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.45 No.3 2012—7

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

大規模科学計算システム関連案内

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

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

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

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

本誌の名前「SENAC」の由来

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

[共同研究成果]

「かぐや」月レーダサウンダが見た月の海

小林敬生 1)　加藤雄人 2)　熊本篤志 2)　小野高幸 2)
1) 韓国地質資源研究院 国土地質研究本部 地質調査研究室
2) 東北大学大学院 理学研究科 地球物理学専攻

1.　はじめに

日本の月探査機「かぐや」は 2007 年 9 月 14 日に宇宙航空研究開発機構（JAXA）種子島宇宙センターから打ち上げられ、同年 11 月に月周回軌道上での観測を開始、 2009 年 6 月 11 日の制御落下による運用終了時まで 1 年 7 ヶ月にわたり 15 の搭載機器による全球規模の各種科学観測を行なった[1]。月レーダサウンダ（Lunar Radar Sounder: LRS）は 15 の搭載機器のひとつで、月の地下構造探査を主要観測目標とする短波（5MHz）レーダである[2]。LRS は 1972 年のアポロ 17 号によるレーダ探査実験（Apollo Lunar Sounder Experiment）[3]以来 35 年振りの、そして史上初の全球規模の月レーダ探査を実現した。LRS は月の表面が平坦な「海」と呼ばれる領域の多地点で地下のレーダ反射信号をとらえることに成功し、この結果は初期成果として発表された[4]。「かぐや」の運用終了後、取得された大量の LRS データは空間分解能と微弱な地下反射信号の S/N（信号対雑音比）を改善するため合成開口レーダ（Synthetic Aperture Radar: SAR）処理を施され[5]新たなデータセット、LRS SAR データとして生まれ変わった。大量の LRS データに SAR 処理を施す作業は 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 で行なった。SX でのデータ処理については、すでにこの SENAC 誌上で紹介したとおりである[6]。 LRS SAR データは月科学における LRS データ利用の出発点である。本稿では、LRS SAR データを解析して明らかになった月の海領域の様相について紹介したい。

2.　LRS SAR データ

LRS は高度 100km の極軌道を周回しながら月全面をカバーするように観測を行なった。全観測期間を通じて行なった観測（レーダパルス送信）の総数は 1 億を超え、平均すると月面上の 600m 四方毎に LRS 観測（レーダパルス送信）がひとつなされたことに相当する。LRS の空間分解能は軌道方向に 600m（合成開口長 5km の場合）、軌道直交方向は 5km で、レンジ（深さ）方向は 150m である。レンジ分解能は真空中の値である。地下では、地下媒質（岩石）の誘電率の値に依存してこの値の 1/2～1/3 程度の値になる。

LRS SAR データを構成する個々の要素データは信号値対レンジ（探知距離）の 1 次元配列データである。図 1 は典型的な観測例についてそれを図示したものである。軌道上の各観測点に対応して図 1 に示されるような 1 次元配列データがひとつ得られる。

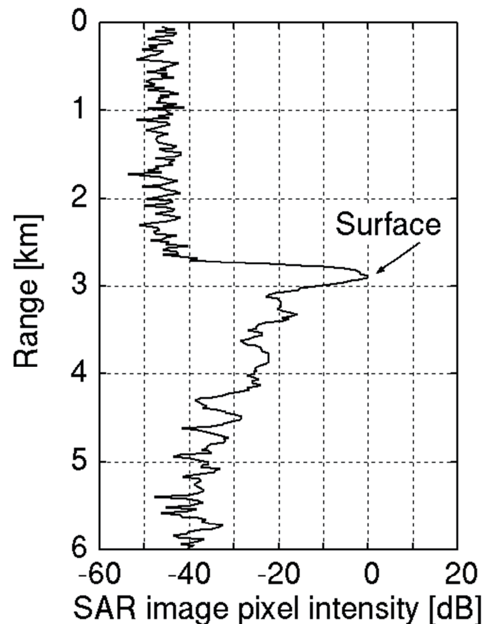


図 1. LRS 受信データ（1 パルス）例。

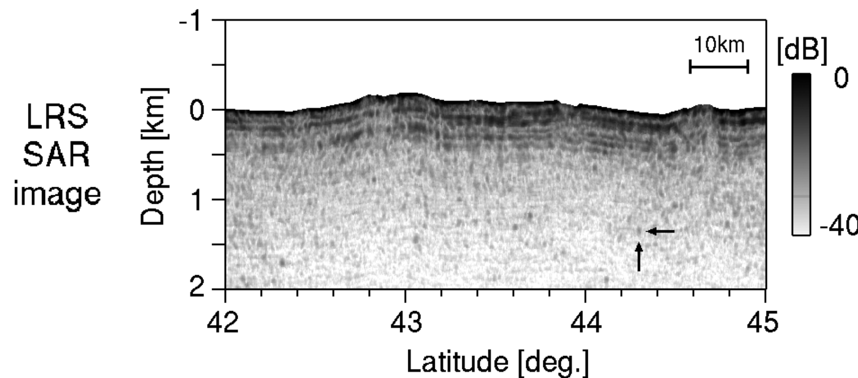


図２． LRS SAR データによる月地下断面画像の例。 縦軸に示す深さは地下媒質の比誘電率を 6.25 と仮定した場合の値。

図１では、任意のレンジを基準としてレンジ値を定義している。観測領域の月面が平坦な場合、信号の最大値はかぐや直下点からの表面反射波のものである。図１では、レンジ 2.9km に直下点表面が見つかったことになる。図では、この表面反射波に続いて弱い強度の信号が受信されているが、これら表面レンジより遠くからやってくる反射波は地下反射面からのものであると考えられる。これらの小ピークは対応する深さにおける地下反射面（地層境界面）の存在を示唆するが、図１のデータだけでは地下の地層境界面からの反射波なのか、表面の同レンジ地点からの斜め反射波なのかはっきりしない。地下反射波と同定するためには、図１のデータを観測経路に沿ってならべて図２のような地下断面イメージを作って地下の層構造を確認し、さらに表面の光学画像データから地下反射波と紛らわしい斜め反射波を返すような地形がないことを確認する必要がある。

図１に示すよう、個々の LRS SAR 要素データには月直下点表面反射波信号と直下点地下反射波信号が含まれ、それに表面斜め反射波信号が重畳している。表面が平坦な海領域では、表面斜め反射波信号は 直下点地下反射波信号に比べ十分微弱であることがわかって来た。つまり、任意の深さの地下イメージを表面反射の誤認の恐れなく構成することができるということである。以下では、LRS の電波で「見た」月の表面および地下の様相を紹介する。

3.　表面

月の表面（図３）は大別して 2 種類の地形領域に分類できる。ひとつは、「海」と呼ばれる領域で、過去の火山活動で噴出した溶岩に満たされてできた平坦な地形領域である。肉眼（可視光）では暗く見えるが、これはこの溶岩（玄武岩）の色による。もうひとつは、「高地」と呼ばれる領域で望遠鏡で見るとクレータに覆われた 起伏の激しい地形領域であることがわかる。「高地」は明るく見えるが、これは高地の岩石に多く含まれる斜長石が白いためである。「海」は月面全体の 2 割程度の表面を覆う。月の裏側はほぼ全面「高地」である。東北アジアの国々では、「海」の暗い部分をうさぎの姿に見立てて、月にいるうさぎの話を語らってきた。

LRS データから表面反射波だけを取り出し、それを画像ピクセル情報として軌道情報に従って月面座標の当該点に置いて行けば LRS の月面反射波強度マップができる。これは、言わば LRS の電波で「見た」月表面イメージである。図４は地球から見える月の表側の大部分を含む、南北方向南緯 40 度から北緯 70 度、東西方向西経 90 度から東経 100 度の範囲の LRS による「月表面イメージ」である。図の「明るさ」は月面の LRS 反射波強度を表しており、その相対強度はグレイス

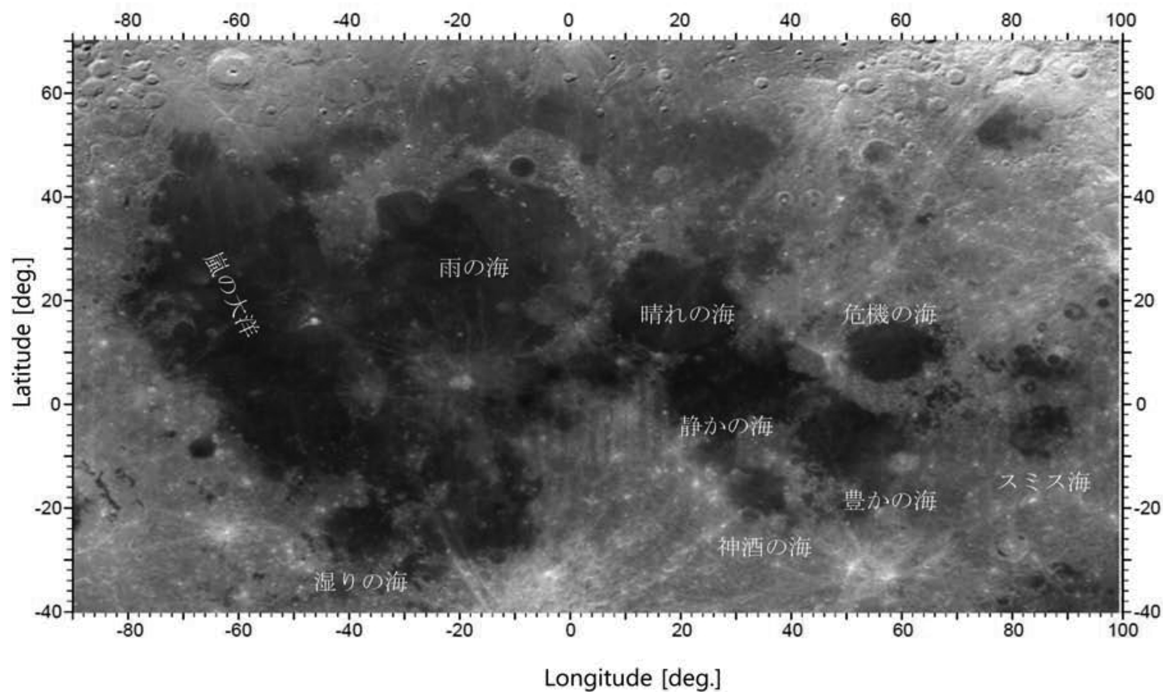


図３．月表側の可視光イメージ[7]。主な海の名前を図中に示す。

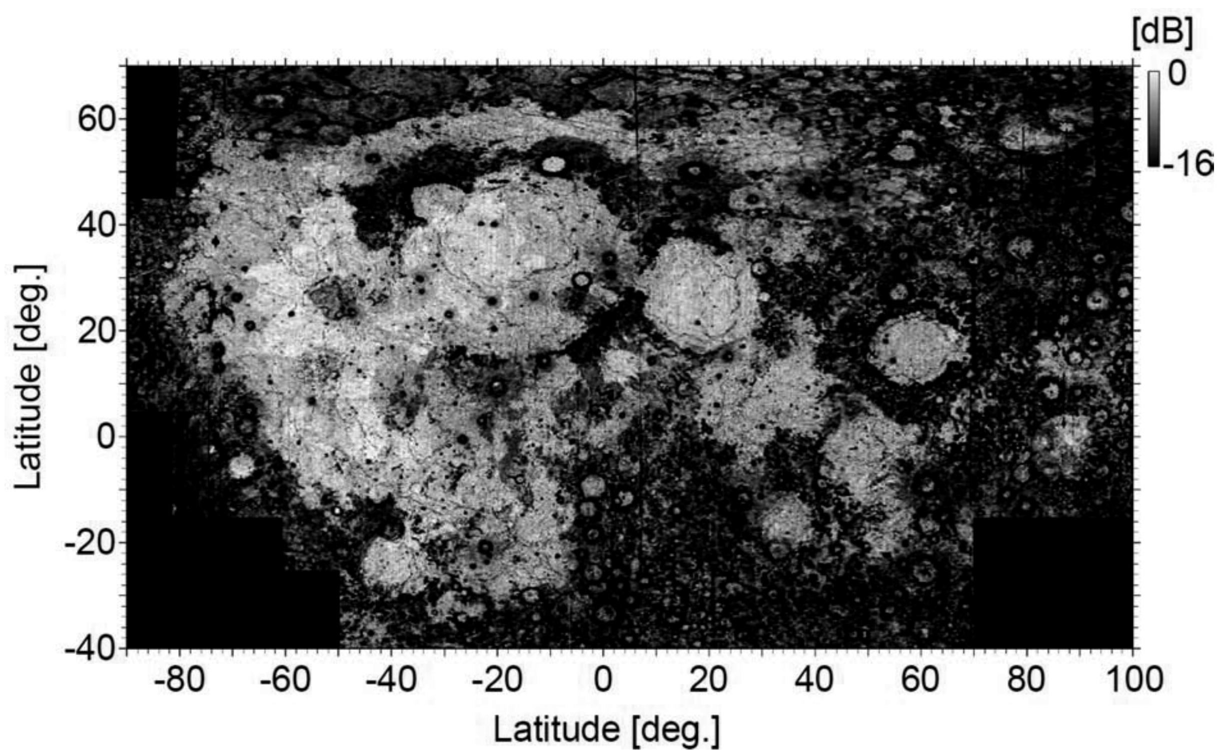


図４． LRS SAR データによる月表側表面画像 。

ケール（単位：dB）が示すとおりである。画像ピクセルのサイズは 0.1 度×0.1 度だが、これは赤道では 3km×3km に相当する。かぐやの観測軌道投影（フットプリントパス）は一様に月面をカバーしているわけではないので一部の領域でデータ欠損が生じている。データ欠損領域は近接データを線形補間して補った。そのため、空間分解能にむらが生じている。

LRS が見る月面（図４）は我々が普段見慣れている月のイメージ（図３）と明暗が逆転している。我々の目には暗く見える「海」が LRS には明るく見え、逆に、我々の目に明るく見える「高地」は LRS には暗く見えるのだ。これは、波長 60m の LRS の電波観測の場合、表面の物質の違いによる反射率の違いよりも表面の起伏の大小による散乱の影響の違いのほうが大きいためである。すなわち、「海」の表面は平坦なため鏡面反射がほぼ実現され LRS の観測では強い表面反射波が受信されるが、「高地」では表面の大きな起伏のために LRS のレーダパルスが強い散乱を受けて弱い反射波しか受信されないためである。高地領域の中でも、直径が 100km を越える大きなクレータは底が比較的平坦なため、明るく見えるものもある。

「海」の部分を詳しく見るとさまざまな様相が見えてくる。図５は「嵐の大洋」から「雨の海」西部を含む領域を拡大した図である。図５は電波で見た月面とはいえ、陰影に富む豊かな様相を呈している。まず、「海」の中にも明るい領域と暗い領域があることに気づく。我々の研究の結果、「海」の明るさはその表面を覆っている溶岩の流出年代と関係があることが分かってきた。暗い部分は溶岩の流出による表面形成時期が 30 億年以上前の古いもので、明るい部分は表面形成時期が 30 億年前から 15 億年前ころまでの比較的新しいものである。

赤線で囲まれた 3 つの領域は北から順に、リュンカー、アリスタルコス台地、マリウス丘陵と呼ばれており、いずれも火山性の地形である。リュンカーとアリスタルコス台地は周囲よりも暗く見えているが現時点でその理由ははっきりしていない。マリウス丘陵にはゴマ粒のような多数の黒点状のイメージが見られるがこれら一つ一つは個々の火山の溶岩ドームに対応する。

