



TOHOKU
UNIVERSITY

ISSN 0286-7419

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.46 No.1 2013—1



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) kaikai@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。

[巻頭言]

2013 年東北大学及びサイバーサイエンスセンターへの期待

公益財団法人仙台応用情報学研究振興財団

理事長 野口 正一

2012 年は我が国にとって必ずしも明るい年ではなかったと思います。

一向に改善されない日本の経済、伸びない GDP、かつて日本を牽引したエレクトロニクス産業の急速な衰退、そして領土問題等。これら多くの要因で我々の周囲には大きい閉塞感が漂っていました。

今、2013 年に向けて日本はどのように復活できるのであろうか。この目的に向けて大学を中心とするアカデミアはどのように貢献できるのであろうか。

この問題が 2013 年の初頭、我々に課せられた大きい課題だと思います。

即ち、2013 年に向けて日本の進むべき進路、そしてどのような輝く未来を具体的に我々は描くことができるのか。

この中で重要な課題の一つが日本の産業構造のドラスティックな再構築の問題です。

日本の産業力はかつて世界的なコモディティの分野で圧倒的な力を持ち、現在の日本のポテンシャルをここまでに高めることができたわけです。

しかしながら、この分野は新たに台頭してきた新興国の産業の力のもとで自動車を除く多くの分野で決定的な差をつけられてきました。

この分野で再び日本が覇権を取り戻すためにはコモディティのビジネスで決定的に他に勝てる製品開発のシナリオがなければ不可欠ですし、当面は大変に難しいと思います。

一方、現在の日本は世界最高水準の物作りの技術、また新素材の開発の分野でも圧倒的な力を持っています。当然この技術を具体的に産業化していくことは大変大事なことです、これだけでは日本を支える産業基盤として考えることは難しい。とすれば、これらの技術を集大成し、新しいビジネスモデルを構築していくことが大事となります。

異種分野の技術の知的資源を活用し、総合的な社会システムのインフラの構築技術ができれば、他国に先駆けて日本にとっての大きいビジネスが創出できるものと思います。そしてこれを支える技術は異種の巨大データベースを活用し、社会インフラに対応するクラウドシステムの構築技術、つまり大規模なスマートコンバージェンスを可能とする技術です。

対象となる分野としてはエネルギー資源を中心とした Smart City、水資源を中心とした環境社会システム、交通、Logistic を中心とした社会システム等数多くの分野で多くの産業が創出される可能性があります。同時に多くの新しい雇用も生まれます。

以上のことを換言すれば、すでに日本が絶対的な力として獲得している各産業分野の高度の知的資源をベースに高度なスマートコンバージェンスをベースとした新しい総合的な情報処理システムの開発が早急に解決すべき課題ということです。

しかしながら、この問題を解決するための基本的な技術はいまだ十分に開発されているとは思えません。

この技術を達成するための最も重要な課題は **heterogeneous** な巨大データをどのように一元化し、目的とする分野に最大限有効な情報を提供させる技術です。このためにはいくつかの重要な研究課題を解決しなければなりません。

第一が **heterogeneous** な情報環境における巨大データベースをどのように効率的に一元化するかということ。

このためには **heterogeneous** な環境のデータを一元化する共通プラットフォーム構築の問題が重要となります。当然データ送受信の要である新しい通信プロトコルの開発も必要となります。これらの問題をどのようにして開発していくか、特に共通プラットフォーム問題解決を含めての国際標準化への問題は重要となります。

次の重要な課題は巨大データ処理のための知的エンジンとそれを管理する巨大センターの開発です。

このセンターはシステムを構成する各個別の要素に実時間で必要とする情報を提供しなければならない。以上、総括すれば夫々の目的に対応した分野のための巨大クラウドの構築技術の開発が中心的課題です。

以上のような一般論の上で東北大学はどのようなプロジェクトを行うことができるのでしょうか。

あくまで私見ではありますが、次の三つの大きい分野があると思います。

第一は現在医学系を中心に推進されている巨大な人間のゲノム情報の活用です。この情報を新しい医療、健康、介護、福祉等の世界に活用する技術が生まれれば大きい新しい産業の創出が可能となります。

第二に東北大学が世界に誇る多様な分野の材料物性の巨大データベースの活用です。新しい材料の創出とこれをベースに知的生産システムとのコンバージェンスにより、新しい産業が創出されるでしょう。

第三が現在東北大学の通研が中心に NICT との間で研究開発を推進している“大震災に対応できる次世代情報・通信システム構築”のプロジェクトです。

このプロジェクトが産業創出の面からさらに解決すべき一つの重要な課題は国、自治体、企業の持つ情報システムと本次世代情報・通信システムとの総合的な融合技術です。

このプロジェクトが開発されれば日本のみならず諸外国でも展開できる新しい産業創出の基盤となります。

以上のような状況下で上に述べたプロジェクトが産業創出の面からも推進されれば、これらは東北大学の更なる社会貢献の柱になると思います。

最後に以上のプロジェクトを推進する上で巨大な知的資源を保有し、活用している当サイバーサイエンスセンターはスマートコンバージェンス技術推進の中核の機関として重要なミッションを果たすことができると強く期待しています。

[共同研究成果]

ダウンスケールシミュレーションにおける台風の再現性について

台風 Choi-wan (2009) の事例解析

濱田真之、岩崎俊樹
東北大学理学研究科

台風予報は数値予報の重要な開発課題である。全球数値予報モデルによる進路予報は、近年精度向上が著しく、5 日先までの予報が提供されている。しかしながら、台風の強度の再現には高解像度の雲解像モデルが必要であり、多大な計算機資源が必要であることと、現状で予測性能が十分ではないため、まだ数値モデルによる台風強度予測は実用化には至っていない。現在、非静力学数値モデルを用いたダウンスケールシステムを利用し、台風強度予測の可能性を調べている。このようなフレームワークでは、台風強度予測に関しては数値モデルそのものの性能に加えて、初期条件や境界条件もその精度に影響する。そこで、本研究では現状の再解析を初期値・境界値に用いた完全予報(事後予報)実験における、台風の進路と強度の再現精度を事例解析により調べる。

実験対象は防災上大きな問題となる急発達を示した台風 Choi-wan (2009) を選んだ。特に、台風進路と強度の再現精度は初期値・境界値の精度に依存する。このため、初期値・境界値に JRA25/JCDAS と ERA-Interim の 2 つの異なる再解析を用いて、再現された台風の強度を調べた。

本研究のケースでは台風強度についてはどちらの場合も過小評価であった。2 つの再解析の影響については、進路・強度とも ERA-Interim を使用した実験のほうがベストトラック(観測値)に近かった。また、初期値と境界値の組み合わせを変えた実験から、進路の予報精度を上げるためには境界値を、強度の精度を向上させるには初期値の再現性を向上させる必要があることが明らかになった。

1. はじめに

台風は北西太平洋で発生・発達し、大災害をもたらす気象現象である。最近では Talas (2011) や Megi (2010) などによる災害が記憶に新しい。台風の進路や強度を正確に予報することは重要である。

台風を含む気象現象を予報するために主要国の予報機関は独自の数値モデルを開発してきた。近年では、観測データの増加[1]、4 次元データ同化手法の高度化、予測モデルの高解像度化、パラメタリゼーションの改善などにより、予報精度は大幅に向上している。気象庁の台風モデルについても予報誤差の縮小に伴い、狭領域の TYM(気象庁台風モデル、[2])から TEPS(Typhoon Ensemble Prediction System : [3])へ移行され、5 日間予報が行われている。

進路予報に関しては TIGGE(The THORPEX Interactive Grand Global Ensemble)を使用した解析から進路予報の精度を向上させるためには指向流が再現されているかに依存することが明らかになっている[4]。一方、強度予報では台風内部の小規模現象(e. g., Eyewall Replacement Cycle)を再現できる高解像度の雲解像モデルが必要となる[5]。高解像度モデルは膨大な計算機資源を必要とし、予測精度も満足いくレベルに達していないため、台風強度の数値予報は実用化されていない。

現在、強度予報に関しては非静力学数値モデルと力学的ダウンスケーリングを用いた研究が行われている[6]。この手法では領域モデルを使用するため、予報精度は数値モデルの誤差の他に初期値や境界値に左右される。また、ダウンスケーリングを用いた台風の強度予報実験では各予報機関が作成した再解析データが初期値として使用されている[7][8]。再解析データとは、過去の観測データを数値予報モデルに同化し、過去の大気の 3 次元分布を再現させたものであり、台風や集中豪雨等の極端現象を再現する際に有効である。

本研究ではJCDAS(Japan Climate Data Assimilation System: 以下 JRA) [7]と ERA-interim(以下 ERA) [8]の2つの再解析データを初期値と境界値として使用した実験を行う。JRA25(Japanese 25-year Reanalysis)/JCDAS は気象庁の再解析で、ERA-Interim(ECMWF re-analysis)は ECMWF(European Center for Medium-Range Weather Forecasts)の作成した ERA と JMA(Japan Meteorological agency)が作成した再解析である。また、2009年に発生した台風 Choi-wan を対象として気象庁非静力学モデル(JMA-NHM)を用いて再現実験を行った。台風 Choi-wan は2009年に発生した台風の中でも急激に発達した台風であり、防災上重要な問題を持つため本研究の対象として選択した。本研究を通じて、強度と進路が初期値と境界値のどちらに影響を受けるか事例解析を行った。また、本計算は東北大学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64並列の計算を約15時間実行した。

2. 実験概要

2.1 台風 Choi-wan(2009)の概要

気象庁ベストトラックから Choi-wan は2009年9月12日18:00UTCに15.4°N、150.9°Eで台風となった。急発達しながら北西へ進み、9月15日12:00UTCに最低気圧が915hPa、最大風速が55m/sに達した。その後も強度を維持したまま北西に進み、9月18日00:00UTCに転向した。転向後は強度が弱まり、9月20日12:00UTCに熱帯低気圧に変化した。

本稿では Choi-wan が急発達した段階である9月13日06:00UTCから成熟状態に達した9月16日12:00UTCまでを計算時間とする。

2.2 数値モデルの概要

本研究では JMA-NHM[9]を数値モデルとして使用した。JMA-NHM は気象研究所/数値予報課統一非静力学モデル(MRI/NPD-NHM)を基にしたモデルであり、メソ数値モデルシステムの現業モデルとして採用されている。このモデルは静力学近似を用いていないため細かい水平解像度を要求する現象の再現が可能である。また、雲の微物理過程をモデルに含む。これは、相変化や放射に伴う熱収支や雲を構成する微粒子の質量変化によって降水過程だけではなく熱力学や力学過程を変化させるため顕著な降水を伴う気象現象を扱う場合には微物理過程を含むモデルを使用することが必要となる。

本研究で対象とする台風は積乱雲の集合体であり、静力学近似を使用することは適切ではない。さらに、台風は激しい降水を伴う気象現象であるため雲物理過程を含むモデルを使用することが望ましい。この2点から非静力学モデルを用いることが適当である。

JMA-NHM の設定は表1に示す。

本研究では以下の3つの条件で実験を行った。

- EI-EB(era)
初期値 ERA、境界値 ERA
- EI-JB(era_bc_jra)
初期値 ERA、境界値 JRA
- JI-JB(jra)
初期値 JRA、境界値 JRA

ERA と JRA は実験で使用する水平解像度2kmと比べて粗いため、水平解像度を30km(201×171格子)→10km(481×391格子)→2km(1501×1001格子)へとダウンスケーリングした。各ダウンスケーリングの際にスピンアップとして3時間ずつ計算時間をずらしている。

積雲対流パラメタリゼーションとして水平解像度が30kmと10kmの計算をする際には Kain-Fritsch 法を用いた。対流パラメタリゼーションは解像度が粗いモデルで表現することが出来

表1 JMA-NHM の設定

水平解像度	2km
計算期間(UTC)	2009.09.13.12- 2009.09.16.12
水平解像度	1501×1001
鉛直層	50層(top ~27.8km)
予報時間	72h
初期値	dx=10kmの結果
タイムステップ	6s
対流パラメータ	雲物理過程
乱流クロージャー	Improved Mellor- Yamada Level3
海面温度	NOAA Daily OI SST

ない雲を仮想的に与えて計算をする手法である。しかし、台風の強度はアイウォール内部の現象 (e. g., Eyewall Replacement Cycle、2 重眼構造) の影響を受ける [4] ためメソ渦を表現することが望ましい。また、積雲パラメタリゼーションを用いずにメソ渦を再現するためには水平解像度を 2.5km 以下にすることが求められる [10]。よって、水平解像度 2km の計算では雲物理過程のみを用いて計算を行った。

2.3 MPI (Maximum Potential Intensity)

台風の潜在的な強度を評価する方法として MPI [11] [12] を用いた。計算式は、

$$V_m^2 = \frac{(T_s - T_0) C_k (k_0^* - k_a) \Big|_m}{T_s C_D} \quad (1)$$

$$p_c = p_m \exp \left(- \frac{V_m^2}{2 R_d T_s} \right) \quad (2)$$

である。また、式中のパラメータについては、 V_m は最大接線風速、 T_s は海面温度、 T_0 は台風の温度、 C_k/C_D は熱の交換係数、 k_0^* は海面のエンタルピー、 k_0 は大気のエントルピー、 p_c は中心気圧、 p_m は最大風速半径での気圧、 R_d は気体定数である。

MPI は初期条件から台風の最大可能強度を求める手法であり、内部構造に左右されず強度を求めることが出来る点でメリットがある。一方、台風は成長の過程で海面を混合させ、海面温度を低下させるため成長を抑える効果がある [5]。また、海面から供給される熱とアウトフローとして放出される熱の差によって生じるエネルギーと、地表面の摩擦による消失エネルギーが等しくなるときに最大可能強度としているため、強度を過大評価する傾向がある点に考慮する必要がある。

3. 結果

3.1 構造

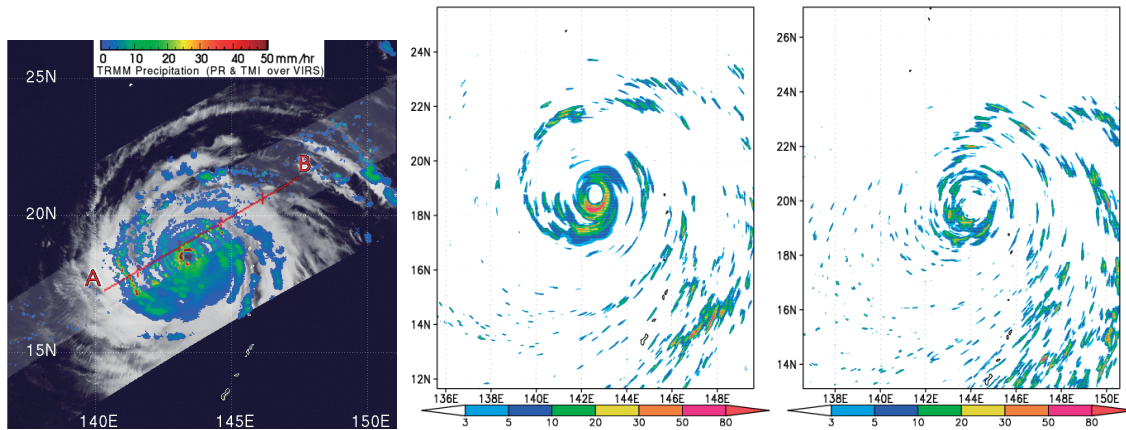


図1 TRMM の観測値(左図)と計算結果(EI-EB(中央図)と JI-JB(右図))

TRMM は 2009 年 9 月 15 日 23:20UTC、計算結果は 2009 年 9 月 15 日 23:00UTC のデータを使用

再現性を確かめるために TRMM(Tropical Rainfall Measuring Mission) の降水データと計算結果を比較する(図 1)。図 1 から EI-EB ではアイウォールの顕著な降水やレインバンドを再現しているのに対して JI-JB では渦の再現性が悪い。また、JI-JB は強い降水を再現できていない。この結果から、Choi-wan(2009) のケースでは降水分布に関しては EI-EB の再現性が高い。

3.2 進路

台風の経路を図2に示す。予報時間は72時間である。計算結果からEI-EBの結果が気象庁ベストトラックと同方向に進路をとる。しかし、中心からの誤差では常に50kmから150kmの誤差を持ち、進行速度がベストトラックと比べて速いことが分かる。また、JRAを境界値としたEI-JBとJI-JBはマリアナ諸島に接近後、ベストトラックに比べて相対的に北向きの進路をとる。この結果から、本事例では台風の進路は境界値に依存する。

図3は2009年9月13日06:00UTCにおけるERAとJRAの動径方向の風速である。JRAは第2象限に北西方向の強い風速を持つ。一方、ERAは第1象限に南西方向の強い風速を持つ。計算結果から境界値がERAのEI-EBは境界値をJRAとしたEI-JBとJI-JBと比較して南のルートをとる点から計算開始時の流れに依存している。この結果から、台風は大規模場の流れを指向流として移動するため、進路は境界値に依存したと考えられる。

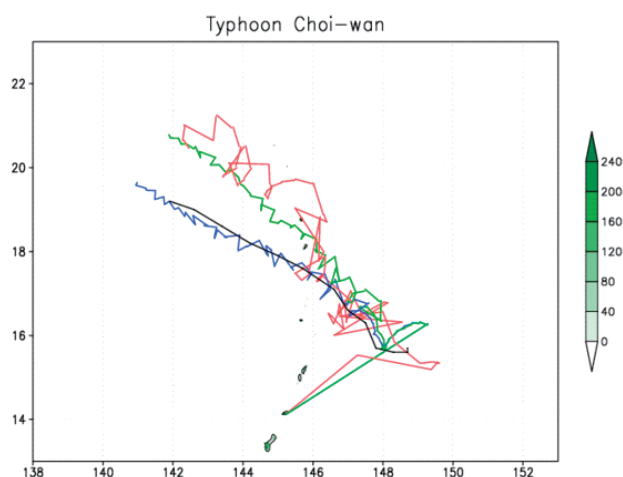


図2 台風 Choi-wan (2009) の経路図
計算結果は1時間毎、ベストトラックは6時間毎のデータをプロット

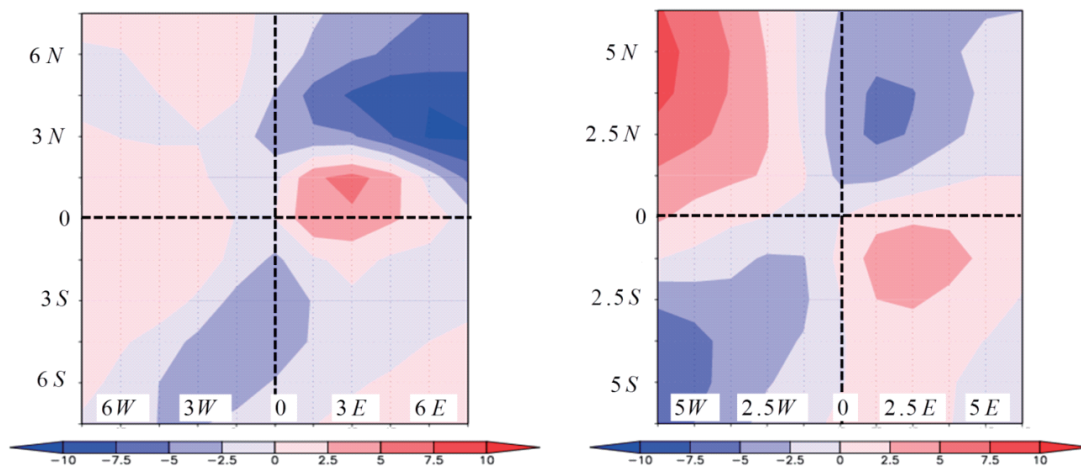


図3 風速の動径方向成分(JRA(右図)、ERA(左図))
赤色が中心から離れる方向、青色が中心に向かう方向

3.3 強度

図4は台風の強度(最大接線風速、最低気圧)である。計算時間内で最も低い気圧はEI-EBで954.8hPa、EI-JBで961.0hPa、JI-JBで986.2hPaとなった。最大風速についてはEI-EBは43.9m/s、EI-JBは40.7m/s、JI-JBは27.8m/sである。最大接線風速及び最低気圧はEI-EBで最もベストトラックに近い値となる。EI-EBと同じ初期値を使用したEI-JBについてもEI-EBと同じ強度変化をしているが計算時間30時間目から乖離し、計算時間後半では6hPa程度の差が生じた。JI-JBについては計算開始から発達が見られなかった。この結果から台風の強度は初期値に依存することが明らかになった。

強度についてベストトラックと比較するとすべての実験結果で大きな差が生じた。計算を行った初期場が発達できる環境場であったか調べるためにMPIを使用する。MPIの結果とEI-EBとJI-JBの最

大風速、最低気圧を表 2 に示す。MPI の結果から初期場は発達が可能であることが分かる。また、ERA と JRA では初期場の発達に大きな違いがないことが分かる。従って、急発達を対象とした本事例では環境場よりも台風の初期渦の再現性を高めることが重要であると示唆される。

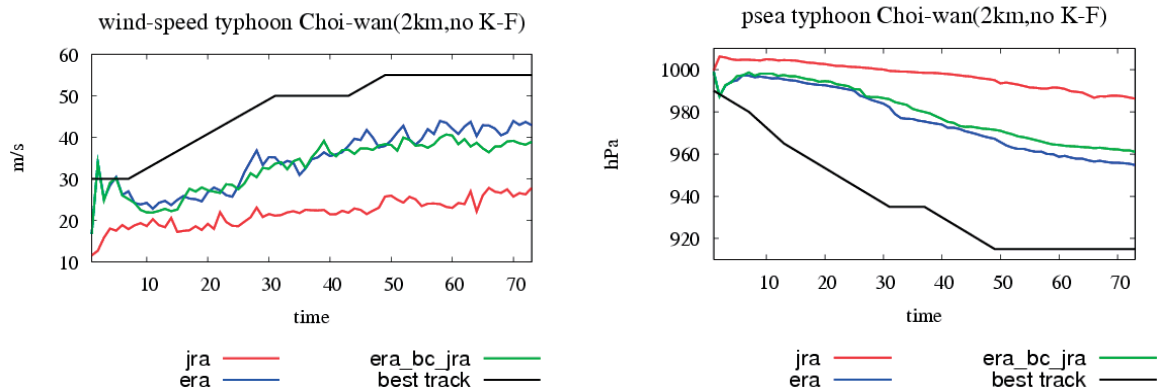


図 4 最低気圧(右図)と最大接線風速(左図)

計算結果は 1 時間毎、ベストトラックは 6 時間毎のデータをプロットした。また、横軸は 2009 年 9 月 13 日 12:00UTC(time=1h)から 2009 年 9 月 16 日 12:00UTC(time=73h)

表 2 MPI と EI-EB(ERA)、JI-JB(JRA) の結果の比較

	最大風速(m/s)	最大気圧(hPa)
MPI(JRA)	78.3	895.5
MPI(ERA)	80.2	886.5
ERA(dx2km)	43.9	954.7
JRA(dx2km)	27.8	986.2

4. まとめ

ERA-Interim と JRA25/JCDAS の 2 つの再解析データを初期値と境界値として JMA-NHM を用いて台風 Choi-wan(2009) の再現実験を行った。本事例では進路と強度で ERA を使用した場合の方がベストトラック(実況監視データ)に近かった。特に降水分布は衛星データと比較的よく一致した。また、初期値と境界値の組み合わせを変えた実験から、領域モデルを使用して台風の数値実験をする際には、進路は境界値により強く依存することが分かった。台風の進路は比較的大規模な周囲の風を指向流として感じているためだと考えられる。これに対して、台風の強度の再現には、境界値より初期場の方が重要であった。MPI(最大可能強度)は、JRA と ERA では違いは大きくはなかった。従って、急発達の 1~2 日前を初期条件とする今回の実験の場合は、初期渦の構造が台風強度の再現にとってより重要であることを示している。もちろん、数値モデルについても、雲物理過程や大気境界層などまだ課題は多い[13]。今後事例を重ねて数値モデルの問題点を洗い出し、数値モデルを改良していく必要がある。

5. 謝辞

気象庁より気象庁非静力学モデル(JMA-NHM)を貸与して頂いた。また、本研究は東北大学サイバーサイエンスセンターを利用することによって実現することが出来た。ここに謝意を表する。

