサブルーチンの引数に数値を代入することで，call 文の引数変数に代入される数値を“戻り値”といいます．戻り値を使えば，サブルーチンの動作で得られた結果をコール側で受け取ることができます．ただし，call 文の引数には定数を与えたり計算式を書いてもよい，と述べましたが，戻り値を指定する引数は別で，必ず変数か配列にしなければなりません．理由を次節で説明します．

1.4　ルーチン間におけるデータの受け渡しと間接アドレス

ルーチン間のつながりをもう少し詳しく考えてみましょう．各ルーチンは基本的にプログラム領域とデータ領域から構成されています．プログラム領域とは文字通りプログラム命令が記録されている領域のことであり，データ領域とは変数や配列のために用意されたメモリ領域のことです．プログラム的には，変数や配列の宣言文がデータ領域の指定を意味しています．

変数や配列が各ルーチンごとに宣言されなければならないのは，データ領域が各ルーチンそれぞれに付属しているからです．このため，図２のように異なるルーチンで同じ名前の変数を宣言しても，それらは別のデータ領域に所属しています．これがローカル変数です．ローカル変数は，そのルーチン内部からしか参照することができません．

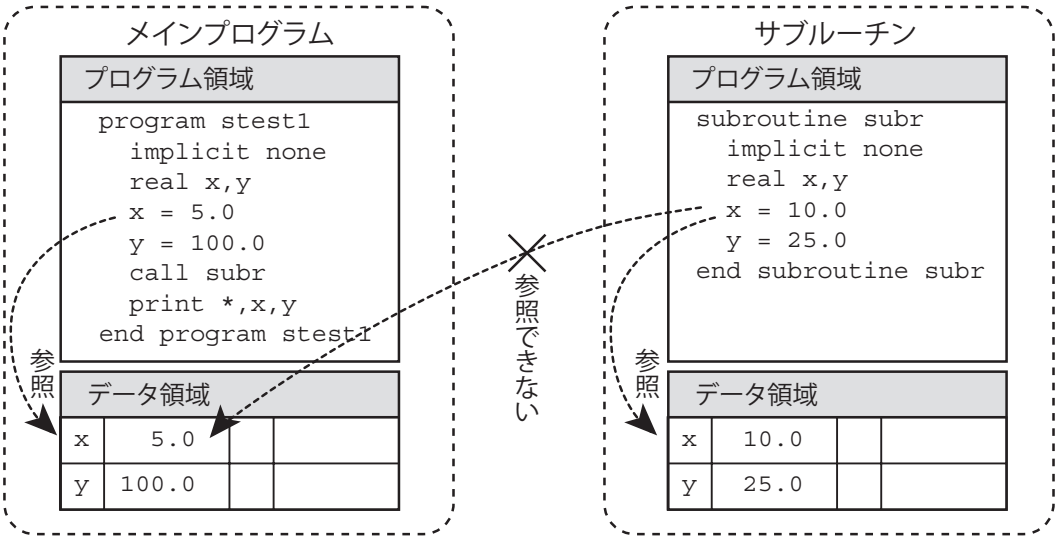


図２．プログラム領域とデータ領域

このため，あるルーチンで計算した数値を別のルーチンで使うには特別な手続きが必要になります．これが引数です．引数を使えば，コール側のルーチンとサブルーチンの間でデータをやりとりすることができます．では，具体的にどうやってデータの受け渡しをしているのでしょう．

データをサブルーチンに渡す手段として，もっとも単純に考えられるのは，引数の数値を直接サブルーチンの変数に代入することです．図３を見て下さい．図のように，コール側の引数 x と y の内容（この例では 5.0 と 100.0）が，サブルーチンの引数として宣言された変数，a と b に代入されています．この方式は，引数の値を直接渡して呼び出す，という意味で“call by value”

と呼ばれています．“call by value”は，コール側からサブルーチン側への値の引渡しについては問題ありません．C 言語はこの方式を採用しています．

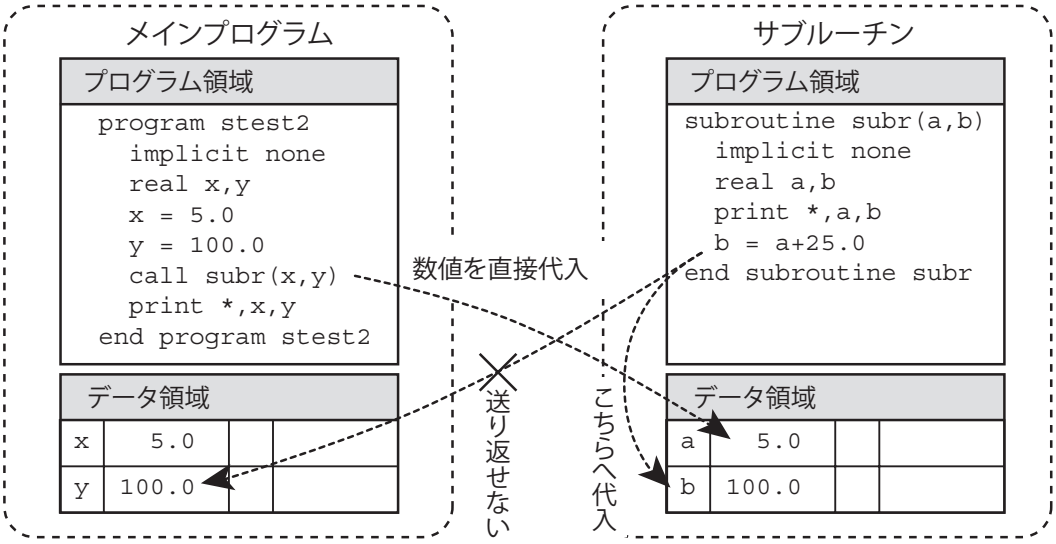


図３．“call by value”を使ったデータ渡しでは，値を送り返せない

しかし，この方式では，サブルーチン側で作成したデータをコール側に戻すことはできません．サブルーチンは，コール側から送られてきた“値”はわかるのですが，それ以上の情報がないので，コール側のどこにデータを代入すればいいかわからないからです．図３のように，サブルーチン中の代入文，b=a+25.0，でサブルーチンの変数 b に 30.0 という値を代入しても，サブルーチンのデータ領域の b に代入されるだけで，メインプログラムのデータ領域には影響がありません．

この問題を解決するために，Fortran は“call by value”ではなく，“call by reference”という方式を採用しています．“call by reference”とは，変数の内容ではなく，変数の存在場所（メモリアドレス）をサブルーチンに伝えて呼び出す方式です．

コンピュータのメモリには，“番地”と呼ばれる通し番号がついています．メモリアドレス（あるいは単にアドレス）とはこの番地のことで，プログラムの実行中に，あるメモリのデータを読み書きするときは，そのメモリのアドレスを指定して行います．例えば，簡単な代入文，

```
a = 1500.0
b = a
```

を考えてみましょう．この結果，b のメモリに 1500.0 という値が書き込まれますが，代入式 b=a において，a（アドレスを[103]とする）という変数からデータを取って来て b に代入するという流れは，

```
a のアドレス指定　→　[103]番地のメモリ内のデータ（1500.0）の呼び出し
　　→　b に代入
```

となります．流れを図に描けば図４のようになります．この方式を“直接アドレス”といいます．

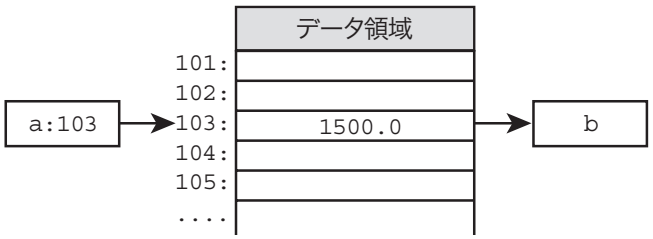


図４．直接アドレスによるメモリ参照

これに対し，あるアドレス（addr1）のメモリに入っているのが別のメモリのアドレス（addr2）で，アドレス addr1 を指定して，アドレス addr2 のメモリに入っているデータを読み書きする方式を“間接アドレス”といいます．Fortran では，“call by reference”でサブルーチンを呼び出すので，引数には call 側ルーチンのメモリアドレスが入っています．このため，サブルーチン内での引数の読み書きは間接アドレスを使って行います[1]．例えば，

```
program stest2
  implicit none
  real a
  a = 1500.0
  call subr(a)
end program stest2

subroutine subr(s)
  implicit none
  real b,s
  b = s
end subroutine subr
```

というプログラムを考えます．サブルーチン subr の中の代入文 b=s によって，変数 b のメモリの値は 1500.0 になりますが，引数変数 s には a の内容である 1500.0 ではなく，メインプログラムの変数 a のアドレスが入っています．このため， b=s という代入文で，変数 s（メモリアドレスを[107]とする）からデータを取ってきて変数 b に代入する流れは，

```
s のアドレス指定　→　[107]番地にあるメモリアドレスデータ（103）の呼び出し
　　→　[103]番地にあるデータ（1500.0）の呼び出し
　　→　b に代入
```

となります．図示すれば図５のようになります．

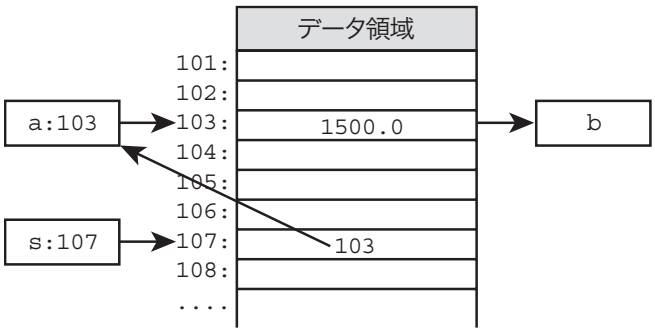


図５．間接アドレスによるメモリ参照

間接アドレスによる読み書きは直接アドレスよりも時間がかかります．よって，b=s の代入文において，代入する値を保持している右辺の変数 s が間接アドレス指定の場合にはメリットがありません．しかし，代入される左辺の変数 b が間接アドレス指定の場合には，この仕組みを利用することで，コール側ルーチンの変数にサブルーチン側のデータを代入することができます．

例えば，図６のようなプログラムを考えます．サブルーチンの引数変数 b に a+25.0 の結果である 30.0 を代入すると，b にはメインプログラムの変数 y のアドレスが入っているので，間接アドレス指定により y に 30.0 が代入されます．