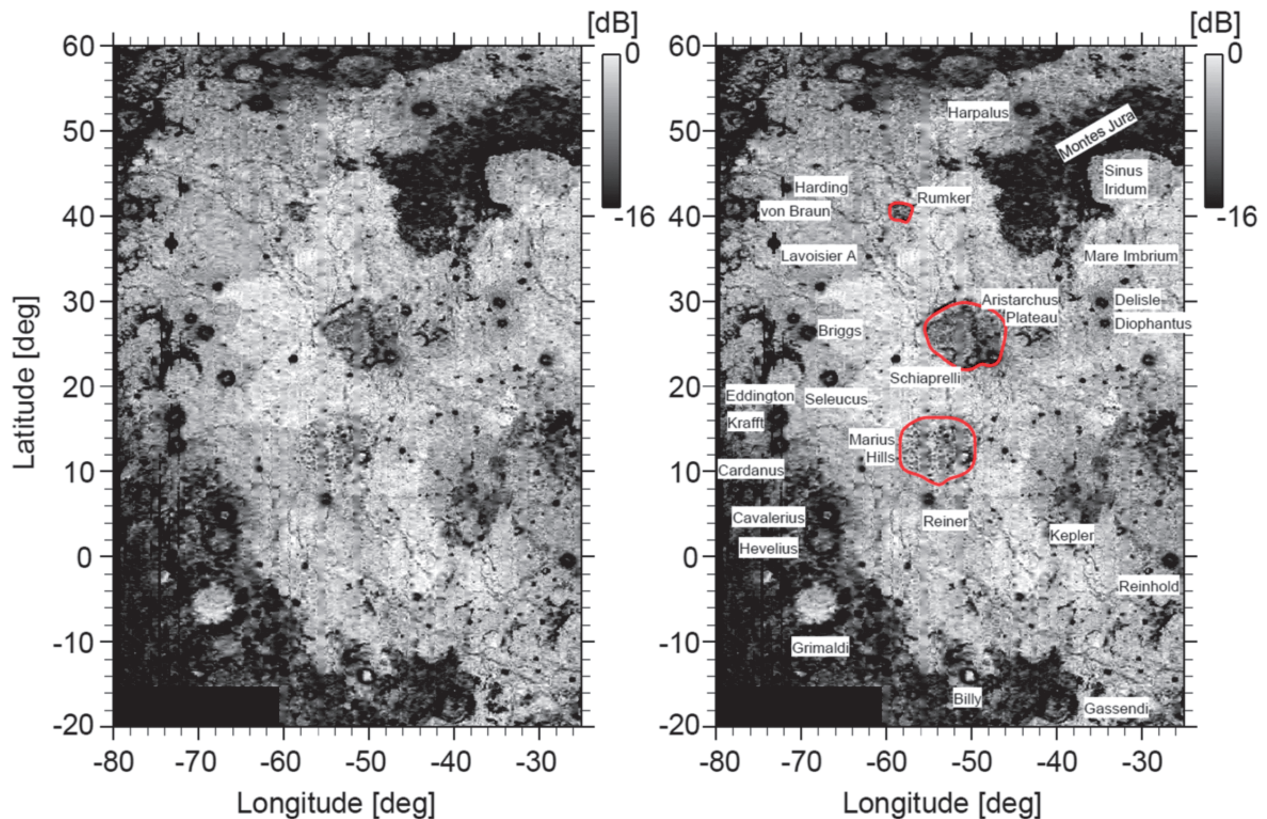


図５． LRS SAR データによる月表側表面画像、「嵐の大洋」領域拡大図。右図は左図に代表的な地形、クレータ等の名称を表示したもの。

図５（左）では多数の黒く丸いイメージが「海」のあちらこちらに認められる。同図（右）に示すように、これらの黒丸イメージは直径数十キロメートルのクレータである。直径がさらに小さいクレータは黒い点のイメージとして認められる。これら、クレータが黒く見えるのは、クレータのふちから外に広がる斜面がクレータの生成（隕石衝突）時に放出された瓦礫（イジェクタ）が積もってできた起伏の激しい斜面なので、入射する LRS レーダパルスが強い散乱を受けて反射波の受信強度が著しく低下するためである。

図５では、ひも状の暗い構造も見られる。これは、リンクルリッジと呼ばれる地形に対応している。リンクルリッジは「海」を満たした溶岩が冷えて固まるときに起きる溶岩全体の体積収縮に伴って生じたと考えられている。リンクルリッジは幅が数キロメートルから 10 キロメートル程度、 長さは数百キロメートルにおよぶ。起伏があり周囲からの高さは数百メートル程度である。暗く見えるのはリッジの起伏による幾何学的な要因によるものとも考えられるが、リッジの表面から十数メートル程度の浅い部分の空隙率が大きいことによる強い散乱・吸収効果も否定できない。

最後に、「ジュラ山脈（Montes Jura)」を見てみよう。ジュラ山脈は図５の図中右上の部分に広がるひときわ暗い部分である。ここは、溶岩に埋もれることなく残った古い高地であるので、月の海が生じる前の隕石重爆撃を受けてクレータで飽和した古い地形を呈している。その起伏の激しい表面のため LRS レーダパルスは強い散乱を受け、結果としてジュラ山脈全体が非常に暗く見えている。ジュラ山脈の東側に大きな入り江のような地形があるが、これは直径 260km の衝突クレータで「虹の入り江（Sinus Iridum)」と呼ばれている。虹の入り江には「雨の海」からの溶岩が流入して内側を満たしており、これが入り江を形づくっている。虹の入り江は 2014 年に計画されている中国の「嫦娥３号」の着陸地点の候補地に選ばれている。

4.　地下

LRS による「海」領域の観測ではあちらこちらで地下の層構造を示すデータが得られた[4]。合成開口処理を施す前はその層構造も２層程度までしか認められていなかったが、処理後のデータではさらに多層の構造がよりはっきりと認められるようになった。図６は LRS データの合成開口処理によって実現された代表的な地下構造可視化の例を示している。層構造を示す地下反射波は従来火山活動に伴って流出した溶岩が作り出した溶岩層の境界からの反射波であると考えられてきたが、我々は、それが、単純な溶岩層境界面ではなく、過去の火山活動休止期に隕石衝突によって粉砕された溶岩層表面に作られたレゴリス層からの反射波であると考えている。単純な溶岩層境界面からの反射だけでは３層以上の多層地下構造の反射波強度をうまく説明できなくなるからである。この考え方で図６を見ると、月の「海」領域における過去の火山活動は領域毎に異なる休止期を伴う活動度で生じてきたことが分かる。

さて、これら検出された地下反射波はどのような分布をしているのだろうか。図７は LRS SAR データの見かけ深さ 300m から 1200m までのデータを深さ毎に正規化してデータのダイナミックレンジをそろえた後、さらに深さ方向に積分して作った地下反射波検出分布図である。図中、白く見える領域が特にはっきりと地下反射波が検出された領域であり、暗く見える部分は地下反射波を識別することができなかった領域であることを示す。LRS データの初期解析では月面表面物質の二酸化チタン（TiO2）含有量と LRS の地下反射波検出との間に逆相関の関係があることが見つかったが[7]、S/N が向上した LRS SAR データによる図７を見ると当初見られたほどにはその逆相関関係は強そうには見えない。今後、定量的な再解析が必要である。

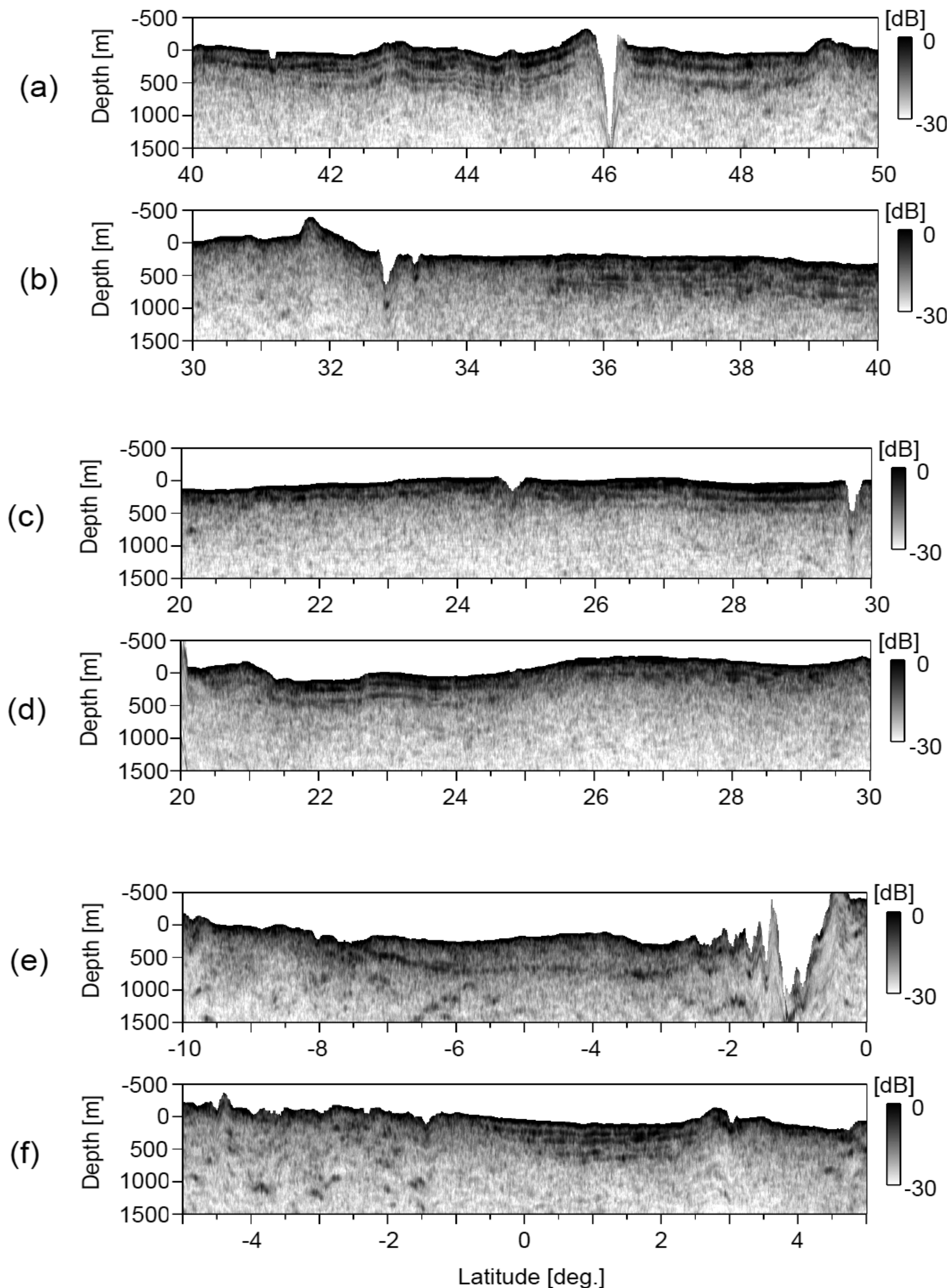


図６．さまざまな領域のLRS地下断面画像。(a)嵐の大洋（西経68.1度）、(b)雨の海（西経8.9度）、(c)晴れの海（東経20.9度）、(d)危機の海（東経59.0度）、(e)神酒の海（東経37.7度）、(f)スミス海（東経87.6度）。縦軸に示す深さは地下媒質の比誘電率を6.25と仮定した値。一部著しく画像が乱れている部分があるが、これは信号処理上の問題で生じたもので、実際の地形を表すものではない。

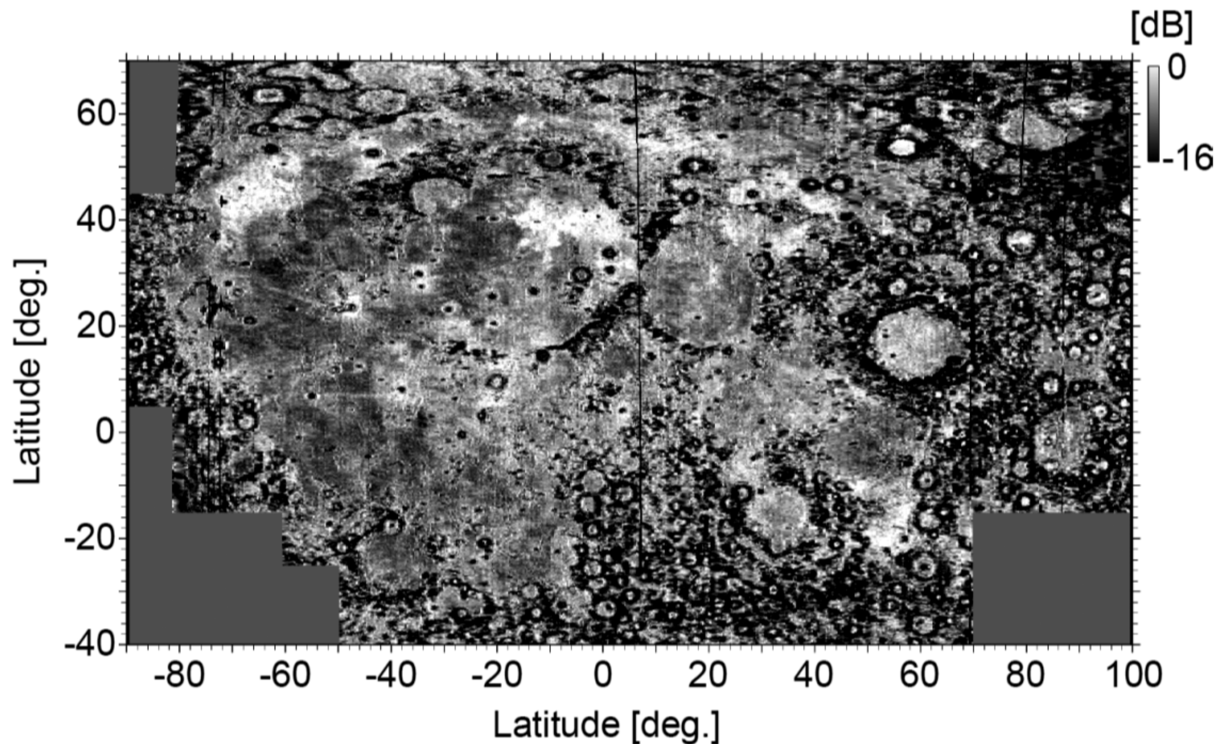


図７．　LRS 地下反射波検出分布図　。みかけ深さ 300m から 1200m までの観測データを積分。表示範囲は図３に同じ。

5.　むすび

本稿では、これまでに明らかになった「かぐや」月レーダらサウンダの観測による月の様相の一部を紹介した。地球上からは、LRS の観測周波数、5MHz による月のレーダ観測は電離層の遮蔽効果のために不可能である。これは宇宙空間からしかできない観測である。この、かぐや LRS の観測・データ処理は東北大学が率いる観測グループによって進められてきた。データの SAR 処理が終了した現在、本格的なデータの解析と研究が進行中である。研究の今後が楽しみである。

謝辞

本研究は、韓国地質資源研究院基本研究事業「地球・惑星進化追跡源泉技術開発」の中で行なわれた。本研究における大量のデータ処理は,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に有益なご指導とご協力をいただいた。

参考文献

[1] M. Kato, S. Sasaki, Y. Takizawa, and the Kaguya project team, The Kaguya Mission Overview, Space Sci. Rev., 2010, DOI 10.1007/s11214-010-9678-3
[2] T. Ono et al., The Lunar Radar Sounder (LRS) Onboard the KAGUYA (SELENE) Spacecraft, Space Sci. Rev., 2010, DOI 10.1007/s11214-010-9673-8
[3] R. J. Phillips et al., Apollo Lunar Sounder Experiment, APOLLO 17 Preliminary Science Report ch. 22, NASA, 1973