参考文献

- [1]Elsberry, R. L., T. D. B. Lambert, and M. A. Boothe, : Accuracy of Atlantic and Eastern North Pacific Tropical Cyclone Intensity Forecast Guidance. *Wea. Forecasting*, **22**, 747–762, 2007
- [2]Iwasaki, T., H. Nakano, and M. Sugi, : The performance of typhoon track prediction model with cumulus parameterization. *J. Meteor. Soc. Japan*, **65**, 555–570, 1987
- [3]Yamaguchi, M., R. Sakai, M. Kyoda, T. Komori, and T. Kadowaki, : Typhoon Ensemble Prediction System Developed at the Japan Meteorological Agency. *Mon. Wea. Rev.*, **137**, 2592–2604, 2009
- [4]Yamaguchi, M., T. Nakazawa, and K. Aonoshi, : Tropical cyclone track forecasts using JMA model with ECMWF and JMA initial conditions. *Geophys. Res. Lett.*, **39**, L09801, doi:10.1029/2012GL051473, 2012
- [5]Chen, H., D. Zhang, and J. Carton, : On the rapid intensification of hurricane Wilma(2005). Part I : Model prediction and structural changes. *Wea. Forecasting*, **26**, 885–901, 2011
- [6]Wada, A., : Numerical study on the effect of the ocean on tropical-cyclone intensity and structural change. Atmospheric Model Applications, *In Tech*, 43–68, ISBN 979-953-307-335-3, 2012
- [7]Onogi, K., J. Tsutsui, H. Koide, M. Sakamoto, S. Kobayashi, H. Hatsushika, T. Matsumoto, N. Yamazaki, H. Kamahori, K. Takahashi, N. Mannoji, and R. Taira, : The JRA-25 Reanalysis, *J. Meteor. Soc. Japan*, **85**, 369–432, 2007
- [8]Dee, D. P., S. M. Uppala, A. J. Simmons, P. Berrisford, P. Poli, S. Kobayashi, U. Andrae, M. A. Balmaseda, G. Balsamo, P. Bauer, P. Bechtold, A. C. M. Beljaars, L. van de Berg, J. Bidlot, N. Bormann, C. Delsol, R. Dragami, M. Fuentes, A. J. Geer, L. Haimberger, S. B. Healy, H. Hersbach, E. V. Holm, L. Isaksen, P. Lallberg, M. Kohler, M. Matricardi, A. P. McNally, B. M. Monge-Sanz, J.-J. Morcrette, B.-K. Park, C. Peubey, P. de Rosnay, C. Tavolato, J.-N. Thepaut, and F. Vitart, : The ERA-Interim reanalysis: configuration and performance of the data assimilation system. *Q. J. R. Meteorol. Soc.*, **137**, 553–597, 2011
- [9]Saito, K., J. Ishida, K. Aranami, T. Hara, T. Segawa, M. Narita, and Y. Honda, : Nonhydrostatic atmospheric models and operational development at JMA. *J. Meteor. Soc. Jpn*, **85B**, 271–304, 2007
- [10]Noda, A., and H. Niino, : Critical grid size for simulating convective storms: A case study of the Del City supercell storm. *Geophys. Res. Lett.*, **30**, 1844, doi:10.1029/2003GL017498, 2003
- [11]Emanuel, K., : An air-sea interaction theory for tropical cyclones. Part I. *J. Atmos. Sci.*, **42**, 586–604, 1986
- [12]Emanuel, K., : Sensitivity of tropical cyclones to surface exchange coefficients and a revised steady-state model incorporating eye dynamics. *J. Atmos. Sci.*, **52**, 3969–3976, 1995
- [13]Sawada, M., and T. Iwasaki, : Impacts of Evaporation from Raindrop on Tropical Cyclone. Part I: Evolution and Axisymmetric Structure. *J. Atmos. Sci.*, **67**, 71–83, 2010

[共同研究成果]

ヤマセ研究のための再解析・気候モデルデータのダウンスケーリング

島田照久 沢田雅洋* 岩崎俊樹

東北大学大学院理学研究科

(*現在 東京大学大気海洋研究所)

1. はじめに

気候変動の影響が顕在化するにつれて、気候変動研究は、気候変動そのものの解明に取り組むという本来の目的に加えて、気候変動の影響評価に貢献する必要性があると強調されるようになってきている。気候変動の予測には不確実性が伴うが、気候変動の起こりうる範囲を考慮したうえで、気候変動の影響評価が求められるようになってきているのである。このような影響評価は、温暖化適応策や気候変動に起因する様々なリスク管理を推進する上で重要な基礎情報となる。さらに、気候変動の影響評価は、地域レベルで求められるようになっており、それに応える科学的知見が必要である。このような社会的要請に応えるための方法の一つが、気候モデルの力学的ダウンスケーリング(数値気象モデルを用いた再計算によるデータの詳細化のこと。以下、ダウンスケーリングという)である。つまり、低解像度の気候モデルによる大規模場の計算結果をもとに、高解像度の地形等を入力することで下部境界の状態に大きく依存した小さな現象を解像し、気候予測の結果に付加価値を付けるのである。気候モデルの予測から、地域レベルでの気候変動の予測と影響評価を行うためにダウンスケーリングに関する様々なプロジェクトが実施されている[1-4]。

本研究では、気候モデルのダウンスケーリングによって、東北地方の夏季気候に重要な役割を果たすヤマセとその将来変化の理解を進めることを目的としている。ヤマセとは、夏季に間欠的に発達するオホーツク海高気圧から北日本の太平洋側に吹きつける冷たい東風のことであり、背の低い(<1000m)下層の冷気層は、逆転層を伴って安定成層しており、地形の影響を大きく受ける。そのため、下層雲や霧が低地を覆い、放射冷却により低温を維持する。一方で、オホーツク海高気圧の消長に合わせて、ヤマセの発生頻度や低温の程度は顕著な経年変動を起こす。ヤマセは、地域気象・気候、そして社会活動(農業、海洋安全、航空安全等)に影響を与えるため、ヤマセの理解は、特に東北地方の冷害の歴史[5]を背景に、社会的に重要な課題でありつづけている。その一方で、気候変動に伴うヤマセの将来変化を予測し、ヤマセの局地気候への影響の理解をさらに進めることが重要な課題となってきた。しかし、温暖化シミュレーションを行うほとんどの気候モデルの空間解像度は100kmより粗く、気候モデルではヤマセにとって重要な役割を果たす北日本の地形(脊梁山脈等)を十分に解像することはできていない。そこで、私達は、再解析データや気候モデルデータのダウンスケーリングによって、長期間にわたって地域スケールの気象・気候解析ができるデータセットを作成し、ヤマセとその将来変化について研究を進めている[2]。本稿では、これまでの取り組みの概要について述べる。

2. ダウンスケーリング

気象庁・気象研究所の数値気象モデル(非静力学モデル JMA-NHM [6])を用いて、北日本全体(東北地方と北海道)を含む領域について、10km 解像度までダウンスケーリングする(図1)。北日本全体を計算領域に含めることによって、ヤマセの影響が顕著な北日本の太平洋側だけに着目する

のではなく、北日本の太平洋側と日本海側の気候の対照性を明らかにすることができる。また、今回は長期間の気候データセットを作成することが目的であるので、ダウンスケーリングする解像度は、計算時間とのトレードオフで決める必要がある。今回選択した 10km という解像度は、北日本全体の長期積分(現在気候から将来気候まで)に必要な計算時間が妥当な範囲でありながら、ヤマセにとって重要な地形(脊梁山脈、山地と平野、海峡や地峡、半島等)を解像し、地域スケールの解析が可能となる解像度である。実際、20-30km 解像度では、北日本の脊梁山脈全体は表現できるものの、平野の形状再現や海峡の解像が難しく、標高の過小評価が著しい。鉛直方向には 38 層設定し、1 時間間隔でデータを出力する。データ作成の対象期間は、各年の 6-8 月である。

今回、作成したデータセットは次の通りである。まず、再解析データを境界条件とする現在気候データを作成した(1979-2011 年)。用いたデータは、再解析データ JRA-25(1.25 度格子)[7]と 0.25 度格子の海面水温[8](1979-1981 年は、COBE-SST[9]を用いた)である。10km にダウンスケーリングを行うために、30km 解像度の計算領域を介し、2 回のネスティングを行う。次に、気候モデルデータをダウンスケーリングした。今回用いたのは、気象研究所のグループが開発した水平解像度約 20km (TL959)鉛直 60 層の超高解像度全球大気モデル (MRI-AGCM) によるタイムスライス実験データである。現在気候(1979-2003 年)、将来気候(2075-2099 年)の各 25 年間を解析対象とする。温室効果気体の濃度は、SRES A1B シナリオに基づいている。下部境界条件の海面水温は、現在気候実験では、英国ハドレーセンターの HadISST データセットを用いている。将来気候実験については、第 3 次結合モデル相互比較プロジェクト(CMIP3)の 18 の大気海洋結合モデルの平均から作成されており、線形トレンドを除去した後の経年変動については、現在気候の経年変動に置き換えられている[10]。以上のようにして作成したデータセットは、東北地方の地域特性を考慮できる空間解像度(10km)と日変化を解析できる時間解像度(1h)を有し、ヤマセの経年変動を扱える長期間(25, 33 年)の均質な(同じモデル、スキーム、計算領域)データとなり、このような特徴を持つデータセットは初めてである。

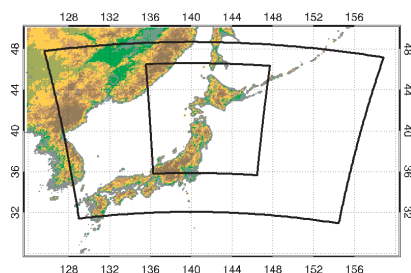


図 1 モデルの計算領域(外側が 30km 解像度、内側が 10km 解像度)。

数値気象モデルによるデータ作成は、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 SX-9 で行った。各年の 6-8 月の 3 ヶ月分の JRA-25(MRI-AGCM)の 2(1)段階のダウンスケーリングにかかる時間は、ノード内の 8CPU(ジョブクラス p8)を用いた MPI による並列処理で、約 25(12)時間である。この計算を JRA-25 については 33 年分、MRI-AGCM については計 50 年分実施した。このような長期間の気候データセットの作成は、スーパーコンピュータを利用することで可能になった。

3. 解析事例

(1) ダウンスケーリングの効果

まず、空間解像度の観点からダウンスケーリングの効果を見てみる。ヤマセを特徴づけるインデックス(ヤマセインデックス) [11]を用いて、ヤマセインデックスが1標準偏差を越える時(つまり、ヤマセが卓越する時)の6-8月の地表気温の気候平均と偏差を図2に示す。ここでは、稚内と仙台の気圧差で定義され、ヤマセ卓越時の北高型の気圧配置をよく表すヤマセインデックス[12]を用いた。再解析データ JRA-25(図2左)の1.25度格子の解像度では、オホーツク海からの冷氣が日本列島とユーラシア大陸の東側に沿って舌状に伸びていることがわかる程度であるが、10km場(図2右)では、北海道から北東北にかけて山地の東側で低温偏差が大きいことがわかる。つまり、10kmの解像度のデータでは、北日本の主要な地形を考慮した局地気候が解析可能であると言える。一方で、時間解像度の違いを見てみる(図3)。ダウンスケーリングによって計算された1時間毎の八戸の気温は、気温の日変化とともに、ヤマセ卓越時(下層雲量が多いときに対応)に気温の日振幅が減少する様子が再現されている。このような特徴を、6時間間隔のJRA-25データから解析することは難しい。つまり、ダウンスケーリングにより時間解像度も改善され、ヤマセに起因する日周期の変動を考察することができる。例えば、日変化の周期帯のウェーブレットパワーが、ヤマセのよい指標となることを示した[11]。

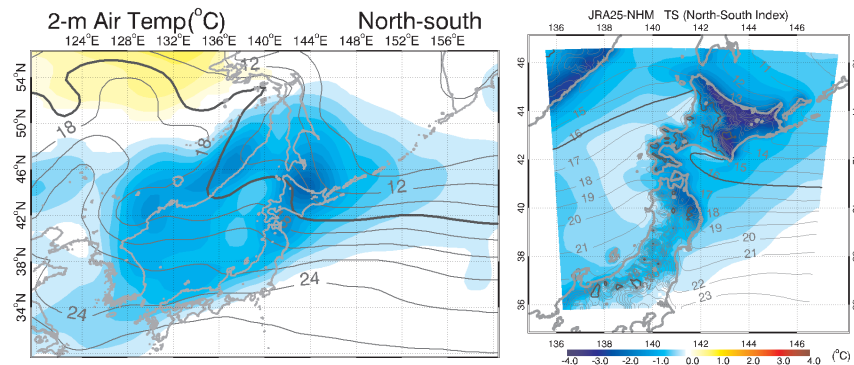


図2 ヤマセ卓越時の気候平均の気温分布(コンター)と平年偏差(カラー) (6-8月, 1979-2011年)。(左)JRA-25 と(右)10km ダウンスケーリングデータ。

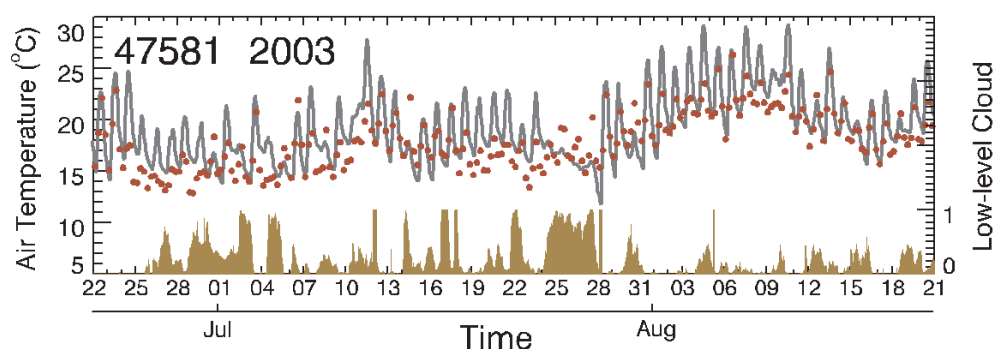


図3 八戸の気象官署に対応する格子の10kmデータの1時間毎の気温(灰色実線)とJRA-25の6時間毎の気温(赤点)(2003年)。下層雲量を合わせて示す。

(2) 気温の変動

現在気候の再現性と将来の気温上昇を確認するために地表気温をデータセット間で比較した。図4には、八戸について、各データセットのヒストグラムと、観測値に対するQ-Qプロットを示す。現在気候のデータセット(JRA-25とMRI-AGCM)は観測に対しやや高温のバイアスを持つが、分

布形はほぼ一致することが確認できる。将来気候の気温は、夏季平均では現在気候よりも 3.2°C 上昇するが、将来気候の気温のヒストグラムの全体の形状は現在気候のものに近い。高温バイアスの原因の一部は、MRI-AGCM で用いられた海面水温の解像度の問題、雲物理過程の解像度依存性にあることがわかっている。

各データセット間で気温の頻度分布には違いが見られるものの、ヤマセに伴う気温の変動パターンを検証してみると、各データセット間でよい一致が見られる。図 5 には、各データセットについて、気象官署の観測点における気温の主成分解析による第 3 モードの空間パターンを示す。北日本全体の気温変動を表す第 1 モード(寄与率 80%)、南北変動パターンを示す第 2 モード(寄与率 7%)に加えて、北日本の山地を境にした太平洋側と日本海側の変動パターンを示す第 3 モード(寄与率 3%)が各データセット間に共通してみられる。このモードは、ヤマセ卓越時に特に強調される[11]。この結果は、現在気候の再現性の良さを示すとともに、ヤマセ卓越時の気温変動パターンが将来気候データにおいても見られることを示す。ただ、MRI-AGCM の結果では、福島から茨城にかけては、日本海側と同符号になっており、気候モデルではヤマセによる冷気の南下の表現が弱いことが示唆される。

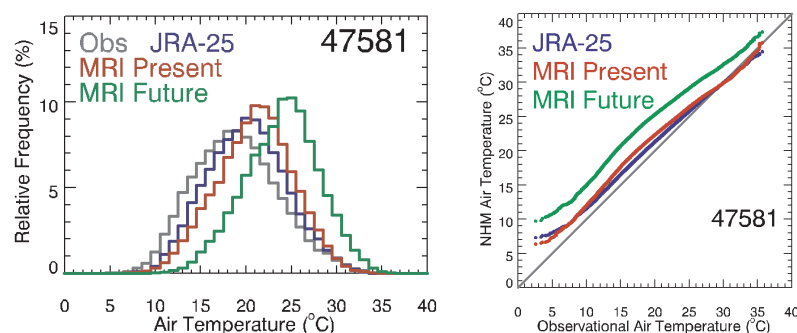


図 4 八戸における気温のヒストグラムと観測に対するモデルの気温の Q-Q プロット(6-8 月)。気象官署における観測(灰色)、JRA-25 からのダウンスケーリングデータ(青)、MRI-AGCM からの現在気候(赤)と将来気候(緑)のダウンスケーリングデータ。現在気候については 1979-2003 年で共通で、将来気候は 2075-2099 年である。

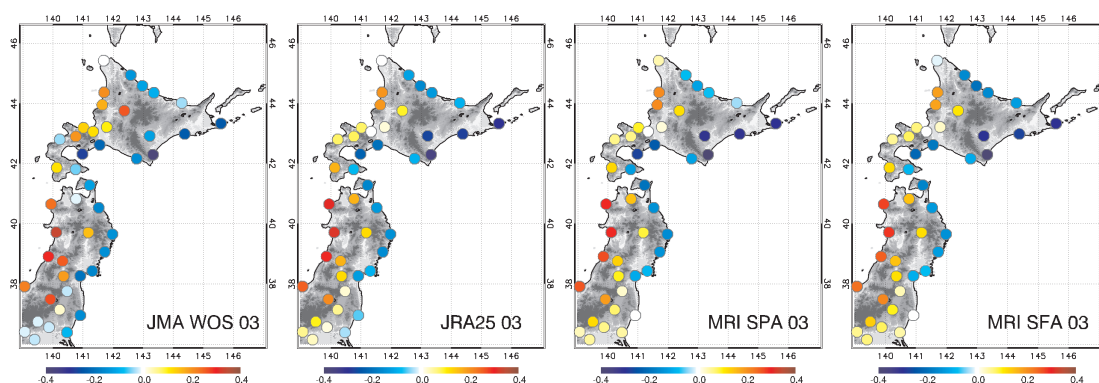


図 5 気象官署の観測地点における気温の主成分解析の第 3 モードの空間分布(6-8 月)。左から、気象官署の観測、JRA-25 からのダウンスケーリングデータ、MRI-AGCM からの現在気候と将来気候のダウンスケーリングデータを用いた結果を示す。

(3) ヤマセの将来変化

ヤマセが卓越する頻度が現在気候と将来気候でどのように違うかを、ヤマセインデックスを用いて検証した(図 6)。ここでも、ヒストグラムと観測に対する Q-Q プロットを示す。ヒストグラムの分布は各データセット間でほぼ一致し、極値にわずかな違いがみられるものの変動幅がほぼ同じであることがわかる。実際に、ヤマセ卓越時の海面気圧を比較してみる。図 1 と同様に、ヤマセインデックスが、1 標準偏差を越えるときのコンポジット場を作成した(図 7)。海面気圧の分布は、MRI-AGCM と JRA-25 からダウンスケーリングした現在気候、MRI-AGCM からダウンスケーリングした現在気候と将来気候は、ともに北高型の気圧配置を示す。上記の結果は、MRI-AGCM の現在気候の再現性の良さを示すとともに、気温上昇した将来気候においても、北高型の海面気圧分布が現在気候と同様に出現することを示す。つまり、ヤマセは将来気候においても現在気候とほぼ同じ頻度と強度で発生していると言える。ただし、将来気候と現在気候の差を詳しく見ると、将来気候においては、北日本付近では、約 1hPa の気圧傾度の弱化が示される。一方、地表気温分布についても、将来気候の平年偏差にしてみると、図 1 右にみられるようなヤマセ卓越時の気温分布が出現することがわかった。しかし、地形の東側の低温偏差は、現在気候に比べ 0.2℃ほど小さくなっていることを確認しており、ヤマセの弱化を示唆する。

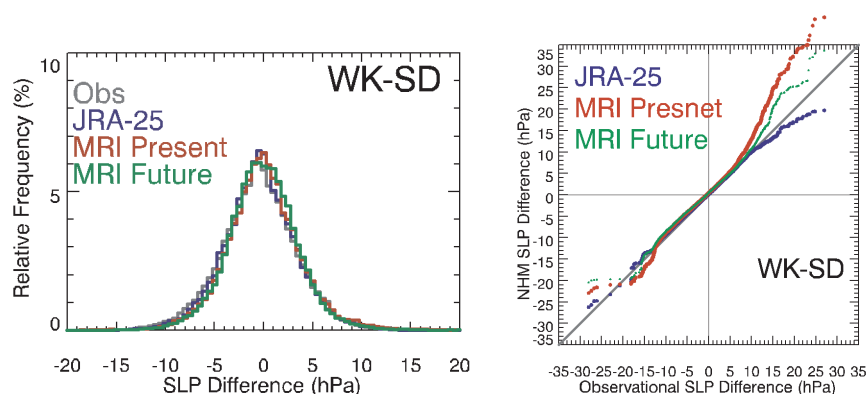


図 6 ヤマセインデックス(稚内と仙台の海面気圧差)のヒストグラムと観測に対するモデルの Q-Q プロット(6-8 月)。気象官署の観測(灰色)、JRA-25 からのダウンスケーリングデータ(青)、MRI-AGCM からの現在気候(赤)と将来気候(緑)のダウンスケーリングデータ。

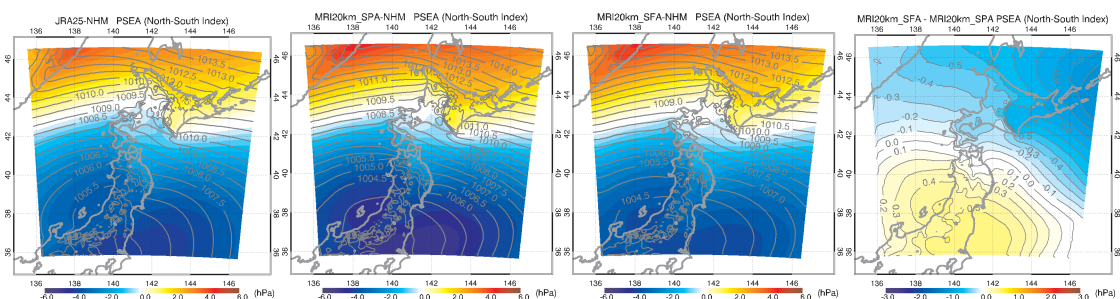


図 7 ヤマセ卓越時の海面気圧のコンポジット(コンター)と各データセットの気候平均からの偏差(カラー)。左から、JRA-25 からのダウンスケーリングデータ、MRI-AGCM の現在気候と将来気候からのダウンスケーリングデータ。右端は、MRI-AGCM の将来気候と現在気候の差である。

4. まとめと今後の方針

本稿では、東北地方の夏季気候に重要な役割を果たすヤマセの将来変化を解析するために、再解析データと気候モデルデータから 10km にダウンスケーリングしたデータセットの概要と解析結果を報告した。ダウンスケーリングにより、北日本の主要な地形を解像でき、日変化が表現できるデータセットを現在気候と将来気候について作成することができた。そして、MRI-AGCM の将来気候データでは、地球温暖化によって気温上昇した後も、将来気候の平年偏差でみるとヤマセが出現していると言え、下層大気に対する地形の影響は現在気候のものと同様であることがわかった。今後は、ヤマセの影響の地域性や顕著なヤマセ事例について解析する予定である。また、今回作成したデータセットは、その長所を活かして、例えば、農業気象分野における気候変動の影響評価(生育予測モデルへの適用等)への応用が期待される。

一方で、気候モデルによる将来気候の不確実性を考慮するために、別の気候モデルのダウンスケーリングを行うことを予定している。さらに、多数の気候モデルの中で、ダウンスケーリングに用いた気候シナリオと気候モデルの位置づけを把握することが重要である。ヤマセの再現性においても、気候モデルの再現性の差が大きいことが示されている[13]。ここでは、最新の気候モデルデータを用いて、オホーツク海南部の領域(東経140-155度、北緯45-50度)の平均海面気圧(オホーツク海高気圧の消長の指標となる)について、7月の統計(2075-2099年)を気候モデルごとに比較してみる(図8)。これらの気候モデルは、気候変動に関する政府間パネル(IPCC)の第5次報告書に用いられる、第5期結合モデル相互比較計画(CMIP5)の気候モデルのうち、本稿作成時に入手可能なものである。ここでは、CMIP5 に採用されている気候変動予測シナリオ(RCP: Representative Concentration Pathways)のうち、RCP4.5 のデータを用いた。気候モデル間で、平均値の差は最大約 7hPa にもなり、変動幅の違いも大きいことがわかる(RCP8.5 でも同様な傾向を示す)。MRI-AGCM は、この中ではオホーツク海高気圧の気圧が高い方から 11 番目に位置する。この位置は、現在気候の比較においても同じで、かつ再解析データ JRA-25 の結果に最も近いことを確認している。現在、これらの気候モデルの比較や解析利用が世界中で進んでいる。今後、各モデルの特徴(現在気候の再現性や気候感度)が明らかになるにつれて、ダウンスケーリングに利用した気候モデルの位置づけがより明確になってくると考えられる。

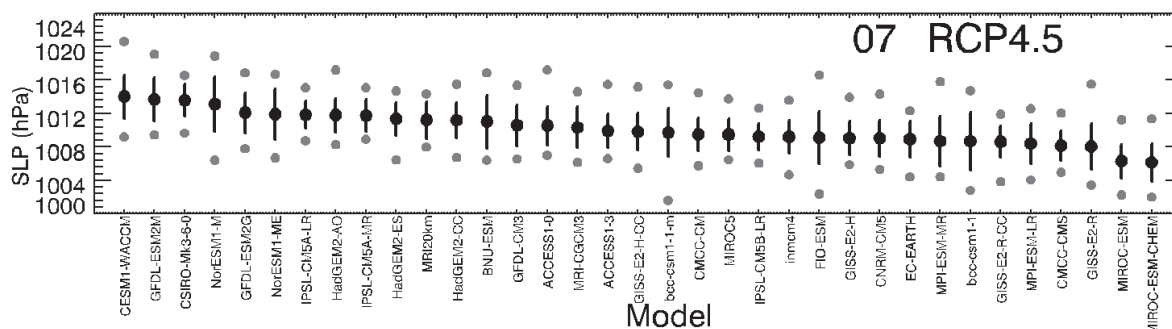


図8 CMIP5 の RCP4.5 シナリオを用いた気候モデルと MRI-AGCM(図中の MRI20km)から求めた 7 月平均のオホーツク海南部の領域平均海面気圧の 2075-2099 年の統計。黒丸と棒が平均値と ± 1 標準偏差、灰色の点が、最大値と最小値を示す。左から平均値の大きい順に並べた。

謝辞

本研究は、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を利用することで実

現することができた。また、研究にあたっては同センター関係各位から有益なご指導とご協力をいただいた。気候モデルデータは、気象研究所から提供いただいた。本研究は、文部科学省の委託事業「気候変動適応研究推進プログラム」に採択されている課題「東北地域のヤマセと冬季モンスーンの先進的ダウンスケール研究」（代表 岩崎俊樹）の一環として実施した。

参考文献

- [1] 環境省環境研究総合推進費 S-8 「温暖化影響評価・適応政策に関する総合的研究」
http://www.nies.go.jp/s8_project/index.html
- [2] 気候変動適応研究推進プログラム <http://www.mext-isacc.jp>
- [3] ENSEMBLES project <http://www.ensembles-eu.org>
- [4] North American Regional Climate Change Assessment Program (NCARCCAP)
<http://www.narccap.ucar.edu/index.html>
- [5] ト蔵建治, 2001: ヤマセと冷害: 東北稲作のあゆみ, 成山堂書店.
- [6] Saito, K., J. Ishida, K. Aranami, T. Hara, T. Segawa, M. Narita, and Y. Honda, 2007: Nonhydrostatic atmospheric models and operational development at JMA. *J. Meteor. Soc. Japan*, 85B, 271–304.
- [7] Onogi, K., J. Tsutsui, H. Koide, M. Sakamoto, S. Kobayashi, H. Hatsushika, T. Matsumoto, N. Yamazaki, H. Kamahori, K. Takahashi, S. Kadokura, K. Wada, K. Kato, R. Oyama, T. Ose, N. Mannoji and R. Taira, 2007: The JRA-25 Reanalysis, *J. Meteor. Soc. Japan*, 85, 369–432.
- [8] Reynolds, R. W., T. M. Smith, C. Liu, D. B. Chelton, K. S. Casey and M. G. Schlax, 2007: Daily High-resolution Blended Analyses for sea surface temperature. *J. Climate*, 20, 5473–5496.
- [9] Ishii, M., A. Shouji, S. Sugimoto, and T. Matsumoto, 2005: Objective Analyses of Sea-Surface Temperature and Marine Meteorological Variables for the 20th Century using ICOADS and the Kobe Collection. *Int. J. Climatol.*, 25, 865–879.
- [10] Murakami, H. and B. Wang, 2010: Future Change of North Atlantic Tropical Cyclone Tracks: Projection by a 20-km-Mesh Global Atmospheric Model. *J. Climate*, 23, 2699–2721.
- [11] Shimada, T., M. Sawada, and T. Iwasaki, Indices of cool summer climate in northern Japan: Yamase indices, in submission.
- [12] Kanno, H., 2004: Five-year cycle of north-south pressure difference as an index of summer weather in northern Japan from 1982 onwards, *J. Meteor. Soc. Japan*, 82, 711–724.
- [13] Endo, H., 2012: Future changes of Yamase bringing unusually cold summers over northeastern Japan in CMIP3 multi-models. *J. Meteor. Soc. Japan*, 90A, 123–136.