戻り値を利用するときは，対応する引数にコール側変数のアドレスを与えなければなりません．これが変数または配列を指定しなければならない理由です．戻り値指定の引数に定数や計算式を与えると実行時エラーになります．

---

[1] 配列を実現するのにも間接アドレスが使われています．例えば，a(10)は a(1)から数えて 10 番目のメモリです．よって，配列の先頭要素 a(1)のアドレスをメモリに記録し，それに 9（=10-1）を加えて間接アドレスを用いれば，a(10)のメモリを読み書きすることになります．

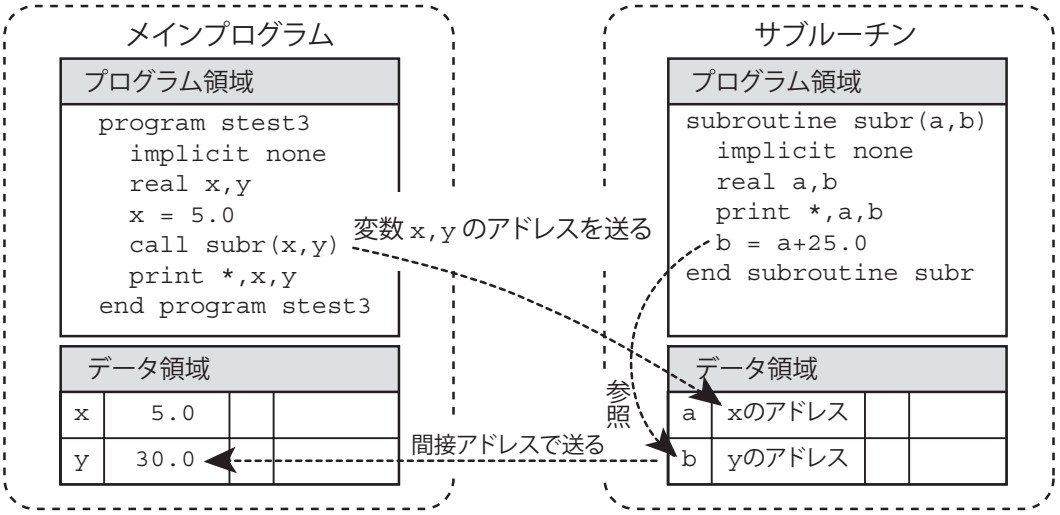


図６．Fortran では“call by reference”により，データを送り返すことができる

1.5　配列を引数にする場合

配列を call 文の引数にするときは，配列名を引数にすることも，配列要素を引数にすることも可能です．配列名は配列の先頭要素アドレスを示すので，配列名を引数にすることと，その配列の第１要素を引数にすることは同じ意味になります．例えば，real a(10)と宣言した１次元配列に対して，

```
call sub(a)　と　call sub(a(1))
```

は同じ動作をします．

これを受けるサブルーチンでの宣言は，サブルーチン内部でそのデータをどう使うかで決めます．一例を示します．

```
subroutine sub(x)
  implicit none
  real x
  x = 10.0
end subroutine sub
```

この例では引数 x が単一変数として扱われています．このため，call 文で指定した a(1)だけがサブルーチン内部で利用でき，他の要素は使えません．これは，call sub(a)のように，配列名を与えた場合でも同じです．

これに対し，

```
subroutine sub(x)
  implicit none
  real x(10)
  integer i
  do i = 1, 10
      x(i) = i
  enddo
end subroutine sub
```

のように引数 x を配列宣言すれば，あたかもコール側ルーチンの a という配列がサブルーチン内部の配列 x になったかのように取り扱うことができます．ここで“あたかも”という言葉を使ったのは，サブルーチンでの宣言文の書き方によってはそうならないことがあるからです．例えば，サブルーチンでの配列宣言 x(10)の要素数 10 はコール側と同じ数にする必要はありません．100 のように大きくても，1 のように小さくてもかまわないのです．これは，引数 x が配列の先頭要

素アドレスを保持している１個の変数にすぎないからです. サブルーチンでの引数配列の宣言は, 配列の形状（次元や下限値）を表示するためのダミーにすぎません.

このため，コール側ルーチンで宣言した要素数とサブルーチンで宣言した要素数を一致させる必要はありません[2]. 与えられた配列をどう宣言するかは，サブルーチン内部の計算の都合に合わせて, サブルーチン側で決めることができます. よって, 汎用性のあるサブルーチンにするには, 配列のアドレス以外に，配列の要素数も引数にしてサブルーチンに伝える必要があります.

配列の形状さえ分かればいいのですから，サブルーチンで引数配列を宣言するときは，数字の代わりに“*”を書くことができます．例として，要素数 n の１次元配列 a のデータを，同じ長さの１次元配列 b にコピーするプログラムを考えてみましょう．最も単純な答えは，以下のようなものです.

```
      subroutine copy(a,b,n)
        implicit none
        real a(*),b(*)
        integer n,i
        do i = 1, n
          b(i) = a(i)
        enddo
      end subroutine copy
```

このように，サブルーチンの引数配列 a や b の宣言を“*”にすることで，要素数が不定の１次元配列であることを明示しています.

“*”による宣言は２次元以上の配列でも使うことができますが，制限があります．例えば，a(*,*)という形の宣言はできません．なぜなら，２番目の要素を進めるのは１番目の要素が最大値，すなわち宣言した上限数に達したときなので，１番目の要素が不定では情報不足で進められないからです．このため，“*”が使えるのは，一番右側の要素数のみです．例えば，real a(3,*)と宣言すると，第１要素数が３の２次元配列として計算されます.

しかし, これではどんな２次元配列でも使えるサブルーチンを作ることはできません. そこで, “整合配列”と呼ばれる機能が用意されています. 整合配列とは subroutine 文の引数の中にある整数型変数を利用して宣言した形の引数配列のことです．例えば，以下のように宣言することができます.

```
      subroutine copy2d(a,b,m,n)
        implicit none
        real a(m,n),b(m,n)
        integer m,n,i,j
        do j = 1, n
          do i = 1, m
            b(i,j) = a(i,j)
          enddo
        enddo
      end subroutine copy2d
```

この例では，m と n が引数の中にあるので，これらを使って引数の配列 a と b を宣言することができるのです．a(m,n)のような単純な宣言だけではなく，a(2*m*n,n-1)のように，計算式を使って宣言することも可能です.

このサブルーチンの使用例を次に示します. サブルーチンの引数 a, b にどんな要素数の２次元配列を与えても，コール側ルーチンにおける配列宣言と同じ要素数を m, n に与えれば，正しく動

[2] 要素数だけではなく，次元さえ一致させる必要はありません．コール側が２次元配列で，サブルーチンでの宣言が１次元配列でも動作は可能です．例えば，上記のサブルーチン copy は，２次元配列でも使用できます．これは，２次元配列もメモリ上では１次元的に並んでいるからです．ただし，要素数を与える引数 n には全要素数を指定する必要があります．例えば，a(10,10)と宣言された配列ならば，n=100（=10×10）を与えます.

作します.

```
program stest3
  implicit none
  real a(10,20),b(10,20),c(100,200),d(100,200)
    .......
  call copy2d(a,b,10,20)
  call copy2d(c,d,100,200)
end program stest3
```

なお，配列宣言の際には，コロンを付加して下限を指定することができますが，これをサブルーチンに引き継ぐにはサブルーチン側でも同じ下限指定をする必要があります．例えば，

```
program stest4
  implicit none
  real a(-1:100)
  call sub(a,a,100)
end program stest4

subroutine sub(x,y,n)
  implicit none
  real x(-1:n),y(*)
  integer n
  x(10) = 10.0
  y(10) = 10.0
end subroutine sub
```

と書いた場合，メインプログラムと同じ下限指定をした配列 x では x(10)がメインプログラムの a(10)に対応するので，a(10)に 10.0 が代入されます．しかし，下限指定をしていない配列 y では下限が 1 ですから，a(-1)から数えて 10 番目の a(8)に 10.0 が代入されます．任意の下限を持つ配列に対して動作するプログラムにするには，下限の数値を引数にして整合配列を使います．

1.6　関数副プログラムと色々な組み込み関数

sin(x)のように，引数を使って内部で計算した結果を計算式中で使うことができる“関数”を自作することもできます．これを関数副プログラムといいます．関数副プログラムは，サブルーチンとほとんど同じ構造をしていますが，以下のような違いがあります．

(1) subroutine の代わりに function を書く
(2) 関数名を変数として宣言し，それに計算結果（関数値）を代入しなければならない

これ以外は，引数ありサブルーチンと同じです．引数に関する注意もサブルーチンと同じで，戻り値を与える引数を含めることもできます．関数副プログラムとは，引数以外に戻り値を１個持つサブルーチンの変種だといえます．この戻り値が関数値になります．

関数副プログラムの一例を示します．

```
function square(x)
  implicit none
  real square,x
  square = x*x
end function square
```

このように，function 文で開始し，end function 文で終了します．また，関数名 square を実数型宣言し，引数 x の二乗を代入して終了しています．すなわち，この例は，引数 x に実数型の数値を与えると，x^2 を関数値として与える関数になります．

関数副プログラムを使うことは,“関数名”という変数を使うことであると考えます.このため，作成した関数を使用するルーチンでも関数名を型宣言する必要があります．例えば，上例の square を使うには，

```
program ftest1
   implicit none
   real x,y,square
   x = 5.2
   y = 3.0*square(x+1.0) + 50.5
   print *,x,y
end program ftest1
```

のように，square という関数名を関数副プログラムでの宣言と同じ型で宣言しておかなければなりません．関数副プログラム中で宣言した関数名の型と，その関数を使用するルーチンで宣言した関数名の型が異なる場合には実行時エラーになります．

さて，前回紹介したように，sin や sqrt のような数値計算でよく使う関数はあらかじめ用意されています．これらを“組み込み関数”といいます．組み込み関数は，コンパイラが型を認識しているので，使用に際しての型宣言は不要です．表２に，前回示した関数以外で，よく使うと思われる組み込み関数をいくつか示します．

表２．型変換関数などの組み込み関数

組み込み関数	名　称	引数の数値型	関数値の数値型	関数の意味
real(n)	実数化	整数	実数	実数型へ変換する
int(x)	整数化	実数	整数	整数型に変換する
real(z)	複素数の実部	複素数	実数	z の実部
imag(z)	複素数の虚部	複素数	実数	z の虚部
cmplx(x,y)	複素数化	２個の実数	複素数	x+iy
conjg(z)	共役複素数	複素数	複素数	z の共役複素数
abs(z)	絶対値	複素数	実数	z の絶対値
mod(m,n)	剰余	２個の整数	整数	m を n で割った余り