[4] T. Ono et al., Lunar Radar Sounder Observations of subsurface layers under the nearside maria of the Moon, Science, 2009, DOI: 10.1126/science.1165988
[5] T. Kobayashi et al., Synthetic Aperture Radar processing of Kaguya Lunar Radar Sounder data for lunar subsurface imaging, IEEE Trans. Geoscience and Remote Sensing, 2012, DOI: 10.1109/TGRS.2011.2171349
[6] 小林敬生 小野高幸、スーパーコンピュータ SX と月探査――「かぐや」による月地下構造探査――、SENAC, Vol.42, pp.101-105, 2009
[7] http://www.google.com/moon/
[8] A. Pommerol et al., Detectability of subsurface interfaces in lunar maria by the LRS/SELENE sounding radar: Influence of mineralogical composition, Gephys. Res. Lett., 2010, DOI: 10.1029/2009GL041681

［共同研究成果］

金属ナノ構造を含む一般フォトニック結晶の光学応答計算コード MPI 化による高速化

岩長 祐伸
物質・材料研究機構，科学技術振興機構さきがけ

周期長が光の波長程度である人工周期構造体（フォトニック結晶）に構成要素として金属ナノ構造を含む一般的なフォトニック結晶における光学応答を高精度に数値計算するためには，巨大なメモリを使用しながら 10000×10000 程度の一般複素数値行列の演算を大量に実行する必要がある．数年来，この計算コードの高速化，並列化に漸進的な改良を行ってきたが，さらなる高速化を実現するために MPI 化を実施した．本稿では，MPI 化によって得られた高速化の現状を中心に報告する．

1. はじめに

電磁波の数値計算に関しては多くのソフトウェアが存在し，商用シェアウェアで利用できるものとしてマクスウェル方程式を時間差分して電磁場を計算する FDTD（Finite Difference Time Domain）法，空間を有限要素で分割してマクスウェル方程式を解く有限要素法，マクスウェル方程式をフーリエ変換した方程式を解いて反射率・透過率などを算出する RCWA（Rigorously Coupled Wave Approximation）法などが比較的よく知られている．

いずれの方法でもマクスウェル方程式を数値的に解くことになるので，空間のグリッド分割が必要になる．定性的には，フォトニック結晶内における電磁波の波長 λ の数十分の 1 程度のグリッドの細かさが必要となる（RCWA 法では空間座量から電磁波の波数へのフーリエ変換が行われるので，さらにフーリエ波数 $2\pi i m/\lambda$ も大きな m 次まで取ることになる）．フォトニック結晶の単位胞に金属ナノ構造が含まれていると，ナノ構造のサイズ程度で電磁波が固有の分布を形成するため，ナノ構造の数十分の 1 程度のグリッドというさらに細かい空間分割が必要となってくる．図 1 は金属フォトニック結晶の細線で外枠を示す単位ドメイン内における金属部表面を分割した一例を示している．この例は有限要素法によるもので境界 yz，zx 面に周期境界条件を課して計算を行う．円柱状のロッドは直径 150 nm，x 軸方向の周期は 900 nm，z 方向の金属の厚さは 150 nm である．金属ナノロッドの直径 150 nm に対して，その数十分の 1 は 2~5 nm になる．

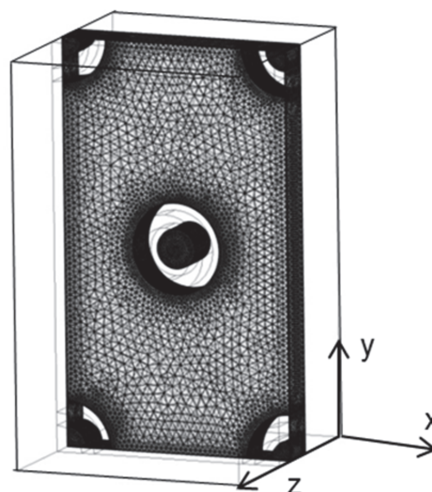


図 1 一般フォトニック結晶単位ドメイン（細線）内の金属部分のグリッド分割の例

FDTD 法や RCWA 法においても単位ドメインに対応する分割を設定して，対象の光学応答を算出することになる．結果として周期構造の単位ドメインを 10^8 程度の要素に分割して，各グリッドに物質パラメータである複素誘電率を割り当て，マクスウェル方程式の膨大な数値計算（最終的には数値行列演算）を行うことになる．

研究の現場で未知のフォトニック結晶の性質を明らかにするためには，試行錯誤を含めて多くの数値計算を実行しながら，計算精度を追求することが必要となるので，筆者は RCWA 法の原著論文［1］を基に有限次切断フーリエ展開を高速に取り扱うアルゴリズムを取り入れ，周期構造が積層した 3 次元的なフォトニック結晶を数値的に安定に取り扱うことができる散乱行列法［2］を組み込んだ数値計算コードを開発し，SX-9 上で運用してきた．これまでの RCWA・散乱行列コードのアルゴリズムの詳細，運用実例，改良については本誌上で報告してきた［3－7］ので繰り返さないが，漸進的に高速化・並列化の向上を実現し，MPI 化を除けば，改良の上限に達していた．

計算機環境についても読者の関心があると思われるので，ここで簡単に触れておきたい．近年のワークステーションの進歩に伴い，100 GB 程度のメモリをもつワークステーションを研究室に備えて計算を実行していくことも選択肢としてありうるが，RCWA 計算では対象次第では 1 TB に近いメモリを要する場合も稀ではないため SX-9 上でのコード運用に大きな利点を筆者は感じている．また，メモリだけならばクラスター型の計算機なら対応できるが，数値行列演算に関して SX-9 のほうが数倍から数十倍速かったという経験もある．コード自体に多くのチューニングを施さないと高速化できないクラスター計算機はユーザーにとっては恩恵を感じにくいと想像する．一方，SX-9 上では自動並列化オプション（-Pauto）のみで相当程度の並列化率を得て高速化できる利点がある［7］.

本稿では，以上の状況を受けて，今回 RCWA・散乱行列コードの MPI 化とその結果について述べる．通常，フォトニック結晶というと，シリコンなどの半導体材料または透明誘電体であるオパールなどからなるものを通常想定するので，以下では金属ナノ構造も含む，より一般的なフォトニック結晶を単に一般フォトニック結晶と呼ぶことにする．

2. MPI 化の概要

一般フォトニック結晶の光学応答では線形過程によって生じるもののみを考える．この場合，反射光，透過光に加えて回折光が生じうる．これらの応答の特性はスペクトルとして表現し，その性質を吟味することになる．図 2 は図 1 で示した金属フォトニック結晶の模式図と反射（R）スペクトル（赤線），透過（T）スペクトル（青線）である．入射角度 θ が 0 度（実線）と 10 度（点線）の場合を示している．

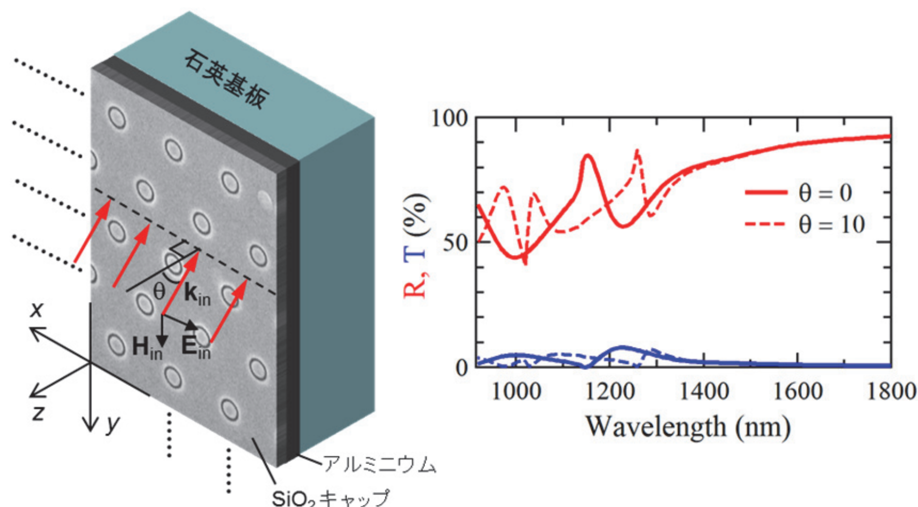


図 2　金属フォトニック結晶の模式図と反射（赤線）・透過（青線）スペクトル

図２左では入射光の配置を示している．入射面（入射光と反射光の進行ベクトルが張る平面）は xz 面であるように設定し，入射偏光は電場ベクトル $\mathbf{E}_{in}$ が xz 面内にある *p* 偏光とした．対応する実験も行い，計算結果とよい一致を示すことを明らかにした［8］．図２右では横軸を光の波長で表示した反射・透過スペクトルを示し，入射角度によってピークやディップが大きく変化していることが分かる．これらのスペクトル特性から共鳴状態を明らかにするのが筆者の数値計算を通じた物理学的な主題であり，実際に起源の異なるプラズモン共鳴として分類できるのであるが，考察結果と構造パラメータの詳細は文献［8］に譲り，ここでは数値計算の条件などについて述べていく．

図２の反射・透過スペクトルは RCWA 法によって SX-9 上で計算した結果であり，xy 面を 5×5 nm^2 のグリッドで分割して，フーリエ波数を±20 次まで取った．この計算を p16（16 CPU 下）で実行した結果，使用したメモリは 59.6 GB でスペクトル上の１点を計算するのに 9.8 分要した．１つのスペクトルには 139 点のデータ点があるから，22.7 時間かかったことになる．

光学スペクトルの各点は線形過程では互いに独立であるから，１つのスペクトル（列データ）を計算するために MPI 化で強制的に並列化することは大幅な計算時間の短縮につながることが期待できる．単純に言えば，図２の例を p64 で 16 CPU×4 の実行を行うことができれば，計算時間は 1/4 に短縮できる．この大幅な時間短縮，つまり高速化を実現するために今回コードの MPI 化を実施した．

MPI 化の方針は図３に図式化した．基本的な方針としては，スペクトル計算の主要ループを強制的に並列化すればよい．ただし，主要ループの前で共通保持できるポインターは確保し，物質パラメータなどを割り当てている．

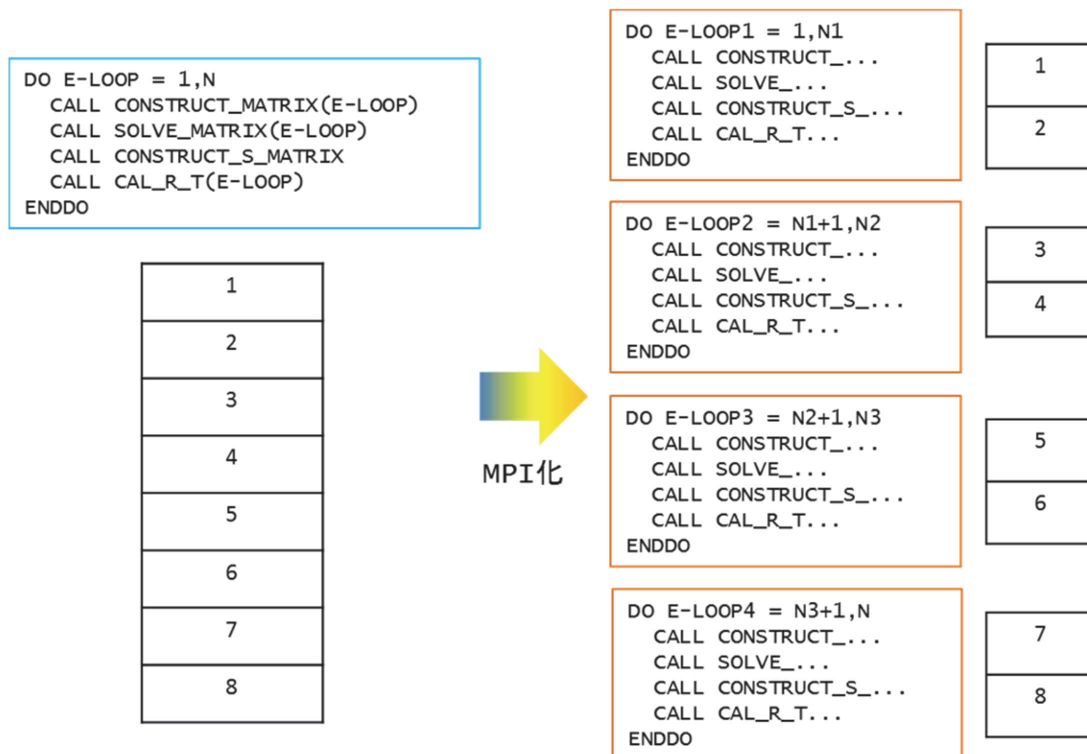


図３　MPI 化の概念図

計算実行時のメインループは図３で簡略化して示しているように E-LOOP が担う．このループはスペクトルの横軸に相当する量を掃引する．主な計算処理は，

1.　フーリエ変換したマクスウェル方程式の係数行列を構成する CONSTRUCT_MATRIX

2. 各周期層のなかのマクスウェル方程式を解く SOLVE_MATRIX
3. 各周期層の固有モードを使って一般フォトニック結晶全体の応答を表現する散乱行列を構成する CONSTRUCT_S_MATRIX
4. 最後にインプット（入射光）に対してアウトプット（反射光，透過光など）を散乱行列から算出する CAL_R_T

がある．各処理で使う配列が E-LOOP ごとに独立であるから，今回の MPI 化は実行できた．物理量として対象の線形応答からスペクトルを算出する場合は，今回のように電磁波の問題に限らず，同様の MPI 化が一般に可能である．

図３の左側は自動並列化のみでの実行を想定しており，p16 または p8 または s での実行に対応する．一方で図３右は MPI 化後に p64 に 16 CPU×4 並列で実行することを模式化している．スペクトルのデータ点が 8 点であれば，2 点×4 並列となり，理想的には計算時間は 1/4 になる．一般にはデータ点が 4 で割り切れるとは限らないが，その場合は端数を E-LOOP1 と E-LOOP2 に割り振る実装になっている．スペクトル計算において通例データ点は 100 から 1000 点ほどあるので，実際の運用において端数が出たことによる遅延が問題になることはなかった．

表 1 は p16 と p32（16 CPU×2 並列），p64（16 CPU×4 並列）での計算時間と使用メモリの変化の典型的な例である．テストケースとして 3 層積層からなる一般フォトニック結晶の反射・透過スペクトルを計算した（図１，２の構造とは異なる）．

	p16	p32	p64
データ１点の計算時間（分）	13.2	3.38	1.78
使用メモリ（GB）	50.15	100.67	200.34

表１　MPI 化の有無による計算時間，使用メモリの変化

表 1 から，p64 での計算時間が 1.78/13.2 = 1/7.4 と想定の 1/4 よりさらに高速化しているように読み取れる．この例では p16 でのデータ点数を 21 点に抑えたため，偶然時間のかかる波長域で計算してしまった可能性がある．共鳴状態の波長においては SOLVE_MATRIX のなかで call している行列の固有値・固有ベクトルを求める関数の実行に時間がかかる傾向がある．表 1 以外の例では，およそ 1/4 程度の時間短縮が得られており，当初の目標通りの高速化が実現できていると言える．また，p32 と p64 を比較すると，1.78/3.38 = 1/1.9 と p64 で約 1/2 の実行時間で計算を実行できている．

使用メモリに関しては，MPI による並列化で同じ配列を並列数だけ確保するので，p16 に対して p64 では約 4 倍，p32 では約 2 倍のメモリを使用している．これも想定通りであり，MPI の実装がうまくできていることの証左である．