[共同研究成果]

Convergence Property of Flexible GMRES for the Method of Moments Based on a Volume–Surface Integral Equation

Hidetoshi Chiba : Mitsubishi Electric Corporation

Toru Fukasawa : Mitsubishi Electric Corporation

Hiroaki Miyashita : Mitsubishi Electric Corporation

Yoshihiko Konishi : Mitsubishi Electric Corporation

Qiang Chen : Tohoku University

Kunio Sawaya : Tohoku University

This paper presents flexible inner-outer Krylov subspace methods, which are implemented using the fast multipole method (FMM) for solving scattering problems with mixed dielectric and conducting object. The flexible Krylov subspace methods refer to a class of methods that accept variable preconditioning. To obtain the maximum efficiency of the inner-outer methods, it is desirable to compute the inner iterations with the least possible effort. To this end, we construct a less-accurate but much cheaper version of the FMM by intentionally setting the truncation number to a sufficiently low value, and then use it for the computation of inaccurate matrix-vector multiplication in the inner solver. The main focus of this study is to clarify the relationship between the overall efficiency of the flexible inner-outer Krylov solver and the accuracy of the FMM within the inner solver. Numerical experiments reveal that there exists an optimal accuracy level for the FMM within the inner solver, and that a moderately accurate FMM operator serves as the optimal preconditioner. The accomplishments presented in this paper would contribute to not only the facilitation of the utilization of numerical methods for electromagnetic wave problems but also the further development of the computer aided engineering (CAE) technology.

1. Introduction

Amongst various numerical methods, the method of moments (MoM) is one of the most powerful techniques for solving large-scale electromagnetic wave problems, and it is usually implemented in conjunction with iterative linear system solvers. Recently, in the area of scientific computations, Krylov subspace methods (KSM), which are used for iteratively solving linear systems, have enjoyed widespread success and popularity in scientific computing because of significant advantages such as low memory requirements and good approximation properties.

Since the fast multipole method (FMM) [1] does not explicitly generate the coefficient matrix, it has been still a challenging problem to establish a strategy to design an efficient preconditioner for iterative methods implemented with the FMM. In recent years, novel iterative methods, which are classified as so-called flexible KSM, have been proposed [2][3]. The flexible KSM refer to a class of methods which accept preconditioning that can change from one step to the next. Malas [4] and Fan [5] have studied the flexible KSM in the context of multipole techniques; the former takes into account only the near field interaction for the preconditioner, whereas the latter includes also the far field interaction expressed by the multipole expansion as well as the near field interaction for further enhancement of the preconditioner. However, the flexible KSM have been less extensively studied or practiced so far for the electromagnetic wave problems. In this paper,

we propose an efficient implementation of the inner-outer flexible GMRES (FGMRES) in the context of the multipole techniques. The main contribution of this paper is to clarify the relation between overall efficiency of the inner-outer FGMRES and the accuracy of the inner solver's matrix-vector multiplication (MATVEC).

2. Inner-Outer Flexible GMRES

The flexible Krylov subspace methods belong to a class of methods that allow variable preconditioning; in other words, these methods accept preconditioning that can vary at each iteration step. Assume that the system $\mathbf{Ax}=\mathbf{b}$ is solved, where $\mathbf{A}$ denotes the coefficient matrix, $\mathbf{b}$ is the right hand side vector, and $\mathbf{x}$ represents the unknown vector. In the conventional right-hand preconditioned algorithm, the right-hand preconditioning operation $\mathbf{z}=\mathbf{K}^{-1}\mathbf{v}$ is calculated in each iteration, where $\mathbf{z}$ and $\mathbf{v}$ are vectors and $\mathbf{K}$ is the preconditioning matrix. The preconditioner $\mathbf{K}$ must be a good approximation to $\mathbf{A}$, and it should be relatively cheap to construct. Here, the operation $\mathbf{z}=\mathbf{K}^{-1}\mathbf{v}$ can be considered to be a method for approximately solving $\mathbf{Az}=\mathbf{v}$. Hence, we can replace the computation $\mathbf{z}=\mathbf{K}^{-1}\mathbf{v}$ by roughly solving $\mathbf{Az}=\mathbf{v}$ with an iterative solver to obtain $\mathbf{z}$. Here, the iterative solver for the original linear system is generally referred to as the “outer” solver, and the iterative solver that performs the preconditioning is referred to as the “inner” solver. This inner-outer concept implies that different values of $\mathbf{K}$ are obtained at each step of the KSM; hence, the outer solver must be able to work with variable preconditioners. This is facilitated by the use of flexible KSM. In addition, we also note that the particular case where the flexible GMRES (FGMRES) is employed for the outer solver and the GMRES is used for the inner solver is usually called “inner-outer flexible GMRES.”

To obtain maximum efficiency of the inner-outer methods, it is desirable to compute the inner iterations with the least possible effort. Hence, in general, inaccurate MATVEC is performed in the inner solver within a short computation time. This is realized by using a particular feature of the multipole techniques. The accuracy and computational cost of the FMM can be controlled by selecting the truncation number, which indicates the number of multipoles used to express far-field interactions. On the basis of this fact, we construct a less-accurate but much cheaper version of FMM by intentionally setting the truncation number to a sufficiently lower value, and then use it for inaccurate MATVEC computation in the inner solver. Accordingly, two FMM operators with different levels of accuracy are used; the one is highly accurate and was used for the MATVEC within the outer solver, whereas the other operator, which is less accurate and a cheaper version of FMM, is employed for the computation of the MATVEC within the inner solver.

To control the accuracy and computational cost of the FMM operator, we introduce two parameters, L_{low} and p . L_{low} defines the truncation number for the lowest MLFMA level within the inner solver. The parameter p defines the increasing rate of the truncation number when the level increases from the lowest to the highest, and it indicates the overall accuracy and computational cost. By using these two parameters, we can define the truncation number for the FMM within the inner solver, L_i , as follows:

$$L_i = c(ka)^p, \quad (1)$$

where a denotes the cluster size of a level, and c is a constant that is pre-computed prior to the solver execution according to the value of L_{low} , that is, c is set such that L_i for the lowest MLFMA level becomes equal to L_{low} . We notice that for the determination of the two parameters L_{low} and p , both parameters should be set such that all the truncation

numbers for the inner iteration (L_i) are less than those for the outer iteration. From Eq. (1), it is inferred that p affects the overall accuracy and computational cost; as p increases, the FMM within the inner solver becomes more accurate and increasingly more expensive.

3. Numerical Experiments

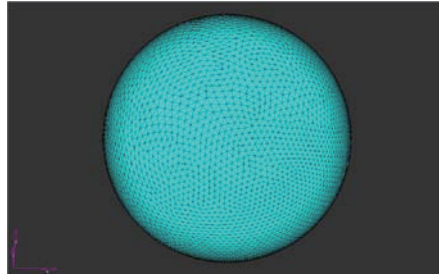
In this section, some numerical results will be presented to verify the efficiency of the inner-outer flexible GMRES with the proposed inner solver implemented in the multipole context.

We consider the following three geometries in the numerical experiments:

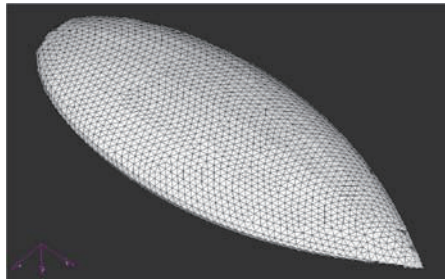
- (a) Dielectric-coated conducting sphere,
- (b) Dielectric-coated conducting NASA almond,
- (c) Frequency selective surface (FSS) structure.

The first geometry (a) is a dielectric-coated conducting sphere. Since the analytical solution (Mie series) is available for this test case, it provides a reference solution for evaluating the accuracy of our software. The core of the conducting sphere has a radius of 5λ , and the thickness of the dielectric layer, having a relative permittivity $\epsilon_r = 1.5 - i0.5$, is 0.25λ . The volume of the dielectric layer and the surface of the PEC sphere are discretized into 338,074 tetrahedrons and 53,432 triangular patches, respectively, leading to a total of 812,114 unknowns. In this test case, the solver generates four MLFMA levels with the truncation number for the outer solver's FMM being $\{10, 15, 24, 41\}$. Next, a dielectric-coated NASA almond is considered as the second test case (b). The dimensions of the PEC body are $32.02\lambda \times 12.35\lambda \times 4.20\lambda$, and the dielectric layer has a thickness of 0.1λ , with a relative permittivity $\epsilon_r = 1.5 - i0.5$. 93,042 tetrahedrons and 38,208 triangles are generated for the volume and surface region, respectively, resulting in a total of 274,463 unknowns. For this example, the MLFMA operator in the outer solver consists of five levels, and $\{9, 13, 20, 33, 58\}$ are used as the truncation numbers. The third test example (c) deals with an FSS structure that is $10\lambda \times 10\lambda \times 0.1\lambda$, which is discretized into 60,796 tetrahedrons and 206,400 triangles, where the degrees of freedom for the resultant linear system become 424,504. The relative permittivity of the dielectric layer is $\epsilon_r = 1.1$. This test case yields four levels MLFMA operator within the outer solver, with the truncation number being $\{10, 14, 23, 39\}$. All the truncation numbers are defined using the equation shown in [7]. This last example is the most realistic and complicated problem and is the most difficult to solve among the three geometries. In all of the cases, the scatterers are illuminated by an x-polarized and -z-traveling incident plane wave.

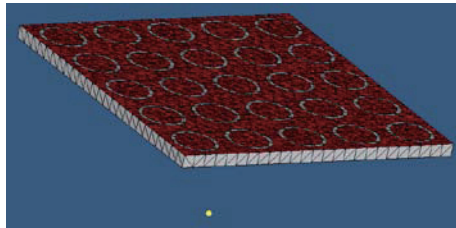
For the aforementioned three test cases, we conduct comparative experiments with respect to various sets of L_{low} and p introduced in the previous section. In addition, for comparison purpose, the strategies for the FMM operator within the inner solver employed in Malas [4] and Fan [5] as well as non-preconditioned GMRES case will also be investigated. Hereafter, we refer to the strategy proposed in [4] as “Malas’ strategy,” and that adopted in [5] as “Fan’s strategy.” The settings and conditions for the inner-outer flexible GMRES for all the strategies are summarized in Table 1. The same settings are used for all the three test cases. As shown in Table 1, we use two stopping criteria for both of the inner and outer solver; one of the criteria is error-bound (tolerance) and the other is the maximum number iterations. The settings for the Maras’ and Fan’s strategies shown in Table 1 are consistent with those in their original studies of [4] and [5].



(a) Dielectric-coated conducting sphere



(b) Dielectric-coated conducting NASA almond



(c) FSS structure

Fig. 1 Geometry of three test examples.

Table 1: Settings and conditions for inner-outer flexible GMRES.

(a) Proposed strategy

Inner GMRES	Maximum number of iterations	20
	Tolerance	0.1
	Definition of the truncation number	Eq. (23)
	Restart cycle	20
Outer FGMRES	Maximum number of iterations	1000
	Tolerance	1.0E-4
	Restart cycle	40

(b) Malas' strategy [4]

Inner GMRES	Maximum number of iterations	10
	Tolerance	0.1
	Definition of the truncation number	Eq. (2) in [4]
	Restart cycle	10
Outer FGMRES	Maximum number of iterations	1000
	Tolerance	1.0E-4
	Restart cycle	45

(c) Fan's strategy [5]

Inner GMRES	Maximum number of iterations	7
	Tolerance	0.1
	Definition of the truncation number	—
	Restart cycle	7
Outer FGMRES	Maximum number of iterations	1000
	Tolerance	1.0E-4
	Restart cycle	47

(d) Non-preconditioned GMRES

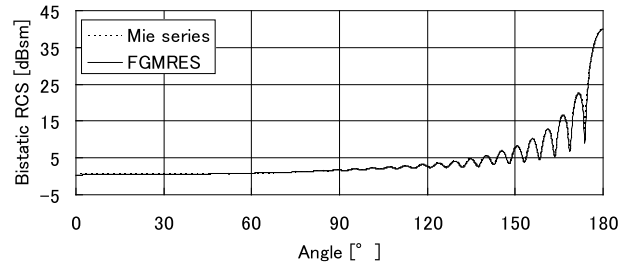
Inner GMRES	Maximum number of iterations	—
	Tolerance	—
	Definition of the truncation number	—
	Restart cycle	—
Outer FGMRES	Maximum number of iterations	2000
	Tolerance	1.0E-4
	Restart cycle	100

It should be pointed out that Malas' strategy has a noticeable feature in that the truncation number for FMM within the inner solver is equivalent to that for the FMM within the outer solver, and Fan's strategy does not take into account the far-field interaction expressed by the multipole expansion within the inner solver. It should be also noted that the comparison among all the strategies is fair with regard to memory requirements. In fact, for the same restart value, the storage requirements for FGMRES are twice that for the standard GMRES, because FGMRES also stores the preconditioned vectors of the Krylov basis as well as the original Krylov basis. Further, for the inner solver, we do not restart and perform a prescribed number of full GMRES iterations. All the runs have been performed in single precision on a parallel computer Express5800 of Cyberscience Center, Tohoku University, Japan.

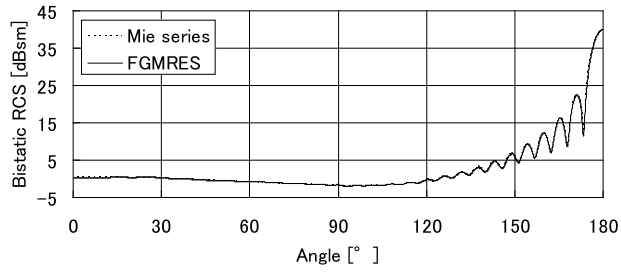
Fig. 2 displays the bistatic RCS for the dielectric-coated conducting sphere of test case (a), calculated by the inner-outer flexible GMRES and Mie series. It can be observed that inner-outer flexible GMRES agrees quite well with the Mie series for both polarizations, validating the accuracy of the solver code that we developed.

Table 2 tabulates the iteration times and the CPU time required for the convergence for the presented strategy with various sets of L_{low} and p , along with Malas' strategy [4], Fan's strategy [5], and non-preconditioned GMRES(100), and Fig. 3 compares the convergence history for the presented strategy with $(L_{low}, p) = (6, 0.75)$, Malas' strategy [4], Fan's strategy [5], and non-preconditioned GMRES(100). From Table 2, it can be inferred that the combination $(L_{low}, p) = (6, 0.75)$ perform quite well among all of the sets for all three test cases. Especially, it is worth noticing that, in the test case (c), the proposed method is the one that attains the solution, and all the other methods fail to converge. This observation reveals that there is an optimal accuracy for the FMM within the inner solver, and that a moderately accurate FMM operator is optimal. This implies that when the FMM within the inner solver is deteriorated, the inner solver can no longer serve as the preconditioner, and thus in both the cases, the total performance of the solver is not sufficiently improved. Additionally, it should be noted that the proposed strategy was proven to be superior to those of both Malas and Fan.

Table 3 lists the memory requirements for the presented strategy with $(L_{low}, p) = (6, 0.75)$, Malas' strategy, Fan's strategy, and non-preconditioned GMRES(100). Fan's strategy needs no extra memory storage as compared to non-preconditioned GMRES since it does not take into account the far-field interaction within the inner solver, thus, the memory requirements for Fan's strategy are always equal to that for the non-preconditioned GMRES. Malas' strategy has an advantage with regard to memory utilization as compared to the proposed strategy. This is because the truncation number for FMM within the inner solver is equivalent to that for the FMM within the outer solver; therefore, the $q2M$ and $L2p$ translators for the outer solver can be reused for the inner solver. Consequently, Malas' strategy does not require extra memory storage for the $q2M$ and $L2p$ translators for the inner solver. However, the strategy does not sufficiently reduce the CPU time, as shown in Table 2. On the other hand, the proposed strategy requires a slightly larger amount of memory storage compared to other methods; however, the impact is not very significant as compared to the basic requirements for the FMM operator within the outer solver. From the observations provided, we can state that the proposed strategy with $(L_{low}, p) = (6, 0.75)$ achieves the optimal performance with regard to the balance between the memory requirements and convergence behavior.



(a) V-V polarization



(b) H-H polarization

Fig. 2 Bistatic RCS of a dielectric-coated conducting sphere for V-V and H-H polarizations computed by inner-outer flexible GMRES and Mie series.

Table 2: Comparison of CPU time for presented strategy with various sets of L_{low} and p , along with Malas' strategy [4], Fan's strategy [5], and non-preconditioned GMRES(100);
Acronyms: N.C. $\equiv$ "not converged."

(a) Dielectric-coated conducting sphere

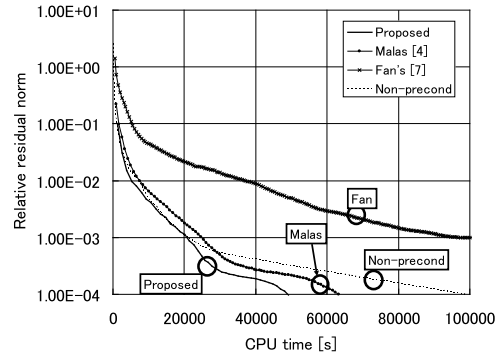
L_{low}	p	L_i	Iteration times	CPU time [s]
3	0.5	3, 4, 6, 8	200	260118.7 2
	0.85	3, 5, 9, 17	76	94765.04
	1.0	3, 6, 12, 24	72	61750.63
6	0.25	6, 7, 8, 10	115	140056.4 3
	0.55	6, 8, 12, 18	43	49121.40
	0.75	6, 10, 16, 28	40	45445.70
10	0.1	10, 10, 11, 12	N. C.	N. C.
	0.4	10, 13, 17, 22	39	51128.32
	0.58	10, 14, 22, 33	39	54802.14
Malas' [4]		10, 11, 12, 16	78	63192.40
Fan's [5]		0, 0, 0, 0	716	300452.3 9
Non-preconditioned GMRES(100)			1922	98959.89

(b) Dielectric-coated NASA almond

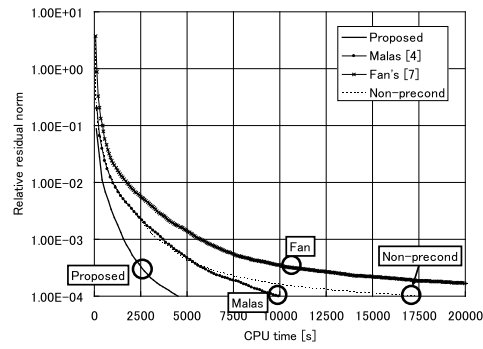
L_{low}	p	L_i	Iteration times	CPU time [s]
3	0.5	3, 4, 6, 8, 12	123	25499.16
	0.85	3, 5, 9, 17, 31	52	8840.75
	1.0	3, 6, 12, 24, 48	48	8567.96
6	0.25	6, 7, 8, 10, 12	116	21706.89
	0.55	6, 8, 12, 18, 27	29	6416.50
	0.75	6, 10, 16, 28, 48	27	4499.97
9	0.1	9, 9, 10, 11, 11	N. C.	N. C.
	0.4	9, 11, 15, 20, 27	29	6892.78
	0.58	9, 13, 20, 30, 44	27	9103.85
Malas' [4]		9, 9, 11, 13, 18	68	9927.53
Fan's [5]		0, 0, 0, 0	N. C.	N. C.
Non-preconditioned GMRES(100)			1056	17420.14

(c) FSS structure

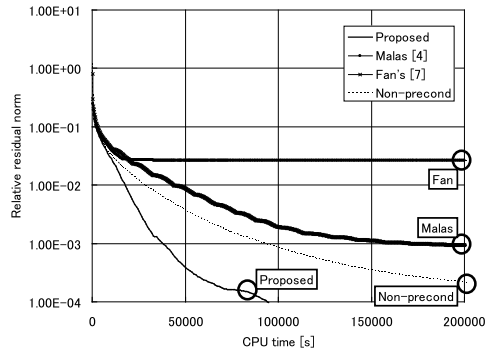
L_{low}	p	L_i	Iteration times	CPU time [s]
3	0.5	3, 4, 6, 8	N. C.	N. C.
	0.85	3, 5, 9, 17	N. C.	N. C.
	1.0	3, 6, 12, 24	225	206705.4 4
6	0.25	6, 7, 8, 10	N. C.	N. C.
	0.55	6, 8, 12, 18	149	113951.0 2
	0.75	6, 10, 16, 28	108	91884.92
10	0.1	10, 10, 11, 12	N. C.	N. C.
	0.4	10, 13, 17, 22	110	94949.39
	0.58	10, 14, 22, 33	107	99498.96
Malas' [4]		10, 11, 12, 15	N. C.	N. C.
Fan's [5]		0, 0, 0, 0	N. C.	N. C.
Non-preconditioned GMRES(100)			N. C.	N. C.



(a) Dielectric-coated conducting sphere



(b) Dielectric-coated NASA almond



(c) FSS structure

Fig. 3 Convergence history for presented strategy with $(L_{low}, p) = (6, 0.75)$, Malas' strategy [4], Fan's strategy [5] and non-preconditioned GMRES(100).

Table 3: Comparison of the memory requirements for presented strategy with $(L_{low}, p) = (6, 0.75)$, Malas' strategy, Fan's strategy, and non-preconditioned GMRES(100).

(a) Dielectric-coated conducting sphere

	Memory usage [GByte]
Proposed strategy with $(L_{low}, p) = (6, 0.75)$	46.7
Malas' strategy [4]	44.8
Fan's strategy [5]	44.7
Non-preconditioned GMRES(220)	44.7

(b) Dielectric-coated NASA almond

	Memory usage [GByte]
Proposed strategy with $(L_{low}, p) = (6, 0.75)$	9.09
Malas' strategy [4]	8.22
Fan's strategy [5]	8.12
Non-preconditioned GMRES(220)	8.12

(c) FSS structure

	Memory usage [GByte]
Proposed strategy with $(L_{low}, p) = (6, 0.75)$	25.2
Malas' strategy [4]	24.0
Fan's strategy [5]	23.9
Non-preconditioned GMRES(220)	23.9

4. Conclusions

In this paper, we have reported the performance of the inner-outer flexible GMRES, implemented in the context of FMM techniques. Specifically, we have investigated the relationship between the overall performance of the inner-outer flexible GMRES and the accuracy of the MVM within the inner solver. We introduced two parameters for controlling the accuracy and computational cost of the inner FMM operator with the solver. In the numerical experiments, we employed the volume-surface integral equation for solving scattering problems with mixed dielectric and conducting objects. These numerical experiments revealed that there is an optimal accuracy for the FMM within the inner solver, and that a moderately accurate FMM operator serves as an optimal preconditioner. By using the preconditioner with the optimal accuracy, even though we require a slightly larger amount of memory storage compared to conventional methods, the proposed method significantly enhanced the convergence behavior.

Acknowledgement

All the numerical results presented in this paper were obtained from the parallel computer Express5800 which is owned by Cyberscience Center, Tohoku University, Japan.