型変換関数は，サブルーチンの引数に，数値型の異なる変数を与えるときに使います．また，整数化は切り捨て演算を含んでいるので，それを積極的に利用するときにも使います．例えば，正の実数 x の小数点以下を四捨五入した整数は，int(x+0.5)で計算することができます．

最後の剰余を計算する関数 mod は意外によく使います．do 文で繰り返し計算をする場合，“n 回毎に出力する”というように，一定間隔で動作を変えたいことがよくありますが，ここで mod が使えます．例えば，繰り返し計算の中で 10 回に 1 回出力するなら，

```
do m = 1, 10000
  ......
  if (mod(m,10) == 0) print *,m,x,y
  ......
enddo
```

のように書きます．プログラムというのは，基本的に全て計算であり，流れのコントロールも計算結果を使って行わなければなりません．このようなケースで mod を使うのは，パターンの一つとして覚えておくと良いでしょう．

1.7　モジュールを使ったグローバル変数の利用

ローカル変数のおかげで，各ルーチンの独立性は保たれますが，引数でしかルーチン間のデータ受け渡しができないのは不便です．引数は，かっこを使った並びで指定するので，受け渡す変数が多いと書くのが大変です．しかも並びが重要なので，一カ所順番を間違えただけでもエラーが発生します．そもそも引数による受け渡しはアドレス情報を経由したものなので，コール側とサブルーチン側で完全に同じ変数であるという保証はありません．

計算機シミュレーションでは，内容に応じてサブルーチンを作成すると述べましたが，この用途においては，ルーチン間で共用する変数が多数必要です．しかし，共用変数を全て引数にするのは効率が悪く，エラーも発生しやすくなります．

そこで，ルーチン内部でのみ意味を持つ“ローカル変数”に対して，“グローバル変数”が用意されています．グローバル変数とは，どのルーチンから参照しても共通した値を保持している変数のことで，Fortran では，モジュールと use 文の組み合わせで利用可能です[3]．

モジュールとは，変数やサブルーチンなどを集めて一つのパッケージにしたもので，module 文で開始して end module文で終了します．サブルーチンと同じ構造をしていることからわかるように，モジュールの記述は，メインプログラムやサブルーチンと同レベルです．例えば，

```
module data1
   integer nmin,nmax
   real tinitial,amatrix(20,30)
end module data1
```

のように書きます．先頭の module 文で，module の後に指定した文字列（この例では data1）を“モジュール名”といいます．最後の end module文にもモジュール名を指定します．サブルーチンや関数副プログラムと異なり，モジュールはプログラムに記述しただけでは利用することができません．利用するルーチンの先頭に，use 文を使ってモジュール名を指定し，その利用を宣言する必要があります．use 文は以下のような形式です．

```
use モジュール名
```

use 文は implicit 文よりも前に書かなければなりません．モジュールの使用例を以下に示します．

```
module global
   real xaxis,yaxis
end module global

program stest5
   use global
   implicit none
   xaxis = 5.0
   yaxis = 100.0
   call subr
   print *,xaxis,yaxis
end program stest5

subroutine subr
   use global
   implicit none
   print *, xaxis,yaxis
   yaxis = 25.0
end subroutine subr
```

モジュールの中で宣言された変数や配列は，メインプログラムやサブルーチンとは独立して存在したデータ領域にあり，use 文で利用を宣言すれば，どのルーチンからでも参照することができます．このプログラムのメモリイメージを図示すれば，図７のようになります．

モジュールは，名前が異なれば，一つのプログラムに複数作成することも可能であり，一つのルーチンが複数のモジュールを利用することも可能です．ただし，モジュールは use 文で利用を宣言するより前で（プログラムの上方で）定義されている必要があります．このため，通常はこの例のように全てのルーチンより前に記述しておきます．

---

[3] モジュールや use 文は Fortran90 から採用された比較的新しい仕様です．Fortran77 以前では common 文でグローバル変数を実現していました．しかし，common 文には不便な点が多いので本稿では省略します．

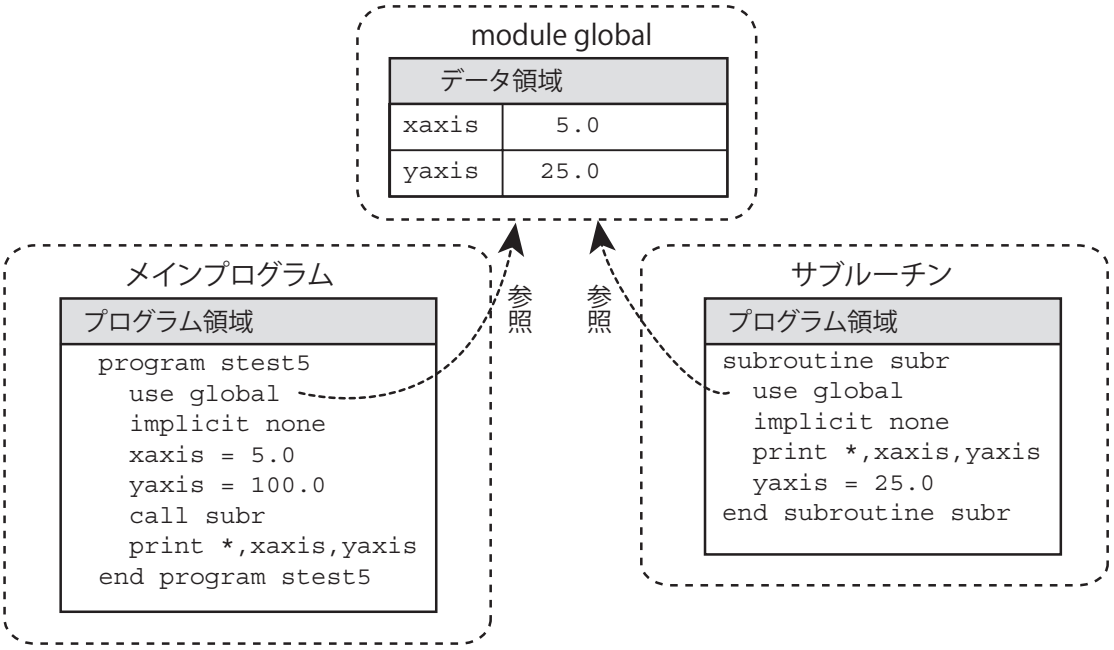


図７．モジュールで宣言されているグローバル変数の参照

モジュールで宣言されている変数は，use 文を書くだけで利用できるという利便性がありますが，ルーチン内で明示的に宣言されていないので，不用意に使ってしまう可能性があります．そこで，次のような only 句を使って，ルーチン内で必要な変数だけを明示することができます．

```
use モジュール名, only : 変数１, 変数２, ...
```

例えば，前の例で，

```
use global, only : yaxsis
```

のように書くと，yaxsis 以外の変数（xaxsis）は，このルーチンでは宣言されていないのと同等です．宣言されていない変数は，ローカル変数として別途宣言することも可能です．

グローバル変数は，多用するとルーチンの独立性が失われてしまいます．基本的にはローカル変数でプログラムを作り，ルーチン間で共用する必要がある変数のみ，グローバルにしてください．また，グローバル変数は広い範囲で存在する可能性があるので，できるだけ意味のある長めの名前を付けます．これは，前回出てきた“大域的に有効な変数名の付け方”です．

2.　データ出力の詳細とデータ入力

2.1　データ出力先の指定

これまでもたびたび登場しましたが，もっとも単純な出力命令は，print 文です．

```
print form, データ１, データ２　...
```

print 文で出力すると，画面（正確には標準出力）にデータが表示されます．*form* は出力形式を指定するためのものですが，とりあえず“*”を書いておけば，データの数値型に応じた標準形式で出力します．例えば，

```
print *, x, y(i), x**2+5, 'abc = ', 10
```

のように，データの位置には，変数や配列を与えても良いし，計算式や定数を与えることも可能です．また，文字列を適当に入れて，データの意味を表示することもできます．

画面ではなく，ファイルに出力するときは write 文を用います．print 文との違いは，出力先を指定する整数値 *nd* と出力形式指定の *form* をかっこで記述することです．

```
write(nd,form) データ１，データ２　...
```

nd を装置番号といいます．装置番号は，出力ファイルを識別する数字で，任意に選ぶことができます．ただし，原則として *nd*≧10 にして下さい．これは，Fortran コンパイラの仕様によって，一桁の数値は予約されている可能性があるからです．例えば，*nd*=6 にすれば print 文と同じ意味になって，画面に表示されます．*nd* には，変数や，整数値になる計算式を与えることもできるので，条件に応じて出力先を変更することも可能です．

nd に適当な整数を与えて write 文を実行すると，“fort.*nd*”という名のファイルに出力されます[4]．例えば，

```
write(20,*) x,y,z
```

と書けば，x,y,z の値が“fort.20”という名前のファイルに出力されます．print 文と同じく，*form* に“*”を与えれば，標準形式で出力します．

2.2　配列の出力，do 型出力

出力文において，データの位置に配列名を書けば，全要素が並んで出力されます．例えば，

```
real a(4)
do i = 1, 4
   a(i) = i
enddo
print *,a
```

というプログラムを実行すると，

```
 1.00000000000000   2.00000000000000   3.00000000000000   4.00000000000000
```

のように，配列要素が先頭から順に並んで出力されます．このとき，配列要素が多いと適当に改行を入れて出力されます．よって，この方法は配列要素が少ないとき以外はあまりお勧めできません．例えば，２次元配列を，

```
real a(3,3)
do j = 1, 3
   do i = 1, 3
      a(i,j) = i*j
   enddo
enddo
print *,a
```

のように出力すると，a(1,1),a(2,1),a(3,1),a(1,2),...,a(3,3) という並びで 9 個の要素が出力されます．そこで，次のように do 文で出力する方がいいでしょう．

```
do j = 1, 3
   do i = 1, 3
      print *,a(i,j)
   enddo
enddo
```

しかし，print 文や write 文は，１文ごとに改行をするので，複数の print 文を使って，横に続けて書くことはできません．よって，この例の出力では全部で 9 行必要です．行数を減らすために，1 行に行列の 1 行を出力するには，次のように書かなければなりません．

```
do i = 1, 3
   print *,a(i,1),a(i,2),a(i,3)
enddo
```