3. メタマテリアルに関する最近の結果

この節では SX-9 上での光学応答計算コード運用から得られた，この１年の結果［8－10］のなかからメタマテリアルに関する成果［9,10］について述べる．メタマテリアルとは，一般フォトニック結晶のなかで考察する波長域よりも周期長が小さいものであり，回折光が生じないことから，従来の固体媒体では実現困難な性質を備えた新しい電磁波伝播媒体となることが期待されている．

3.1　フィッシュネット・メタマテリアルにおける固有モードの分散計算

フィッシュネット・メタマテリアルとは，金属・絶縁体・金属の積層構造に貫通孔を周期的にあけたフォトニック結晶のことを指す．図４左は構造模式図であり，図４右は p 偏光下の波数・光エネルギー分散図を示している．分散のデータ点は光学応答から得られた吸収スペクトルのピーク位置をプロットすることで得た．光吸収量 A は入射光の電磁エネルギーを 1 と規格化したと

きに $A = 1 - R - T$ で表される（R ：反射率，T ：透過率）．最低次下枝 a の分散式は

$$\hbar\omega = -\alpha k_x + \hbar\omega_1 \tag{1}$$

と表される．ただし，$\alpha > 0$，k_x は波数，$\hbar\omega_1$ は $k_x = 0$ での最低次のエネルギーである．

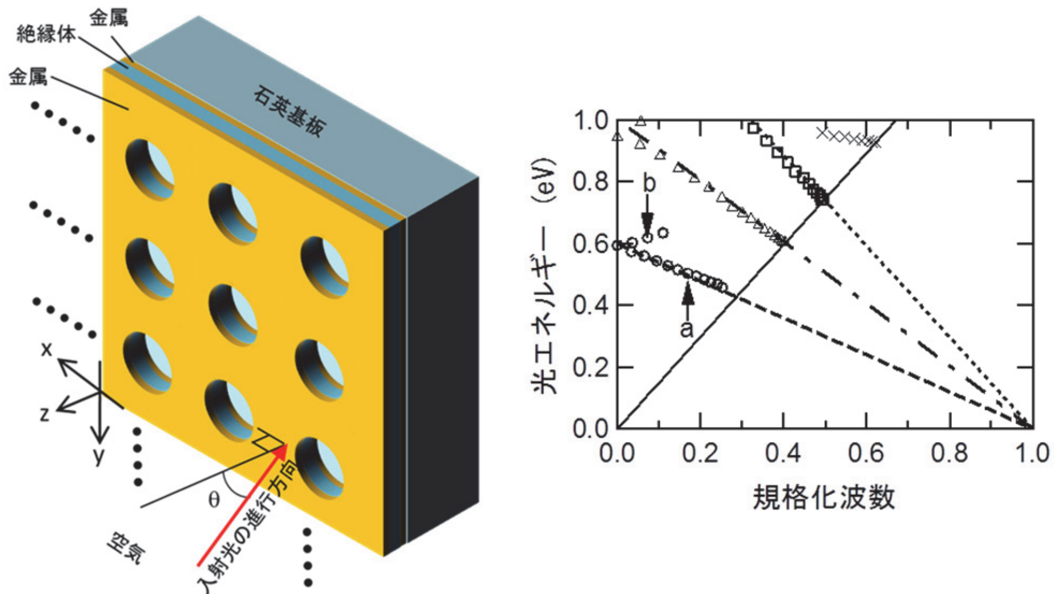


図４　フィッシュネット・メタマテリアルにおける固有モードの波数・エネルギー分散

金属層に挟まれた絶縁体層内部の導波路モードの群速度 v_g は

$$v_g = \frac{\partial\omega}{\partial k_x} \tag{2}$$

で与えられ，式（1）から $v_g = -\alpha < 0$ となる．つまり，最低次下枝 a の導波路モードは負の群速度を持つことが分かった．このような特異な条件が満たされることで，このフィッシュネット・メタマテリアルは，斜め入射下で負の屈折現象が生じる媒体となっていることが今回初めて明らかにできた［9］．なお，最低次のモードであっても上枝 b の群速度は正であり，負の屈折現象は生じない．

3.2　金属ナノロッド列からなるメタマテリアルにおける構造共鳴の変化

金属ナノロッドが周期的に配列した場合に，単独のロッドだけでは生じない集団的な共鳴状態を見出した．図５はその具体的な構造と透過スペクトルを示している．周期構造に上から入射光が当たる配置を考える．

金属ナノロッド間の距離（ギャップ）を 0 nm から 20 nm まで少しずつ変えていく．ギャップが 0 nm のときは入射直線偏光が ψ=45 度（ψ は x 軸と直線偏光のなす角）に対して波長 1500 nm 以上で透過率がほぼ 0 になっている．一方，入射偏光 ψ=135 度に対しては同じ波長域で 40%程度の透過率がある．このことは ψ=45 度では広帯域な共鳴状態が形成されて，光エネルギーが吸収されていることを示唆している．実際，電磁場分布の解析からナノロッド間の集団的な共鳴が生じていることを明らかにした［10］．ナノロッド間の集団共鳴を示したのは，筆者の知るかぎりにおいてこの例が初めてである．この集団共鳴はロッド間の距離に非常に敏感でわずか 10 nm のギ

ャップでもほぼ失われてしまう．構造設計の精度が非常に重要であることを示唆している．

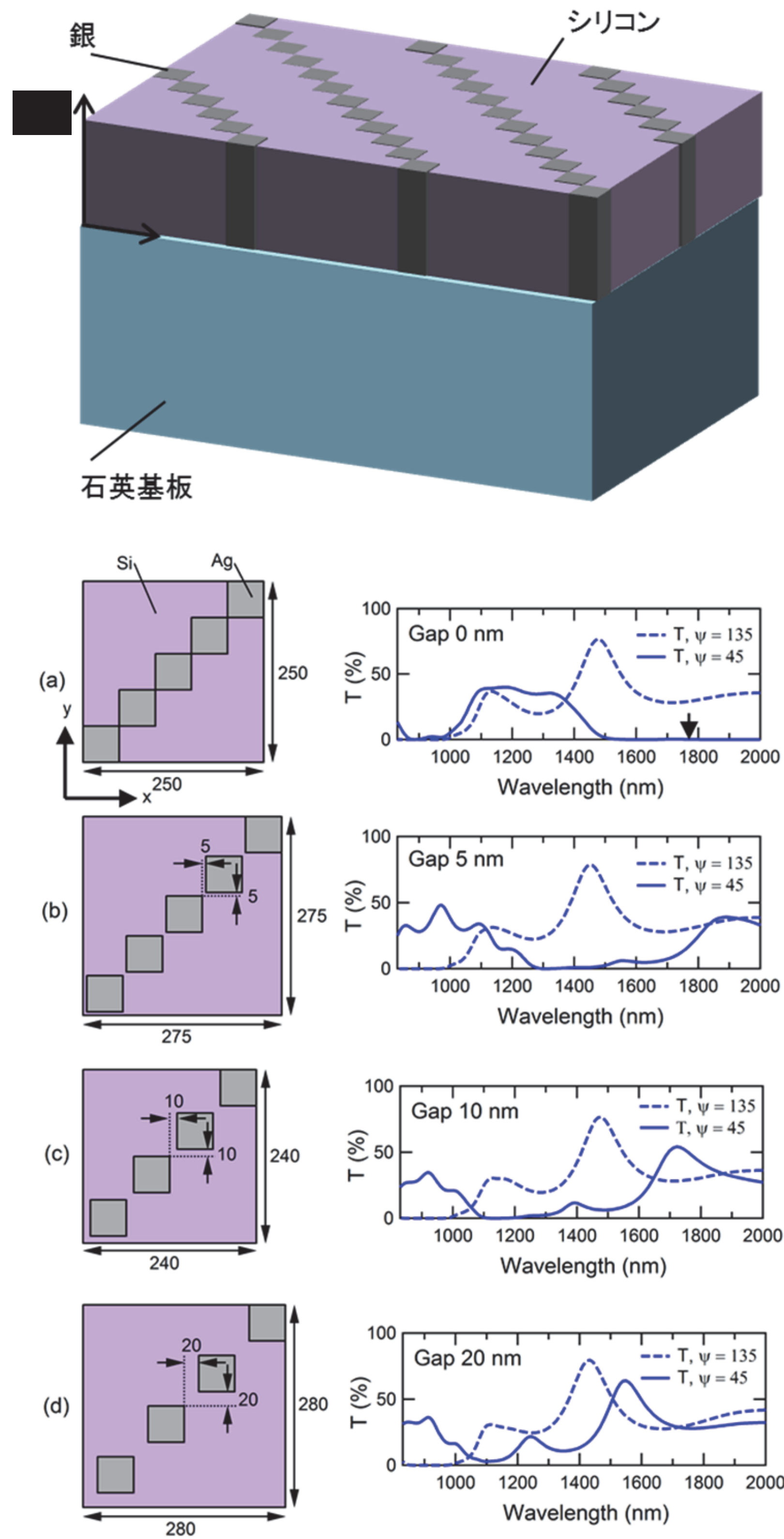


図５　金属ナノロッド列における構造変化と偏光透過スペクトル

一方で入射偏光 ψ=135 度のとき，つまり入射直線偏光がナノロッド列と直交しているとき，透過スペクトルはほとんどギャップに依らない．この結果は集団的な共鳴状態が偏光選択性をもつことを示している．このようにフォトニック構造と光の状態（偏光）の組み合わせを含めて，対象の光機能を設計していく必要がある．

4. まとめと今後の展望

一般フォトニック結晶の光学応答を数値計算するコードの MPI 化を実装し，高速化（計算時間の短縮）を試みた．当初の期待通り，p64 では p16 の約 1/4 の実行時間でコードの運用ができるようになった．光学スペクトルの計算が電磁波の線形応答をもとにしたものであったので，各データ点を独立に扱うことができ，自ずと MPI 化に適していた．電磁気学の問題に限らず，線形スペクトル計算の MPI 化は物理学一般において有効であると考えられる．

計算時間の短縮は様々な対象をより多く扱うことを可能にするので，コードを運用するユーザーとしても大きな恩恵にあずかることができる．この 1 年だけでも様々な対象を詳細に研究できた［8－10］ことも端的な成果と言える．

本稿で紹介した例では p16 実行下で 50 GB 程度のメモリを使ったものが多かったが，その他の事例ではメモリ 500 GB というものもあった．SX-9 でもメモリの上限をそろそろ意識しなくてはならない領域に踏み込みつつある．私見ながら，SX-9 のスペックは電磁気学の問題を解くのに適していると感じている．量子力学の問題の多くは現在の大型計算機を持ってしても解けていないことは周知であり，チャレンジングな課題であることは確かであるが，単に計算機のスペック不足なのか，それとも良いアルゴリズムを見つけることができていないだけなのか，筆者にはいまだに判然としない．今回用いた RCWA 法に関して，実用に耐えるアルゴリズムの発見［1］までに約 30 年を要した歴史的経緯がその思いを強めている．

今回の MPI 化の効果として，さらに膨大な計算量を必要とする数値的な新規フォトニック構造探索が可能になった．遺伝アルゴリズム［5］と今回の光学応答計算コードの融合による構造探索を現在実行している．比較的単純な構造に潜む共鳴状態を解くことがこれまでの主な研究であったが，人間の想像を超える新しい人工ナノ構造の発見につなげていくことを目標に SX-9 を今後活用していきたいと考えている．

謝辞

本研究における MPI 化の実装は SX-9 開発元 NEC のご協力を得て，東北大学サイバーサイエンスセンターと共同で実施したものであり，期待にたがわぬ MPI 化を行っていただきました．この場を借りて厚くお礼を申し上げます．また，本研究の一部は科学技術振興機構さきがけ，科学研究費補助金（No. 22760047）の支援を受けて行われました．

参考文献

[1] L. Li, “New formulation of the Fourier modal method for crossed surface-relief gratings,” J. Opt. Soc. Am. A **14** (10), 2758–2767 (1997).
[2] L. Li, “Formation and comparison of two recursive matrix algorithm for modeling layered diffraction gratings,” J. Opt. Soc. Am. A **13** (5), 1024–1035(1996).
[3] 岩長祐伸,「散乱行列法を用いたフォトニック結晶の光学応答解析」SENAC **39** (3), 25–32 (2006).
[4] 岩長祐伸,「メタマテリアルにおける有効光学定数の決定法と応用」SENAC **40** (3), 5–14 (2007).
[5] 岩長祐伸,「遺伝アルゴリズムを用いた光機能性人工構造体の探索」SENAC **41** (3), 43–51 (2008).
[6] 岩長祐伸,「メゾ周期構造体における電磁波散乱の高精度数値計算」SENAC **42** (4), 9–18 (2009).
[7] 岩長祐伸,「積層プラズモニック結晶における光機能性発現」SENAC **44** (2), 49–56 (2011).
[8] M. Iwanaga, N. Ikeda, and Y. Sugimoto, “Enhancement of local electromagnetic fields in plasmonic

crystals of coaxial metallic nanostructures," Phys. Rev. B **85** (4), 045427 (2012).
[9] M. Iwanaga, "In-plane plasmonic modes of negative group velocity in perforated waveguides," Opt. Lett. **36** (13), 2504–2506 (2011).
[10] M. Iwanaga, "Collective Plasmonic States in Metallic Nanorod Array and Their Application," in *Nanorods*, edited by O. Yalcin (InTech, Rijeka, 2012) Chap. 4. (http://dx.doi.org/10.5772/36198)

[研究成果]

大規模計算システムにおけるBCMの性能評価

†‡小松 一彦　†¶曽我 隆　†‡江川 隆輔　§‡滝沢 寛之　†小林 広明
†東北大学サイバーサイエンスセンター
§東北大学情報科学研究科
¶NEC システムテクノロジー株式会社
‡JST CREST

*BCM(Building Cube Method)*は，大規模計算システムでの流体シミュレーションを念頭に設計された次世代 CFD(Computational Fluid Dynamics)ソルバーである．BCM では等間隔直交格子を採用しており，流体シミュレーションに必要な演算を容易に均等分割することができるため，大規模計算システムを用いた並列計算に適している．本報告では，様々な大規模計算システムの特徴を考慮して BCM の実装・最適化を行い，その性能評価に基づき実効性能を議論する．

1.　緒言

1960 年代以降，コンピュータを用いた数値計算が流体力学におけるシミュレーションや解析に用いられてきた．一般的に CFD の分野では，複雑な形状を再現するために，境界適合格子(boundary-fitted mesh)や非構造格子(unstructured mesh)が広く用いられる．これらの格子を用いることにより，流体解析の対象となる形状を忠実に再現することが可能なため，より正確な CFD を行うことができる．しかしながら，近年，飛行機や高速鉄道，自動車などのように，非常に形状が複雑な物体が CFD の主な対象となっている．そのため，非構造格子を用いた CFD においては，モデルデータからの格子作成などの処理を行う前処理，複雑なモデルの流体解析計算，その解析結果に基づく可視化やデータ抽出などの後処理に非常に膨大な時間が必要となる．また，大規模計算システムを用いた並列処理に必要となる分割方法も複雑化し，演算負荷を均等に保つのが難しいため，非構造格子に基づく大規模 CFD の効率的な並列処理は困難になっている．

この問題を根本的に解決するために，次世代の CFD ソルバーである BCM が提案されている[1,2,3]．BCM は大規模計算システムを用いて様々な流体現象を効率的にシミュレーションするために，格子の形状が単純な直交格子(Cartesian mesh)を採用している．これにより，複雑化・煩雑化した前処理・流体解析・後処理を，単純にすることができる[4]．さらに，BCM は CFD に必要となる演算を容易に均等分割することができるため，大規模計算システムにおいて負荷のバランスを保った効率的な並列処理を実現できる．