References

- [1] J. M. Song and W. C. Chew, “Multilevel fast-multipole algorithm for solving combined field integral equations of electromagnetic scattering,” *Micro. Opt. Tech. Lett.*, vol. 10, no. 1, pp. 14–19, Sept. 1995
- [2] Y. Saad, “A flexible inner-outer preconditioned GMRES algorithm,” *SIAM J. Sci. Comput.*, 14(1993), pp. 461–469.
- [3] Y. Notay, “Flexible conjugate gradient,” *SIAM J. Sci. Comput.*, 22 (2000), pp. 1444–1460.
- [4] T. Malas, Ö. Ergül, and L. Gürel, “Approximate MLFMA as an efficient preconditioner,” in *Proc. IEEE Antennas and Propagation Soc. Int. Symp.*, pp. 1289–1292, 2007
- [5] Z. H. Fan, R. S. Chen, E. K. N. Yung and D. X. Wang, “Inner-outer GMRES algorithm for MLFMA implementation,” 2005 *IEEE Antennas and Propagation Soc. Int. Symp. Digest*, Vol 4A, pp 463–466, July 2005
- [6] S. M. Rao, D. R. Wilton, and A. W. Glisson, “Simple and Efficient Numerical Methods for Problems of Electromagnetic Radiation and Scattering from Surfaces,” *IEEE Trans. Antennas Propagat.*, vol. AP-30, no. 3, may 1982
- [7] W. C. Chew, J.-M. Jin, E. Michielssen, and J. M. Song, eds., *Fast and Efficient Algorithms in Computational Electromagnetics*. Boston-London: Artech House Publishers, 2001.

Fortran スマートプログラミング

— 第3回 もっとスマートなプログラムにしよう —

田口 俊弘

摂南大学理工学部電気電子工学科

第1回でFortranプログラムの基本的書式とその考え方を示し、第2回でサブルーチンと入出力について説明しました。数値計算や計算機シミュレーション用のプログラムを書くための知識はここまでで十分です。数値計算とは、コンピュータの基本動作である“高速数値演算”と“大容量メモリの読み書き”を利用するのが主目的で、それに“入出力インターフェース”を加えれば処理は完結するからです。しかし、Fortranにはこの他にも様々な文法が用意されています。その中には、プログラムを書く手間を省いたり、メンテナンスを容易にしたり、コンピュータの動作を細かく制御するための書式が含まれています。最終回である今回は、この中から便利なものをいくつか紹介します。例えば、“配列演算”を使えば、数値の集合を一つの変数で代表し、集合の要素間の演算を、集合と集合の演算の形で記述することができます。また、変数宣言の拡張形や、文字列変数などについても紹介します。プログラム作りに慣れてきたら、このようなちょっと進んだ文法を使って、プログラムをさらにスマートにしてみましょう。

また、数値計算をするときには、高速フーリエ変換や多倍長計算など、自分で一から作成するのは難しいプログラムが必要になることがあります。幸いなことに、これらの多くはインターネットからダウンロード可能なサブルーチンライブラリとして公開されているので、最後にFortranから利用できる筆者がお勧めのライブラリをいくつか紹介します。

1. 知っておくと便利な文法

本節では、覚えておくと便利な書式や、これまでの基本的文法では記述することができない、コンピュータの細かな動作指定について説明します。

1.1 継続行と複文

計算式が非常に長くて、作成に使っているエディタウィンドウの横幅を越えると読みづらくなります。また、コンパイラによっては1行に書ける文字数に制限があるので、そのままではエラーになることもあります。そこで、1行の文を途中で改行して、複数行に分割して書くことができます。このとき、次の行に継続するという意志を示す印として、行末に“&”の文字を書きます。例えば、

```
print *, alpha, beta, gamma, delta, epsilon, zeta, eta, iota, kappa, lambda, mu
```

という1行は、

```
print *, alpha, beta, gamma, delta, epsilon &  
      , zeta, eta, iota, kappa, lambda, mu
```

と分割して書くことができます。継続行は何行にわたってもかまいません。

```
print *, alpha, beta, gamma &  
      , delta, epsilon &  
      , zeta, eta, iota &  
      , kappa, lambda, mu
```

と書いても同じです。最後の行に“&”を付加するとエラーになるので注意して下さい。

逆に、1行に複数の文を書くことも可能です。これを“複文”といいます。2個以上の文を1行で書くには、文と文を分離する記号として“;”の文字を書きます。例えば、

```
x = 1; y = 2; z = 3
```

のように書くことができます。ただし、複文の使用はこのような短い代入文を書く場合に限定して下さい。プログラムは1行で一つの完結した動作を表すのが基本です。1行に複数の文を書くと、動作の流れが見えにくくなるので、多用しない方が良いでしょう。

1.2 拡張された宣言文

これまで変数や配列の宣言に使っていた宣言文は、

```
型指定 変数 1, 変数 2, ...
```

という形式でした。ここで、“型指定”には“integer”とか、“real”のような数値型を記述し、“変数”には変数名か、配列名とその要素数を記述します。この宣言文は、データ領域におけるメモリの確保という意味もあります。

さて、Fortran90 から宣言文の記述形式が拡張されました¹。拡張宣言文を使うと、メモリを確保すると同時に初期値を代入したり、メモリの特性を指定することができます。拡張宣言文は以下の形式です。

```
型指定 [, 属性 1, 属性 2, ...] :: 変数 1 [=数値 1], 変数 2 [=数値 1], ...
```

ここで、角かっこ、“[”と“]”は、この中が省略可能であるという意味で使っているだけなので、角かっこ自体は書かないで下さい。拡張宣言文を使用するときは、型指定部と宣言する変数や配列の間にコロン2個“::”を書く必要があります。また、“属性”には宣言する変数の特性を指定するための予約語を記述します。属性も数値も省略して、単に型指定と変数の間に“::”を書いただけの宣言文は、書かずに宣言した旧来の書式と同じ意味になります。

属性を書かずに、単に数値を代入した形で宣言すると、その変数に指定した数値を代入して、プログラムの動作が開始することを意味します。例えば、

```
integer :: imax=10, jmax=100
real :: xx=1.0, yy=2.0
```

のように宣言すると、それぞれの変数に指定された値を代入した状態で動作が開始します。ただし、この宣言文中での数値代入は一度だけです。サブルーチン中で数値代入した変数を宣言しても、コールするたびに数値が代入されるものではありません。このため、その変数を実行文で変更すると、その変更した結果が残ります。例えば、

```
program stest1
  implicit none
  call subr1
  call subr1
  call subr1
end program stest1

subroutine subr1
  implicit none
  integer :: n=1
  print *,n           ! コールするたびに n は増加する
  n = n + 1
end subroutine subr1
```

というプログラムでは、メインプログラムでサブルーチン subr1 が3回コールされていますが、サブルーチン中の print 文の出力は、1回目が1、2回目が2、3回目が3、となります。宣言文における代入は、プログラム開始時の初期値だと考えて下さい。

さて、上記のサブルーチンの動作を考えると、一つ注意しなければならないことがあります。

¹ 本稿では、拡張された宣言文のことを“拡張宣言文”と呼んで、旧来の型指定だけの宣言文と区別します。

前回のサブルーチンの節で、ローカル変数は各ルーチンのデータ領域に所属しているという説明をしましたが、もう少し詳しく言うと、サブルーチンのデータ領域には2種類あります。一つは、サブルーチンがコールされた時点で一時的に生成されるメモリ領域で、もう一つは、サブルーチンの呼び出しに関係なく常に確保されたメモリ領域です。本稿では、前者を“一時メモリ領域”と呼び、後者を“固定メモリ領域”と呼ぶことにします。旧来の宣言文で宣言したローカル変数は、一時メモリ領域に所属します²。一時メモリ領域は、サブルーチンを呼び出すたびに場所や内容が変わる可能性があるため、同じサブルーチンを2回コールした場合、1回目のコールでローカル変数に代入した値が、2回目にコールした時点でそのまま残っているとは限りません。

例えば、

```
program stest2
  implicit none
  call subr2(1)
  call subr2(2)
end program stest2

subroutine subr2(n)
  implicit none
  integer n
  real x,y
  if (n == 2) print *,x,y      ! この出力値は不定
  x = 10.0
  y = 100.0
end subroutine subr2
```

のようなプログラムを書いたとします。1回目のコール文、call subr2(1)で、ローカル変数 x と y に、それぞれ 10.0 と 100.0 という値が代入されますが、2回目のコール文、call subr2(2)を実行したときに、print 文で出力した x と y が 10.0 と 100.0 になる保証はないのです³。ローカル変数に代入した値を保証するには、固定メモリ領域に所属させなければなりません。拡張宣言文を使って初期値を代入したローカル変数は、一時メモリ領域ではなく、固定メモリ領域に所属します。最初のサブルーチン例、subr1 の変数 n が、コールするごとに1ずつ増加した値を出力するのは、初期値 1 を代入して宣言することで固定メモリ領域に所属させたためです。

数値を代入せずに、ローカル変数を固定メモリ領域に所属させるには、変数に save 属性を指定します。例えば、上記のサブルーチン subr2 を以下のように書き換えれば、2回目のコールで 10.0 と 100.0 が出力されます。

```
subroutine subr2(n)
  implicit none
  integer n
  real, save :: x,y           ! save 属性を指定
  if (n == 2) print *,x,y
  x = 10.0
  y = 100.0
end subroutine subr2
```

変数の初期値が未定のときや、ローカル配列を固定メモリ領域に所属させたいときは、save 属性を使って下さい。また、初期値を代入する場合でも、固定メモリ領域に所属させることを積極的に利用するのであれば、次のように save 属性を付加して、固定メモリ領域にあることを明示した方が良いでしょう。

² これに対し、メインプログラムの変数とモジュール内のグローバル変数は、宣言文の記述にかかわらず、全て固定メモリ領域に所属します。

³ コンパイラによっては、全てのローカル変数を固定メモリ領域に所属させるものがあり、このプログラムでもうまくいく場合があります。しかし、文法的には保証されていないのですから、当てにするべきではありません。

```
integer, save :: n=1
```

その他の便利な属性をいくつか紹介します。まず、同型の配列を多数宣言するときのために、dimension 属性が用意されています。例えば、 10×10 の 2 次元実数型配列を 5 個用意するとき、通常の宣言では、

```
real a(10,10), b(10,10), c(10,10), d(10,10), e(10,10)
```

と書きますが、dimension 属性を使えば、これを次のように書くことができます。

```
real, dimension(10,10) :: a, b, c, d, e
```

すなわち、dimension 属性に配列の形状（次元や要素数）を書いておけば、宣言する変数の位置には、配列名を記述するだけになります。属性は複数付加することができるので、dimension 属性に save 属性を加えて宣言することも可能です。

もう一つ、配列を宣言するときに便利なのが、parameter 属性を指定した変数、parameter 変数です。parameter 変数には必ず値を代入しなければなりません。例えば、

```
integer, parameter :: imax=10, jmax=200
```

のように書くと、imax と jmax が parameter 変数になります。

さて、配列を宣言するとき、その要素数は整数定数で指定しなければなりません。このため、配列の長さを変更するときには、宣言した要素数に関連した数値を全て変更する必要があり、手間がかかるだけでなく、見落とししたり、書き間違える可能性があります。例えば、

```
real a(100),b(100)      ! 100 が 2 カ所
integer i
do i = 1, 100           ! 100 が 1 カ所
  a(i) = i*i
  b(i) = a(i)**2
enddo
```

というプログラムにおいて、配列要素数の 100 は、宣言文だけでなく、do 文の終了値にも入っています。このため、配列要素数を変更するときは 3 カ所変更しなければなりません。このとき、配列宣言の要素数を parameter 変数に代入する形で用意しておくと、配列要素数をその parameter 変数で置き換えることができます。例えば、上のプログラムは次のように書き換えることができます。

```
integer, parameter :: imax=100
real a(imax),b(imax)
integer i
do i = 1, imax
  a(i) = i*i
  b(i) = a(i)**2
enddo
```

このプログラムならば、imax に代入する数値を変更するだけで、配列要素数の変更は完了です。

parameter 変数を含んだ計算式を別の parameter 変数への代入値にしたり、配列宣言の要素数に使うこともできます。これらは、その計算式の結果で宣言したことに相当します。例えば、

```
integer, parameter :: imax=100, imax2=imax**2
real a(imax-1),b(imax*2),c(imax2)
```

という宣言は、以下の宣言文と同等です。

```
integer, parameter :: imax=100, imax2=10000
real a(99),b(200),c(10000)
```

ただし、他の parameter 変数に代入するときは、代入される変数（この例では imax2）が、代入

する計算式に使う変数（この例では `imax`）より後で宣言されなければなりません。

実は、`parameter` 属性を指定した変数は、“変数＝数値”の形式で変数名と数値の対応関係を示しているだけで、メモリとしての実体はありません。この対応関係を使った“変数”から“数値”への置き換えは、コンパイルの段階で行われるので、非実行文である宣言文でも使えるのです。逆に言えば、`parameter` 変数に実行文を使って代入することはできません。例えば、

```
integer, parameter :: imax=100
real a(imax),b(imax)
imax = 5           ! これはエラー
```

と書いた場合、“`imax=5`”は、コンパイルエラーです。これは、プログラムに“`100=5`”と書いたことに等しいのですから当然だと言えます。

コンパイル時に数値の置き換えをするのですから、代入という実行動作が不要になり、少しですが計算時間の短縮になります。また、同時にメモリの節約にもなります。そこで、計算で使う定数を `parameter` 変数にしておくとう便利です。例えば、

```
real, parameter :: dl3=1.0/3.0, pi=3.141592653589793, pi2=pi*2
real, parameter :: clight=2.99792458e8, hplanck=6.62606896e-34
```

のように、数学定数や物理定数を `parameter` 変数で用意しておけばいいでしょう。これらをモジュールの中に入れてグローバル変数にしておくと、さらに良いと思います。`parameter` 変数は、不用意に書き換えられることがないので、このような定数の定義にはうってつけです。

1.3 数値データの精度指定

第1回で、基本的な数値型には整数型と実数型があり、実数型には単精度実数型と倍精度実数型があるという話をしました。Fortran のデフォルト実数型は単精度ですが、コンパイラの自動倍精度機能を使えばデフォルトを倍精度実数型に変更できるので、本稿ではこれまで数値の精度に関して特に意識をしない説明をしてきました。しかし、大量のデータをバイナリ形式で保存するときには保存容量を圧縮するために精度を落とすことが考えられますし、“4倍精度実数”という倍精度よりも有効桁数の多い実数型を使って、高精度計算をすることもできます⁴。このことから、精度を指定する方法を知っておくと便利です。

変数の精度を指定するには、3種類の方法があります。ここでは、基本的な宣言文だけを紹介しますが、属性や数値代入を含む拡張宣言も可能です。

```
型指定*精度数 変数 1, 変数 2, ...
型指定(精度数) 変数 1, 変数 2, ...
型指定(kind=精度数) 変数 1, 変数 2, ...
```

ここで、“精度数”のところには、数値型の byte 数を整数で指定します。実数型の場合、単精度なら 4、倍精度なら 8、4倍精度なら 16 です。例えば、単精度実数型変数を宣言するには、

```
real*4 xs1,ys1,as1(100)
real(4) xs2,ys2,as2(100)
real(kind=4) xs3,ys3,ds3(100,100)
```

などと書きます。また、通常の実数型は 4byte ですが、最近は“倍精度整数”という 8byte の整数型を使うことができるので、これを利用するときには、

```
integer*8 n81,m81,k81(100)
integer(8) n82,m82,k82(100)
integer(kind=8) n83,m83,ka83(100,100)
```

などと書きます。

3種類の書式の中でどれが良いかは一概には言えません。一番目の“*”を使った書式は、古い

⁴ ただし、4倍精度実数型は、必ず実装しなければならない規格ではないようで、コンパイラによっては使えない場合があります。gfortran でも、バージョン 4.8 では使えますが、4.2 では使えないようです。

Fortran から使われているので、コンパイラが古い場合や、昔のプログラムを継承する場合のために覚えておいた方が良いでしょうが、これからプログラムを書き始める場合には、二番目か、精度数の意味を明記した三番目の方が良いでしょう。これらは、次のように精度数を parameter 変数を使って指定することができます。

```
integer, parameter :: kp=16
real(kp) xq2, yq2, aq2(100)
real(kind=kp) xq3, yq3, dq3(100, 100)
```

これに対し、“*”を使った書式では、必ず定数で指定しなければなりません。parameter 変数で精度指定をしておけば、計算機環境に応じて精度を変更する必要があるときに便利です。

ただし、高精度の計算をする場合は、定数も高精度にしなければなりません。例えば、“1.23”と書けば、デフォルトの実数型になるので、本稿では倍精度実数です。よって、このまま4倍精度の計算に使うと精度が落ちてしまいます。 $A \times 10^B$ で表される実数を4倍精度で表現するときは“ AqB ”と書きます。このため、“1.23”は“1.23q0”と書かなければなりません⁵。

しかし、parameter 変数で精度を指定しているときに、e や q のような文字で精度指定をしていると、精度を変更する際に、定数の変更が別途必要です。そこで、数値の後にアンダースコア “_” と精度数を付けて定数の精度を指定することができます。例えば、倍精度の“1.23”は“1.23_8”と書け、4倍精度の“1.23q0”は、“1.23_16”と書くことができます。この精度指定には parameter 変数を使うことができるので、次のように書くことができます。

```
integer, parameter :: kp=16
real(kp) xx, yy
xx = 1.23_kp
yy = 1.2345e-15_kp*xx**3
```

この例の yy に代入している実数の指数指定は e を使っていますが、精度数が 16 なので、4倍精度定数になります。

1.4 include 文

parameter 変数を使えば、配列の要素数だけでなく数値型の精度まで指定することができますが、それでもルーチンごとに宣言をしていると、変更が必要になったときに、使用している全てのルーチンで parameter 変数の値を変更しなければなりません。そこで、

```
module global_param
  integer, parameter :: imax=300, jmax=200
end module global_param
```

のように、parameter 変数だけを記述したモジュールを用意して、共有するルーチンで use 宣言をするのが良いでしょう。use 宣言はモジュールの中でも使うことができるので、以下のように記述することが可能です。

```
module global_param
  integer, parameter :: imax=300, jmax=200
end module global_param
module mod_array
  use global_param          ! モジュール内でも use 宣言可能
  real abc(imax, jmax), cd(jmax*2-1)
end module mod_array
program mtest1
  use mod_array              ! use global_param は不要
  implicit none
  integer km(imax)
```

⁵ ちなみに、デフォルト実数が単精度の時、倍精度実定数は“ AdB ”と書きます。例えば、“1.2”は“1.2d0”です。

ここで、あるモジュールの中で別のモジュールを use 宣言した場合、前者のモジュールを use 宣言したルーチンでは、後者のモジュールも合わせて use 宣言したことになります。上記のメインプログラム mtest1 ではモジュール mod_array しか use 宣言をしていませんが、mod_array の中で use 宣言をしているモジュール global_param も同時に宣言したことになり、その中にある変数 imax もメインプログラムで利用可能になります。ただし、次のように、メインプログラムに global_param の use 宣言を加えて、その使用を明示することもできます。

```
program mtest1
  use global_param          ! 書いてもエラーではない
  use mod_array
  implicit none
  integer km(imax)
```

これは、同じモジュールの use 宣言を重複して書いてもエラーにはならないからです。

さて、プログラムが完成して、後は問題に応じて parameter 変数を変更するだけになると、parameter 変数の変更のためだけにプログラム本体を変更するのは煩わしいし、変更するときに誤って本体を書き換えてしまうリスクもあります。プログラムを変更すると、プログラムファイルの日付が変更されるので、いつの時点で完成したプログラムかもわからなくなります。これらは、特に他人にプログラムを提供するときに問題となります。

そこで、parameter 変数の部分だけをプログラム本体から分離して保存し、必要に応じて結合すると便利です。結合する一つの手段に include 文の利用があります。include 文とは、以下のように include の後に、ファイル名を文字列で記述した文です。

```
include 'ファイル名'
```

include 文を書くと、コンパイラはその位置に“ファイル名”で指定したファイルの内容を挿入したプログラムを生成して、それをコンパイルします。例えば、上記のモジュール global_param を“global.inc”というファイルに書き込んで保存しておけば、次のように記述することができます。

```
include 'global.inc'      ! ここに global_param が挿入される

module mod_array
  use global_param
  real abc(imax, jmax), cd(jmax*2-1)
end module mod_array

program mtest1
  use mod_array
  implicit none
  integer km(imax)
  .....
```

include 文が指定するファイルの内容は、それを include 文の位置に挿入したときに、プログラム全体が正しく動作すればいいので、モジュールやサブルーチンのような完結したプログラムである必要はありません。また、同じファイルを一つのプログラムの複数の場所で include 指定することもできます。例えば、

```
integer, parameter :: imax=300, jmax=200
```

だけをファイルに書いて、サブルーチンごとに include 指定することも可能です。

2. 文字列の活用

これまで print 文や write 文に挿入して、数値の意味などを表示するのに使ってきた文字列ですが、この他にも様々な用途があります。また、固定した文字の並びだけではなく、文字列変数を使って、条件に応じてその内容を変更したり、文字列と文字列を連結したり、一部を取り出し

た文字列を作成するなどの“文字列演算”をすることもできます。本節では文字列をより積極的に活用する手法を紹介します。

2.1 文字列定数と文字列変数

Fortran における文字列とは 2 個の「'」または「"」で囲んだ文字の並びのことです。例えば、

```
'abc' 'Taguchi_T.' "(123.5678+X^2)" "漢字も書けます"
```

等が文字列です。文字列にはスペースや記号を入れることもできます⁶。「'」と「"」は同等なので、どちらを使うかは自由です。例えば、通常は「'」で囲い、文字列の中に「'」を使いたいときは、「"Teacher's"」のように全体を「"」で囲む、というように決めておけば良いでしょう。

コンピュータの“文字”は文字コードという整数値と対応していて、文字列は文字コードを表す整数の配列で実現されています。このため、整数定数の 1 と文字列の '1' は全く異なるものです。文字コードは半角英数字なら 1 文字あたり 1byte の整数、全角の漢字やひらがななら 1 文字あたり 2byte の整数です。半角英数字のコードは、現在ほとんど全てのコンピュータで ASCII コードが使われています。このため、半角英数字だけでプログラムを書けば移植性の問題はありません。しかし、全角文字は OS や利用環境によって文字コードが違う可能性があるため、プログラム中に日本語を記述すると、別のコンピュータに移したときに文字化けすることがあります⁷。Fortran において日本語が使えるのはコメント文とテキスト表示くらいなので、それほど重要ではありません。以下、文字列中の“文字”は 1byte の ASCII コードであると仮定して説明します。

「'」または「"」で囲んだ文字列は、“文字列定数”ですが、文字列を代入して保存する文字列の変数を作ることができます。文字列変数を作るには character 宣言を使います。character 宣言は、以下のような書式です。

```
character 文字列変数 1*文字数 1, 文字列変数 2*文字数 2, ...
```

文字数とは、文字列変数に代入可能な最大の文字数です。指定できる最小の文字数は 1 です。また、拡張宣言文にして属性などを付加することも可能です。例えば、

```
character c1*10, c2*20, cc*1, chr(20)*30, chs(100, 200)*50
```

と宣言すると、文字列変数 c1 には 10 個まで、c2 には 20 個までの文字を代入することができます。これに対し cc には 1 文字しか入りません⁸。また chr や chs のように文字列の配列を宣言することも可能です。chr は 30 文字まで入る 1 次元配列で、chs は 50 文字まで入る 2 次元配列です。

同じ文字数の文字列変数を複数個宣言するときは、以下の書式で宣言することができます。

```
character*文字数 文字列変数 1, 文字列変数 2, ...
character(文字数) 文字列変数 1, 文字列変数 2, ...
character(len=文字数) 文字列変数 1, 文字列変数 2, ...
```

ここで、“文字列変数”には、変数名か、配列名とその要素数を記述します。この 3 種類の相違点は、1.3 節の精度数の指定と同様で、最初の“*文字数”が旧来の書式、それ以外が新しい書式です。新しい書式の場合には、parameter 変数を利用して文字数を指定することができます。

文字列変数に文字列を代入するには、数値の代入と同じで、イコール「=」を使います。例えば、上記の 10 文字の文字列変数 c1 に文字列定数を代入するには、

```
c1 = 'abc'
```

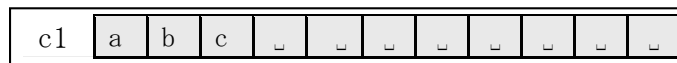
のように書きます。このとき、代入される文字 'abc' は 3 文字なので、10 文字の変数 c1 の先頭

⁶ 本節では半角スペース記号を明記するときに “_” を使います。

⁷ パソコンで使われているのは Shift JIS コードが多く、Linux などの UNIX 系 OS で使われているのは EUC コードが多いようです。最近では、コード体系をより統一化した Unicode を使う OS が増えてきました。

⁸ 文法的には“*文字数”を省略して変数名だけで宣言すると、文字数が 1 の文字列になります。しかし、プログラムがわかりにくくなるので、常に“*1”を付加して文字数 1 を陽に指定した方が良いでしょう。

から順に 1 文字ずつ代入され、残りの領域は半角スペースが代入されます。スペースも文字ですから、c1 の文字数は、代入した文字列の文字数にかかわらず、10 のままです。図に描けば、以下のようなイメージです。



逆に、代入する文字列の文字数の方が多い場合には、その文字列の先頭から、代入できる最大文字数までが代入され、残りは無視されます。拡張宣言文を使えば、次のようにあらかじめ文字列定数を代入した文字列変数を用意することも可能です。

```
character :: chr*10=' abcde'
```

文字列変数に代入された文字列は部分的に取り出すことができます。これを“部分文字列”といいます。部分文字列は、文字列変数の先頭から数えて何番目から何番目という範囲を「:」を使って指定します。例えば、c1 という文字列変数の n1 番目から n2 番目の文字を取り出すには、

```
c1(n1:n2)
```

と指定します。この文字列は、n2-n1+1 文字の文字列として扱われます。例えば、c1='abcdefg' ならば、c1(3:5) は、'cde' です。n1 と n2 を等しくすることで 1 文字を取り出すこともできます。例えば、

```
c1 = 'abcdefg'
do i = 1, 7
  print *, c1(i:i)
enddo
```

とすれば、1 文字ずつ縦に出力されます。

文字列配列の部分文字列を取り出す場合には、要素指定を先に、部分文字列指定を後に書きます。例えば、1 次元の文字列配列 chr に対し、k 番目の要素の部分文字列は、

```
chr(k)(n1:n2)
```

のように指定します。かっこが連続するので、順番に気を付けて下さい。

文字列は連結することもできます。文字列の連結には演算子「//」を用います。例えば、

```
c1 = 'abc' // 'xyz'
```

と書くと、c1 には 'abcxyz' という文字列が代入されます。文字列の連結は、文字列定数と文字列変数、文字列変数と文字列変数という組み合わせでも可能です。ただし、文字列変数に代入された文字列を連結するときには注意が必要です。例えば、

```
character c1*10, c2*20
c1 = 'abc'
c2 = c1 // 'xyz'
```

と書いた場合、c2 に 'abcxyz' という文字列が代入されると思ったら間違いです。正しくは

```
'abc          xyz'
```

が代入されます。これは、上記のように 10 文字の文字変数 c1 に 3 文字の 'abc' を代入しても、c1 の文字数は変わらないからです。末尾の不要なスペースを削除するには、部分文字列を使う必要があります。

```
character c1*10, c2*20
c1 = 'abc'
c2 = c1(1:3) // 'xyz'
```

このプログラムならば、c2 に 'abcxyz' が代入されます。

しかし、この方法は変数に代入されている文字数が不明のときには使えません。そこで、関数

trim が用意されています. trim は末尾のスペースを除去した文字列を返す組み込み関数です. 例えば, 上記の例は,

```
character c1*10, c2*20
c1 = 'abc'
c2 = trim(c1)//'xyz'
```

と書くことができます. この結果も c2 には 'abcxyz' が代入されます.