---

[4] “fort”の部分はコンパイラに依存しますが，多くは fort のようです．また，コンパイラによっては環境変数で指示することにより，必要な装置番号に任意の名前のファイルを割り当てることも可能です．

しかし，この例のように列の数がわかっているときは良いのですが，列数が多いときや変数で指定したいときには不便です．そこで do 型出力が用意されています．do 型出力とは，

```
（データ１,データ２,...,整数型変数=初期値,終了値）
```

のような，かっこで囲んだ形式です．これをデータの位置に記述すれば，do 文のように，整数型変数が初期値から終了値まで１づつ増えていき，その変数で指定されるデータが順に出力されます．do 型出力を複数並べたり，通常のデータと並べて書くこともできます．例えば，

```
print *,(a(i,j),j=1,3)
```

は，さきほどの“print *,a(i,1),a(i,2),a(i,3)”と書くことと同等です．また，

```
print *,5*x,(a(i),b(i),i=1,3)
```

は，“print *,5*x,a(1),b(1),a(2),b(2),a(3),b(3)”と書くことと同等です．

do 型出力は多重にすることもできます．例えば，

```
print *,((a(i,j),j=1,3),i=1,3)
```

は，“print *,a(1,1),a(1,2),a(1,3),a(2,1),a(2,2),a(2,3),a(3,1),a(3,2),a(3,3)”と書くことと同等です．この例のように，多重のときには内側の繰り返し変数が先に進みます．

なお，do 型出力の整数型変数は，do 文のカウンタ変数に相当します．このため，do 型出力の入っている出力文を do ブロックの中に入れるときには，カウンタ変数が重複しないようにしなければなりません．

2.3　format による出力形式の指定

標準出力形式（“*”指定）で実数を画面に出力すると，有効数字 15 桁の数字を使って表示されます．このため，あまり多くのデータを横に並べることができないし，結果を見るだけなら，それほどの有効数字は必要ありません．また，標準形式の出力には 1 行の出力文字数に制限があるため，出力数が制限を超えると自動的に改行されてしまいます．このため，同じ出力を繰り返すと，行ごとに小数点の位置が違うこともあり，大量に出力するのには向きません．

これらの問題は，出力形式（format）を指定することで解決できます．これまで“*”を書いていた *form* の位置に format を指定すれば，小数点以下の桁数を小さくしたり，必要に応じて数字と数字の間にスペースを空けたり，改行を入れたりすることができます．また，自動改行されることがないので，幅広く出力することも可能です．

format の指定方法は２通りあります．まず，format 文による指定です．一例を示します．

```
      real x,y
      integer n
      x = 1.5
      y = 0.03
      n = 100
      print 600,x,y,n
  600 format('  x = ',f10.5,'    y = ',es12.5,'    n = ',i10)
```

最後の文が format 文です．format 文は，サブルーチンの引数のように，出力形式の指定をリストにしてかっこで囲み，先頭に文番号を付けます．この文番号を print 文や write 文の *form* に指定すれば，その format にしたがって出力データが整形されます．この例では，600 が *form* 指定です．文番号は重複できないので，ルーチン内では format ごとに異なる数字をつけなければなりません[5]．

複数の print 文や write 文が同じ format 文を指定するのは可能です．例えば，次のように共用することができます．

---

[5] 文番号の付け方に決まったルールはありませんが，“format 用である”という意味をどこかに入れておいた方が良いでしょう．この例で 600 を使っているのは，“6”が昔の出力装置番号だったころの習慣です．

```
      real x,y,u,v
      integer n,k
       ......
      print 600,x,y,n
      write(20,600) u,v,k
  600 format('  x = ',f10.5,'   y = ',es12.5,'   n = ',i10)
```

format を指定するもう一つの方法は，文字列を使って *form* の位置に直接 format の内容を記述することです．例えば，上記の format を print 文や write 文に埋め込んで，

```
      print "(' x = ',f10.5,'   y = ',es12.5,'   n = ',i10)",x,y,n
      wriye(20,"(' x = ',f10.5,'   y = ',es12.5,'   n = ',i10)") u,v,k
```

のように書くことができます．このとき，“format”の文字は不要ですが，両端のかっこは必要です．なお，Fortran での文字列は「'」で囲むのが基本ですが，「"」も使えるので，format 内部に「'」が入っているときには「"」で囲みます．

format を出力文中に書き込む方法は，文番号を必要としない点は良いのですが，出力文が長くなるし，同じ format を何度も使うときには不便です．そこで，文字列変数を利用して書く方法があります．文字列変数の利用については次回説明します．

出力形式の指定方法を上記の format を例にして説明しましょう．まず，format 中の文字列はそのまま出力されるので，必要に応じて適宜挿入します．この例では，' x ='や' y ='は，スペースも含めてそのまま出力されます．

次に，出力文中のデータ値の並びに対し，それぞれの出力形式を指定する“編集記述子”を選んで，前から順に記述します．この例では，f10.5，es12.5，i10，が編集記述子で，print 文の並びに対し，

```
      print 600,       x        ,       y        ,       n
                       ↓                ↓                ↓
  600 format('  x = ',f10.5,'   y = ',es12.5,'   n = ',i10)
```

という対応で出力形式を指定しています．文字列と，編集記述子で指定したデータ値は，その並び順に出力されます．よって，この print 文を実行したときの出力は以下のようになります．

```
      x =    1.50000    y =  3.00000e-02    n =         100
```

出力形式を指定する編集記述子の主要なものを表３に示します．データの数値型に応じた編集記述子を使用しないと正しい値が出力されないので注意して下さい．なお，この表で斜体文字（w, m, d, r）は整定数で指定します．

表３．主要な編集記述子

編集記述子	数値型	編集の意味
Iw	整数	幅 w で整数を出力する
I$w.m$	整数	幅 w で整数を出力する 出力整数の桁が m より小さいときには，先頭に 0 を補う（$w \geqq m$）
F$w.d$	実数	幅 w で実数を固定小数点形式で出力する d は小数点以下の桁数（$w \geqq d$+3）
E$w.d$	実数	幅 w で実数を浮動小数点形式で出力する d は小数点以下の桁数（$w \geqq d$+8） 仮数部の１桁目は 0 になる
G$w.d$	実数	幅 w で実数を出力する 指数部の大きさに応じて F 編集と E 編集を切り替える

表３（続き）．主要な編集記述子

編集記述子	数値型	編集の意味
ES$w.d$	実数	幅 w で実数を浮動小数点形式で出力する 0 以外の数値を出力すると，仮数部の 1 桁目は 1 から 9 になる
EN$w.d$	実数	幅 w で実数を浮動小数点形式で出力する 0 以外の数値を出力すると，仮数部の絶対値は 1 以上 1000 未満になり， 指数部は 3 で割り切れる数になる
A	文字列	文字列をそれ自身の長さの幅で出力する
Aw	文字列	幅 w で文字列を出力する
/	なし	改行する
rX	なし	r 個スペースを挿入する

最後の二つ（改行とスペースの挿入）は，出力動作を記述するものなので，文字列と同様に，出力文中のデータとの対応はありません．また，ここでは編集指定の文字（F や ES など）を指定数（w など）と区別するために大文字で書きましたが，小文字でも同じ意味です．例えば，i10 は整数型値を幅 10 文字で出力することを意味し，f10.5 は実数型値を幅 10 文字，小数点以下 5 桁で出力することを意味しています．このため，

```
real x,y
integer m,n
x = 1.5
y = 0.03
m = 100
n = 10
print "(f10.5,f10.5,i10,i10.5)",x,y,m,n
```

というプログラムの出力は，

```
    1.50000   0.03000       100     00010
+----+----+----+----+----+----+----+----+
```

となります．２行目の目盛りは位置を確認するために書いたものですが，10 文字の中に，右寄りで出力されているのがわかります．なお，出力文字数が指定の幅 w を越えると，「*****」のように「*」が w 個出力されます．

E 編集を使って実数を浮動小数点形式で出力すると，小数点の前が 0 になります．例えば，

```
real x,y
x = 1.5
y = 3.14e10
print "(e15.5,e15.5)",x,y
```

の出力結果は，

```
    0.15000e+01     0.31400e+11
```

となります．これでは感覚的にわかりにくいし，表示字数が 1 個無駄になるので，ES 編集や EN 編集を使う方が良いでしょう．例えば，

```
real x
x = 3.14e10
print "(es15.5,en15.5)",x,x
```

の出力結果は，

```
    3.14000e+10    31.40000e+09
```

となります．

各編集記述子と出力の数値は１対１対応にしなければならないので，配列を出力するときには出力要素数だけ編集記述子を書かなければなりません．このとき，同じ編集記述子を繰り返すならば，編集記述子の前に整数 *r* を付加して，*r* 回反復するという指定ができます．例えば，“3f10.5”は“f10.5,f10.5,f10.5”と書くことと同等です．

さらに，実数，整数，実数，整数のような繰り返しのときには，かっこでくくって反復指定をすることができます．例えば，次のように書くことができます．

```
real x,y
integer m,n
x = 1.5
y = 0.03
m = 5
n = 100
print "(2(f10.5,3x,i10))",x,m,y,n
```

この print 文の format は，“f10.5,3x,i10,f10.5,3x,i10”と書くことと同等です．

なお，format 中の編集記述子の数よりも出力文のデータ値の方が多い場合には，編集記述子の数だけ出力した後で改行し，同じ format を再度使って残りのデータ値を出力します．文字列が入っていれば，文字列も再度出力されます．

逆に，format 中の編集記述子の数よりも出力文のデータ値の方が少ない場合には，指定したデータ値を出力した段階で終了し，あまった編集記述子は無視されます．無視された記述子以降は文字列等が入っていても全て無視されます．配列を出力するときなどは，反復指定に大きめの数値を与えておくことができます．

2.4　書式なし write 文

write 文は，次のように *form* を省略して，装置番号のみの指定で出力することができます．これを“書式なし write 文”といいます．

```
write(nd) データ１，データ２　...
```

書式なし write 文で出力すると，データ形式が“バイナリ形式”になります．これに対し，これまで説明した *form* のある出力文で出力したときのデータ形式は“テキスト形式”です．元来，計算機内部の数値は 2 進数データであり，“10”とか，“1.000e20”とかの文字ではありません．このため，画面に表示するときには“2 進数→10 進数文字表現”というデータ変換を行って表示しています．この表示形式がテキスト形式です．