本報告では，大規模計算システムを用いて BCM の性能評価・解析を行い，BCM の特性を明らかにする．近年の大規模計算システムの性能向上は著しく，多様な大規模計算システムが登場している．汎用プロセッサを搭載したスカラ型大規模計算システムや描画処理用プロセッサ(Graphics Processing Unit, GPU)を計算に応用するアクセラレータ型大規模計算システム，大量の計算を同時に処理するベクトル型大規模計算システムなど，プロセッサのアーキテクチャやシステム構成によりその特徴が異なる．それぞれの大規模計算システムの特徴を考慮して BCM の実装と最適化を行い，その性能評価・解析を行うことで，BCM の特徴を明らかにし，更なる高速化の検討を行う．

2.　等間隔直交格子を用いた Building Cube Method の概要

BCM は，高密度格子を必要とするような複雑な物体周りの大規模 3 次元流体解析のために設計されている．図 1 に示すように，BCM では CFD の解析領域を ***cube*** と呼ばれる部分領域に分割し，さらにそれぞれの cube を ***cell*** と呼ばれる高密度の等間隔直交格子で分割する．cube のサイズは物体の形状や流れの特徴に応じて決められる[4]．格子の形状が非構造格子のように複雑ではな

いため，BCM のアルゴリズムをシンプルにすることができ，前処理や流体解析，後処理の高速化が期待できる．

BCM は演算を容易に均等分割することができるため，並列処理に適している．cube ごとの計算は完全に独立しており，各 cube の計算に必要となる演算量とデータ量は同一である．そのため，cube 単位で計算を分割することにより，均等な演算に分割することができる．さらに，cube 内の隣接する cell 同士には依存関係があるが，各 cell の計算に必要となる演算量とデータ量は同一である．そのため，cell 同士の依存関係を解消することができれば，更なるデータ並列性を抽出することが可能になる．

図 2 に BCM による非圧縮性流体解析のフローチャートを示す[3, 4]．BCM では，流れの基礎方程式に非圧縮性 Navier-Stokes 方程式を用いており[5, 6, 7]，スタガード配置(staggered arrangement)の要素に対して有限差分法(finite difference scheme)による部分段階法(fractional-step method)で計算を行う．部分段階法では，流体解析を時間ステップごとに，仮速度場計算ステージ，圧力場計算ステージ，速度場計算ステージの主に 3 つのステージに分けている．各ステージには場の計算と隣接する cube 間とのデータ交換が含まれている．

これらのステージの中で，SOR(Successive over-relaxation)法を用いてポアソン方程式を解く圧力場の計算が最も支配的である．これは，隣接する 6 つの cell を用いたステンシル計算を，圧力場の差分が収束するまで，全ての cube の全ての cell に対して繰り返し計算を行う必要があるためである．

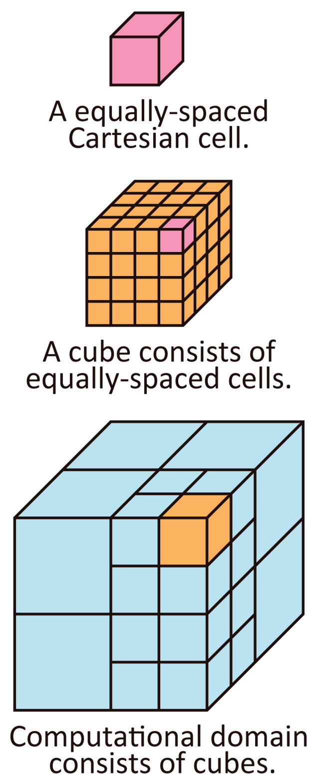


図 1　BCM における計算格子

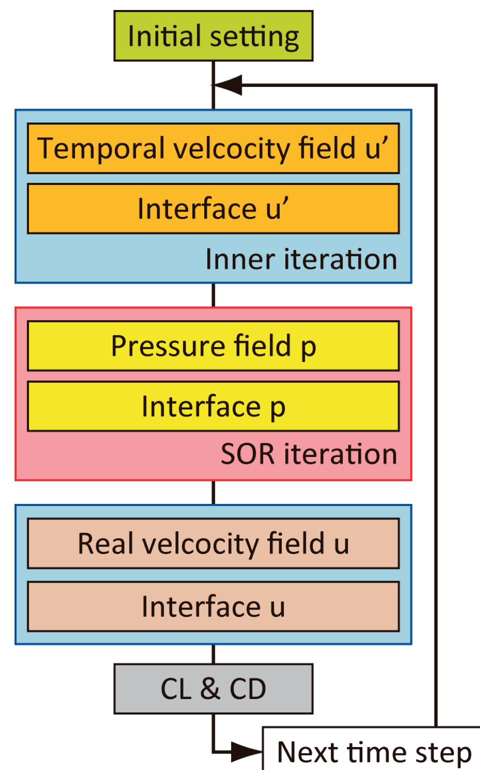


図 2　BCM における流体解析の流れ

3.　大規模計算システムへのBCMの実装

大規模計算システムによる並列処理によってBCMの計算時間を短縮させるためには，プロセッサのアーキテクチャやシステム構成を考慮した実装と最適化が必要になる．本章では，表1に示すプロセッサを搭載する大規模計算システムの概要を述べ，それぞれの大規模計算システムへのBCMの実装・最適化について述べる．

表1　大規模計算システムに搭載されるプロセッサの諸元

システム	理論演算性能(Gflops/s)	メモリバンド幅(GB/s)	コア数	オンチップメモリ	B/F値
Nehalem EP	46.93	25.6	4	256KB L2/core 8MB shared L3	0.55
Nehalem EX	74.48	34.1	8	256KB L2/core 24MB shared L3	0.47
FX-1	40.34	40.0	4	6MB shared L2	1.0
SR16000 M1	245.1	128	8	256KB L2/core 32MB shared L3	0.52
SX-9	102.4	256	1	256KB ADB	2.5
Tesla C1060	78	102	1	16KB/SM	1.3

3.1　スカラ型大規模計算システムにおけるBCMの実装

Intel Nehalem EPクラスタやIntel Nehalem EXクラスタ，富士通FX-1，日立SR16000 M1は，それぞれNehalem EP，Nehalem EX，SPARC64VII，IBM Power 7といった多数の汎用型スカラプロセッサを搭載するスカラ型大規模計算システムである．表1に示すような複数のコアを持つスカラプロセッサを1つまたは複数搭載し1ノードを構成し,このノードをInfinibandなどの高速なネットワークで多数接続することで，スカラ型大規模計算システムが構築されている．スカラプロセッサは複数のデータに対して同じ演算を実行することが可能なSIMD(Single Instruction Multiple Data)命令や，局所性の高いデータへのアクセスレイテンシを削減するための大容量オンチップキャッシュメモリを備えている．

BCMをスカラ型大規模計算システムで実装するためには，多数のスカラプロッセッサの有効活用が重要となる．BCMの高いcube並列性を利用し，cubeを計算システムのノード，そしてノード内のプロセッサへ，プロセッサ内のコアへと階層的に割り当てることで，多数のコアを用いた並列処理を行う．cube毎の演算量は完全に同一であるため，ノード間，プロセッサ間，コア間の演算は均等となり，多数のスカラプロセッサを利用した効率的な計算が可能になる．スカラプロセッサにおける大容量キャッシュメモリを効率的に利用するために，データへのアクセスが連続であり局所性が高い通常のSOR法を用いる．また，SIMD命令を利用するために，コンパイラによる最適化を行っている．より効率的にSIMD命令を利用するためには，スカラプロセッサごとにそれぞれ定義されているインラインアセンブラなどを用いた実装が必要になる．

3.2　ベクトル型大規模計算システムにおけるBCMの実装

NEC SX-9は大規模な*SMP(Symmetric Multi Processing)*ノードから構成されるベクトル型大規模計算システムである．各SMPノードは102.4Gflops/sのベクトルプロセッサを16個搭載している．SX-9ベクトルプロセッサでは256要素の同一演算を同時に処理することができる．

SX-9による効率的な処理を実現するためには，BCMの並列性をできる限り抽出し，できるだけ多くの要素を同時に演算する必要がある．そのため，SX-9におけるBCMの実装には，cubeの並列

性だけでなく，cell の並列性も利用可能な，Red-Black SOR 法を採用する．Red-Black 法では隣接する cell 同士の依存関係を解消するために，cube 内のすべての cell を図 3 に示すような red グループと black グループに分ける．red グループの cell 同士は依存関係がないため，red グループのすべての cell の圧力場を同時に計算することが可能となる．同様に，red グループの圧力場の計算が終わった後に，black グループのすべての cell の圧力場を同時に計算する．この Red-Black SOR 法により，cube の並列性だけでなく cell の並列性も利用して，複数の cell の圧力場の計算をベクトルプロセッサで同時に処理することができる．

しかしながら，一般的にベクトル処理による Red-Black SOR 法では，同時に処理できる要素数が減ってしまいベクトル演算性能が低下してしまう可能性がある．これを抑制するために，マスクテーブルを用いて red グループと black グループを判別することにより，ベクトル演算に十分な要素数を確保する．

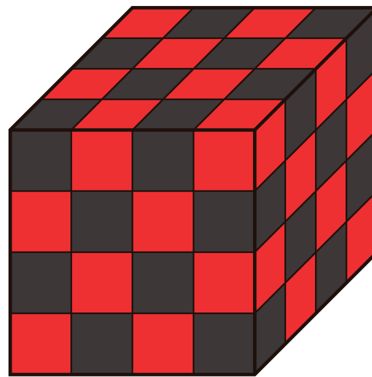


図 3　Red cell グループと black cell グループ

```
・・・
        !cdir ON_ADB(prs)
        do icolor=1, 2
                do icell=icolor,max_cell*max_cell*max_cell,2
                        if(mask(icell,1,1,icolor).eq.1) then
                                p0   = prs(icell,1,1)
                                pxm = prs(icell-1,1,1)
                                pym = prs(icell-max_cell,1,1)
                                pzm = prs(icell-max_cell*max_cell,1,1)
                                pxp = prs(icell+1,1,1)
                                pyp = prs(icell+max_cell,1,1)
                                pzp = prs(icell+max_cell*max_cell,1,1)
        ・・・
```

図 4　SX-9 における圧力場計算の疑似コード

また，SX-9 の性能をさらに引き出すために，プロセッサに搭載されているオンチップメモリである *ADB(Assignable Data Buffer)* を活用する．ADB は 256KB の容量を持つソフトウェア制御が可能なオンチップキャッシュメモリである．プログラマに指示されたデータに一度アクセスされると ADB に保存され，次のアクセスの際に ADB からデータを取得することができる．図 4 に示すように，圧力場のステンシル計算の際に 7 回再利用されるデータを ADB に保存する ON_ADB ディレクティブをソースコードに挿入する．これによって，メインメモリと ADB の両方から別々

のデータをベクトル演算器に供給することができるため，より高い実効メモリバンド幅を実現し，BCM の計算を高速に実行できることが期待できる．

3.3　アクセラレータ型大規模計算システムにおける BCM の実装

代表的なアクセラレータ型大規模計算システムの 1 つである *GPU(Graphics Processing Unit)* クラスタでは，1 つまたは複数の GPU を搭載したノードが多数接続されている．GPU は *CUDA(Compute Unified Device Architecture)* では，数百の *SP(Stream Processors)* からなるメニーコアのプロセッサと考えることができる[5]．CUDA では SP が ***SM(Stream Multiprocessor)*** としてグループにまとめられ，SM の中の SP が同時に動作する．

GPU クラスタでは，1 つの GPU に数百コアも搭載されているため，システム全体のコア数は膨大になる．また，GPU はメモリアクセス中に他の演算を実行することで，メモリアクセスレイテンシを隠蔽することが可能なため，GPU の高い性能を引き出すためには並列処理が可能な処理を可能な限り増やす必要がある．そのため，GPU クラスタへの実装においても cube と cell の両方の並列性を抽出可能な Red-Black SOR 法を用いる．図 5 に示すように，cube をサブグループに分けて，各ノードへ割り当てる．割り当てられた cube をさらに GPU の SM に割り当てる．cube 中に含まれる各 cell の計算を thread に割り当て，SP で並列に実行する．各 cell の演算量は同一であるため，複数ノードの GPU を用いた効率的な大規模並列処理が期待できる．

GPU の高い演算性能をさらに引き出すためには，演算の割り当てのみではなく SP へのデータ供給が非常に重要となる．GPU のメモリは階層構造になっており，***共有メモリ(shared memory)*** は各スレッドから共有されている．共有メモリの容量は小さいが，メモリアクセスレイテンシは非常に短い．一方，***大域メモリ(global memory)*** は，容量が大きいオフチップメモリであり，メモリアクセスレイテンシは長い．最大限に共有メモリを活用し，大域メモリへの効率的なアクセスを行うなど，これらの両方のメモリを適切に利用することで GPU の性能を引き出すことができる．

容量の小さな共有メモリを効率的に利用するために，図 6 に示すように，共有メモリ上にリングバッファを構築する．ステンシル計算に利用される cell を平面単位でリングバッファに保存することによって，大域メモリへのアクセス数を削減するだけでなく，限られた共有メモリの容量を効率的に利用することが可能になる．

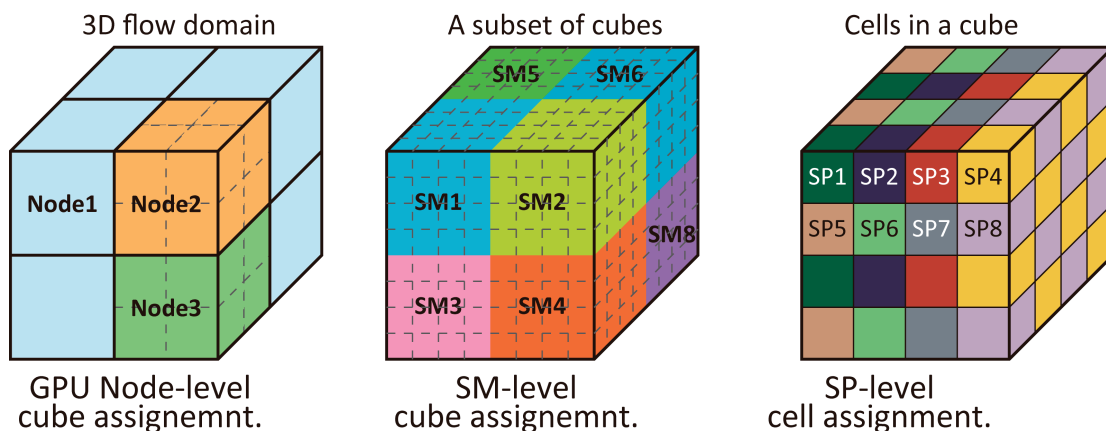


図 5　GPU クラスタ大規模計算システムのおける計算割当

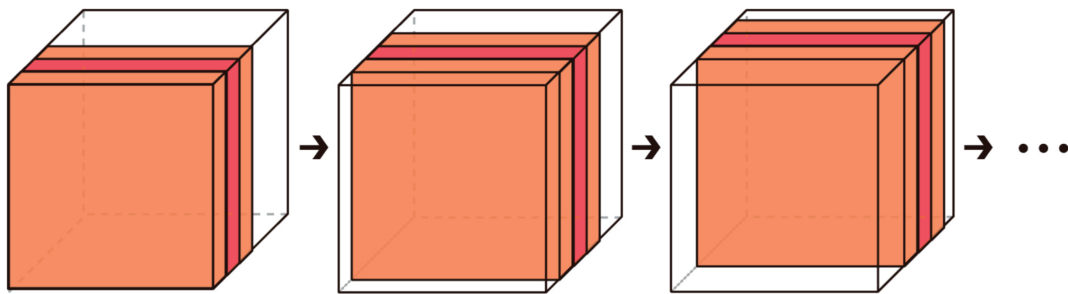


図 6　リングバッファを用いた効率的な共有メモリの活用

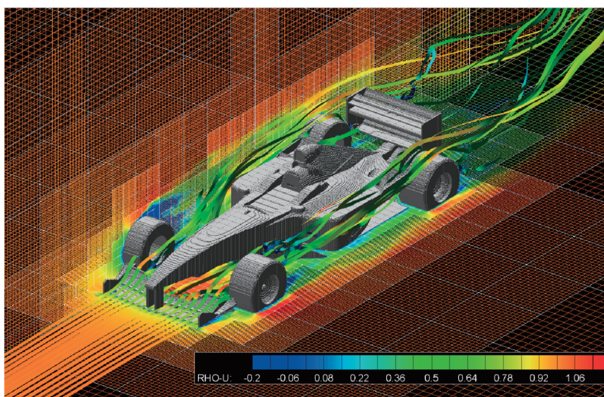


図 7　F1 モデル（1 億 cell）

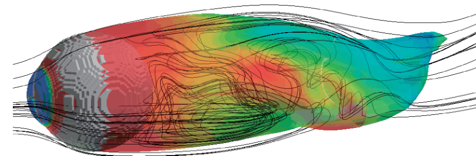


図 8　球体モデル（500 万 cell）

4.　大規模計算システムにおける BCM の性能評価

表 1 に示す大規模計算システム上で，BCM による 3 次元テストモデル周りの流体解析を実行して，その実効演算性能を評価した．3 次元テストモデルには図 7 に示す F1 モデルと図 8 に示す球体モデルを用いた．F1 モデルは複雑で大規模なモデルデータを想定しており，約 1 億 cell の格子からなる．球体モデルは比較的小規模なモデルデータを想定しており，約 500 万 cell の格子からなる．