文字列をサブルーチンの引数にするときは, 「(*)」を指定して宣言します. 例えば,

```
subroutine subr3(chr)
  implicit none
  character(*) chr      ! 文字数は不要
```

の chr のように宣言します. Fortran の文字列には, 文字コードの並びだけではなく, 文字数の情報も含まれています. このため, (*) 指定のように文字数が明記されていなくても, コール側で指定した文字数の文字列として使用することができます. 例えば,

```
program ctest1
  implicit none
  call subr3('abcde')
end program ctest1

subroutine subr3(chr)
  implicit none
  character(*) chr
  print *, chr, len(chr) ! 文字列 'abcde' と文字数 5 が出力される
end subroutine subr3
```

のようなプログラムにおいて, サブルーチン subr3 の print 文の出力は, 文字列 'abcde' と文字数 5 になります. ここで, 関数 len は, 文字列の文字数を取得する関数です. 文字列に関する組み込み関数は, 2.4 節で説明します.

文字列には大小関係があり, これを利用して if 文で条件分岐をすることもできます. 2 個の文字列を比較するときは, 以下の手順で行います.

- (1) 文字数の長さが異なるときは, 短い方の文字列の末尾にスペースを追加して文字数を等しくする
- (2) 2 個の文字列を先頭から 1 文字ずつ比較して行って, 全て同じならば, “等しい (==)”
- (3) 異なる文字があれば, 最初に異なる文字の ASCII コードを比較して, コード値の大小が文字列の “大 (>)” または “小 (<)”

ASCII コードは, “スペース” < “数字” < “英大文字” < “英小文字” の順で大きくなり, 個々の文字には次のような大小関係があります.

```
'_' < '0' < '1' < ... < '9' < 'A' < 'B' < ... < 'Z' < 'a' < 'b' < ... < 'z'
```

例えば, 文字列の比較を使って次のようなプログラムを書くことができます.

```
character c1*10, c2*20
c1 = 'abcde'
c2 = 'abdce'
if (c1 < c2) print *, 'c1 < c2'
if (c1 == c2) print *, 'c1 == c2'
if (c1 > c2) print *, 'c1 > c2'

! 3 番目が 'c'
! 3 番目が 'd'
```

この結果は, 'c1 < c2' です. なぜなら, 3 番目の文字が初めて異なり, c1 は 'c', c2 は 'd' ですが, 'c' < 'd' だからです. なお, c1 は 10 文字, c2 は 20 文字ですが, 後ろにスペースを補うので, この例の結果には無関係です.

2.2 出力における文字列の利用

最もよく文字列を使う場面は、print 文や write 文の中に入れて表示の補助に使うことです。例えば、

```
real spd,pres
spd = 10.0
pres = 960.0
print *, ' Wind Speed = ', spd, ' Pressure = ', pres
```

と書けば、出力が

```
Wind Speed =    10.000000000000000    Pressure =    960.0000000000000
```

のようになって、どの数値がどの変数の値なのかが一目でわかります。

ここで、文字列はその長さに合わせて出力されています。これは、print 文が文字列に入っている文字数の情報に合わせて出力するからです。この機能は、format を使って文字列の出力指定をするときにも使うことができます。文字列の出力指定には、A 編集を用いますが、A 編集は、「a」だけ書くと、文字列の文字数が出力幅になります。例えば上記の print 文を、

```
print 600, ' Wind Speed = ', spd, ' m/s'
print 600, ' Pressure = ', pres, ' hPa'
600 format(a, f8.2, a)
```

と書けば、次のような文字幅に合わせた出力結果が得られます。

```
Wind Speed =    10.00 m/s
Pressure =    960.00 hPa
```

前回の format の説明で紹介しましたが、出力形式指定を print 文や write 文の中に埋め込む書式も、文字列の利用方法の一つです。例えば、

```
print 600, x
write(10, 600) x
600 format(' x=', es12.5)
```

というプログラムは次のように書き換えることができます。

```
print "(' x=', es12.5)", x
write(10, "(' x=', es12.5)") x
```

すなわち、format 文のかっこ以下を、両端のかっこを含めて文字列にして print 文や write 文の書式指定の位置に書き込みます。このとき、文字列変数を使えば、複数の場所で同じ format を使うときに便利だし、出力内容に応じて、実行時に format を変更することも可能です。例えば、

```
real x
character form*20
if (abs(x) >= 1.e5) then
  form = "(' x=', es12.5)"
else
  form = "(' x=', f10.5)"
endif
print form, x
```

とすれば、x の絶対値が 10^5 以上のときには es12.5 編集で、さもなくば f10.5 編集で出力されます。

また、部分文字列を利用して変更することも考えられます。例えば、

```
real x
character form*20
form = "(' x=', f10.5)"
if (x >= 100.0) form(8:11)=' 12.3'    ! 10.5 から 12.3 に変更
print form, x
```

のように書けば, x が 100.0 以上のときは f12.3 編集で, さもなくば f10.5 編集で出力されます.

2.3 数値・文字列変換

ここまでは write 文や read 文の書式指定の位置に文字列を使用していましたが, 装置番号の位置に文字列を記述することもできます. これは“文字列”から“数値”への変換, またはその逆変換を行うときに使用します. 上記のように 123 という数値と '123' という文字列は計算機内部の表現が異なりますが, 処理の過程で 123 という数値からそれに相当する文字を作ったり, 逆に '123' という文字列を数値として計算に利用したい場合があります. そもそも, 書式あり read 文はファイルなどから“文字で表現された数値”を入力して“計算機内部の数値”に変換する動作であり, 書式あり write 文は“計算機内部の数値”を“文字で表現された数値”に変換して出力する動作です. 装置番号の位置に文字列を利用すると, その変換機能だけを使うことができるのです.

“数値” → “文字列” 変換をするには write 文を用います. 例えば,

```
real x
character ch*20
x = 123.5
write(ch, "(f10.5)") x
```

と書けば, 文字列変数 ch に '123.50000' という文字列が代入されます. ただし, 文字列に変換するときは, 出力文字数以上の長さを持つ文字列変数を用意しておく必要があります.

open 文を使ってファイルをオープンするとき, ファイル名は文字列で指定する必要があります. このため, file01.dat, file02.dat, ..., file10.dat という通し番号の付いた 10 個のファイルが存在して, それぞれを装置番号 11~20 としてオープンするときは, 次のように書きます.

```
character name*20
integer m
do m = 1, 10
  write(name, "(' file', i2.2, '.dat')") m
  open(10+i, file=name)
enddo
```

ここで, 整数 m の出力編集を i2.2 にしているのは, スペースを '0' で埋めて, 整数の出力を 01, 02, ... のようにするためです.

逆に, “文字” → “数値” 変換をするには read 文を用います. 例えば,

```
real x
character ch*20
ch = '12345.0'
read(ch, *) x
```

と書けば, 実数型変数 x に 12345.0 という“数値”が代入されます.

2.4 文字列に関する組み込み関数

文字列に関する組み込み関数は, trim の他にも色々あります. 代表的なものを表 1 に示します.

表 1. 文字列に関する組み込み関数

文字列関数	引数の型	関数値の型	関数の意味
trim(c)	文字列	文字列	末尾の空白を削除
len(c)	文字列	整数	文字列の文字数
len_trim(c)	文字列	整数	末尾の空白を削除した文字列の文字数
adjustl(c)	文字列	文字列	文字列を左にそろえた文字列
adjustr(c)	文字列	文字列	文字列を右にそろえた文字列

表 1 (続き). 文字列に関する組み込み関数

文字列関数	引数の型	関数値の型	関数の意味
index(c, cp)	2 個の文字列	整数	文字列 c 中で文字列 cp が含まれていれば, その開始位置を返す. 無ければ 0 を返す
repeat(c, n)	文字列と整数	文字列	文字列 c を n 個連結した文字列
char(n)	整数	1 文字の文字列	ASCII コード値を与えると, それに対応する半角英数文字を返す
ichar(c)	1 文字の文字列	整数	半角英数文字を与えると, それに対応する ASCII コード値を返す

使用例をいくつか示します. 2.1 節に出てきた関数 len は, 文字列の文字数を取得するための関数ですが, 使用の際は注意が必要です. 例えば,

```
character c1*10
integer m, n
m = len('abc')    ! m = 3
c1 = 'abc'
n = len(c1)       ! n = 10
```

とすると, m の値は 3 ですが, n の値は 10 になります. これは, 文字列変数の文字数が宣言時の文字数だからです. 末尾のスペースを除去した文字列の長さを取得するには, 関数 trim を使って削除してから関数 len で調べるか, 関数 len_trim を使います. 例えば, 上のプログラムで, 最後の n への代入文を

```
n=len(trim(c1))   または   n=len_trim(c1)
```

で置き換えれば, n=3 になります.

最後の char と ichar の使用例を示します. この二つの関数を組み合わせれば, read 文や write 文を使わなくても, 文字と数値の変換をすることができます. 例えば, 2 桁の整数を 3 桁の 8 進数で表示するプログラムは以下ようになります.

```
integer zero, num
character c1*10
num = 12
c1 = '000'
zero = ichar('0')           ! '0' の文字コードを整数値に変換
c1(3:3) = char(zero+mod(num,8)) ! num の 1 の位の文字
c1(2:2) = char(zero+mod(num/8,8)) ! num の 8 の位の文字
c1(1:1) = char(zero+mod(num/64,8)) ! num の 64 の位の文字
print *, 'decimal(', num, ') = octal( ', trim(c1), ' )'
```

ASCII コードでは, '0', '1', '2', ..., '8', '9' の順で値が 1 ずつ増加しています. そこで, 先頭の '0' のコード値に 1 桁の数字を加えれば, その数字のコード値が得られます. 同様に, 大文字の英数字や, 小文字の英数字も 1 ずつ増加して並んでいるので, 先頭の文字である 'A' や 'a' のコード値に, その文字の順番の差の数字を加えて逆変換すれば, 対応する文字を得ることができます. 整数の 16 進数表示をするプログラムを考えてみてはいかがでしょうか.

3. 配列計算式

文字列の処理は, これまで説明してきた数値の計算と少し毛色が違います. 数値計算が 1 個の数値と 1 個の数値の間の演算を基本としているのに対し, 文字列の処理では, “文字列” という文字コードの集合をひとまとめにして取り扱い, 文字列と文字列の連結や文字列変数への代入を一つの式で実行することができます. 文字列とは文字コードの配列なのですから, これを数値計算的に処理するなら

ば、本来は1文字ずつ処理しなければなりません。文字列の演算が可能なのは、Fortran コンパイラが、“文字列”という数値集合の処理を、内部で1文字ずつ処理するプログラムに変換してくれるからです。

Fortran では、この概念を一般の配列に拡張した“配列演算”が可能です。配列演算とは、数値の集合を配列名で代表させ、配列要素と配列要素の演算を、配列名と配列名の演算の形で記述するものです。配列演算を使えば、do 文を使わずに配列を使った計算を記述することができます。これを“配列計算式”と呼びます。配列計算式を使う利点は、単にプログラムが短くなるだけではありません。計算が最適になるように、繰り返しの順序やメモリ配置をコンパイラが決めてくれるので、場合によっては、do 文を使って書くより高速に計算させることができます。

3.1 基本的な配列計算式

配列演算は、配列の全要素に対して、全て同じパターンの演算をするのが基本です。このため、1行の配列計算式で用いる配列は、次元と各次元の要素数が全て等しい、“同型の配列”でなければなりません。その点に注意をすれば、配列の全ての要素に同じ定数を代入するときや、2個の配列の対応する要素間で全て同じ四則演算をするときに、あたかも配列名が一つの変数であるかのような記述をすることができます。例えば、次のような記述が可能です。

```
real x(10,10), y(10,10), z(10,10)
x = 1.0
y = 2.0
z = x + y*3.5
```

このプログラムを実行すると、配列 x の全ての要素は 1.0 になり、配列 y の全ての要素は 2.0 になり、配列 z の全ての要素は 8.0 (=1.0+2.0*3.5) になります。このプログラムは do 文を使った以下のプログラムと同じ結果になります⁹。

```
real x(10,10), y(10,10), z(10,10)
integer i, j
do j = 1, 10
  do i = 1, 10
    x(i, j) = 1.0
    y(i, j) = 2.0
    z(i, j) = x(i, j) + y(i, j)*3.5
  enddo
enddo
```

配列計算式中の定数は、全要素に対して共通の値として計算します。

次のような、関数を使った配列計算式も可能です。

```
real a(10,10), b(10,10), c(10,10)
...
a = sqrt(b*sin(c)/(3.2*c + 1.5e-3))
```

この配列計算式を do 文で表すと、次のようになります。

```
do j = 1, 10
  do i = 1, 10
    a(i, j) = sqrt(b(i, j)*sin(c(i, j))/(3.2*c(i, j) + 1.5e-7))
  enddo
enddo
```

配列計算式においては、式中に含まれる全ての配列が同型でなければならないので、異なる次元や要素数の配列が混じっているとエラーになります。ただし、下限は異なってもかまいません。

⁹ ただし、ループの処理はコンパイラが自動的に生成するので、実際にこの通りの順序で計算するかどうかはわかりません。あくまでも、このプログラムと同じ結果になるという意味だと考えてください。

例えば,

```
real a(3,3), r(-1:1, 0:2)
...
a = 3.14*r**2
```

のように計算することができます。ただし、この配列計算式を do 文で表せば,

```
do j = 1, 3
  do i = 1, 3
    a(i, j) = 3.14*r(i-2, j-1)**2
  enddo
enddo
```

のように、対応する要素の位置がずれるので注意して下さい。

このように、配列計算式を使うとプログラムがシンプルになります。逆に、単一変数の計算式と配列計算式との区別がなくなるので、両者が入り交じるとプログラムがわかりにくくなり、エラーが見つげにくくなるという欠点もあります。

なお、サブルーチンの引数配列を使って配列計算をするときには注意が必要です。例えば,

```
subroutine subr(a, b)
  implicit none
  real a(*), b(*)
  a = b**2      ! これはエラーになる
  ...
```

のようなプログラムでは、配列計算式はエラーになります。なぜなら、“*”を入れて配列宣言をした場合は、要素数が不定だからです。これに対し,

```
subroutine subr(a, b, n)
  implicit none
  real a(n), b(n)
  integer n
  a = b**2      ! この場合は OK
  ...
```

のように、整合配列を使うと、配列計算が可能になります。

3.2 部分配列

配列名だけを使った配列計算式では全ての要素について計算をしますが、常に全要素の計算が必要とは限りません。そこで、配列の一部を取り出した配列、“部分配列”を指定することができます。部分配列は“:”(コロン)を使って要素番号の範囲を指定します。例えば、1次元配列 a に対して,

```
a(n1:n2)
```

と書けば、a(n1)~a(n2)という n2-n1+1 個の要素から構成された 1 次元配列として計算に使うことができます。また、2次元配列 b に対して,

```
b(n11:n12, n21:n22)
```

と書けば、b(n11, n21)~b(n12, n21)~b(n11, n22)~b(n12, n22)という (n12-n11+1) × (n22-n21+1) の 2 次元配列として計算に使うことができます。また、ある次元の要素番号を固定して,

```
b(n, n21:n22)
```

と書けば、b(n, n21)~b(n, n22)という n22-n21+1 個の要素から構成された 1 次元配列として計算に使うことができます。

配列計算式で部分配列を使うときは、その部分配列がその他の配列と同型であれば良いので、宣言文での次元や要素数が一致している必要はありません。例えば 1 次元配列なら,

```
real a(10),b(5)
...
a(3:5) = b(1:3)**2
```

のように書くことができます。また、2次元配列の部分配列を使って、

```
real a(10,10),b(5,4),c(10)
...
a(3:5,6:8) = b(1:3,1:3)**2
```

など書くことができます。この配列計算式を do 文で表せば、

```
do j = 6, 8
  do i = 3, 5
    a(i,j) = b(i-2,j-5)**2
  enddo
enddo
```

となります。ある次元の要素番号を固定した2次元配列は1次元配列として扱えるので、2次元配列と1次元配列が混在した計算も可能です。例えば、

```
real a(10,10),c(10)
...
c(3:8) = a(3,3:8)**2
```

と書くことができます。この配列計算式を do 文で表せば、

```
do j = 3, 8
  c(j) = a(3,j)**2
enddo
```

となります。

なお、要素のところに“:”だけを記述すると“その次元の全要素”という意味になります。そこで、全配列要素を使った配列計算式を書くときにも“:”を使って配列であることを明示することができます。例えば、前節の2次元配列計算式、

```
a = sqrt(b*sin(c)/(3.2*c + 1.5e-3))
```

は、次のように書くことができます。

```
a(:, :) = sqrt(b(:, :)*sin(c(:, :))/(3.2*c(:, :) + 1.5e-3))
```

この方が、単一変数の計算と区別できるので良いでしょう。本稿でもこれ以降はこの書式を使って記述します。

3.3 where 文による条件分岐

配列演算は便利ですが、全ての要素について同じ形の計算をするので、要素の条件に応じて異なる処理をさせることができません。そこで、配列要素の条件に応じて動作を分岐させるための where 文が用意されています。where 文は、以下のような形式です。

```
where (配列条件) 配列計算式
```

これは、“配列条件”に合った要素に対してのみ、“配列計算式”を実行するという意味です。例えば、

```
real a(10,10),b(10,10)
...
where (a(:, :) > 0) b(:, :) = a(:, :)**2
```

のように書きます。この where 文の部分を do 文で表せば、次のようになります。

```

do j = 1, 10
  do i = 1, 10
    if (a(i, j) > 0) b(i, j) = a(i, j)**2
  enddo
enddo

```

where 文はブロック構文にすることもできます. ブロック where 文では, 次のように else where 文を付加して, 条件に一致しない場合の記述をすることもできます.

```

where (配列条件 1)
  配列計算式 1
  .....
else where (配列条件 2)
  配列計算式 2
  .....
else where
  配列計算式 0
  .....
endwhere

```

この場合, 配列条件 1 を満足する要素は配列計算式 1 のブロックを実行し, 配列条件 1 を満足せず, 配列条件 2 を満足する要素は配列計算式 2 のブロックを実行し, ... という具合に続いて, 全ての条件を満足しない要素は配列計算式 0 のブロックを実行するという動作になります. 中間の else where に関するブロックは省略可能です. if 文と違って, “then” が不要なこと, “全ての条件以外”を表す文が “else where” であること, ブロックの最後に “endwhere” 文を付加すること, などに注意して下さい. 例えば,

```

real a(10,10), b(10,10)
...
where (3.0*b(:, :) > 0)
  a(:, :) = b(:, :)**2
else where
  a(:, :) = b(:, :)**3
endwhere

```

のように書くことができます. このブロック where 文を do 文で表せば, 次のようになります.

```

do j = 1, 10
  do i = 1, 10
    if (3.0*b(i, j) > 0) then
      a(i, j) = b(i, j)**2
    else
      a(i, j) = b(i, j)**3
    endif
  enddo
enddo

```

where ブロック中で使用する配列は, 全て同型でなければなりません. このため, where ブロックの中に, 単一変数の計算式や, 次元や要素数の異なる配列計算式を入れることはできません.

3.4 配列定数

配列演算を使えば, do 文を使わなくても配列要素の計算ができますが, ここまでの書式では配列要素ごとに異なる定数を使用することができません. そこで, 1 次元配列だけですが, 配列定数が用意されています. n 個の要素からなる 1 次元の配列定数は以下の形式です.

```
(/定数 1, 定数 2, ..., 定数 n/)
```

このとき, n 個の定数は全て同じ型の数値型でなければなりません. 例えば,

```
real a(5)
a(:) = (/1.0, 2.0, 3.0, 4.0, 5.0/)
```

と書けば、右辺は実数型の 1 次元配列定数であり、この配列計算式は、 $a(1)=1.0$ 、 $a(2)=2.0$ 、 $a(3)=3.0$ 、 $a(4)=4.0$ 、 $a(5)=5.0$ という 5 個の代入文を実行したことに相当します。この配列定数は、次のように拡張宣言文の初期値代入に使うこともできます。

```
real :: a(5)=(/1.0, 2.0, 3.0, 4.0, 5.0/)
```

しかし、この形式では数値を一個一個書かねばならず、要素数が多いと大変です。そこで、前回の入出力文の説明に出てきた、“do 型形式”を使って定数設定をすることができます。これを do 型定数といいます。do 型定数とは、以下のような形式です。

(データ 1, データ 2, ..., 整数型変数=初期値, 終了値)

この形式では、まず整数型変数（カウンタ変数）を初期値にして、データ 1、データ 2、... と並べ、次に、カウンタ変数を 1 増加して、再度、データ 1、データ 2、... と並べ、カウンタ変数が終了値になるまで繰り返して得られる配列定数になります。このため、配列要素数は“データ並びの数×カウンタ変数のカウント回数”になります。例えば、

```
(/ (n, n=1, 5) /)
```

という do 型定数を使った配列定数は、整数型の配列定数、

```
(/1, 2, 3, 4, 5/)
```

と同じです。また、

```
(/ (n, n**2, n=1, 5) /)
```

という do 型定数を使った配列定数は、整数型の配列定数、

```
(/1, 1, 2, 4, 3, 9, 4, 16, 5, 25/)
```

と同じです。複数の do 型定数を並べたり、通常の変数と混在させることも可能です。例えば、

```
(/ (real(n), n=0, 2), 1.5, 1.8, (real(n), n=3, 5) /)
```

という実数型の配列定数は、

```
(/0.0, 1.0, 2.0, 1.5, 1.8, 3.0, 4.0, 5.0/)
```

と同じです。このとき、型変換関数の $\text{real}(n)$ を単に n と書くとエラーになるので注意して下さい。これは、 n が整数型なので、 n だけを書くとその配列定数の中に実数型と整数型が混在するからです。

do 型定数は多重にすることも可能です。例えば、

```
(/ ((i+j, i=1, 3), j=1, 4) /)
```

という do 型定数を使った整数型の配列定数は、

```
(/2, 3, 4, 3, 4, 5, 4, 5, 6, 5, 6, 7/)
```

と同じです。多重にした場合、カウンタ変数は内側が先に進みます。

本節最初の例は、次のように do 型定数を使って書くことができます。

```
real a(5)
a(:) = (/ (n, n=1, 5) /)
```

ここで、右辺は整数型の配列で、左辺は実数型の配列ですが、この場合は全要素に対して整数型から実数型への変換が行われて代入されるのでエラーにはなりません。なお、do 型定数は拡張された宣言文での初期値代入に使うこともできますが、カウンタ変数はその宣言文より前に宣言されている必要があります。

配列定数は 1 次元配列しか用意されていません．このため，2 次元以上の配列計算で使うときは，1 次元ごとに計算する必要があります．例えば，

```
real a(3,4)
integer i,j
do j = 1, 4
  a(:,j) = (/sin(0.5*(i+j)), i=1,3)/
enddo
```

のように書きます．これは以下のプログラムと同等です．

```
real a(3,4)
integer i,j
do j = 1, 4
  do i = 1, 3
    a(i,j) = sin(0.5*(i+j))
  enddo
enddo
```

ただ，配列計算で do 文を使うのはあまりスマートではありません．そこで，配列の形状を変換する組み込み関数 `reshape` が用意されています．`reshape` を使って 1 次元の配列定数を多次元配列に変換すれば，そのまま多次元の配列計算式に記述できます．この方法は次節で説明します．

3.5 配列に関する組み込み関数

配列を引数に持つ組み込み関数は色々ありますが，主として配列情報に関する関数を表 2 に示します．

表 2．配列に関する組み込み関数

配列関数	引数の型	関数値の型	関数の意味
<code>sum(a)*</code>	配列	配列要素の型	全ての配列要素の和
<code>product(a)*</code>	配列	配列要素の型	全ての配列要素の積
<code>minval(a)*</code>	配列	配列要素の型	全ての配列要素の最小値
<code>maxval(a)*</code>	配列	配列要素の型	全ての配列要素の最大値
<code>minloc(a)</code>	配列	整数型配列	全ての配列要素の最小値の位置
<code>maxloc(a)</code>	配列	整数型配列	全ての配列要素の最大値の位置
<code>lbound(a)*</code>	配列	整数型配列	配列の各次元の最小要素番号リスト
<code>ubound(a)*</code>	配列	整数型配列	配列の各次元の最大要素番号リスト
<code>shape(a)</code>	配列	整数型配列	配列の各次元の要素数リスト
<code>size(a)*</code>	配列	整数	配列の全要素数
<code>reshape(a, s)</code>	配列と整数型配列	a の型の配列	配列 a を配列 s で指定された型の配列に変換する
<code>allocated(a)</code>	配列	論理	配列 a が割り付けられていれば真，さもなければ偽

ここで，* の付いた関数は，引数配列の後に次元を示す整数型の引数を追加して，その次元に関する値を取り出すこともできます．例えば，

```
real array(3,4)      ! 3×4 の配列
integer m,n1,n2
m = size(array)      ! 全要素数 3×4=12
n1 = size(array,1)   ! 第 1 次元要素数 3
n2 = size(array,2)   ! 第 2 次元要素数 4
```

と書けば，`m` には，`array` の全要素数である 12 が代入され，`n1` には `array` の第 1 次元の要素数で

ある 3 が, `n2` には array の第 2 次元の要素数である 4 が代入されます.