しかし，2 進→10 進変換には時間がかかる上に，文字に変換することでデータ量が増えます．これは，文字データが 1 文字あたり 1byte 必要だからです．例えば，実数を有効数字 15 桁で文字表示するには 15byte 以上必要です．しかし，倍精度実数の 2 進数表現は 8byte ですから，そのままで保存すれば 8 文字分で済みます．この内部の 2 進数データがバイナリ形式です．大量のデータをディスクに保存するときには，バイナリ形式による保存が有効です．

ただし，Fortran の書式なし write 文を使って出力したバイナリ形式ファイルは，2.5.1 節で説明する書式なし read 文で読まなければならないので，あくまでも Fortran プログラムから入力して利用するのが基本です．他の言語で作成したプログラムで使う場合や，データ解析ソフトを使って解析するためのファイルを作成するときは，テキスト形式で出力した方が良いと思います．

2.5　データの入力方法

プログラムが実行しているとき，そのプログラム中の指定した変数に外部からデータを代入することを“入力する”といいます．計算条件を設定するための変数にデータを入力できるようにしておけば，プログラムの実行を開始してから条件を設定して，それに応じた計算をさせることができます．これにより，プログラムはその動作だけを利用する“ブラックボックス”になり，コンパイルしたプログラムを“アプリケーション”として他の人に提供することも可能です．

2.5.1　入力文の一般型

データ入力にはread文を用います.

```
read(nd,*) 変数１，変数２　...
```

*nd*は装置番号で,*nd*に適当な整数を与えると“fort.*nd*”という名のファイルから入力します[6].装置番号に関する条件や注意事項は出力の場合と同じで，原則として *nd*≧10 にして下さい．出力ファイルと違うのはfort.*nd*という名のファイルが存在していなければエラーになるので，あらかじめ用意しておかなければならないことです.なお,“*”の位置には,write文のようにformatによる書式指定ができますが，あまり使うことがないので説明は省略します.

例えば，fort.30という名前のファイルに,

```
5.2　　1.5　　3
```

と書いて保存しておき，プログラム中に,

```
read(30,*) x,y,z
```

と書けば，このread文実行後，x=5.2，y=1.5，z=3.0となって実行が継続します．入力ファイルに改行が入っていても，read文の変数入力が完了するまで読み込みを続けるので，fort.30の入力数値は次のように３行に分けて書くこともできます.

```
5.2
1.5
3
```

read文実行時に，ファイルが存在しなかったり，データが足らない場合には，エラーになってプログラムが強制終了します．逆に，ファイル中の数値が多い場合には，あまった数値は無視されます.

read文の変数の位置には，配列名や，do型出力と同型のdo型入力を書くこともできます．これらは，出力と入力という方向が異なりますが，入力要素数や繰り返しの意味は同じです.

ファイルではなく，キーボード（正確には標準入力）を使って入力したいときには，以下のように書きます.

```
read *, 変数１，変数２　...
```

この文を実行すると，プログラムの実行が一時停止し，キーボードからの数値入力を待つ状態になります．そこで，適切な数値をキーボードから入力すると，その数値を所定の変数に代入した後，実行が再開します．このため，read文のタイミングを考慮して数値を入力しなければ，いつまでたっても停止したままです.

read文でも装置番号のみを指定することができます.これを“書式なしread文”といいます.

```
read(nd) 変数１，変数２　...
```

書式なしread文で入力する場合は，入力データが“バイナリ形式”だと仮定されるので，書式なしwrite文で作ったファイルを使う必要があります．また，出力したときの数値と同じ数値型の変数を同じ順番で並べる必要があります．例えば,

```
real x,y
integer n
x = 10.0
y = 100.0
n = 10
write(20) x,n,y
```

[6] 出力ファイルと同様，fortの部分はコンパイラに依存します．環境変数でファイル名が変更できるコンパイラもあります．ちなみに，標準出力の装置番号は6でしたが，標準入力の装置番号は5です.

というプログラムで作成された fort.20 というファイルから入力するには，

```
      real x,y
      integer n
      read(20) x,n,y
```

のように書かなければなりません．この場合，x も n も同じ 10 だからと思って，

```
      read(20) n,x,y
```

と入力すると，n と x には，正しい値が代入されません．

書式なし write 文１行で書き込まれたデータ数よりも入力データが少ないのは問題ありません．このとき入力しなかったデータは読み飛ばしたことに相当します．逆に言えば，２回の read 文を使って分けて入力することはできません．例えば，上記のデータを，

```
      read(20) x,n
      read(20) y
```

のように２行の read 文に分けて入力しようとした場合，x と n には正常な値が代入されますが，２行目の read 文実行時に，入力データがないというエラーで強制終了します．

2.5.2　入力時のエラー処理

ファイルからデータを入力するとき，要求したファイルが存在しなかったり，書き込まれたデータ数より多くのデータを入力しようとすれば，実行時エラーになって，プログラムは強制終了します．これを防ぐため，read 文中にエラー処理指定を入れることができます．

```
      read(nd,*,err=num) 変数１，変数２　...　　！テキスト形式入力
      read(nd,err=num) 変数１，変数２　...　　！バイナリ形式入力
```

ここで，*num* には文番号を与えます．この read 文を実行したとき，入力エラーが起こると *num* で指定した文番号の行へジャンプします．例えば，

```
      do k = 1, 100
        read(10,*,err=999) x,y,z
        .......
      enddo
  999  x = 100
```

と書けば，エラーが起こると文番号 999 の行にジャンプして，その行から処理を継続します．

もし“ファイルの終了”，すなわち，データを入力するときに，それ以上入っていなかった，という場合を検知するだけなら，err=*num* の代わりに end=*num* と書くこともできます．データの出力回数が不明のときには，err か end 指定を入れておき，データ終了時点で次の処理に進むようなプログラムにしておくと良いでしょう．

2.5.3　namelist を用いた入力

便利なテキスト形式入力として，namelist を用いる方法があります．例えば，

```
      read(10,*) x,y,n
```

という入力文では，入力ファイル fort.10 を，

```
      10.0    1.e10    100
```

のように作成しますが，作成するためには入力変数の対応を常に憶えておかなければなりません．必要なデータを全部書き込まなければならないし，順番を間違えることもできません．

これに対し，namelist 入力文では，入力データを“変数＝データ”という代入形で記述するので，どの変数に代入するかを入力ファイルの中で明示することができます．

namelist 入力文を使うときは，まず入力する可能性のある変数や配列名を namelist 文で登録します．namelist 文は次のような形式です．

```
namelist /ネームリスト名/ 変数１，変数２　...
```

namelist 文は非実行文なので，全ての実行文より前に書かなければなりません．また，変数や配列名の登録だけなので，型宣言は別途必要です．例えば，

```
real x,y,a(10)
integer n
namelist /option/ x,y,n,a
```

と書きます．ローカル変数だけではなく，use 文で指定されたグローバル変数も登録可能です．

この namelist に登録された変数に対し，入力文は，

```
read(nd,ネームリスト名)
```

だけです．変数の指定は必要ありません．必要ならば，文番号 *num* を使って，

```
read(nd,ネームリスト名,err=num)
```

のように，err=*num* や end=*num* を追加することで，エラー発生や終了時の処理をすることも可能です．

namelist 入力に対する入力ファイルは次の形式で用意します．変数の順番は任意です．

```
&ネームリスト名
  変数１＝データ１，変数２＝データ２，...
/
```

“&ネームリスト名”から“/”までが namelist 入力文１回で入力されるデータです．例えば上例のようにネームリスト名が option のときには，

```
&option
  x=10.0, y=1.e10, n=100
/
```

のようにファイルに書いておきます．次のように１行づつ書くこともできます．

```
&option
  x=10.0
  y=1.e10
  n=100
/
```

namelist 入力にはもう一つ利点があります．それは，必ずしも登録された変数全部を入力ファイルに記述する必要がないことです．記述しなかった変数には，namelist 入力文の実行前までに代入されていた値がそのまま残ります．このため，あらかじめ全ての登録変数にデフォルト値を代入しておけば，変更したい変数だけ入力ファイルに記述することができます．例えば，

```
real x,y,a(10)
integer n,i
namelist /option/ x,y,n,a
x = 100.0
y = 100.e10
n = 0
do i = 1, 10
   a(i) = i
enddo
read(10,option)
```

のようにプログラムを書いたとします．入力ファイルとして fort.10 という名のファイルに，

```
&option  x=10.0, a(3)=5.0 /
```

と書き込んでおけば，read 文実行後，x は変更されますが，y や n はそのままです．配列 a の場

合には，代入された要素 a(3)のみが変更されます.

なお, namelist に登録された変数の内容は, 次の namelist 出力文で出力することもできます.

```
write(nd,ネームリスト名)
```

この場合，全登録変数が“変数＝データ値”という形で出力されます．もっとも，namelist 出力文を実行すると，登録された全変数のデータが標準形式で出力されるので，変数が多いと煩雑です．取りあえず値を確認したいとき以外は，あまり使わない方が良いと思います.

2.6　ファイルのオープンとクローズ

write 文や read 文を使ってファイルから入出力をする場合, 何も指定がなければ, 装置番号 *nd* を付加した“fort.*nd*”という名のファイルを使用します．これに対し，任意のファイルを使いたいときには，open 文を使って入出力文の実行前にファイル名を指定しておきます．これを“ファイルをオープンする”といいます．open 文は以下の形式です.

```
open(nd,file=name [,form=format][,status=stat][,err=num])
```

[ ]の部分は省略可能です．それぞれの記述（制御指定子）の意味を表４に示します.

表４．open 文の制御指定子の意味

指定子	指定情報	指定子の意味と注意
nd	装置番号	整数を与える（整数変数や整数式を与えることも可能） オープンした後，read 文や write 文の装置番号として使う
name	ファイル名	文字列で指定する（文字変数も可能） ファイル名は大文字・小文字を正しく指定する必要がある
format	ファイル形式	文字列で指定する 省略するとテキスト形式の入出力が仮定される バイナリ形式の入出力を使うときは「'unformatted'」を指定する
stat	ファイル情報	文字列で指定する 既存のファイルを使うときは「'old'」を，存在しないファイルを使うときは「'new'」を指定する 条件に合わないとエラーが発生する
num	文番号	エラーのときにジャンプする行の文番号を指定する 省略すると，エラーが起きたときにはプログラムが強制終了する

例えば，装置番号 10 のテキスト形式ファイルを“test.out”という名にするときは,

```
open(10,file='test.out')
```

と書きます．また，装置番号 30 のバイナリ形式ファイルを“test.dat”という名にするときは,

```
open(30,file='test.dat',form='unformatted')
```

と書きます．なお，存在するファイルを指定して write 文で書き込むと，それまで書き込まれていたデータが上書きされて消えるので注意が必要です. これを回避したいときには, status='new' と err=*num* を指定して，ファイルが存在する場合の処理を用意しておく必要があります.

status=*stat* を省略して open 文を実行すると，指定したファイルが存在しなければ，その名前のファイルが新たに作成されます．このため，read 文で入力するためのファイルをオープンする際にファイルが存在しないと，不必要な空のファイルが作成されてしまいます．これを防ぐために，ファイルが入力用のときは，status='old'を指定して，その存在をチェックした方が良いでしょう.