図 9 に F1 モデルを用いた BCM による流体シミュレーションの実効性能を示す．この図より，SX-9 が他のスカラ型大規模計算システムに比べ高い実効性能を達成しているのが分かる．ステンシル計算はデータ転送あたりの演算が少ないため，実効メモリバンド幅が BCM の実効性能に影響する．SX-9 は，高い理論メモリバンド幅に加え，ADB を効率的に利用することにより，実効メモリバンド幅を高めることができる．これにより，SX-9 が他のスカラ型大規模計算システムに比べ，高い実効性能を達成することができたと考えられる．

FX-1 の理論メモリバンド幅が Nehalem EP や Nehalem EX の理論メモリバンド幅よりも高いにも関わらず，FX-1 の実効性能は Nehalem EP や Nehalem EX に比べ低くなっている．これは，FX-1 の実効メモリバンド幅が Nehalem EP や Nehalem EX の実効メモリバンド幅よりも低いためである．実効メモリバンド幅を測定するためのベンチマークソフトである STREAM ベンチマークを実行した結果と，Nehalem EP や Nehalem EX はそれぞれ 17.0GB/s，17.6GB/s のメモリバンド幅を達成しているにもかかわらず，FX-1 は 10.0GB/s と実効メモリバンド幅が低いことが分かる．このため，Nehalem EP や Nehalem EX の実効性能が FX-1 の実効性能を上回った．

図 10 に球体モデルを用いた BCM による流体シミュレーションの実効性能を示す．球体モデルの結果には GPU クラスタによる結果も含まれている．この結果を見ると，GPU の大域メモリの容量が限られるため F1 モデルのような大きなモデルに対しては実行できないが，球体モデルのようなモデルにおいては，GPU クラスタの性能が SX-9 とほぼ同等，スカラ型大規模計算システムよりも高いことが分かる．これは多数の SP に効率的に演算を割り当てることで，高いデータ並列処理能力を発揮できたためである．また，共有メモリを効率的に利用することによって，スカラプロセッサに比べ高い実効メモリバンド幅を引き出すことができたことも要因の 1 つである．もう 1 つの要因としては，球体モデルのデータが小さいため，SX-9 や他のスカラ型大規模計算システムにおいては十分な性能を引き出すことができないことが挙げられる．図 9 に示す F1 モデルから得られた実効性能と比較すると，球体モデルから得られた実効性能が低くなっているのが分かる．

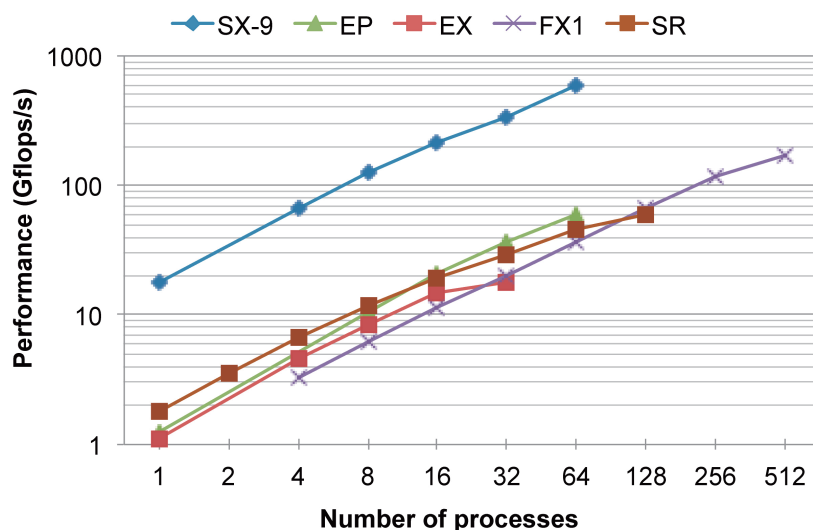


図 9　F1 モデルを用いた場合の BCM の実効性能

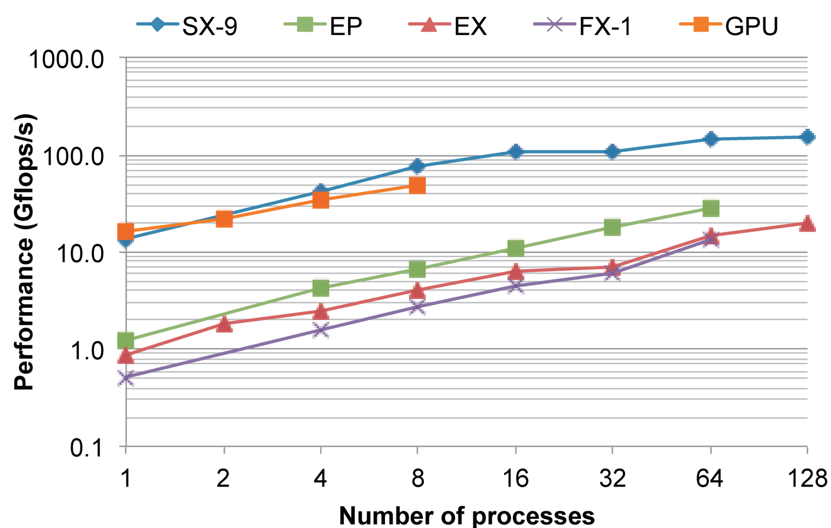


図 10　球体モデルを用いた場合の BCM の実効性能

GPU による流体シミュレーションにおいては，圧力場の計算時間の大部分がデータ転送で占められる．これは，GPU の活用によって演算時間が短縮されたが，ノード内の GPU と CPU や他のノードとのデータ転送に時間がかかるためである． GPU クラスタを用いたさらなる高速化を実現するためには，演算中にデータを転送しデータ転送時間を隠蔽するなどの工夫が必要となる．

図 9 と図 10 において，理論性能に対する実効性能の比率である実行効率を見ると，SX-9 が約 17%と他の大規模計算システムの 1.5～3.2%と比べ非常に高いことが分かる．これは高い実効メモリバンド幅を実現することによって，ベクトルプロセッサが効率的にベクトル処理を行うことができたためである．

図 11 に示されている F1 モデルにおけるスケーラビリティをみると，すべての大規模計算システムにおいて高いスケーラビリティが実現できているのが分かる．これは，F1 モデルの並列度が高いため，十分な並列処理可能な計算を割り当てることができたことと，ノード間をつなぐネットワークのバンド幅が十分であったためだと考えられる．図 12 に示す球体モデルにおけるスケーラビリティは F1 モデルのスケーラビリティと比べ低い．GPU クラスタにおいては，データ転送のオーバーヘッドが要因でスケーラビリティが低下したと考えられる．他の大規模計算システムにおいては，球体モデルでは並列可能な処理が不足したため，スケーラビリティの低下が見られる．

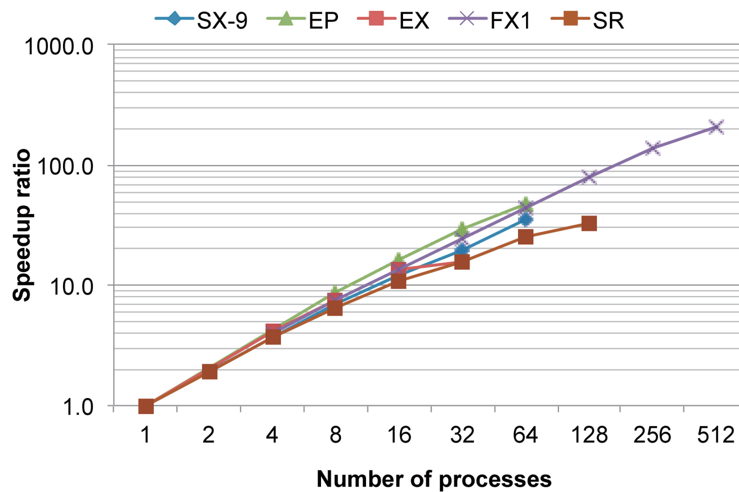


図 11　F1 モデルにおける BCM のスケーラビリティ

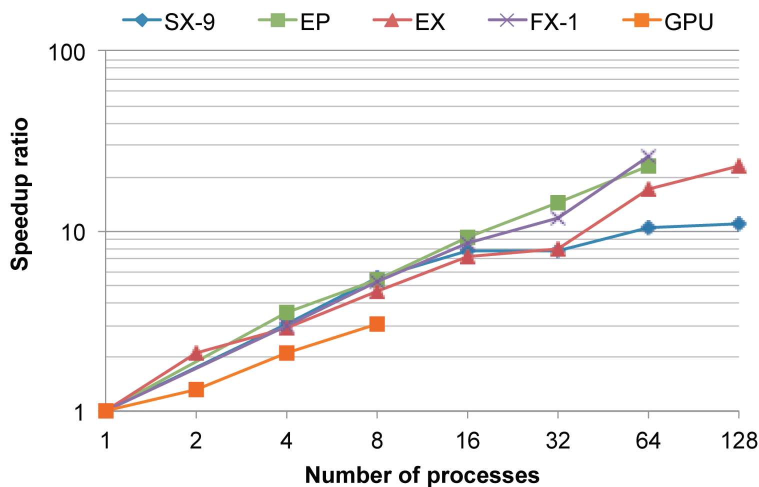


図 12　球体モデルにおける BCM のスケーラビリティ

5. 結言

本報告では，スカラ型・ベクトル型・アクセラレータ型の大規模計算システムを考慮した BCM の実装と最適化を述べ，その性能評価を通じて実効性能を議論した．それぞれの大規模計算システムの性能を引き出すために，大規模計算システムに搭載されているプロセッサのアーキテクチャやシステム構成を考慮して BCM の実装を行った．特に，それぞれの大規模計算システムにおけるプロセッサとそのオンチップメモリを効率的に利用するための最適化を行った．SX-9，Nehalem EP クラスタ，Nehalem EX クラスタ，FX-1，SR16000 M1，GPU クラスタを用いて性能評価を行った結果，実効メモリバンド幅が BCM の性能に大きな影響を及ぼしていることが明らかになった．これにより，SX-9 のような高メモリバンド幅の大規模計算システムが BCM による流体シミュレーションの高速化に適していることが分かった．

謝辞

本研究を進めるにあたり JAXA 中橋和博博士，金沢工業大学 佐々木大輔講師，東京農工大学大学院 高橋俊助教，NEC 撫佐昭裕博士，東北大学サイバーサイエンスセンター関係各位には大変有益なご助言をいただいた．また，本研究は，北海道大学情報基盤センター，東北大学サイバーサイエンスセンター，名古屋大学情報基盤センター，ドイツ シュトゥットガルト大学 HLRS 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，本研究の一部は，文部科学省科研費研究(S)(21226018)と文部科学省科研費若手研究(B)(23700028)と科学技術振興機構(JST)戦略的創造研究推進事業(CREST)の助成を受けている．

参考文献

[1] Nakahashi, K., “High-density mesh flow computations with pre-/post-data compressions,” AIAA paper, pp. 2005–4876, 2005.

[2] Takahashi, S., Ishida, T., Nakahashi, K., Kobayashi, H., Okabe, K., Shimomura, Y., Soga, T., Musa, A., “Study of high resolution incompressible flow simulation based on cartesian mesh,” AIAA paper 47th AIAA Aerospace Sciences Meeting, pp. 2009–563, 2009.

[3] Takashi, S., “Study of large scale simulation for unsteady flows,” Ph.D. thesis, Tohoku University, 2009.

[4] Ishida, T., Takahashi, S., Nakahashi, K., “Efficient and robust cartesian mesh generation for building-cube method,” Journal of Computational Science and Technology **2**(4), pp. 435–445, 2008.

[5] Kim, J., Moin, P., “Application of a fractional-step method to incompressible navier-stokes equation,” Journal of Computational Physics 59, pp. 308–323, 1985.

[6] Perot, J.B., “An analysis of the fractional step method,” Journal of Computational Physics 108, pp. 1–58, 1993.

[7] Dukowicz, J.K., “Approximate factorization as a high order splitting for the implicit incompressible flow equations,” Journal of Computational Physics 102, pp. 336–347, 1992.