表 2 で, 関数値の型が“整数型配列”になっている関数は, 引数に与えた配列の次元を要素数に持つ 1 次元整数型配列が戻り値です. その際, その 1 次元配列の各要素には, 引数配列の各次元の情報が代入されています. 例えば, 下限と上限を取得するための関数, `lbound` と `ubound` は以下のように使います.

```
real bb(-10:10,4)      ! 2 次元配列
integer blow(2), bupp(2) ! 2 次元なので要素数 2 の配列
blow = lbound(bb)
bupp = ubound(bb)
print *, blow, bupp
```

ここで, 2 次元配列 `bb` は, 第 1 次元の下限が -10, 上限が 10, 第 2 次元の下限が 1, 上限が 4 ですから, 定数で書けば, `blow` は `(/-10, 1/)`, `bupp` は `(/10, 4/)` になります. このとき, `ubound(bb, 2)` のように, 次元に関する引数を加えると, その結果は一個の整数 (この例では 4) になります.

`reshape` は, 配列の形状を変換する関数です. そもそも, どんな次元の配列もメモリ上では 1 次元的に並んでいるので, 2 次元配列 `a(3, 4)` と 1 次元配列 `b(12)` のように, 全要素数が等しい配列は同じように取り扱えるはずですが, 配列演算は, 次元および各次元の要素数が等しい, 同型の配列間でしか許可されていません. そこで, `a(3, 4)` と `b(12)` の間で演算をするには, 形式上, 同型になるような変換が必要です. これを実行するのが関数 `reshape` です. `reshape` には, 最初の引数に変換したい配列を与え, 2 番目の引数に変換後の形状を表す 1 次元配列を与えます. ここで形状を表す 1 次元配列とは, 変換後の配列が, $k_1 \times k_2 \times \dots \times k_n$ の n 次元配列であれば,

`(/k1, k2, ..., kn/)`

与えられる要素数 n の 1 次元整数型配列のことです. 例えば, 3.4 節最後の例文は,

```
real a(3,4)
integer i, j
a(:, :) = reshape((/(sin(0.5*(i+j))), i=1, 3), j=1, 4)/), (/3, 4/))
```

と書くことができます. `reshape` の第 1 引数は `do` 型定数なので 1 次元配列ですが, これを第 2 引数の `(/3, 4/)` で 3×4 の 2 次元配列に変換するよう指定しているわけです.

この配列の形状を表す 1 次元配列は, 関数 `shape` を使って取得することができます. 例えば, 上記の `a(3, 4)` という宣言をした配列に対し, `shape(a)` は, 1 次元整数型配列, `(/3, 4/)` になります. そこで, 上記の配列計算式は次のように書くことができます.

```
a(:, :) = reshape((/(sin(0.5*(i+j))), i=1, 3), j=1, 4)/), shape(a))
```

しかし, これではまだ `do` 型変数における `i` や `j` の範囲指定が定数のままです. そこで, ここは先ほど説明した関数 `size` で書き直すと良いでしょう.

```
a(:, :) = reshape((/(sin(0.5*(i+j))), i=1, size(a, 1)), j=1, size(a, 2))/), shape(a))
```

この配列計算式なら, 宣言文の要素数を変更するだけで変更は完了です.

3.6 配列の動的割り付け

配列を宣言するときには, 整数定数を与えて要素数を明示しなければなりません. これは, その情報に基づいて, プログラムの動作開始時に計算で使用するメモリ量を確定するためです. しかし, 最近のコンピュータではプログラムの実行中に必要になったメモリを確保したり, 不要になったメモリを解放するという動作が頻繁に行われています. このような, プログラムの実行中に必要に応じてメモリを確保することを“動的割り付け”といいます. Fortran も Fortran90 から動的割り付けが可能になり, プログラムの動作に必要な要素数を持つ配列を実行中に確保できるようになりました. 動的割り付けを使うと, メモリ効率の良い汎用性のあるプログラムを作成することができます. ここでその方法を紹介します.

まず、サブルーチンの中で必要に応じた要素数の配列を確保する簡単な方法として、サブルーチンの引数を使った配列宣言があります。例えば、次のサブルーチンで real 宣言されている 2 次元配列 b がこれに相当します。

```
subroutine memory1(a,m,n)
  implicit none
  real a(m,n),b(m,n),x           ! 引数 m と n を使って宣言する
  integer m,n,i,j
  x = 10.0
  b(:, :) = x*reshape((/(i*j, i=1, m), j=1, n)/), shape(b))
  a(:, :) = b(:, :)**3
end subroutine memory1
```

配列 b は、このサブルーチンがコールされたときに、引数 m と n の値を使って確保されます。このとき、b(m,n) のような単純な宣言だけではなく、b(0:2*m, n-1) のような、下限指定や計算式を使った宣言も可能です。なお、配列 a も引数を使って宣言されていますが、a は引数に含まれているので前回説明した整合配列です。整合配列は、引数配列の取り扱いを決める仕組みにすぎず、配列自体はサブルーチン側にありません。

引数に含まれていない配列 b はローカル配列であり、宣言されたサブルーチンの一時メモリ領域に所属します。1.2 節で説明したように、一時メモリ領域に所属するローカル変数は、サブルーチン呼び出した時点で生成されます。引数の値はサブルーチン開始時に確定しているので、これを使って配列に必要なメモリを確保することが可能なのです。ただし、これはあくまでもサブルーチンで宣言する配列についてのみ使える機能であり、メインプログラムの配列やモジュール中のグローバル配列としては使えません。

そこで、メモリの動的割り付けを明示的に行う仕組みが用意されています。Fortran で動的割り付けを行うには、二つの手続きが必要です。一つは、割り付けたメモリを配列として使用するための配列名の宣言です。これは、割り付けたメモリの先頭アドレスを保持するためのメモリを用意することだと考えられます。もう一つは、その配列名に、実際のメモリを割り付ける動作です。前者は宣言文なので非実行文、後者はプログラム実行中に割り付けるので実行文です。

メモリを割り付ける予定の配列名は、allocatable 属性を付けて型宣言をします。このとき、配列の次元情報をコロン “:” を使って示す必要があります。配列の次元はコロンの数で決まり、実行時に変更することはできません。例えば、

```
real, allocatable :: ab(:), z2(:, :)
integer, allocatable, dimension(:, :) :: km
```

のように宣言します。1 行目のように、名前の後ろに次元情報を付加しても良いし、2 行目のように、dimension 属性を使うことも可能です。この例の場合、ab は 1 次元実数型配列、z2 は 2 次元実数型配列、km は 2 次元整数型配列として割り付けることができます。

配列の割り付けは allocate 文で行います。allocate 文は、次のように配列の要素指定をサブルーチンのようにかっこで囲んで記述します。

```
allocate ( ab(100), z2(0:m, -m:m), km(k-1, 2*k) )
```

かっこ内は、配列宣言と同じ形式です。ab や km のように、要素数に整数だけを与えると、その次元の下限は 1 になるし、z2 のように “:” を使って下限指定をすることも可能です。割り付けるときは、配列の変数型が混在していてもかまいません。allocate 文は実行文なので、z2 や km のように整数型変数や整数式の計算結果を使って割り付けることも可能です。一旦割り付けられた配列は、通常の宣言文で宣言した配列と同様に使用することができます。

ただし、むやみに割り付けを繰り返すと、コンピュータで利用できる容量を超える “メモリオーバー” になる可能性があります。そこで、不用になった配列のメモリ領域は deallocate 文で解放することができます。deallocate 文は、次のように配列名だけをかっこで囲んで指定します。

```
deallocate ( ab, z2, lm )
```

メモリの割り付けや解放はそれほど時間がかかる処理ではないので、必要なときに割り付けて、不要になったら解放するようにしましょう。例えば、先ほどのサブルーチン `memory1` は、以下のよう書き換えることができます。

```
subroutine memory1(a,m,n)
  implicit none
  real a(m,n), x
  real, allocatable :: b(:, :)      ! 割り付け用 2次元実数型配列
  integer m,n,i,j
  allocate (b(m,n))                ! m×n の 2次元配列を割り付け
  x = 10.0
  b(:, :) = x*reshape((/(i*j, i=1,m), j=1,n)/), shape(b))
  a(:, :) = b(:, :)**3
  deallocate (b)                    ! メモリの解放
end subroutine memory1
```

なお、引数変数を使って宣言したローカル配列は一時メモリ領域に所属しますが、`allocate` 文で割り付けた配列は固定メモリ領域に所属します¹⁰。サブルーチンにおける一時メモリ領域は固定メモリ領域より容量が少ないことが多いので、要素数の大きな配列を割り付けるときは `allocate` 文を使う方が良いでしょう。`allocate` 文を使った動的割り付けは、メインプログラムでも利用できるし、モジュールの中で配列宣言をして、グローバル配列として使用することも可能です。

ただし、`allocate` 文で割り付けられた配列を、解放しないで再度 `allocate` 文で割り付けようとするると実行時エラーになります。また、割り付ける前にその配列を使用しようとしても実行時エラーになります。そこで、配列がすでに割り付けられているかどうかを確認する関数 `allocated` が用意されています。`allocated` は論理型の値を返す関数で、`allocatable` 属性を持つ配列を引数に与えると、その配列がすでに割り付けられていれば“真”，割り付けられていなければ“偽”を返します。論理型の関数は、そのまま `if` 文の条件として使うことができるので、例えば次のように利用することができます。

```
real, allocatable :: a(:)
integer i,n
.....
if (allocated(a)) then
  a(:) = (/(100*i, i=1,n)/)      ! 代入 1
else
  allocate ( a(n) )
  a(:) = (/(200*i, i=1,n)/)      ! 代入 2
endif
```

この場合、配列 `a` がすでに割り付けられていれば“代入 1”を実行し、割り付けられていなければ、`allocate` 文で割り付けてから“代入 2”を実行します。

4. 数値計算を補助するフリーのサブルーチンライブラリ

本稿における Fortran プログラミングの説明は以上です。これまで紹介した文法を使って色々なプログラムを書いてみて下さい。ただ、複雑な計算アルゴリズムを使ったプログラムをすべて一から書くのはかなりの手間がかかります。アルゴリズムによってはインターネット上にサブルーチンライブラリとして公開されているので、これらを利用してプログラムを作ると手間が省け

¹⁰ ただし、サブルーチンを複数回コールするために固定メモリ領域に所属させることを利用する場合には、配列名の宣言に `save` 属性を加えて、配列名も固定メモリ領域に所属させなければなりません。

ます。ここでは筆者が使ったことのあるライブラリの中からお勧めをいくつか紹介します¹¹。

4.1 高速フーリエ変換

偏微分方程式の解法やデータ解析で良く使う高速フーリエ変換 (Fast Fourier Transform, FFT) は、アルゴリズムが複雑で、効率の良いプログラムを自作するのは大変です。筆者はもっとも簡単なレベルのプログラムを作ったことはありますが、最近ではネットで公開されているものを使っています。お勧めなのは、FFTW (Fastest Fourier Transform in the West) です。

```
http://www.fftw.org/
```

このライブラリには、Fortran から使用できるサブルーチンが入っています。

4.2 多倍長計算

Fortran での倍精度実数は 8byte で、有効数字は 15 桁程度です。4 倍精度実数は 16byte で有効数字が 30 桁程度に増加しますが、それ以上は無理だし、コンパイラによっては 4 倍精度実数が使えないこともあります。筆者は多項式の根を計算するときに高精度計算が必要になったことがあります。数十次の多項式を計算するときは、最大次数の項と最小次数の項との数値の差が非常に大きくて、桁落ちが深刻な問題になるからです。

このときに便利なのが多倍長計算ライブラリです。これは、任意の精度で実数計算をすることができます。通常の計算より処理に時間がかかりますが、精度優先のときにはやむを得ません。ネット上には何種類か公開されていますが、筆者が使って便利だったのは、FMLIB です。

```
http://myweb.lmu.edu/dmsmith/FMLIB.html
```

これは Fortran のモジュールになっていて、四則演算を “+−*/” の記号で書けるし、sin や cos などの関数も充実しています。本稿では説明していない構造体の知識が少しだけ必要ですが、変数宣言程度なのでそれほど難しくはありません。筆者は、倍精度計算用に作った多項式の根を計算するプログラムを、それほど大きな変更をすることなく多倍長で動作させることができました。

4.3 乱数発生

計算機シミュレーションには、時間発展の方程式を解くことで未来を予測する決定論的手法と、サイコロを振って確率的に将来の可能性を探る確率論的手法があります。後者の確率論的手法で、サイコロに相当する働きをするのが乱数です。Fortran には、乱数発生用の組み込みサブルーチン random_number が用意されているので、

```
call random_number(x)
```

のように書けば、乱数 x を得ることができます。しかし、乱数は周期長が問題で、周期の短い乱数を使うと良いシミュレーション結果が得られません。random_number では周期長が不足するときにお勧めなのは、周期が非常に長い乱数として有名な、Mersenne Twister です。これを考案した松本真さんが以下のサイトでプログラムを公開されています。

```
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html
```

松本さんが公開されているのは C 言語で書かれたものですが、リンク集の中に Fortran 用の公開先が入っています。

4.4 数値計算ライブラリ

連立方程式の解法や数値積分など、一般的な数値計算ライブラリがあれば、数値計算のプログラムを作るのが楽になります。ただ、残念ながら筆者はフリーの良いライブラリを知りません。ネットで公開されていて使えそうなものに、GSL (GNU Scientific Library) があります。

```
http://www.gnu.org/software/gsl/
```

¹¹ サイバーサイエンスセンターの計算機には商用のサブルーチンライブラリがインストールされています。詳しくは、“<http://www.ss.isc.tohoku.ac.jp/super/library.html>” をご覧下さい。

ただし、基本的にC言語用なので、Fortran で使うには若干テクニックがいるようです。また、ちょっと使ってみました、必ずしも速くありません。ベッセル関数などの特殊関数は、その高速バージョンを公開しているサイトからダウンロードした方が良いでしょう。筆者は、数値計算の教科書を読んで原理から自作するか、参考文献[2, 3]に掲載されているものをコピーして使うことが多いです。

4.5 グラフィックライブラリ

計算機シミュレーションは物理的に起こっている現象を模擬するものですが、出力される数値だけで現象を読み取るのは容易ではありません。そこで必要不可欠なのが可視化ソフトです。フリーの可視化ソフトとして有名なものに gnuplot があります。

<http://www.gnuplot.info/>

プログラムの結果をファイルに保存しておけば、gnuplot からそれを読み込んで、様々なグラフに加工して表示することができます。

最近のパソコンは基本操作が全てビジュアルであり、グラフィック出力機能はOSに組み込まれています。Fortran にはグラフィック出力に関する文法はありませんが、他の言語で書かれたグラフィックライブラリとリンクすれば、Fortran プログラムからサブルーチンをコールする形式で図を描くことが可能です。その一つに、筆者が作った Frames があります。

<http://www.pp.teen.setsunan.ac.jp/frames/>

Frames は、Fortran で使うためのグラフィックツールとして開発してきたもので、本稿で説明した知識だけで簡単に2次元や3次元のグラフを描くことができます。また、描画した図形をイメージやアニメーション形式で出力して、プレゼンテーションに使うことも可能です。良かったら試してみてください。

連載を終えるに当たって

今回は、前回までに説明した書式の拡張と、文字列や配列計算式の使い方を説明しました。Fortran には、この他にも色々な文法が用意されていて、構造体を使ったオブジェクト指向的なプログラミングも可能ですし、最新の Fortran2008 には並列処理の記述まで入っています。しかし、筆者が計算機シミュレーションをするために使っている文法は、この3回の内容でほぼ全てです。数値計算や計算機シミュレーションは、計算機を効率良く動かしさえすればそれで良いのであり、新しい文法は必ずしも必要ではありません。コンピュータの動作とプログラムとの関係が理解できれば、後はいくらでも複雑な計算をさせることができます。Fortran を使ってコンピュータを“スマートに”使いこなし、数値計算や計算機シミュレーションをもっともっと楽しんでもらえればと思います。

本稿は、筆者が摂南大学の卒研生向けに書いた Fortran のテキストがベースになっています。これを、大阪大学レーザーエネルギー学研究センターの福田優子氏が Web で公開できる Fortran テキストを探しているというので提供したところ、回り回ってこちらで連載させていただくことになりました。最後に、このような機会を与えて下さった福田優子氏に深い感謝の意を表して、本稿を終えたいと思います。

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
- [2] Numerical Recipes in Fortran 77, Second Edition, W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Cambridge University Press, 1996.
- [3] Numerical Recipes in Fortran 90, W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Cambridge University Press, 1996.

[全国共同利用情報基盤センター研究開発論文集 No. 34 より]

東北大学サイバーサイエンスセンターにおける ユーザコードの高速化支援活動

佐々木大輔[†], 山下毅[†], 小野敏[†], 大泉健治[†]
江川隆輔[‡], 小林広明[‡]

[†]東北大学 情報部 情報基盤課

[‡]東北大学サイバーサイエンスセンター スーパーコンピューティング研究部

1 はじめに

東北大学サイバーサイエンスセンター(以下, 本センター)は, その前身の一つである大型計算機センターの設立(1969 年)以来, 研究室のレベルを遙かに超える最高性能の計算機システムを設置し, 最先端の学術研究を強力に支援してきた. さらに, 利用者にとって使い勝手の良いシステムの構築, 他では実行できない大規模プログラムの実行環境の整備を行うと共に, スーパーコンピュータの性能を最大限に発揮することを目的に, 利用者, 本センター教員・技術職員, ベンダーが一体となってプログラムの高速化技術および新しいシミュレーション技術の研究・開発に取り組み, 計算科学・計算機科学の発展に貢献してきた. 本稿では, はじめに本センターのシステム概要および自動バッファリング機能を持つコンパイラへのアップデート, 次に共同研究を通じた高速化支援活動の取り組みについて述べる.

2 大規模科学計算システム

2.1 システム概要

本センターでは, 全国のベクトル型計算機ユーザの高いニーズに応えられるシステム構成を目的に, ベクトル型スーパーコンピュータSX-9とスカラ型並列コンピュータExpress5800を運用している.

ベクトル型スーパーコンピュータSX-9は1ノードあたり16個のCPUで構成され, シングルコアのCPUは1CPUあたり102.4Gflop/sの演算性能を有する. また, 1ノードあたり1Tバイトの共有メモリと1CPUあたり256Gバイト/秒と高いデータ転送能力を有し, 大規模かつ高性能な処理を可能にしている. ベクトル型のスーパーコンピュータは, 高いメモリバンド幅とループ中で繰り返し処理されるような大規模な配列データを一括して演算実行することで, 流体解析や気象解析, 電磁界解析をはじめとするメモリバンド幅に対する要求の高い大規模科学シミュレーションを高効率で実行可能である.

Express5800はSX-9システムのフロントエンドサーバとして機能するばかりでなく, 汎用のアプリケーションや, ベクトルスーパーコンピュータに適さないアプリケーションの実行に用いられている. このように, 本センターではベクトル・スカラと2種類のスーパーコンピュータを配備・運用することで多様なユーザの計算要求に応えられる計算サービスを提供している.

2.2 コンパイラによるADB向け最適化の評価

本センターは, FortranコンパイラおよびC/C++コンパイラに対し年に数回のアップデートを実施し, 最新の環境を利用者に提供している. 本年は既に2回のアップデートを行い, 機能強化, 機能追加およびバグ修正に対応している. ベクトル化促進等の最適化強化や性能解析ツールの強化が主な内容だが, 本年はADBを用いた自動ベクトルデータ・バッファリング機能を新たに追加している.

2.2.1 ADB(Assignable Data Buffer)

SX-9ではCPUと主記憶装置間にADBと呼ばれるベクトルデータをバッファする機構が新たに追加されている。図1にADBの概念図を示す。ADBはCPU—主記憶装置間よりメモリバンド幅が広く、レイテンシ（遅延）が短いという特徴を持っている。また、通常のキャッシュメモリとは異なり利用者が選択的にデータをバッファリングすることが可能である。ベクトル計算機は、大きいメモリバンド幅を持ち、高いデータ供給能力を有する。データ供給能力は、ベクトル演算性能あたりのメモリバンド幅であるB/F(Bytes/FLOP)で表され、これまでのベクトル計算機は4B/Fと高いデータ供給能力を有していた。ところが、近年マイクロプロセッサの様々な設計制約により、SX-9のデータ供給能力は2.5B/Fに低下している。そこで、ADBを用いる事で、SX-9のデータ供給能力を2.5B/Fから4.0B/Fに向上させることが可能になる。

2.2.2 自動ベクトルデータ・バッファリング機能

ベクトルデータ・バッファリングとは、再利用性のあるベクトルデータをADBに保持することである。プログラム中で頻繁にアクセスされる配列をADBに保持することでデータ供給能力が向上し、アプリケーションの高速化が期待できる。これまで、バッファリングをするには指示行を用いて、保存する配列を指定する必要があり、ADBを効果的に利用するには利用者の判断によるところが大きかった。しかし、本年3月のコンパイラアップデートにより自動ベクトルデータ・バッファリング機能が追加され、FortranおよびC/C++ともにバッファリングの自動化が可能になった。センターで利用されている各種プログラムを、自動バッファリング機能の有無で比較した結果を図2に示す。前コンパイラは自動バッファリング”なし”，現コンパイラは自動バッファリング”あり”の場合の性能を示している。各プログラムとも実行時間にして概ね数パーセントの性能向上が得られていることがわかる。性能が低下しているプログラムも一部見られるがこれはスカラデータ・命令がベクトルデータによりADBから追い出されたためと考えられる。この結果から、本センターで実行されている多くのプログラムが自動化の効果をめられると判断し、規定値で自動化を有効とし利用者への提供を開始している。

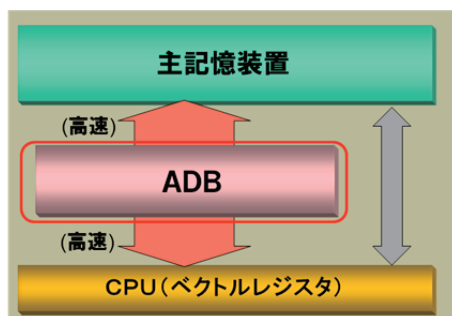


図1 ADBの概念図

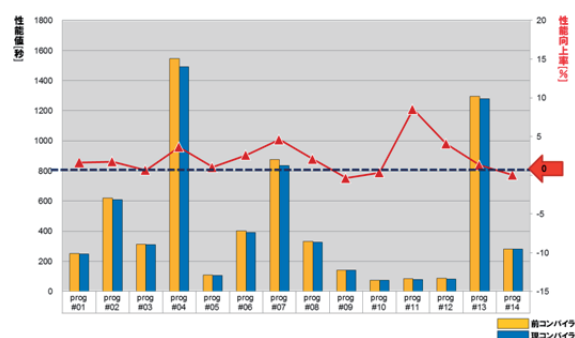


図2 自動バッファリング機能の有無による性能比較

3 高速化支援活動

スーパーコンピュータシステムの潜在能力を最大限に享受し、大規模かつ高精度な計算機シミュレーションを実現するためには、コンピュータシステム、およびプログラミング技術に関する高度な知識が求められるのが実情である。このような状況下で、本センターでは「最先端、かつ世界最高性能クラスのコンピュータシステムの導入、利用環境の整備・運用・研究・開発」という使命を果たすために、1997年

より利用者・センター運用スタッフ（技術系職員・スーパーコンピューティング研究部教員）、ベンダーの技術者から構成される高速化支援体制を整備している。

本センターでは、この支援体制のもと、自動ベクトル化、自動並列化機能のみに頼ることなく、実シミュレーションコード解析に基づく臨床学的研究開発を行い、ユーザプログラムの高速化支援活動に取り組んできた。表 1 に1999年から2012年にかけて本活動を通して、取り組んできた高速化支援の結果を示す。この間、合計163件のプログラムの高速化に取り組み、平均で単体性能向上比は39.3倍、並列性能向上比12.2倍を実現しており、高速化支援が高い成果を上げていることがわかる。