例えば, バイナリ形式ファイル“test.inp”を入力ファイルとして装置番号 20 に指定するには,

```
open(20,file='test.inp',form='unformatted',status='old',err=999)
```

のように書きます．この open 文には，err=999 というエラー処理が書かれているので，オープンした時点でファイルが存在しなければ，文番号 999 の行へジャンプします．

ファイルへ出力する場合，実行プログラムでの出力命令のタイミングと実際にディスクに書き込まれるタイミングは必ずしも一致していません．これは入出力ハードウェアを効率よく運用するために，一時記憶領域への読み書きが介在するためです．このため，確実に書き込みを完了させたいときにはファイルをクローズします．クローズは，次の close 文で行います．

```
close(nd)
```

nd はクローズするファイルの装置番号です．close 文を実行すると，その時点までに装置番号 *nd* に出力した全てのデータがディスクに書き込まれます．もっとも，プログラムが正常に終了すれば，全てのファイルが自動的にクローズされるので，通常は close 文を書かなくても問題ありません．

ファイルをクローズすると，装置番号 *nd* と open 文で指定したファイル名の関係は途切れます．このため，同じ装置番号に，別のファイルを新たに指定してオープンすることも可能です．また，同じ名前のファイルを再度オープンすることもできます．ただし，再オープンしてもそれ以前の読み書きの継続にはならず，ファイルの先頭に戻って読み書きをします．すなわち，“巻き戻し”をすることになります．ファイルを巻き戻すと，read 文による入力は一からやり直すことになり，write 文による出力はファイルの先頭から書き直します．このため，巻き戻してから write 文で出力すると，それ以前に書き込んだデータは消去されます．

なお，巻き戻すのが目的であれば，rewind 文を使うことで，クローズせずともファイルを巻き戻すことができます．rewind 文とは，次のような文です．

```
rewind(nd)
```

nd は再度最初から読み書きするファイルの装置番号です．巻き戻しを利用すれば，まず write 文を使ってファイルにデータを出力しておき，次にそれを巻き戻して，read 文を使ってそのデータを入力して使うことも可能です．

まとめと次回の予告

今回は，プログラムの拡大に伴って必要となるサブルーチンの使い方と入出力の詳細についてお話ししました．数値計算や計算機シミュレーションに必要な Fortran の基本的文法の説明は，これでほぼ完了です．これまで説明した文法を使って，どんどんプログラムを書いて下さい．たくさん書いて経験を積めば，どういう手順で計算機を動作させれば効率が上がるかがわかってくるので，さらにスマートなプログラムが書けるようになります．

さて，ここまでのプログラミングでは数値を個別に計算するのが基本でした．配列は複数のデータを一括して取り扱うことができますが，それを使って計算するときは，あくまでも要素ごとの計算を繰り返す形で記述しました．これに対し，最近の Fortran には，数値の集合を一つのデータ型としてとらえ，その集合間の演算を記述することで，複雑な処理をシンプルに表現する仕組みが導入されています．その一つは，“文字列”です．文字の集合を“文字列”というデータ型で取り扱い，文字列をつないだり，比較したりするという演算ができます．もう一つは“配列計算”です．実数や整数の配列を一つのデータとして取り扱い，do 文を書かなくても，配列間の演算を記述できます．

最終回である次回は，文字列や配列といった集合データ型とその演算について説明します．また，比較的新しい文法の中から，筆者が使って便利だと思われるものをいくつか紹介し，それらを利用してプログラムのメンテナンスを楽にするテクニックを説明したいと思います．

参考文献

[1] 入門 Fortran90 実践プログラミング，東田幸樹・山本芳人・熊沢友信，ソフトバンク，1994 年

［お知らせ］

大規模科学計算システムホームページの更新について

共同研究支援係
共同利用支援係
スーパーコンピューティング研究部

大規模科学計算システムのホームページを初めて利用する人にも、分かりやすいホームページにするために、平成 24 年 6 月からコンテンツ・デザインを、全面的に見直しました。是非、ご覧いただき今後の利用にご活用ください。

なお、ホームページアドレスは　http://www.ss.isc.tohoku.ac.jp/　です。

１．コンテンツの構成

従来のページは、コンテンツタイトルが羅列されている、ページ階層が深すぎる等の理由で目的の情報を探しにくいというご意見がありました。そこで今回は、センターを利用する際に必要な情報をすぐ直観的に探し出せるように、コンテンツ構成を全面的に見直しました。トップページは、閲覧頻度の高いコンテンツタイトルをグループ分けしたレイアウトとし（図１）、サブページで詳細な情報を提供する構成にしました。また、各ページにおいてアイコンを活用し、分かりやすく親しみやすいページになるよう心がけました。

「トップページのコンテンツタイトル」
- ○　スーパーコンピュータ（SX-9）
- ○　並列コンピュータ（Express5800）
- ○　アプリケーションサービス
- ○　共同研究
- ○　利用案内・申請方法
  （学術(一般)利用、民間企業利用、
  各種申請用紙）
- ○　講習会案内
- ○　利用相談
- ○　成果報告
- ○　センター施設利用



図１トップページのコンテンツアイコン

２．主な変更点

・コンテンツタイトルを文字だけでなく、直感的に分かるようにアイコン化し、さらにスマートフォン等でもタッチしやすいサイズに調整しました。

・スーパーコンピュータ、並列コンピュータの利用ガイドを、HTML 形式でも閲覧できるようにしました。従来の PDF ファイルの利用ガイドも、引き続きご利用出来ます。

・学術(一般)利用と民間企業利用で、利用制度が異なりますので、別ページとしました。

・講習会で使用した資料と、サンプルプログラムをダウンロード出来るようにしました。

・利用者の方から頂いたご質問を FAQ 形式で掲載しました。

［報　告］

サイバーサイエンスセンター　オープンキャンパス報告

小松 一彦
スーパーコンピューティング研究部

東北大学では，2012年7月30日から31日にかけて，入学を希望・考慮されている皆さんや市民の方々に，日頃の教育・研究活動や大学施設を広く一般に公開し，東北大学への興味・関心を深めてもらえるよう，オープンキャンパスを開催いたしました．当センターでは，全国の研究者の大規模科学技術計算を支えるスーパーコンピュータや並列コンピュータ，東北のネットワークを支えるネットワーク機器，大規模計算機の歴史を辿ることができるコンピュータ博物館を公開しました．また，ネットワーク研究部，スーパーコンピューティング研究部，先端情報技術研究部の研究活動や最新の研究成果の紹介を行いました．

近年のスーパーコンピュータへの関心の高まりもあり，若い学生達だけでなく一般の方も訪れ，今年度は1000人以上の方々にご来場いただきました．見学者の方達には，スーパーコンピュータの用途や重要性の説明を行い，スーパーコンピュータがどのように科学技術の発展を支えているか，そして，それらの科学技術がどのように私達の生活に活かされているかを知っていただきました．また，普段なかなか見ることが難しいスーパーコンピュータや並列コンピュータをガラス越しではなく直接見学することで，そのサイズの大きさ，空気を流すためのファンの音の大きさ，コンピュータを冷却するための冷風などを肌身で感じていただきました．スーパーコンピュータを少しでも身近に感じていただく良い機会になったと思います．また，博物館の展示では，歴史的な価値のある歴代のスーパーコンピュータや昔のコンピュータの部品など，コンピュータの進化について興味深く見学していただきました．研究開発部の展示では，研究室の研究活動に関する話だけでなく，大学生活や大学での講義，入試対策などについても話をして，大学院生や学生と交流しておりました．来年度のオープンキャンパスも一般に公開する予定ですので，みなさまのお越しをお待ちしております．






［展示室便り⑥］

スーパーコンピュータ SX シリーズ

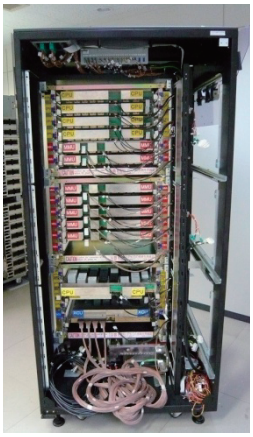
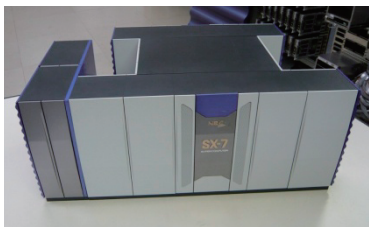
今回は、日本電気（株）製のスーパーコンピュータ SX シリーズです。センターで使用したスーパーコンピュータ（以下スパコン）は、このシリーズの SX-1、SX-2、SX-3R、SX-4、SX-7(1986 年～2007 年)です。これらの計算機はベクトルマシンといわれ当時のメインフレームのコンピュータの 40～100 倍の演算性能を有しました。表 1「スーパーコンピュータの変遷」は、センターで導入したスパコンの約 20 年間の演算処理能力と主記憶容量を表しスパコンの驚異的な性能向上がお分かりいただけると思います。

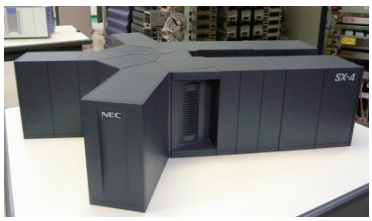
機種	演算性能 GFLOPS	メモリ容量 GB
SX-1　1986-1988 年	0.6	0.1
SX-2N　1989-1993 年	1.1	0.3
SX-3R　1994-1997 年	25.6	4.0
SX-4　1998-2002 年	256.0	32.0
SX-7　2003-2007 年	2119.0	1920.0