Fortran スマートプログラミング

— 第１回　基本的なプログラムの書き方 —

田口　俊弘
摂南大学理工学部電気電子工学科

今回から３回にわたって，数値計算やコンピュータシミュレーションを目的とした Fortran による計算機プログラム作成法を説明します．最近の大学における計算機の講義では C 言語で教えるところが増えてきて Fortran は影が薄くなっていますが，コンピュータシミュレーションの分野では Fortran の方が良く使われています．これは，Fortran が科学技術計算用言語として発展してきたので，数値計算に便利な書式が多数採用されているからです．例えば，複素数計算や行列計算などは文法に組み込まれているので簡単に書くことができます．

Fortran は，改版を重ねて，数値計算用としてかなり進んだ言語になっているのですが，歴史の長い言語でもあるため，古いバージョンとの互換性を保つために若干緩い文法体系になっています．例えば，デフォルトでは変数宣言をしなくてもかまわないため，変数名を書き間違えたときに発見するのが難しく，思わぬエラーを引き起こす可能性があります．これからプログラミングを始めるなら，このあたりを考慮した書き方を覚えるべきです．

そもそもコンピュータは計算する機械にすぎません．その動作は全て我々プログラマーが責任をもって指定しなければならないのです．逆に言えば，コンピュータの内部構造を理解してプログラムを書けば，効率がアップして，より高速に計算させることができます．

本稿では，コンピュータの内部構造の話を交えながら，効率が良くてエラーを見つけやすい，“スマートな”Fortran プログラムの書き方を説明します．必ずしも本稿の説明通りに書く必要はない部分もありますが，そういった部分はプログラムが長くなるにつれて御利益が現れるので，横着せずに書くことをお勧めします．

1.　パソコンでの Fortran 利用

Fortran は，計算機の歴史でいえば，その初期に開発された由緒ある言語で，“FORmula TRANslation”が語源です．当初から数値計算用言語として開発されており，筆者が計算機を利用し始めた 30 年前には，数値計算をするといえば，ほとんどが Fortran だったのです．しかし，逆にこのことが Fortran を近づきにくいものにしたようです．なぜなら，数値計算に特化するということはユーザーの専門分野が限られるということであり，その結果コンパイラの値段があまり下がらず,気軽に使うことができなかったからです.パソコンの出現とともに普及した C 言語は，誕生時点から比較的安価に供給され，これが広く使われている理由の一つです．C 言語はそもそも OS を記述するために開発された言語なので計算機のハードウェア寄りの書式が多く,必ずしも初心者向きの言語だとは思えないのですが，プログラミングの専門家からすれば Fortran も C 言語も大して変わらないので，安い方にシフトしたのはある意味やむを得ないことだと思います．

Fortran は 1991 年に策定された Fortran90 を境に大きく変貌しました[1]．配列演算などの新しい機能を導入すると同時に，開発当初の計算機環境による制約を引きずっていた書式が緩和されて現代的な書式で書けるようになりました．例えば，それまでは 1 行に 72 文字までしか書けないとか，比較記号としての“＞”や“＜”が使えなかったりしたのですが，Fortran90 から 1 行に何文字書いても良くなったし，大小記号も使えるようになりました．Fortran はその後も改版を重ね，現在の最新版は Fortran2008 なのですが，数値計算だけなら特に新しい文法は必要ないので，本稿では Fortran90 レベルの文法で説明をします．

さて，Fortran が使いやすくなったのと時期を同じくして，無料の OS である Linux や FreeBSD

が普及し始め，この OS に有志が開発した無料の Fortran（g77，gfortran など）が付属するようになりました．このため，Linux などをインストールすれば，パソコンでも Fortran が無料で使えます．このフリーの Fortran は Windows や MacOS 上で動作するものも作られていて，インターネットからダウンロードして使うことができます．数値計算プログラムを書くには Fortran の方が書きやすいし，完成したプログラムをそのまま大型計算機で高速に実行させることも可能です．

ここでは，Fortran で書いたプログラムを gfortran を使ってパソコン上で実行させる方法について紹介します．まず，Fortran プログラムを作成する時は，ファイル名の最後に“.f90”という拡張子を付けます．つまり，“文字列.f90”という名前にします．作成したプログラムファイルを計算機で実行させるには，コンピュータが直接解釈できる機械語への翻訳（コンパイル）とライブラリプログラムとの結合（リンク）という二つの過程が必要ですが，これを実行するのが gfortran コマンドです．gfortran でプログラムをコンパイルする場合，もっとも単純には，

```
$ gfortran プログラムファイル名
```

と入力します（最初の“$”はコマンドプロンプトなので入力は不要です）．例えば，test1.f90 というファイル名のプログラムをコンパイルするには，

```
$ gfortran test1.f90
```

と入力します．この時，コンパイル時に文法エラーなどが見つかるとエラーメッセージを出力して終了します．

gfortran コマンドは，コンパイルが成功すると引き続いてリンクを行い，リンクにも成功すると，“a.out”という名前のファイルを出力します．これがパソコンでプログラムを動作させることができる“実行形式ファイル”です．リンクに失敗すると，やはりエラーメッセージを出力して終了しますが，この時 a.out は作成されません．

実行形式ファイルは gfortran と同じレベルのコマンドであり，プロンプトの後に，

```
$ ./a.out
```

のようにファイルを指定することでプログラムに書かれた動作内容をコンピュータで実行させることができます．

以上が gfortran を用いたもっとも単純なプログラム実行までの流れですが，上記のような単純な命令では，どんなプログラムをコンパイルしても a.out という同じ名前の実行形式ファイルになってしまいます．また，Fortran プログラムは計算速度が重要ですから，最適化したコンパイルを行ってできるだけ高速に実行する方が良いでしょう．さらに，計算精度を考えれば倍精度実数で計算をするべきなので，自動倍精度化オプションを付加することもお勧めします．

このため，gfortran でプログラムをコンパイル・リンクする時は最低限，以下の下線部のようなオプションを付けることをお勧めします[1]．

```
$ gfortran -O -fdefault-real-8 test1.f90 -o runtest
```

ここで，“-O”（大文字のオー）が最適化のオプション，“-fdefault-real-8”が自動倍精度化のオプションです．また，“-o”（小文字のオー）のオプションの次の文字列（この例では runtest）が，実行形式ファイル名の指定です．この例では，コンパイルとリンクが正常に終了すると，“runtest”という実行形式ファイルが作成されます．よって，プログラムの実行は，

```
$ ./runtest
```

となります．

---

[1] サイバーサイエンスセンターのコンパイラ f95, sxf90 ではコンパイルオプションが異なります．（センターウェブサイト参照）

1. メインプログラムの開始と終了

プログラムとはコンピュータへの動作指示（命令）を記述した文の集まりです．実行形式コマンドを入力すると，コンピュータは，

実行開始　→　プログラムに記述された命令を順に実行　→　実行終了

というステップで動作します．実行を開始した時に最初に実行する部分を“メインプログラム”といいます．言わばプログラムの本体です．プログラムには，他のプログラムの中から実行開始を指示してその機能を利用する“サブルーチン”がありますが，これについては次回説明します．

Fortran では特別な手続きをせずにプログラムを書くとそれがメインプログラムと仮定されます．しかし，それではサブルーチンと区別しにくいので，program 文を用いて最初にプログラムの名前を書きます．

```
program プログラム名
```

メインプログラムの終了は end program 文で指定します．このため，メインプログラムは次のような構造になります．

```
program code_name
   .......
   .......
   .......
end program code_name
```

この例のようにプログラム名にはアンダースコア（ _ ）も使えるので，これをスペースの代わりに使えば単語をつないだ長いプログラム名を付けることもできます．

program 文と end program 文の間に Fortran の文法に従った文を並べて動作させせたい計算手順を記述します．動作手順を記述する文を“実行文”といいます．しかし，プログラムに記述するのは実行文だけではありません．計算途中で必要となる変数領域を確保するための宣言文も書かねばなりません．このような計算動作に直接携わらない文を“非実行文”といいます．Fortran では，非実行文をプログラムの最初に集約して，実行文はその後に書きます．このため，動作の開始は program 文の次の文ではなく，最初の実行文になります．

完結したメインプログラムの一例を示します．

```
program test_code
  implicit none
  real x,y,z
  x = 5
  y = 100
  z = x + y*100
  print *,x,y
  print *,' z =' ,z
end program test_code
```

このプログラムにおける実行文・非実行文の区分と，動作開始から終了までの実行の流れを図１に示します．実行形式のコマンドを入力すると，最初の実行文から動作を開始し，上から順に一行ずつ実行され，end program 文に到達した段階でプログラムの動作が終了します．後で述べる do 文を使った繰り返しや，if 文による条件分岐など，動作指定によっては所定の位置にバックしたり，条件に応じて実行するかどうかの選択ができますが，それでもそれぞれの文が終了した後に次の文が実行されるという基本的動作は変わりません．

これは計算機が一回に一つの動作しかできないためです．プログラム全体を見渡して実行するのではなく，一行一行を順に実行していくので，バックするような動作指定をしない限り，下方

に書いた実行文は上方に影響を及ぼしません．プログラムはこのことを常に念頭に置いて書かなければなりません．

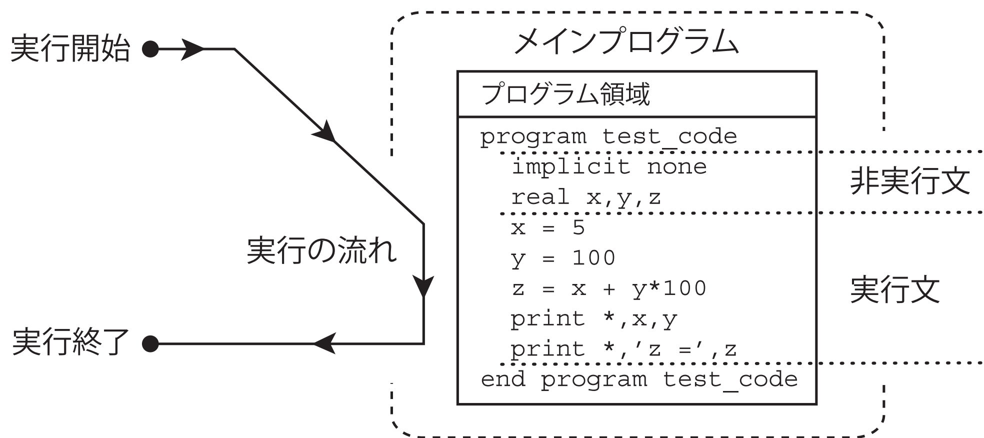


図１．プログラム開始から終了までの流れ

なお，5 節で述べる if 文を使って，条件によって途中でプログラムを終了するときや，次回お話しするサブルーチン内でプログラムを終了するときには stop 文を用います．stop 文を実行すると，その時点でプログラムが終了します．例えば，

```
program fluid_code
  implicit none
  real x,y
  integer m,n
   .......
  if (m<0) stop
   .......
end program fluid_code
```

と書けば，m＜0 の時にプログラムは終了し，それ以降は実行しません．

2. 基本的なプログラム文

2.1　代入文と演算の書式

実行文には，stop 文のように常に同じ動作を指示する文と，いくつかの“数値”を伴って，その数値に応じた動作を指示する文とがあります．プログラム中で“数値”を与える方法は３種類あります．-500 とか 3.14 のような数字を直接書く“定数”，x とか yrz のような単語で示した“変数”，およびそれらを使って x+3 や sin(y-5)のような計算手順を表した“計算式”です．計算式に書かれている手順も計算機の動作なのですが，プログラムにおいての計算式は，その計算結果を動作命令に与える“数値”として位置づけられています．このため，計算式を書いただけでは実行文になりません．

プログラムで最も良く使う実行文は“代入文”です．これは，

```
変数 = 計算式
```

という形をしています．計算式の代わりに，定数や変数を書くこともできます．プログラムにおける“変数”とはコンピュータのメモリ領域のことで，数値を入れる箱だと考えればいいでしょう．代入文は右辺の“数値”を左辺で示した変数メモリに格納する動作を表します．よって，

```
計算式 = 計算式　　　! これはエラー
```

という形は使えません．Fortran において“＝”という記号は，“左辺と右辺が等しい”という意味ではないからです．例えば，

```
x = 4 + 2
y = 9 - x
y = y + 1
```

のようなプログラムを考えてみましょう．プログラムは上から順に実行されるので，まず 4+2 の結果である 6 が変数 x に代入されます．次に，9 から変数 x に保存されている数値を引いた値が変数 y に代入されるので，y には 3 が代入されます．最後の文は数学の等式と見れば変ですが，イコールが“代入する”動作だと理解できれば特に不思議な文ではありません．y はその前の文で 3 が代入されていますから，最後の文によって変数 y に y+1 の結果である 4 が代入されます．

Fortran における基本演算の書き方と使い方を表１に示します．

表１．演算記号の書き方と使い方

演算記号	演算の意味	使用例	使用例の意味
+	足し算	x+y	$x+y$
-	引き算	x-y	$x-y$
*	掛け算	x*y	$x\times y$
/	割り算	x/y	$x\div y$
**	べき乗	x**y	x^y

マイナス記号（-）は -x のように単項演算としても使えます．これらの演算には以下のような優先順位があり，優先順位の高い方が低い方より先に計算されます．

べき乗　＞　掛け算または割り算　＞　足し算または引き算

掛け算と割り算のような同レベルの演算は左から順に計算されます．さらに，かっこを使えばかっこの中が優先的に計算されます．

2.2　数値の型

コンピュータという機械が取り扱う数値は 2 進数で表されていて，基本的には整数です．例えば，1byte の数は，8bit，つまり 0 か 1 のどちらかを選択できる数が 8 個並んだものなので，10 進数では $0\sim2^8-1$（=255）までの整数を表すことができます．Fortran では小数点のない数字，56 とか -1112 などの数字を“整数型”といいます．Fortran の整数型数は 4byte（32bit）で表現されていて，$-2^{31}\sim2^{31}-1$ の整数値を扱うことができます．

しかし，数値計算やシミュレーションをする場合には，3.14 のように小数点以下を含んだり，3×10^{19} とか，1.6×10^{-27} とかいった様々なスケールの数値を取り扱うため，整数型だけでは不便です．特に，整数型数は小数点以下の表現ができないので，割り算をすると全て切り捨てになります．これを忘れてプログラムを書くと，よく間違いを起こします．例えば，

```
x = (5/2)**2
y = x**(3/2)
t = 1/2*y*x
```

と書くと，x，y，z はいくつになると思いますか？答えは x=4，y=4，t=0 です．これは整数の割り算が切り捨てになるため，5/2→2，3/2→1，1/2→0 になるからです．

そこで，整数型の他に，3.14 のような少数点以下を含んだり，1.6×10^{-27} のような，$A\times10^B$ という形式の数値を取り扱うことができます．これを“実数型”といい，A の部分を“仮数部”，B の部分を“指数部”といいます．通常，数値計算は実数型で計算しなければなりません．

実数型には有効数字の違う 2 種類が用意されています．単精度実数と倍精度実数です．単精度実数とは 4byte で表される実数のことで，仮数部の有効数字は 7 桁程度です．これに対し，倍精度実数とは 8byte（64bit）で表される実数のことで，仮数部の有効数字は 15 桁程度です．7 桁あれば問題ないと思われるかもしれませんが，何百万個もの数字の合計を計算したり，次数の高い

多項式の計算をするときなどでは，計算回数の増加につれて有効桁数が減少することを考慮しなければなりません．このため，通常は倍精度実数で計算をします．

Fortran における基本的な実数型は単精度であり，倍精度実数型を使用する時にはそれを意識した書式で書かなければならないのですが，gfortran の説明で示したように最近のコンパイラはオプションを指定すればデフォルトの実数を倍精度にする“自動倍精度化機能”を持っているので，本稿では単精度実数と倍精度実数を使い分ける書式の説明は省略し，単に“実数型”と表現します．よって，特に単精度で計算する必要がなければ，コンパイル時に自動倍精度化オプションを付加して倍精度で計算して下さい．多くの計算機は倍精度実数計算を高速で行えるハードウェアを内蔵しているので，倍精度計算をしてもさほど実行速度は下がりません．

プログラム上で，整数型の定数と実数型の定数は小数点の有無で区別します．例えば，100 とか，-12345 と書けば整数型ですが，100.0 とか，-12345. とか，-0.0314 とか書けば実数型になります．また，1.6×10^{23} を入力したい時には，1.6e23 と書きます．すなわち，$A\times10^{B}$ は AeB と書きます．B が負の場合でも 1.6e-19 のように e の後に続けて書きます．また，6e20 のように eB を付加した場合には小数点が無くても実数型になります．例えば，

$$a = 3.141592\,r^2 + 3\,x^5 + 6.5\times10^{-5}\,x - 10^5$$

という式をプログラムで書けば以下のようになります．

```
a = 3.141592*r**2 + 3.0*x**5 + 6.5e-5*x - 1e5
```

この例のように，r**2 とか x**5 のように整数べき乗の指数(2 とか 5)には整数型を使います．

先ほど整数型を使ったら期待どおりの結果が出ない例を挙げましたが，以下のように実数型を使えば正しい結果が得られます．

```
x = (5.0/2.0)**2
y = x**(3.0/2.0)
y = 1.0/2.0*y*x
```

Fortran の便利な機能の一つは，複素数が使えることです．複素数にも単精度複素数型と倍精度複素数型がありますが，これも自動倍精度化機能を使えば同時に変更できるので，本稿では単に“複素数型”と表現します．複素数型の定数は，

```
(0.0,1.0)
(1e-5,-5.2e3)
(-3200.0,0.005)
```