さらに、1997年から全国に広がる本センターのユーザ（計算科学者）とセンター（計算機科学者）が連携し、サイバーサイエンスセンター共同研究として131件の共同研究を推進している。また、2008年からは文部科学省先端研究施設共用促進事業、2010年からは大規模情報基盤共同利用・共同研究拠点の一拠点として、全国の計算科学者らとの共同研究も推進している。次に、本年度の代表的な高速化支援活動例をとりあげ説明する。

表 1 高速化実績

年	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
件数	8	9	10	7	18	20	8	29	10	15	8	8	13
単体性能 向上比	4.5	2.5	1.6	2.2	6.7	2.9	1.5	3.1	33.0	9.3	381.0	47.0	16.2
並列性能 向上比	31.7	8.6	4.9	2.8	18.6	4.5	4.1	8.0	1.9	5.1	3.6	48.0	17.2

3.1. ベクトル化率促進による高速化

ベクトル処理は DO ループによる配列データの繰り返し演算を、ベクトルユニットを用いて一括に実行するものである。ベクトル型スーパーコンピュータの高い実効性能を得るためには、プログラム全体に対するベクトル処理可能な割合であるベクトル化率を、出来る限り 100%に近づける必要がある。また、ベクトル処理を高速に行うためには、主記憶装置からベクトルユニットに対して、データを滞りなく供給する必要がある。

DO ループのベクトル化を阻害する要因としては、1) 配列要素の定義・引用関係に依存関係がある 2) 配列要素が確定されない IF・CASE 文を含む 3) 配列データのアクセス時に複雑な間接参照を伴う 4) DO ループ中の関数・サブルーチン呼び出しおよび入出力文がある、などが挙げられる。

データ供給の遅延の要因としては、1) CPU 内における同一 CPU ポートへのロード・ストアの集中 2) 同一のメモリバンクへのアクセス競合および CPU と主記憶装置間の経路上で発生するアクセス競合 3) 主記憶装置への不連続なメモリアクセス、などが挙げられる。メモリアクセスの効率化では、前述の ADB を利用することで、メモリバンド幅を補うことが可能となり、性能向上を図れる場合がある。

最適化例として以下で取り上げるユーザコードは、Red Black 法を用いて SOR 法の反復計算を行うものである[1]。オリジナルコードはユーザにより OpenMP による並列化が行われていた。SX-9 の簡易性能解析ツール (Ftrace) によりプログラムの実行プロファイルを取得したところ、プラズマ流の各支配方程式を解くためのサブルーチン A でベクトル化率が低いため、実効性能が低いことが分かった。このサブルーチンのベクトル化率の促進と、OpenMP による並列化の促進について検討を行った。

3.1.1 IF 文の除去によるベクトル化と作業配列の導入による並列化の促進

図 3 に示すサブルーチン A のオリジナルコードでは 5625 行目の if 文が真の場合 5626-5633 行が実行され、さらにこの中の if 文(5628 行目)が真になると RSMAX や IRMU, JRMU, KRMU が更新される構造である。

この構造のままではコンパイラは 5628 行目の if 文に依存関係が存在すると判断し、ベクトル化が行われない。5625 行目の if 文の結果はループ中で不変なので、ループの外側に移動することが可能である。また最外側ループ K を並列化するために作業配列 WKRSMAX を導入した。この作業配列 WKRSMAX により、最外側ループ K における各最大値を保存することができ、OpenMP 指示行による並列化が可能になった。

図 4 に SX-9 の 16CPU でサブルーチン A のオリジナルコードと修正コードを実行した際の Ftrace 結果を示す。オリジナルコードは並列化されていないため 1CPU の値、修正コードは 16CPU (スレッド) の合計値である。修正コードはベクトル化率 98.95% となり、16CPU 合計の FLOPS 値が約 6.5 倍向上した。その結果サブルーチン A の実行時間は約 1/24 に短縮された。

(オリジナルコード)	
5620:	RSMAX=ZERO
5621: +----->	DO K=1, KEND
5622: +----->	DO J=2, JEND
5623: +----->	DO I=2, IEND
5625:	IF (IPHI.EQ. IU) THEN
5626:	RSMU=RESP/RSUO
5627:	RSU(I, J, K)=RESI(I, J, K)/RSUO
5628:	IF (DABS(RSU(I, J, K)).GT. RSMAX) THEN
5629:	RSMAX=DABS(RSU(I, J, K))
5630:	IRMU=I
5631:	JRMU=J
5632:	KRMU=K
5633:	END IF
(略)	
(修正コード)	
6179:	IF (IPHI.EQ. IU) THEN
6180:	RSMAX=ZERO
6181:	RSMU=RESP/RSUO
6196:	!\$OMP PARALLEL DO
6197: P----->	DO K=1, KEND ! P は並列化された DO ループ,
6198:	WKRSMAX(K)=ZERO ! V はベクトル化された DO ループを示す。
6199: +----->	DO J=2, JEND
6200: V----->	DO I=2, IEND
6201:	RSU(I, J, K)=RESI(I, J, K)/RSUO ! Non-dimensionalization
6202:	IF (DABS(RSU(I, J, K)).GT. WKRSMAX(K)) THEN
6203:	WKRSMAX(K)=DABS(RSU(I, J, K))
6204:	IWIR(K)=I
6205:	IWJR(K)=J
6206:	IWKR(K)=K
6207:	END IF
6208: V-----	END DO
6209: +-----	END DO
6210: P-----	END DO
6211:	!\$OMP END PARALLEL DO

図 3 サブルーチン A のオリジナルコードと修正コード

PROC. NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER. TIME [msec]	MOPS	MFLOPS	V. OP RATIO	AVER. V. LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CPU	CONFLICT PORT	CONFLICT NETWORK
original	5100	432.549(7.9)	84.813	3057.3	1472.4	88.65	250.0	79.495	0.252	56.389	0.561	47.425	
modified	5100	17.984(0.7)	7.340	35438.2	9558.2	98.95	235.1	12.754	0.326	1.019	0.499	5.434	

図 4 サブルーチン A のオリジナルコードと修正コードの Ftrace 結果比較

3.2 MPI 並列による高速化

近年の大規模コンピューティングでは分散メモリ並列による実行が不可欠であり、各種の並列化ライ

ブラリを用いて大規模並列計算が行われている。SX-9 では並列化ライブラリとして MPI-2 (Message Passing Interface) をサポートしている。MPI は標準化された並列化ライブラリであるため可搬性が高く、システムのリプレイスの際も従来の MPI 化されたプログラムを大規模な修正なしに実行可能であるなど、そのメリットは大きい。しかしながら、逐次プログラムあるいは OpenMP による並列化プログラムを MPI 化する際には、プログラムを分析し、データ処理の分割方法、通信の内容と方法をユーザが選択し記述する必要がある。これらは実行時の通信性能に大きく影響を与えるため、MPI 化の前には適切な設計が必要である。

MPI 化例として以下で取り上げるユーザコードは、乱流モデルには $k-\omega$ Shear Stress Transport (SST) モデルを用い、3次元差分法を用いて Navier Stokes 方程式を解くものである[2]。現状の計算規模（グリッド点）は $i, j, k = 141 \times 39 \times 45$ である。MPI 化に必要なデータ分割は、データ配列の第3次元目の k で行った。本コードはオリジナルコードの自動並列化版、MPI によるフラット MPI 並列化版および、MPI 化に加えてコンパイラによる自動並列化機能を併用したハイブリッド並列版の作成を行った。

3.2.1 フラット MPI 化コード

ここでは4つのプロセスに分割した場合を例に説明する。MPI による分散並列化は k の上端と下端を除いた、 $k=2, \dots, 44$ の範囲を4等分して $k=2, \dots, 12$, $k=13, \dots, 23$, $k=24, \dots, 34$, $k=35, \dots, 44$ をそれぞれのプロセスに割り当てる。上端と下端を除くのは、多くの配列の更新は $k=2, \dots, nz-1$ の範囲で行われているためである。ただし、インデックス k での配列の更新において、 $k-1$ または $k+1$ のインデックスを参照する場合があるため、各プロセスにさらに $k=1, \dots, 13$, $k=12, \dots, 24$, $k=23, \dots, 35$, $k=34, \dots, 45$ を割り当てる必要がある。

k のループをプロセスごとに分割するとともに、各配列は分割され、プロセスごとに各領域の大きさが小さくなる。この領域分割した部分は並列に実行することができ、並列化前と同じ結果が得られることが確認されている。図5に、MPI 化に必要なデータ分割を行ったソースコードの一部を示す。

```
!宣言 (lz はプロセス 0, 1, 2 で 13, プロセス 3 で 12)
111: allocate( mxf(nx, ny, lz, nsp))
117: allocate(amuis(nx, ny, lz, nsp))
134: allocate( xfai(nx, ny, lz, nsp))
184: allocate( amu(nx, ny, lz) )
!実行コード (local_kend はプロセス 0, 1, 2 で 12, プロセス 3 で 11)
326: +-----> do k=2, local_kend
327: | +-----> do j=2, ny-1
328: || V-----> do i=2, nx-1                                ! V はベクトル化された DO ループ.
329: |||                                                    ! * は展開された DO ループを示す.
330: ||| amumix=0. d0
331: ||| akpmix=0. d0
332: ||| !cdir expand=nsp
333: ||| *----> do l=1, nsp
334: |||| amumix=amumix+amuis(i, j, k, l)/(1. d0+(1. d0/mxf(i, j, k, l))*xfai(i, j, k, l))
335: |||| akpmix=akpmix+akpis(i, j, k, l)/(1. d0+(1. 065d0/mxf(i, j, k, l))*xfai(i, j, k, l))
336: ||| *---- end do
337: |||
338: ||| amu(i, j, k)=amumix
339: ||| anu(i, j, k)=amu(i, j, k)/rhos(i, j, k)
340: ||| akpl(i, j, k)=akpmix
341: |||
342: || V----- end do
343: | +----- end do
344: +----- end do
```

図5 各プロセスにデータを分割した MPI 化コード

3.2.2 ハイブリッド MPI 化コード

ハイブリッド MPI 化コードは MPI による並列化に加えて、コンパイラによる自動並列化機能を併用したコードである。MPI 並列で 16 プロセス生成された場合ループ長は、以下の式より 2 または 3 である。

$$(\text{オリジナルの } k \text{ のループ長 } 43) \div (\text{プロセス数 } 16) = 2 \text{ または } 3$$

MPI プロセスで分割されたこのループを自動並列化の対象とすると、2 または 3 スレッドしか並列実行できないため、高い並列度を実現するには MPI とスレッド並列のループを別のものにする必要がある。自動並列化では一般的に最外側ループを並列化の対象とするため、MPI による並列化が行われていないインデックス j のループを最外側に来るようにソースを修正し、最外側の j のループで自動並列化が行われるようにした。 j のループ長は 37 なので、37 スレッドまで並列実行可能である。

3.2.3 性能測定結果

表 2 にオリジナルコードの自動並列化による SMP 並列化実行、フラット MPI 並列化によるマルチプロセス実行の結果を、表 3 にハイブリッド MPI 並列化コードによるマルチノード実行の結果を示す。

表 2 性能測定結果 (シングルノード)

プログラム		ノード数	ノードあたり プロセス数	ノードあたり スレッド数	実行時間 /sec
オリジナル (自動並列化のみ) フラット MPI 版 (MPI 化のみ)	(a)	1	1	16	42.80
	(b)	1	16	1	26.72

実行時間の測定結果より、1 ノード内実行におけるフラット MPI 版 (b) はコンパイラの自動並列化による (a) よりも 1.6 倍の性能が得られた。このことから本プログラムでは MPI による並列化が高速化に有用であることが示された。

表 3 性能測定結果 (マルチノード)

プログラム		ノード数	ノードあたり プロセス数	ノードあたり スレッド数	実行時間 /sec
ハイブリッド MPI 版 (MPI 化 + 自動並列化)	(c)	2	1	16	32.40
	(d)	2	2	8	30.46
	(e)	2	4	4	29.44

次に、ハイブリッド MPI 版での (c) ～ (e) では、2 ノードを用い、ノード内のプロセス数とスレッド数を変化させて実行したものである。プロセス数を増加させると実行時間が短くなっている。これは、本プログラムは格子点数が並列数で割り切れず、ロードインバランスが発生するが、プロセス数を増加させることで影響を小さくできるためと考えられる。プロセス数およびスレッド数は計算規模（グリッド点）の拡張や、利用可能なノード数（CPU 数）をパラメータとして、計算を実行するシステムに最適な設定値を決定する必要がある。今回のユーザコードの並列化には、汎用性のある MPI ライブラリによる並列化とコンパイラの自動並列化を用いているので、将来的にシステムの大規模化やアーキテクチャの変更があったとしても、コードの長期間の利用が可能である。

4 まとめ

本稿では、本センターの有する大規模科学計算システムと 1999 年から取り組んでいる高速化支援活動の取り組みと成果について述べた。この活動により利用者のプログラムは大幅に性能改善しているが、大規模化・長時間化する様々なプログラムに対応すべく、引き続き高速化支援活動を推進していきたい。

本稿に記載されている成果は、本センターを利用してくださる利用者の方々の多大なるご協力により得られたものである。高速化・MPI 化の事例には東北大学工学研究科機械システムデザイン工学専攻茂田先生、東北大学工学研究科航空宇宙工学専攻渡辺先生のご協力をいただいた。ここにあらためて感謝の意を表する。

参考文献

- [1] 日本電気株式会社 プログラム最適化作業報告
- [2] 日本電気株式会社 渡辺先生コード MPI 化作業報告書

[全国共同利用情報基盤センター研究開発論文集 No. 34 より]

キャンパス無線 eduroam の耐災害性・耐障害性向上と高機能化

後藤 英昭 曾根 秀昭

東北大学サイバーサイエンスセンター

1 はじめに

2006年に日本が国際的な学術無線 LAN ローミング基盤である eduroam (エデュローム)[1]に参加して以来、徐々に参加機関が増え、2011 年末に 27 だった国内の機関数は 2012 年 9 月現在では 39 となり、順調な増加傾向にある[2]。大学 ICT 推進協議会年次大会[3]を始め、各種研究会や学会などにおける広報により、eduroam の知名度も高くなってきており、参加に向けて準備中あるいは検討中の機関も少なくない。しかしながら、日本国内には 1,200 を超える高等教育機関があり、普及率では約 3.1%に留まっている。機関への eduroam 導入の障壁の一つとして、機関ごとの RADIUS サーバの導入・管理にかかる労力および経費の高さが考えられ、この問題に対処するために、サイバーサイエンスセンターでは「eduroam 代理認証システム」を開発して、2008 年よりサービス提供してきた。

代理認証システムは、低い導入・管理コストと高い安定性を実現する eduroam の新しいアーキテクチャとして世界的にも注目されているが、従来の実装では IdP (ID プロバイダ)の機能が単一故障点となる問題があり、実際に 2011 年の東日本大震災では広域停電の影響を受けた[4]。当システムの耐災害性および耐障害性を向上させるために、IdP を地理的分散させたクラウド型の認証基盤を構築したので、本稿で紹介する。

一方、eduroam には機能の高度化の要求もある。サイバーサイエンスセンター CSI 研究室では、OpenFlow 技術を用いて、機関のポリシーと利用者の属性情報に基づいたアクセス制御を実現する方式を開発した。詳細は参考文献[4],[5]で示したが、本稿ではその概略を述べる。

2 eduroam 代理認証システムの耐災害性・耐障害性の向上

代理認証システムは、eduroam の認証基盤をウェブサービスとして代行・提供するものであり、各機関の管理者はウェブ画面から必要数の eduroam アカウント(ID とパスワードのペア)を随時請求、取得できる。eduroam のユーザ認証は、機関ごとの RADIUS IdP ではなく代理認証システムによって行われる。本システムを利用することにより、各機関で RADIUS IdP を導入・運用する必要がなくなり、機関管理者のサインアップのみで eduroam を利用開始できるようになる。もし機関が無線 LAN 基地局の運用をシステムインテグレータやインターネットサービスプロバイダ(ISP)などに委託すれば、基地局を収容する RADIUS proxy の管理も不要となる。

2012 年 9 月現在、代理認証システムは 13 機関に利用されている。このうち 2 機関は、代理認証システムを当初利用していたが、学内のインフラ整備に伴って RADIUS IdP を構築し、補助用として代理認証システムの利用を継続している。4 機関は主システムとして日常的に代理認証システムを利用しており、他の 1 機関は数か月内に自前の RADIUS IdP を立ち上げるとされる。その他の機関は補助用の IdP として利用している。このように、代理認証システムは機関の eduroam 参加の障壁を下げるのに役立っている。

2011 年 3 月 11 日に発生した東北地方太平洋沖地震では、地震の揺れの最中から東日本で大規模な停電が発生し、数時間後に無停電電源装置のバッテリーが切れた後は、東北大学に設置されていた eduroam 関係のサーバ群も停止した。サイバーサイエンスセンターは学内の情報インフラの拠点であり、優先的に電源復旧作業が進められたが、それでも再通電まで約 2 日間を要した。日本国内のインフラである eduroam JP では、トップレベル RADIUS proxy のプライマリサーバが国立情報学研究所(東京都)、セカンダリサーバが東北大学(仙台市)に設置されていたが、このような冗長構成が功を奏して、国内の eduroam が停止することは避けられた。大地震の直後から初期の復旧作業にかけて、部分的ではあるものの、eduroam が緊急連絡手段として有効に使われたことが判明している。この時の様子は文献[6]で報告済みである。

代理認証システムのサーバは東北大学に設置されており、当時はまだセカンダリサーバの構築は計画段階であった。このため、3月の大地震の際には広域停電の影響を受け、停電のなかった関東地区の大学においても、2日間に渡って eduroam が利用できなくなるという問題があった。これは被災時のネットワーク利用環境を維持するという観点でも問題であり、改善が急務であった。2012年に入り、株式会社データホテル(旧ライブドア)より同社クラウドサービス「EX-CLOUD」のサーバ無償提供の申し出があり、今回これを利用して代理認証システムを冗長化し、クラウド型の代理認証システムを構築した。システム構成を以下に説明する。

代理認証システムの構成要素は大きく分けて、(1) ウェブユーザインタフェース(CGI プログラム群)、(2) SQL データベース(PostgreSQL DB)、(3) RADIUS インタフェース(FreeRADIUS) の3つがある。ウェブインタフェースの機能を冗長化するには、データ変更の同期や排他制御の処理が複雑となり、システムの大幅な変更が必要である。しかし、被災時に eduroam の認証処理が継続できることが最も重要であり、新規ユーザ作成などのアカウント管理はそれほど緊急性を要しないとみなせる。そこで、主システムの復旧にはそれほど日数を要しないという前提で、ウェブインタフェースの冗長化は今回見送った。

図1に、代理認証システムの冗長化の様子を示す。主システムであるマスタサーバは従来どおり東北大学に設置、運用継続する。東京にある EX-CLOUD の仮想マシン上に PostgreSQL DB と FreeRADIUS をインストールして、レプリカサーバを構築した。SQL データベースには、ウェブインタフェース用の管理者アカウント情報と、RADIUS 認証に必要な eduroam アカウント情報が格納されているが、後者のみを同期すれば十分である。eduroam JP のトップレベル RADIUS proxy には、代理認証システムのマスタサーバをプライマリサーバ、レプリカサーバをセカンダリサーバとしてそれぞれ登録し、代理認証システム用のレルム名を持つアカウントの認証要求を、これらのサーバに転送するように設定した。通常はマスタサーバのみに認証要求が転送されるが、例えば東北大学の停電や、何らかの障害によってマスタサーバが応答できない場合は、転送先が自動的にレプリカサーバに切り替わる。

マスタサーバとレプリカサーバのデータの同期には Slony-I (2.x 系)を用い、マスタからレプリカへの一方向の同期とした。アカウント管理の機能がマスタ側にしかないので、一方向で十分である。また、アカウントの作成、削除、一時停止などの操作は頻繁に行われるものではなく、多少の遅れがあっても実用上問題がないため、同期確認間隔は Slony-I の最大値である 1.0 秒に設定した。

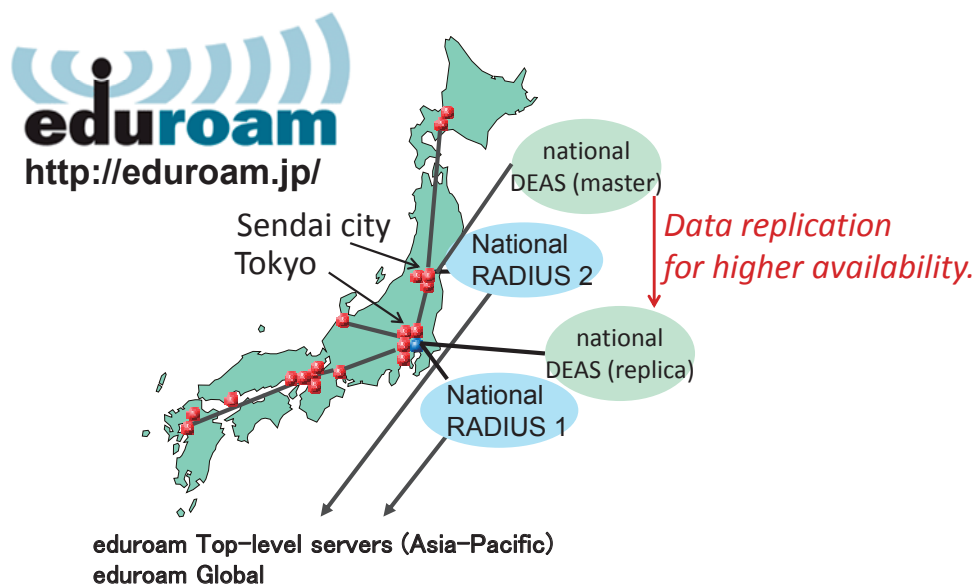


図1. クラウド型代理認証システムにおけるデータベースの冗長構成

株式会社データホテルでは、公衆無線 LAN サービス livedoor Wireless における eduroam サービスの提供[7]に加えて、各大学のキャンパス無線 LAN システムの構築も手掛けている。同社の顧客の多くが代理認証システムを利用していることから、同社の基地局を利用する学内利用者の認証リクエストの多くが日本のトップレベル RADIUS proxy を通過することになり、混雑による性能低下の懸念があること、および、各大学に RADIUS IdP を設置する従来方法と比べて RADIUS proxy のホップ数が多くなり、安

定性が低下する恐れがある。これら問題は、基地局を設置した業者に代理認証システムのレプリカサーバを持たせることで解決できると考えられる。今回構築したシステムでは、同社がレプリカサーバの方をプライマリ IdP とみなして認証要求を転送するように設定することで、RADIUS 認証のバイパス経路を構成でき、安定性の向上が期待できる。このような構成による運用は執筆時点において検討中であるが、負荷分散の有効な手段になると考えられる。

3 利用者の属性情報に基づく eduroam のアクセス制御の高度化

eduroam の運用の現場では、ID のレームを見て機関のメンバーとゲストの端末を区別し、異なるサブネットに收容するといった単純なアクセス制御は従来も可能であったが、より高度なアクセス制御を実現する標準的な仕組みは存在しない。利用者の種別に応じた詳細なネットワークアクセス制御が可能になれば、利便性やセキュリティの向上につながると考えられる。

利用者の所属機関と訪問先機関では、それぞれがアクセス制御のポリシーを有する。一例を以下に示す。

所属機関のポリシー(例)：

- 自機関の教職員・学生のネットワーク利用に特にアクセス制限は設けないが、ファイアウォールを通して、アクセスログを採取する。
- 学生にはトラブルの生じがちな特定のウェブ掲示板やファイル交換サイトを訪問先機関で利用させないようにしたい。利用帯域も制限したい。(訪問先に対するリクエスト)
- 教職員には訪問先でも比較的自由的なネットワークアクセスを許可したい。(訪問先に対するリクエスト)
- 訪問先機関が許可した場合、教職員については訪問先のファイルサーバやプリンタなどの利用を許可する。(訪問先に対するリクエスト)

訪問先機関のポリシー(例)：

- 訪問者の学生には特定のウェブ掲示板やファイル交換サイトを利用させないようにする。利用帯域も制限する。
- 訪問者の学内サーバへのアクセスは原則禁止とするが、非常勤講師には許可する。
- 会議等において、その場で利用権限を付与して、訪問者にファイルサーバやプリンタなどのローカルサービスの利用を許可する。

このような複雑なポリシーを元にアクセス制御を行うには、ポリシーやルールの記述方法の標準化が必要である。しかし、汎用的な記述方法を設計するのは大変難しいと考えられる。そこでまず初めに、ポリシーおよびルールを単純化したモデルを採用することとし、機関のポリシーと利用者の属性情報に基づいたアクセス制御を実現する方式を開発した[4][5]。ローカルサービス(リソース)の利用の可否はネットワークのアクセス制御によって行う。システムの概要を図2に示す。