表 1　スーパーコンピュータの変遷

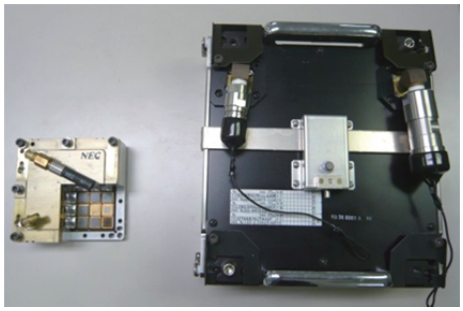
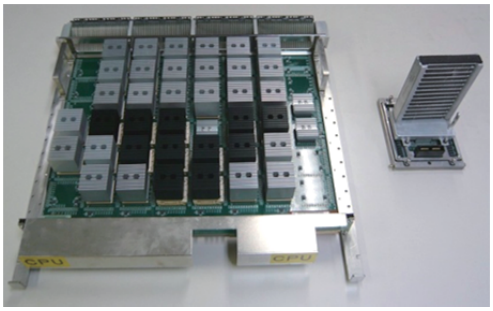
利用形態は、SX-1、SX-2N は前回紹介した ACOS システムのバックエンドプロセッサとしてバッチ処理でサービスを行いました。SX-3R からは オペレーティングシステム(OS)が UNIX となり、利用者の方は会話処理で直接スパコンを利用することができるようになりました。展示品 1 は SX-7、 SX-4 筐体の一部分（ふ線筐体）で CPU とメモリが組み込まれているところです。ここでは、スパコン筐体内での CPU、メモリ、各種のケーブルが実装されていた状態を見ることが出来ます。ミニモックアップは 1 台（1 ノード）の全景を現しています。SX-7 は 7 ノード導入され、1 ノードは 32 個の CPU と 256GB のメモリで構成されていました。



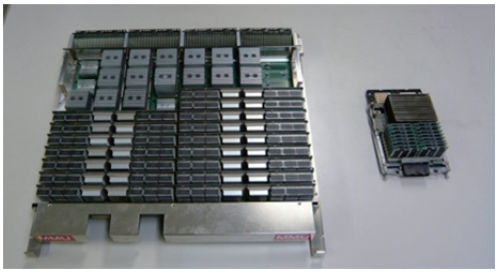


展示品 1　SX-7 筐体（左）、SX-4 筐体(中)、ミニモックアップ（右 2 枚）

展示品２　左から順に SX-2N、SX-3R、SX-4、SX-7 の１CPU

展示品２は各スパコンの 1CPU です。SX-2N、SX-3R、SX-4、SX-7 と大きさを比較し易いよう並べてみました。SX-3R まではバイポーラテクノロジ、SX-4 からは CMOS テクノロジが採用されました。１CPU の演算性能は、1.14GFLOS、6.4GFLOPS、２GFLOPS、8.83GFLOPS です。



展示品３　SX-4、SX-7 のメモリカード

展示品３は SX-4、SX-7 のメモリカード（ユニット）です。各ボードの記憶容量は、256MB と２GB です。年代と共に集積度が上がってきます。



展示品４　高速化推進活動報告

展示品４は「高速化推進研究活動報告」です。報告書では、プログラムの高速化技法や利用者プログラムの高速化事例が紹介されています。2001 年から発行され、現在は第５号まであります。この報告書作成の経緯は、初号作成時、当時の根元義章センター長が「高速化推進研究活動報告の刊行にあたって」の中で以下のように書かれています。

本センターが1998年1月に導入したスーパーコンピュータSX-4/128H4はベクトル型スーパーコンピュータとしては、当時世界最大の演算処理能力を有しており、メモリ共有機構やベクトル処理機構、コンパイラの自動並列化や自動ベクトル化等で世界的にも優れた特徴を有していた。しかし、大学の利用者は、より規模の大きい3次元シミュレーションを行うことを切望しており、計算方法プログラミングを工夫する必要があった。そこで導入決定直後の1997年9月より利用者、本センター、日本電気（株）の3者により高速化推進のための研究会を立ち上げ、約3年にわたって共同研究を進めてきた。本報告はその成果をまとめたものである。

[Web 版大規模科学計算システムニュース]より

大規模科学計算システムニュースに掲載された記事の一部を転載しています。 http://www.ss.isc.tohoku.ac.jp/tayori/

反応経路自動探索プログラム GRRM11 のサービス開始について (No.137)

GRRM は、2002 年に東北大学（教授：大野公一、修士１年：前田理、当時）で制作が開始され、その後開発が進められて、2011 年には GRRM11（開発者： 前田理、長田有人、諸熊奎治、大野公一）として広く利用されるようになりました。GRRM には、以下のような特長があります。

・g09 や g03 プログラムなどを用いた非経験的量子化学計算に基づいて、各化学式で表される構造や反応経路を自動的に探索します。
・平衡構造から出発して、その周囲に存在する反応経路を、ポテンシャルの非調和下方歪みを検出して、系統的に調べ上げる超球面探索アルゴリズムが搭載されており、従来不可能であった５原子以上の反応経路自動探索を行うことができます。

2012 年 8 月 7 日（火）よりサービスを開始しますのでご利用下さい。

サービス開始日　：　2012 年 8 月 7 日 （火）
ホスト名　　：　gen（並列コンピュータ）
バージョン　　：　11.01

利用方法についての詳細は開発元 Web ページ、および当センターの GRRM11 紹介 Web ページをご参照下さい。

【開発者 Web ページ】
http://grrm.chem.tohoku.ac.jp/SRPS/GRRM.HTM

【当センターGRRM11 紹介 Web ページ】
http://www.ss.isc.tohoku.ac.jp/grrm11.html

（共同利用支援係）

コンパイラアップデートのお知らせ　(No.139)

■2012 年 8 月 26 日にスーパーコンピュータ用のコンパイラ、ライブラリを下記のバージョンへアップデートします。

FORTRAN90/SX　　Rev.451
C++/SX　　Rev.093
MPI ライブラリ　　library 8.0.16
■コンパイル方法は下記となります。
なお、コンパイルコマンドに変更はありません。
Fortran コマンド名：sxf90
C/C++コマンド名：sxcc
オプションの説明は、man コマンドでも参照できます。

MPI を利用する場合
Fortran コマンド名:sxmpif90
C/C++コマンド名:sxmpicc
オプションの説明は、man コマンドでも参照できます。

■FORTRAN90/SX
詳細についてはリリースメモをご覧ください。
Rev451 リリースメモ

■C++/SX について
C++/SX の新機能等については下記資料をご覧ください。
Rev093 リリースメモ
Rev.093: 新機能のご紹介
互換性留意事項説明
C/C++言語プリプロセッサ利用の手引

■MPI/SX について
MPI-2.2 使用をサポートしました。
主な新規機能および機能強化は下記のとおりです。

・集団通信新規手続き
　MPI_REDUCE_SCATTER_BLOCK　(全プロセス同一サイズの REDUCE_SCATTER)
　MPI_REDUCE_LOCAL　　(自プロセス内リダクション演算実行)
　MPI_OP_COMMUTATIVE　　(リダクション操作の可換性問合せ)

・集団通信機能強化

下記手続きで MPI_IN_PLACE 引数をサポート。

MPI_ALLTOALL

MPI_ALLTOALLV

MPI_ALLTOALLW

MPI_EXSCAN

・プロセストポロジー新規手続き

Distributed Graph トポロジー生成および問合せ手続きの追加。

MPI_DIST_GRAPH_CREATE

MPI_DIST_GRAPH_CREATE_ADJACENT

MPI_DIST_GRAPH_NEIGHBORS

MPI_DIST_GRAPH_NEIGHBORS_COUNT

・新規データ型

C99 規格の新規データ型および MPI_Aint、MPI_Offset 型に対応した MPI データ型の追加。

・MPI トレース機能強化

詳細につきましては下記のリリースメモをご覧ください。
MPI リリースメモ

（共同利用支援係、共同研究支援係）

非線形構造解析汎用プログラム MSC.Marc2012/Mentat2012のサービス開始について（No.139）

Marc/Mentat のバージョンを MSC.Marc2012/Mentat2012 として運用を始めましたのでお知らせ致します。Mentat2012 では新 GUI を採用しています。従来の Classic GUI は mentat.classic というコマンドでご利用できます。また旧バージョンは暫定的に run_marc.2010、mentat.2010 というコマンドでご利用できます。

2012 年 8 月 27 日（月）よりサービスを開始しておりますのでご利用ください。

サービス開始日　：　2012 年 8 月 27 日（月）
ホスト名　　：　gen.isc.tohoku.ac.jp（並列コンピュータ）

▼ MSC.Marc2012/Mentat2012

バージョン　: 2012

起動コマンド: run_marc, mentat, mentat.classic

新機能の概要、機能の詳細、実行方法についてはマニュアル[1-3]、センターHP、および開発元HPをご覧ください。

マニュアル　: PDF形式マニュアルをgen上で提供しています。
並列コンピュータgen.isc.tohoku.ac.jpにログインし、
acroreadコマンドでご覧ください。各マニュアルは、
gen.isc.tohoku.ac.jpの以下のディレクトリにあります。

gen: /usr/ap/MSC2012/mentat2012/doc/

[1] marc_2012_doc_release.pdf　　: Release Guide(2012 英語)
[2] marc_2012_doc_user.pdf　　: User's Guide(2012 英語)
[3] marc_2012_doc_release_jpn.pdf　: リリースガイド(2012 日本語)

英文マニュアル

[4] vola.pdf　　: Volume A : Theory and User Information
[5] volb.pdf　　: Volume B : Element Library
[6] volc.pdf　　: Volume C : Program Input
[7] vold.pdf　　: Volume D : User Subroutines and Special Routines
[8] vole.pdf　　: Volume E : Demonstration Problems

和文マニュアル(2003版)

gen: /usr/ap/MSC2012/mentat2012/doc/japanese/

[1] vola.pdf　　: A編 理論およびユーザー情報
[2] volb.pdf　　: B編 要素ライブラリ
[3] volc.pdf　　: C編 プログラム入力
[4] vold.pdf　　: D編 ユーザーサブルーチンおよび特別ルーチン
[5] vole.pdf　　: E編 例題集
[6] marc_ug.pdf　　: ユーザーガイド
[7] mt_help_ref.pdf　: Mentat 2003 ヘルプリファレンス
[8] xsec_adden.pdf　: ドキュメント補足資料

センターHP

http://www.ss.isc.tohoku.ac.jp/application/marc_mentat.html

開発元HP

http://www.mscsoftware.co.jp/products/marc/index.php

(共同利用支援係)

ユーザの研究紹介掲載のお知らせ (No.139)

東北大学オープンキャンパスで展示した研究紹介ポスターをホームページでも閲覧できるように掲載致しました。計算機を利用したユーザの研究紹介を大規模科学計算システムトップページの「成果報告」項目、または下記のリンクよりご覧ください。

「成果報告」→「スーパーコンピュータを用いた成果」
http://www.ss.isc.tohoku.ac.jp/report/super.html

(共同研究支援係)

負担金の支払い費目について (No.140)

次回の利用負担金請求(平成 24 年 7 月 1 日から 9 月 30 日までの利用分)は 10 月初旬に行います。負担金の支払費目の指定に関して、これまで shiharai コマンドで指定頂いておりましたが、今年度より廃止いたしました。学内の方については事前に費目の指定は必要ありません(請求金額確定後、センター会計係より各部局の会計担当を通して照会いたします)。

学外の方については、特に支払費目名の入った利用負担金請求書を希望する場合や請求書の適要欄等について不明な点がある場合は会計係(022-795-3405)へご連絡くださるようお願いいたします。また、その他負担金に関することで不明な点がある場合は、共同利用支援係(022-795-6251)へご連絡くださるようお願いします。

(共同利用支援係, 会計係)

センター講習会資料の公開について (No.141)

サイバーサイエンスセンターでは利用者の方のスーパーコンピュータやアプリケーションの利用をサポートするため、センターの講習会で使用したテキスト・演習資料・サンプルデータを公開しております。

講習会資料のダウンロードページへは、大規模科学計算システムのトップページの「講習会案内」のリンク、または下記のリンクよりご覧下さい。

http://www.ss.isc.tohoku.ac.jp/guide/archives.html

（共同利用支援係, 共同研究支援係）

大規模科学計算システム FAQ ページ開設のお知らせ （No. 141）

サイバーサイエンスセンターでは利用者の方のご質問を元にFAQページを作成いたしました。システムの利用に関してご不明な点がありましたらこちらのページをご一読下さい。

FAQページへは、大規模科学計算システムのトップページ右上のリンク、または下記のリンクよりご覧下さい。

http://www.ss.isc.tohoku.ac.jp/faq/

（スーパーコンピューティング研究部, 共同利用支援係, 共同研究支援係）

分散コンピュータ博物館（サイバーサイエンスセンター展示室）ホームページ更新のお知らせ （No. 141）

分散コンピュータ博物館に所蔵されている展示品や資料を、ウェブページから閲覧できるWeb Galley のページを新設いたしました。

分散コンピュータ博物館のホームページへは、サイバーサイエンスセンタートップページのリンク、または下記のリンクよりご覧下さい。

http://www.isc.tohoku.ac.jp/museum/

（共同利用支援係）

— SENAC 執筆要項 —

１．お寄せいただきたい投稿内容

次のような内容の投稿のうち、当センターで適当と判定したものを掲載します。その際に原稿の修正をお願いすることもありますのであらかじめご了承ください。

・一般利用者の方々が関心をもたれる事項に関する論説
・センターの計算機を利用して行った研究論文の概要
・プログラミングの実例と解説
・センターに対する意見、要望
・利用者相互の情報交換

２．執筆にあたってご注意いただく事項

(1)原稿は横書きです。
(2)術語以外は、「常用漢字」を用い、かなは「現代かなづかい」を用いるものとします。
(3)学術あるいは技術に関する原稿の場合､200 字～400 字程度のアブストラクトをつけてください。
(4)参考文献は通し番号を付し末尾に一括記載し、本文中の該当箇所に引用番号を記入ください。
　・雑誌：著者, タイトル, 雑誌名, 巻, 号, ページ, 発行年
　・書籍：著者, 書名, ページ, 発行所, 発行年

３．原稿の提出方法

原稿のファイル形式は Word を標準としますが、PDF での提出も可能です。サイズ*は以下を参照してください。ファイルは電子メールで提出してください。

－Word の場合－
　・用紙サイズ：A4
　・余白：上＝30㎜　下＝25㎜　左右＝25㎜　綴じ代＝0
　・標準の文字数（45 文字 47 行）

＜文字サイズ等の目安＞
　・表題＝ゴシック体 14pt 中央　・副題＝明朝体 12pt 中央
　・氏名＝明朝体 10.5pt 中央
　・所属＝明朝体 10.5pt 中央
　・本文＝明朝体 10.5pt
　・章・見出し番号＝ゴシック体 11pt～12pt
　　*余白サイズ、文字数、文字サイズは目安とお考えください。

４．その他

(1)執筆者には、希望があれば別刷 50 部を進呈します。50 部を超える分については、著者の実費負担とします。別刷の希望部数等は投稿の際に申し出てください。
(2)投稿予定の原稿が 15 ページを超す場合は以下まで前もってご連絡ください。
(3)初回の校正は、執筆者が行って、誤植の防止をはかるものとします。
(4)原稿の提出先は次のとおりです。

東北大学サイバーサイエンスセンター内　情報部情報基盤課共同利用支援係
e-mail　uketuke@isc.tohoku.ac.jp
TEL　022-795-3406

編 集 後 記

今回の SENAC では，センターが利用者の先生方と取り組んでいる共同研究の中から３件の成果報告をしていただきました．共同研究では，「ここの部分が計算のボトルネックになっているのでこうするとプログラムがこれくらい速くなりそうだ」といったプログラムを構成する部分的な計算コードを中心に検討を進めることが多いわけですが，SENAC の記事としてその計算全体のもつ意味を拝見させていただくと，改めてスーパーコンピュータシミュレーション技術がもたらす成果の学術的・社会的影響の大きさを感じることができ，センタースタッフとしてほんの少しでも貢献できたことを大変うれしく思います．

9 月 28 日に理研の「京」の共用が開始され，併せて，「京」とサイバーサイエンスセンターを含む９大学の情報基盤センターのスーパーコンピュータ群を統一的な利用者 ID で利用できる HPCI システムが稼働し始めました．これにより，全国の幅広い分野の研究者や技術者が，我が国の多様なスーパーコンピュータ計算資源を効率よく利用できる環境と支援体制がより強化されました．HPCI システムによりスーパーコンピュータシミュレーション技術に基づく計算科学がより一層進展し，世界をリードする成果が数多く生み出されることを期待したいと思います．(H. K)

今年の仙台は猛暑が 9 月まで続きましたが，10 月に入ると例年と同じ時期に金木犀が香り始めました．今号が発行される頃には，青葉山の木々も色づいてくると思います．

摂南大学の田口先生に，前号に引き続き Fortran プログラミングに関する資料を執筆いただきました。センター職員にも参考になるとたいへん好評です．Fortran に関する資料の１つとして，お近くのみなさまにも紹介いただければ幸いです．(K. S)



サイバーサイエンスセンター前
整備中の青葉山新キャンパス

SENAC 編集部会

小林広明　曽根秀昭　水木敬明　後藤英昭
江川隆輔　早坂哲夫　大泉健治　小野　敏
斉藤くみ子

平成 24 年 10 月発行
編集・発行　東北大学
サイバーサイエンスセンター
仙台市青葉区荒巻字青葉 6-3
郵便番号　980-8578
印　刷　東北大学生活協同組合
プリントコープ

システム一覧

計算機システム	ホスト名	機　種
スーパーコンピュータ	super.isc.tohoku.ac.jp	SX-9
並列コンピュータ	gen.isc.tohoku.ac.jp	Express5800

サービス時間

利用システム名	利用時間帯
スーパーコンピュータ	連　続　運　転
並列コンピュータ	連　続　運　転
館 内 利 用	月曜日～金曜日は 8:30～21:00、 土・日・祝日は閉館

ジョブクラスと制限値

計算機システム	処　理	ジョブクラス	ＣＰＵ時間	メモリ容量
スーパー コンピュータ	会話型	(4cpu)	1 時間	8GB
	バッチ 処理	ss　(4cpu)	1 時間	256GB
		s　(4cpu)	無制限	32GB
		p8　(8cpu)	〃	512GB
		p16　(16cpu)	〃	1024GB
		p32　(32cpu)	〃	1024GB×2
		p64　(64cpu)	〃	1024GB×4
並列 コンピュータ	会話型	(4 並列)	1 時間	8GB
	バッチ 処理	as　(非並列)	無制限	16GB
		am　(Marc 専用)	〃	16GB
		am2　(Marc 専用)	〃	128GB
		a8　(8 並列)	〃	128GB
		a16　(16 並列)	〃	256GB
		a32　(32 並列)	〃	512GB

目次

東北大学サイバーサイエンスセンター
大規模科学計算システム広報　Vol.45 No.4　2012–10

［共同研究成果］
光によって引き起こされる化学反応の量子力学に基づく理論解析
— 非断熱遷移を引き起こす円錐交差の取り扱いについて — …… 山崎 馨 1
菅野 学
河野裕彦

海底圧力観測データから海底鉛直変位を検出するための
海洋モデルの開発 …… 稲津大祐 11
日野亮太
藤本博己

MPI による数値タービンの大規模並列計算手法の開発 …… 笹尾泰洋 19
山本 悟
三宅 哲
岡崎健志

［資料］
Fortran スマートプログラミング
— 第 2 回　サブルーチンと入出力 — …… 田口俊弘 27

［お知らせ］
大規模科学計算システムホームページの更新について …… 51

［報告］
サイバーサイエンスセンター オープンキャンパス報告 …… 小松一彦 52

［展示室便り⑥］
スーパーコンピュータ SX シリーズ …… 53

［Web 版大規模科学計算システムニュース］より
反応経路自動探索プログラム GRRM11 のサービス開始 について（No.137） …… 55
コンパイラアップデートのお知らせ（No.139） …… 56
非線形構造解析汎用プログラム MSC.Marc2012/Mentat2012 の
サービス開始について（No.139） …… 57
ユーザの研究紹介掲載のお知らせ（No.139） …… 59
負担金の支払い費目について（No.140） …… 59
センター講習会資料の公開について（No.141） …… 59
大規模科学計算システム FAQ ページ開設のお知らせ（No.141） …… 60
分散コンピュータ博物館（サイバーサイエンスセンター展示室）
ホームページ更新のお知らせ（No.141） …… 60

執筆要項 …… 61

編集後記 …… 62

VEGETABLE OIL INK