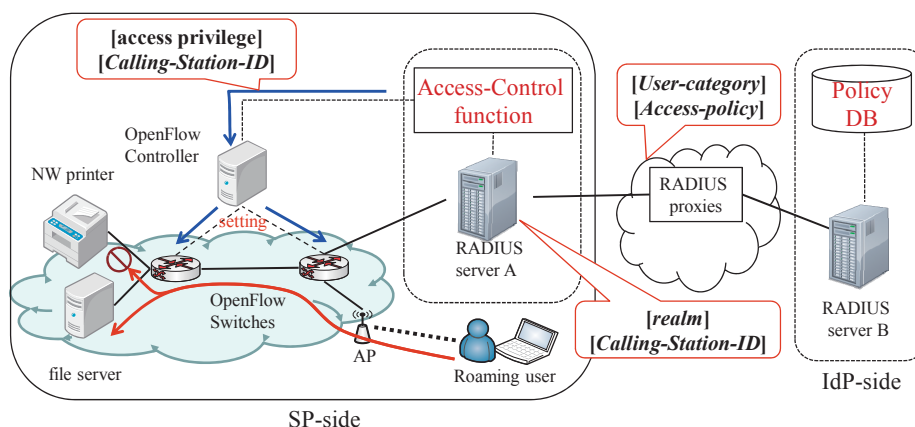


図2. 属性情報に基づいたアクセス制御が可能な eduroam システム

IdP となる RADIUS サーバにはポリシーを格納したデータベースが接続される。利用者ごとの eduroam アカウントには、利用者の種別やアクセス範囲を示す識別子が属性情報として付与されている。ある機関において訪問者がユーザ認証の手続きを開始すると、RADIUS プロトコルの通信パケットに埋め込まれた属性情報が訪問先機関の RADIUS サーバに届けられる。両機関のポリシーを突き合わせ、利用者の属性情報と組み合わせることによって、ネットワークアクセス制御のためのルールが生成される。このルールは OpenFlow コントローラに渡され、OpenFlow スイッチの動作が決定される。図 2 には、利用者がファイルサーバにアクセス可能だが、プリンタは利用できないというシナリオが例示されている。

ポリシーや属性情報が複雑になるにつれて、アクセス制御に必要な VLAN の数も非常に多くなることがある。OpenFlow 技術を用いることで、VLAN の総数に実用上制限のない dynamic VLAN が実現できる。また、システム内のすべての OpenFlow スイッチの制御を一台のコントローラで一括して行うことが可能である。このような OpenFlow の特長を生かして、eduroam のアクセス制御システムを構築することにより、単純なハードウェア構成で所望の機能を実装できた。

4 むすび

本稿では、平成 23～24 年度にサイバーサイエンスセンターで実施した業務の成果として、耐災害性・耐障害性を向上させたクラウド型の代理認証システムと、利用者の属性情報に基づく eduroam のアクセス制御の高度化について紹介した。両方とも、TF-MNM(モバイル通信とネットワークミドルウェアに関する TERENA のミーティング)や国際会議などで、国際的に方式提案を行なっている。

参考文献

- [1] L. Florio, K. Wierenga, “Eduroam, providing mobility for roaming users,” EUNIS 2005, June 2005.
- [2] eduroam JP ウェブサイト: <http://www.eduroam.jp/>
- [3] 後藤英昭, 曾根秀昭, “キャンパス無線 eduroam 導入のメリットと国内外の動向,” 大学 ICT 推進協議会 2011 年度年次大会 論文集 D10-6, pp.259-263, 2011.
- [4] T. Watanabe, S. Kinoshita, J. Yamato, H. Goto, and H. Sone, “Flexible Access Control Framework Considering IdP-side’s Authorization Policy in Roaming Environment,” IEEE 36th International Conference on Computer Software and Applications Workshops, COMPSAC-MidArch 2012, pp.76-81, July 16-20, 2012 (Izmir).
- [5] S. Kinoshita, T. Watanabe, J. Yamato, H. Goto, and H. Sone, “Implementation and Evaluation of an OpenFlow-based Access Control System for Wireless LAN Roaming,” IEEE 36th International Conference on Computer Software and Applications Workshops, COMPSAC-MidArch 2012, pp.82-87, July 16-20, 2012 (Izmir).
- [6] 後藤英昭, 曾根秀昭, “災害時における eduroam 全学無線 LAN の有効性とキャンパスアクセスネットワークの運用,” 電子情報通信学会 2012 年総合大会講演論文集 BS-6-1, pp.98-99, 2012.
- [7] プレスリリース「ライブドアと国立情報学研究所(NII)国際学術無線 LAN ローミング基盤 eduroam の共同実証実験を livedoor Wireless アクセスポイントにて開始」
<http://corp.livedoor.com/press/2010/0308376>

HPCI システムとサイバーサイエンスセンターの提供資源紹介

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

「京」コンピュータを中核とする全国のスーパーコンピュータ群を連携した HPCI システムの共用が、平成 24 年 9 月から開始されています。HPCI システムの利用を検討している方を対象に、HPCI システムの概要とサイバーサイエンスセンターが提供している計算機資源について紹介します。

1. HPCI システム

1.1 概要

革新的ハイパフォーマンス・コンピューティング・インフラ (HPCI) システムは、「京」コンピュータを中核とする全国の基盤センター等の計算機資源を連携した計算環境基盤です。計算科学技術における世界最高水準の成果創出と社会還元に寄与することを目的に、計算科学技術コミュニティを結集した一般社団法人 HPCI コンソーシアムが主導して体制と仕組みを整備し、平成 24 年 9 月 28 日から共用が開始されています。HPCI システムは、HPCI システム構成機関から提供される計算資源によって構成されます (図 1)。計算資源は、計算機資源、HPCI 共用ストレージ、ネットワーク、可視化システム、ライブラリ・アプリケーション等が含まれます。これらの計算資源は HPCI システムとして共通運用され、一括した課題選定によって一部の産業利用を除いて無償で利用することができます。また、ワンストップサービスによる共通窓口、アカウント管理、運用情報等の提供や、シングルサインオンによる共通アカウント・パスワードの機能が提供され、全国の計算資源をシームレスに利用可能な環境を実現しています。

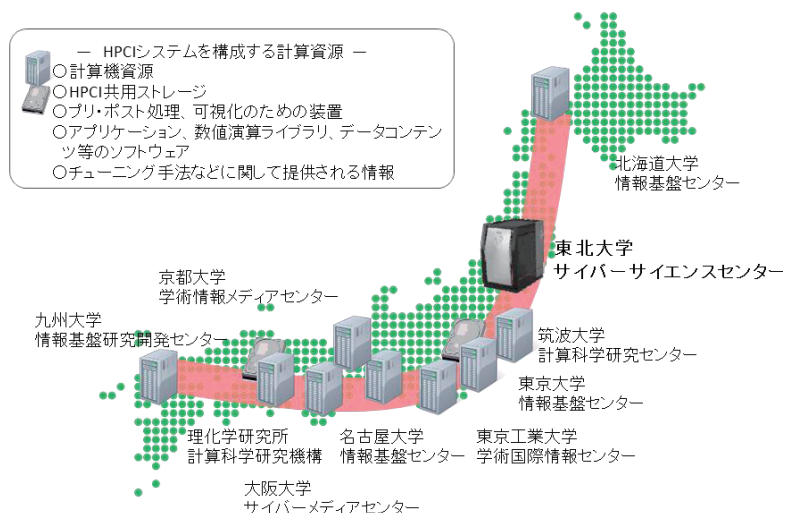


図 1 HPCI システムを構成する計算資源

1.2 HPCI システムを利用するには

HPCI システムを利用するには、利用申請を行い課題の審査を受ける必要があります。HPCI 運用事務局にて一括して課題公募が行われ、第 1 回目の公募はすでに終了しています。今後の公募情報については、HPCI 運用事務局が運営する HPCI ポータルサイト[1]をご参照ください。HPCI システムの利用に関する情報を随時提供しているようですので、このサイトをご覧になりぜひご利用をご検討ください。

2. サイバーサイエンスセンターの提供資源

2.1 提供計算機資源

当センターは、スーパーコンピュータ SX-9 と並列コンピュータ Express5800 を計算機資源として HPCI システムに提供しています (図 2)。SX-9 は、1CPU あたり 100Gflops を超える性能と高いバンド幅を持つ超高速ベクトルプロセッサを搭載し、1T バイトの大規模共有メモリにより高速かつ大規模モデルの演算が可能です[2]。Intel 社の Xeon プロセッサを搭載した Express5800 も 512G バイトの大規模共有メモリを持つシステムです[3]。



図 2 SX-9 (左) と Express5800 (右) の実機

1 ジョブあたりの利用可能な最大資源量は SX-9 が 64CPU・4TB メモリ、Express5800 が 32core・512GB メモリで、一般利用ジョブと計算機資源を共用します。並列数に対応したジョブクラスの設定があり (表 1) (表 2)、利用希望のクラスにジョブを投入すると投入順により資源が割り当てられ実行が開始されます。また、デバッグ作業等を想定した ss 以外の各クラスは実行 CPU 時間を無制限としていますので、長時間ジョブも時間制限にとらわれることなく実行可能です。

表 1 ジョブクラスと制限値 SX-9

クラス名	利用可能 CPU 数	メモリサイズ	CPU 時間
ss	4	256 GB	1 時間
s	4	32 GB	無制限
p8	8	512 GB	無制限
p16	16	1024 GB	無制限
p32	32	1024×2 GB	無制限
p64	64	1024×4 GB	無制限

表 2 ジョブクラスと制限値 Express5800

クラス名	利用可能 CPU 数	メモリサイズ	CPU 時間
as	1	16 GB	無制限
a8	8	128 GB	無制限
a16	16	256 GB	無制限
a32	32	512 GB	無制限

2.2 提供ソフトウェア

スーパーコンピュータ SX-9 と並列コンピュータ Express5800 が提供するコンパイラとライブラリを表 3 に示します。

表 3 提供ソフトウェア

スーパーコンピュータ SX-9	
コンパイラ	Fortran/SX, C++/SX
ライブラリ	MPI/SX, ASL, MathKeisan for SX
並列コンピュータ Express5800	
コンパイラ	Intel Fortran, Intel C++
ライブラリ	MPI, ASL, MKL

2.3 HPCI システムの利用に必要な設定

当センターの HPCI システムを利用する際に必要となる設定項目です。HPCI システム利用に関するマニュアルが HPCI ポータルサイトに用意されていますので、全般的な解説はそちらを参照いただき、ここでは当センターの利用に関わる設定を解説します。同様の解説をホームページにも掲載しています[4]ので併せてご覧ください。

○ログインホスト

SX-9, Express5800 の両システムとも、HPCI システム用のログインホストに接続して利用します。接続するには、ログインツールの導入や電子証明書の発行など、シングルサインオンのための準備が必要になります。HPCI クイックスタートガイド[5]等のマニュアルをご参照の上、事前に準備してください。

サイバーサイエンスセンターのログインホスト `hpcig.isc.tohoku.ac.jp`

○HPCI アカウントのパスワード変更

当センターをプライマリセンターとして登録した HPCI 利用者は、以下のパスワード変更サイトにて HPCI アカウントのパスワードを変更できます。

HPCI アカウントのパスワード変更サイト `https://hpcis.isc.tohoku.ac.jp/pwm`

○HPCI 共用ストレージのマウントポイント (HPCI 共用ストレージ利用者のみ対象)

HPCI 共用ストレージは、ログインホストの以下のディレクトリにマウントしています。

マウントポイント： `/gfarm/ (課題 ID) / (利用者番号) *`

*利用者番号は、サイバーサイエンスセンターが発行したローカル ID です。

○同一利用者が複数の HPCI 課題を持つ場合

課題ごとに利用者番号を発行していますので、課題に対応した利用者番号を用いて HPCI システムを利用します。設定方法・切り替え方法は、当センターの HPCI ページを参照ください[4]。

参考文献

[1] HPCI ポータルサイト：`http://www.hpci-office.jp/`

[2] ベクトル型スーパーコンピュータ SX-9：`http://www.ss.isc.tohoku.ac.jp/super/index.html`

[3] 並列コンピュータ Express5800：`http://www.ss.isc.tohoku.ac.jp/scalar/index.html`

[4] サイバーサイエンスセンター HPCI ページ：

`http://www.ss.isc.tohoku.ac.jp/collabo/hpci.html`

[5] HPCI クイックスタートガイド：`https://www.hpci-office.jp/documents/HPCI-UM01-001.pdf`

[報 告]**後藤英昭准教授と曾根秀昭教授の研究グループが
大学 ICT 推進協議会 2011 年度年次大会・優秀論文賞を受賞しました**

サイバーサイエンスセンターの後藤英昭、曾根秀昭が、大学 ICT 推進協議会 2011 年度年次大会・優秀論文賞を受賞しました。

本受賞は、大学 ICT 推進協議会 2011 年度年次大会における下記の発表論文が高く評価されたものです。

対象論文：

「キャンパス無線 eduroam 導入のメリットと国内外の動向」

後藤英昭，曾根秀昭

(この賞は、大学 ICT 推進協議会年次大会において発表された論文・ポスターのうち、特に優れた業績と認められたものに与えられるものです。)

会議情報：

大学 ICT 推進協議会年次大会

<http://axies.jp/ja/conf>

研究背景：

サイバーサイエンスセンター・CSI 研究室は、国際的な学術系無線 LAN ローミング基盤である eduroam(エデュローム)を 2006 年に日本に初導入し、国立情報学研究所と共同で、国内の基盤である eduroam JP を立ち上げ、その運用および研究開発を行ってきました。1,200 以上もの高等教育機関を有する日本において、キャンパス無線 LAN の安全性向上や、eduroam 導入の技術面および運用面のコスト低減は必須であり、そのための啓発活動、技術開発、および評価・運用を行うとともに、キャンパスネットワークのコビキタス化に向けての研究開発などを推進しています。これらの一連の活動は、研究成果から実運用への適用と、多数機関の eduroam 参加促進の効果が実証されています。

[報 告]

SC12 報告

東北大学サイバーサイエンスセンター 小松一彦

2012年11月10日～16日に米国ユタ州ソルトレイク市において International Conference for High Performance Computing, Networking, Storage and Analysis (通称 SC)が開催されました。SCはスーパーコンピューティングとその関連技術に関する世界最大の国際会議であり、参加者は1万人を超え、その年の最新の研究成果・製品が学界、産業界から発表されてきました。今回、SC12(12は開催年を意味する)に参加し、サイバーサイエンスセンターの研究活動の展示を行って来ましたので、その報告をします。

SC12は技術論文が発表される本会議と大学・研究所・企業等が最新の成果・製品を発表する展示会から構成され、サイバーサイエンスセンターは流体科学研究所と金属材料研究所と連携し、東北大学としてスーパーコンピューティングに関する活動についての展示を行いました。東北大学展示ブースでは、サイバーサイエンスセンターのスパコンシステムの紹介をはじめ、3次元積層型ベクトルプロセッサや将来のHPCIシステムのあり方の調査研究など、次世代高性能計算基盤技術に関する最新の研究成果を紹介しました。また、スパコン研究部に所属する研究員3名、大学院学生2名も帯同させ、各自が取り組む高性能計算技術に関する成果発表も行いました。12～15日の4日間の展示会期間中、東北大学展示ブースへの訪問者は300名以上にものぼり、活発な情報交換を行うことができました。また、訪問者にはセンター紹介パンフレット、手ぬぐい、SX-9折り紙などをくばり、特に折り紙は外国の方々に非常に好評でした。(作り方はyoutubeでも見られます。http://www.youtube.com/watch?v=J60eh_5fnPY)

今年のSC13は11月17日～22日に米国コロラド州デンバー市で行われる予定です。



展示の様子



東北大学展示スタッフ

[展示室便り⑦]

センター刊行物

今回は、大型計算機センター発足直前から現在までの当センターによる刊行物です。展示品1は、準備速報、広報誌 SENAC、大型計算機センター便りです。



展示品 1

大型計算機センターの発足は1969年6月（昭和44年6月）です。展示品の SENAC (Vol.1, No.1 1968-4) は、センター発足一年前に準備室委員会により発行されました。掲載されている内容は、利用に関すること、設置準備に関することなどです。開設に向けての準備の期間であり No.1~No.3 には設置準備の各種委員会、設備の進捗の経過が詳しく書かれています。発行は現在も続いており、本号で Vol.46, No.1 となりました。「SENAC」の名称については展示室便り①で紹介しましたが、東北大学の電気通信研究所で開発された計算機 SENAC (SENdai Automatic Computer) から命名されました。この表紙のデザインはパンチカードに SENAC と穿孔したもので、Vol.29 まで使われました。本号の表紙は三代目です。現在は3ヶ月おきですが、当初は毎月の発刊でした。一方、利用者に対する業務連絡を中心に、速報として「大型計算機センター便り」が1969年から毎週発行されるようになりました。現在は概ね月1回の発行ですが、メールマガジンとして継続しております。No.1 の最初の記事は「昭和44年度計算機利用申請書の受付開始について」でした。

センター発足当時は、計算機メーカーごとにオペレーティングシステムが異なり、利用者

が計算機を利用するために参考となる市販の説明書がほとんどありませんでした。そこで、センタースタッフ、利用者、メーカー担当者が協力してシステムの利用法などの説明書を作成しました。展示品2～4は「計算機の利用法に関する説明書」です。ここでは、3つのカテゴリに分けてタイトルを列挙しておきます（これまでの[展示室便り]中で、紹介したものもいくつか含まれています）。

①システムの利用法に関する説明書（展示品2）

利用の手引き、利用者のためのパンチ（穿孔）の手引き、ジョブの構成法、ファイルの利用法、TSS 入門、TSS の使い方、TSS ライブラリの利用法、ACOS-6 FORTRAN 構造型エディタ FEDIT、スーパーコンピュータ SX-3 の使い方、高速化推進研究活動報告。



展示品 2

②ライブラリ／アプリケーションに関する説明書（展示品3）

数値計算ライブラリ SSL、図形処理ライブラリ説明書 (I) DRFLIB 編、図形処理ライブラリ説明書 (II) GKS 編、会話型データベース管理システム COOD 説明書、UNIQ-1 の使い方、X 線/中性子粉末解析パターン Rietveld 解析システム RIETAN、結晶構造解析プログラムシステム UNICS。



展示品 3

③メインシステムとは別に導入された小規模システムのための説明書（展示品4）

DTシステムの利用法、手書きOCRの利用法、図形入力装置の利用法、画像システムの使い方、東北画像データベース説明書。



展示品 4

展示品5は、[センター概要] です。センターの見学等でセンターを紹介するために作られている冊子で、システム更新時などに改版されてきました。

展示品6は、「十年史、三十年史・軌跡、年報」で、センターの活動をまとめた刊行物です。



展示品 5



展示品 6

読者の中には、若い頃センターを利用されたときに参考にした説明書を上記写真やタイトルから見つけれ思い出された方もいらっしゃるのではないのでしょうか。

紹介したこれらの刊行物はセンター広報活動の一環として発行してきましたが、改めて並べてみると計算機センターの歴史と計算機の進歩を感じます。この展示室便りもこれらの資料を参考にしながら書いています。

[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/>

負担金の支払費目について (No. 143)

今回の利用負担金請求（平成 25 年 1 月 1 日から 2 月 15 日までの利用分）は 2 月下旬に行います。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いたします。

（共同利用支援係, 会計係）

利用負担金（1 月～3 月分）の請求について (No. 143)

大規模科学計算システムの平成 25 年 1 月～3 月分の利用負担金の請求を次のように行います（昨年と同じです）。

《学内支払責任者》

1 月から 2 月中旬までの利用負担金を 2 月下旬に請求します。その後 3 月末までの利用負担金は、翌年度 4 月に請求します（平成 25 年度の大学運営資金で予算の調整を行います）。

《学外支払責任者》

1 月から 2 月中旬までの実績額と 3 月末までの利用見込額を合算した利用負担金を 2 月下旬に請求します。利用見込額は以下の mikomi コマンドで 2 月 15 日（金）までに申告してください。

なお、見込額と実績が異なっても精算は行わないのでご注意ください。

【 mikomi コマンドの使用例 】

ssh gen.isc.tohoku.ac.jp -l 利用者番号

利用者番号@gen.isc.tohoku.ac.jp's password: パスワード

gen% mikomi

*並列コンピュータにログイン
して下線部を入力します。

2 月 5 日 現在の見込み額は次のとおりです。

支払責任者 : ○○ ○○ (u23456)

見込み額指定者 :

見込み額 : 0 円

1. 見込み額の指定 2. 削除 9. 終了

何番の処理を選びますか? 1

見込み額を入力してください(円)? 200000

登録してよろしいですか (yes/no) ? yes

.

gen%

（共同利用支援係, 会計係）

利用負担金額の表示コマンドについて (No. 143)

本センター大規模科学計算システムでは、利用者の利用額と支払責任者ごとの利用額・負担額を表示するためのコマンドとして `kakin`, `skakin` があります。これらのコマンドは、並列コンピュータ (`gen.isc.tohoku.ac.jp`) にログインして使用します。

コマンド名	機 能
<code>kakin</code>	利用者ごとの利用額を各システム、月ごとに表示
<code>skakin</code>	支払責任者ごとに集計した利用額と負担額を表示 (負担額は割引制度に基づいた金額)

いずれも、前日までご利用いただいた金額を表示します。コマンド使用例は大規模科学計算システムウェブページをご覧ください。

負担金の確認

<http://www.ss.isc.tohoku.ac.jp/utilize/academic.html#負担金の確認>

(共同利用支援係)

研究成果リスト提出のお願い (No. 143)

平素、本センターの大規模科学計算システムをご利用いただきありがとうございます。

本センターでは、学術研究を支える世界最高水準の大規模科学計算システムの導入と利用環境の整備・拡充を行い、研究の発展に資することを心掛けてまいりました。今後もシステムの整備を進めていくには、大規模科学計算システムが多くの研究分野で必要不可欠であり、かつ研究成果が得られていることを広くアピールしていく必要があります。

つきましては、本センター大規模科学計算システムを利用して得られた研究成果を、下記により提出くださるようお願い申し上げます。なお、提出していただく研究成果は、平成 24 年度中に発表されたものとします。

記

1. 研究成果リスト：著者名、論文名、掲載誌（巻号頁）、発表年
2. 提出方法 : 電子メールでお願いします。
提出先メールアドレス seika@isc.tohoku.ac.jp
3. 締切り日 : 平成 25 年 4 月 19 日（金）
4. 問合せ先 : 共同利用支援係（022-795-6251, uketuke@isc.tohoku.ac.jp）

論文等への利用の明記について

研究成果を論文等で発表する場合には、本センターを利用した旨を明記くださるようお願いいたします。

－ 記入例 －

「本研究の実験結果の一部は、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を利用して得られた。」

Part of the experimental results in this research were obtained using
supercomputing resources at Cyberscience Center, Tohoku University.

（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=30mm 下=25mm 左右=25mm 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

スタッフ便り*

近年、2011 年 6 月に TOP500 ランキング 1 位になった「京」をはじめとして、スーパーコンピュータへの興味・関心が非常に高まってきています。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や並列コンピュータ、コンピュータ博物館などを公開している東北大学オープンキャンパスでも、たくさんの皆様が見学に来場して頂けることでも、その興味・関心の高さが伺えます。

全国共同利用施設として大学・研究所の研究者・学生を支える本センターは、この時期、卒業シーズンや年度末に近いということもあり、スーパーコンピュータや並列コンピュータの利用がピークに達します。ジョブのスケジューリングや計算資源の割り当てなど、スーパーコンピュータや並列コンピュータを効率的に利用して頂けるよう、少しでも早く計算結果を出せるよう運用しております。日本の研究・科学技術を支える重要な基盤として、研究者・開発者を支援することができるよう取り組んでおります。これからもどうぞよろしくお願いいたします。(K. K)

サイバーサイエンスセンターが NHK の「クローズアップ現代」という番組に取り上げられました。「京」をはじめとして我が国のスーパーコンピュータにおける可能性と課題についての番組で、「産業利用」などにも焦点を当てる狙いのようなのでした。本センターのスーパーコンピュータ SX-9 でシミュレーションすることで実用化が間近に迫っている国産旅客機の設計やタービン設計などが取り上げられていました。予定より遅れて年が明けての放送となりましたが、本センターにとって新年から弾みがつく出来事でした。

本センターは次期スーパーコンピュータシステムを収容する場所と建物を用意しなければならないなど多くの課題を抱えておりますが、利用者の皆様のご協力を得て解決してまいりますので、一層のご愛顧をお願いいたします。(M. C)



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 曾根秀昭 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

* 編集部会では、より多くのセンタースタッフからの話題もお届けできればと考え、今号より「編集後記」を改め「スタッフ便り」といたしました。

平成 25 年 1 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.46 No.1 2013-1

[巻頭言]

2013 年東北大学及びサイバーサイエンスセンターへの期待 野口 正 一 1

[共同研究成果]

ダウンスケールシミュレーションにおける台風の再現性について
台風 Choi-wan (2009) の事例解析 濱田 真之 3
岩崎 俊樹

ヤマセ研究のための再解析・気候モデルデータのダウンスケーリング 島田 照久 9
沢田 雅洋
岩崎 俊樹

Convergence Property of Flexible GMRES for the Method of Moments
Based on a Volume-Surface Integral Equation Hidetoshi Chiba 17
Toru Fukasawa
Hiroaki Miyashita
Yoshihiko Konishi
Qiang Chen
Kunio Sawaya

[資料]

Fortran スマートプログラミング
— 第3回 もっとスマートなプログラムにしよう — 田口 俊弘 29

[全国共同利用情報基盤センター研究開発論文 No.34 より]

東北大学サイバーサイエンスセンターにおける
ユーザコードの高速化支援活動 佐々木大輔・山下 毅・小野 敏 53
大泉 健治・江川 隆輔・小林 広明

キャンパス無線 eduroam の耐災害性・耐障害性向上と高機能化 後藤 英昭 60
曾根 秀昭

[紹介]

HPCI システムとサイバーサイエンスセンターの提供資源紹介 64

[報告]

後藤英昭准教授と曾根秀昭教授の研究グループが
大学 ICT 推進協議会 2011 年度年次大会・優秀論文賞を受賞しました 67

SC12 報告 小松 一彦 68

[展示室便り⑦]

センター刊行物 69

[Web 版大規模科学計算システムニュースより]

負担金の支払費目について (No.143) 72
利用負担金 (1 月～3 月分) の請求について (No.143) 72
利用負担金額の表示コマンドについて (No.143) 73
研究成果リスト提出のお願い (No.143) 74

執筆要項 75

スタッフ便り 76