のように，2 個の実数をコンマでつないで，かっこで囲みます．前半が実部，後半が虚部です．つまりこの例は，それぞれ，i，$10^{-5}-5.2\times10^{3}\ i$，$-3200+0.005\ i$ を表した複素数型定数です．

なお，ここまで読むと整数型は必要がないと思われるかもしれませんが，そうではありません．後で説明する配列要素を指定する数は整数型でなければならないし， do 文で用いるカウンタ変数やオン・オフを表すだけの変数など“順番”や“区別”を示すときは整数型を用います．また実数の整数部を取り出したいときや，剰余（あまり）を計算するときも整数型を利用します．数値計算プログラムの中でも整数型の使い道は多いのです．

計算式中に異なる数値型が混在した時は，精度の高い方の型に合わせて計算し，その型の値が結果になります．例えば，実数型と整数型の計算は整数型を実数型に変換して実数型と実数型の計算を行い，実数型の結果になります．複素数型と実数型の計算の場合にはその実数型を実部とした複素数型にして計算し，結果は複素数型になります．ただし，掛け算と割り算の計算は左から順に行うので注意が必要です．最初の整数型を使った計算例で，

```
t = 1/2*y*x
```

が 0 になるのは，最初に計算されるのが 1/2 という整数型÷整数型だからです．これを，

```
t = y*x*1/2
```

と書けば，切り捨ては起こりません．

2.3　変数の宣言

次に，数式の計算結果を保存する変数の使い方を説明します．変数の名前は，頭文字が a～z のどれかであれば，後は a～z，0～9 をどの様な順序で並べたものでも使えます．例えば，abc とか k10xy 等です．Fortran では大文字と小文字を区別しないため，abc と ABC は同じ変数になります．変数にも型があり，計算結果に応じた型の変数を使わなければ正確に保存することができません．この変数の型を決める文を“宣言文”といいます．型を決めると同時に，変数のメモリ領域を確保します．このため，計算に用いる変数は全て宣言しなければなりません．宣言は一度しかできず，プログラムの実行時に変更することはできません．宣言文は非実行文です．

ところが Fortran には“暗黙の宣言”があり，通常は宣言しなくても文法的な間違いにならないので，タイプミスなどで思わぬエラーが発生する可能性があります．これを防ぐため，プログラムの２行目，すなわち program 文の次の行に必ず以下の implicit 文を書いておきます．

```
implicit none
```

この文があれば，宣言せずに使用した変数があるとコンパイルエラーになり，タイプミスのチェックができます．

数値計算は基本的に実数で行いますが，実数型変数は次のように宣言します．

```
real 変数名, 変数名, ...
```

これに対し，整数型の変数は，次のように宣言します．

```
integer 変数名, 変数名, ...
```

宣言文は非実行文なので，全ての実行文より前に書かなければなりません．

```
program test1
  implicit none
  real x,y,z,omega,wave,area
  integer i,k,n,imin,imax,kmax
   .......
```

real や integer 等の型宣言文は何行書いても良いし，順番も無関係です．

Fortran の暗黙宣言では，変数名の頭文字が a～h と o～z の変数は実数型で，i～n の変数は整数型でした．このため，整数型変数の頭文字を i～n にする習慣があります．全てを宣言する以上，基本的に変数名の付け方に制限はないのですが，整数型は用途が限られているので，頭文字を限定する方が良いと思います．実数型数の名前はあまり頭文字にこだわる必要はありませんが，原則として整数型をイメージする i,j,k,l,m,n の 1 文字変数は使わない方が無難です．

複素数型変数の宣言文は，

```
complex 変数名, 変数名, ...
```

です．複素数型も用途が限定されているので名前の付け方に規則をつける方が良いでしょう．複素数型変数名は，頭文字を c か z にすることが多いようです．

プログラムにおいて，数値を変数に保存する意義は大別して二つあります．一つはプログラム全域にわたって共通してその値を利用するためであり，もう一つは狭い範囲で計算結果を一時的に保存するためです．この二つは意識して使い分けるようにします．特に，前者の大域的に利用する変数には長めで意味のある名前を付けるべきです．これは，１文字のような短い変数名を使うとプログラムが長くなるにつれてどこでその変数を使っているかを検索するのが難しくなるからです．

変数に数値を代入する時は，右辺の計算結果を左辺の変数の型に変換して代入します．このた

め，実数の計算結果を整数型の変数に代入すると小数点以下は切り捨てられます．例えば，

```
real x
integer n
x = 3.14
n = x + 6
```

の結果，n に代入される値は 9 です．

また，複素数型の計算結果を実数型の変数に代入すると，その複素数の実部が代入されます．例えば，

```
real x
complex c
c=(1.0,-2.0)
x=c**2
```

とすると，x=-3.00000 になります．逆に，複素数型の変数に実数型の数値を代入すると，実部に結果が代入され，虚部は 0 になります．例えば，

```
real x
complex c
x=5
c=x**2
```

とすると，c=(25.00000,0.00000)になります．

2.4　数学関数

数値計算上よく使う関数はあらかじめ用意されています．これらを“組み込み関数”といいます．代表的な数学関数を表２に示します．組み込み関数は型宣言をする必要がありません．また，引数の型に応じた精度で計算をする“総称名”機能があるので，引数 x が複素数型でも使えます．表２の必要条件と関数値の範囲は引数が実数型の場合です．この必要条件を満たさない実数型数を与えるとエラーになります．しかし複素数型を与えた場合には必ずしもエラーになりません．

表２．数学関数の書き方と使い方

組み込み関数	名　称	数学的表現	必要条件	関数値 f の範囲
sqrt(x)	平方根	$\sqrt{x}$	x≧0	
abs(x)	絶対値	$\lvert x\rvert$		
sin(x)	正弦関数	$\sin x$		
cos(x)	余弦関数	$\cos x$		
tan(x)	正接関数	$\tan x$		
asin(x)	逆正弦関数	$\sin^{-1} x$	-1≦x≦1	$-\pi/2 \leqq f \leqq \pi/2$
acos(x)	逆余弦関数	$\cos^{-1} x$	-1≦x≦1	$0 \leqq f \leqq \pi$
atan(x)	逆正接関数	$\tan^{-1} x$		$-\pi/2 \leqq f \leqq \pi/2$
atan2(y,x)	逆正接関数	$\tan^{-1}(y/x)$		$-\pi < f \leqq \pi$
exp(x)	指数関数	e^x		
log(x)	自然対数	$\log_e x$	x＞0	
log10(x)	常用対数	$\log_{10} x$	x＞0	

関数の引数には計算式を与えることも可能です．この時は，計算式の結果を引数とした関数値になります．例えば，

```
c = sin(10*x+3) - 2*tan(-2*log(x))**3
```

のように書くことができます.

なお，$\sqrt{2}$ を計算したくて，sqrt(2)と書くとコンパイルエラーになります．なぜなら，“2”は整数型の定数であり，整数型数の平方根は用意されていないからです．必ず sqrt(2.0)のように実数型を書かなければなりません.

2.5　print 文による簡易出力

数値計算を目的としたプログラムでは，得られた計算結果を適宜表示しなければなりません．入出力の詳細については次回説明しますが，最低限，標準フォーマットの print 文を使った数値出力は覚えておく必要があります．print 文の一例を以下に示します.

```
integer n
n = 3
print *, 4+5, n, n*2, 2*n-11
```

このように，“print *,”に続いて変数や計算式をコンマで区切って連ねると，それらの計算結果が横に並んで出力されます．上例の場合には，4+5 の結果である 9 から 2*n-11 の結果である-5 までが以下のように出力されます.

```
          9           3           6          -5
```

なお，“print *,”の“*”は，出力フォーマットが標準形式であることを意味しているのですが，とりあえずは形式的に書くものだと覚えて下さい.

複数の数値を出力するときに数字だけを出力すると，どれがどの数値か分からなくなる可能性があります．このような場合は，文字列を併用して変数の意味を同時に出力します．Fortran における“文字列”とは，２個のアポストロフィ「'」ではさんだ文字の並びのことで，print 文中で使えば，その文字の並びがそのまま出力されます．例えば,

```
real x
x = 3
print *,'x = ',x,'  x**3 = ',x**3
```

というプログラムの出力は,

```
x =    3.00000000000000         x**3 =    27.0000000000000
```

となります.

3.　配列

3.1　配列宣言

ここまで出てきた変数は型に応じた 1 個の数値を記憶することしかできませんでした．しかし，数値計算やシミュレーションでは，数万個のデータを保存してそれぞれを条件に応じて変化させていく，なんていうのが当たり前のように行われます．そこで，数学で $a_1, a_2, \dots, a_n$ のように変数に添字を付けて区別するように，数字で区別した変数を作ることができます．これを“配列”といいます.

配列は変数の一種なので，型宣言文を使って宣言しておかねばなりません．単一変数と異なるのは，宣言時に添字の範囲を示す整数値を付加することです．例えば，次のように宣言します.

```
real a(10),b(20,30)
complex cint(10,10)
integer node(100)
```

ここで，数字が 1 個の配列を 1 次元配列，2 個の配列を 2 次元配列といいます．3 次元以上の配

列を作ることもできます．配列の名前の付け方，頭文字の選択などの原則は単一変数名の付け方と同じにします．

宣言した配列の各部の名称は次の通りです．例えば，real a(10)と宣言した１次元配列について，a は配列全体を表す“配列名”，a(3)と書くとそれぞれのメモリを示す“配列要素”，かっこ内の数字(3)は“要素番号”とか“添字”です．

配列宣言で指定した数値は要素番号の最大値を表し，要素番号の使用可能範囲は１から指定した最大値までです．例えば a(10)の宣言では a(1)から a(10)までの 10 個の配列要素が使用可能になります．また，２次元以上の配列の場合には各次元ごとの最大値を指定しているので，b(20,30)の宣言では全部で 20×30＝600 個の配列要素が使用可能になります．

問題によっては，要素番号として０や負数を使いたい時があります．このような時には“:”を間に入れて，使用可能な要素番号の最小値と最大値を同時に指定します．例えば，

```
real ac(-3:5),bc(-20:20,0:100)
```

と宣言すると，１次元配列 ac は，ac(-3)から ac(5)までの９個が使用可能であり，２次元配列 bc は，bc(-20,0)から bc(20,100)までの(20×2+1)×(100+1)＝4141 個が使用可能です．

3.2　メモリ上での配列の並び

配列はコンピュータ内部において連続したメモリ領域で実現されています．例えば，real a(10)と宣言された配列は，図２左のように，a(1),a(2)...a(10)の順で並んだ実数型のメモリです．

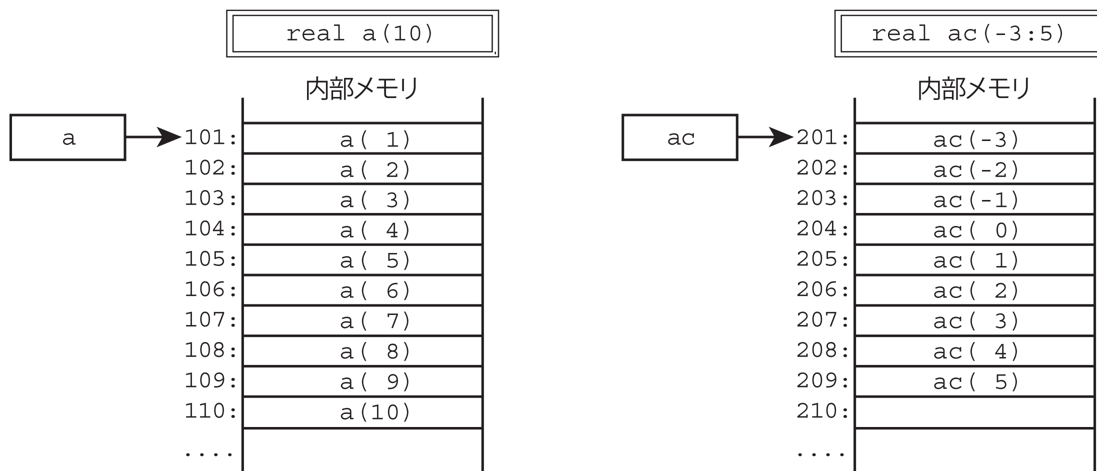


図２．１次元配列のメモリ並び

図から分かるように，a(3)とは a(1)から数えて３番目のメモリのことです．また，a(1000)のように範囲外の要素番号を指定すると予期せぬエラーを引き起こすこともわかります．a(10)の宣言は 10 個しかメモリを確保していないのですから，1000 番目の要素 a(1000)がどのメモリを示すのか不明だし，そもそも存在するという保証さえありません．

Fortran での配列名は配列を代表する名称であると同時に，配列の先頭メモリを示します．たとえば，配列名 a は図２左のように a(1)を示します．また，ac(-3:5)のように最小値も指定して宣言した場合には，図２右のように並んでいて，配列名 ac は ac(-3)を示します．配列名が先頭要素を示すことは配列をサブルーチンの引数として与える時に意識する必要があります．

２次元以上の配列の場合は，左の方の要素番号から先に進むようにメモリ上で並んでいます．例えば，

```
real b(3,2)
```

と宣言した場合，メモリ上での並びは図３のようになります．２次元以上の配列も配列名は先頭要素を示しています．

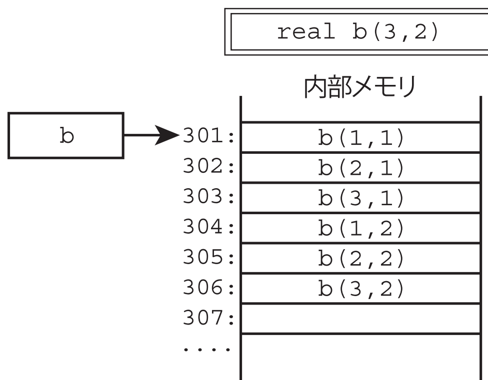


図３．２次元配列のメモリ並び

２次元と３次元の配列の並び方を式で表せば，

```
real b(m,n)の配列宣言で b(i,j)は先頭から数えて i+m*(j-1)番目
real b(ℓ,m,n)の配列宣言で b(i,j,k)は先頭から数えて i+ℓ*(j-1+m*(k-1))番目
```

となります．どちらの公式も一番右側の要素数 n に依存していません．一般的に，どんな次元の配列の公式も一番右側の要素数には依存しないのです．

4．手順の繰り返し ― do 文

実行文は基本的に上から下へ順に実行されますが，それだけでは類似した手順を繰り返す時に，必要な回数だけ同じ文を書かねばなりません．そこで，ある範囲の手順を必要な回数だけ繰り返し行わせる手段として do 文があります．do 文を使う時の基本形は，

```
do 整数型変数 = 初期値，終了値
  .......
  .......
enddo
```

です．最初の do の行が do 文で，do 文と最後の enddo 文で範囲を指定された一連の実行文が繰り返し実行されます．この範囲を“do ブロック”といいます．また，プログラムの流れが循環するという意味で“do ループ”ともいいます．do 文の整数型変数を“カウンタ変数”と呼びます．do ブロックの繰り返しは，カウンタ変数に“初期値”を代入することから開始し，do ブロック内の手順を一回実行するごとにカウンタ変数を 1 増加します．そして，カウンタ変数が“終了値”より大きくなった時点で繰り返しは終了し，enddo 文の次の文に実行が移ります．例えば，

```
do m = 1, 10
  a(m) = m
enddo
```

と書けば，a(1)～a(10)までの配列要素に，1～10 までの数値がそれぞれ代入されます．do 文における，“カウンタ変数＞終了値”の判定は do ブロックの開始時にも行うため，初期値が終了値より大きい場合にはブロック内部が一度も実行されずに enddo 文の次に実行が移ります．

次の do 文のように，整数値をもう一つ追加することで増加量を指定することもできます．

```
do 整数型変数 = 初期値，終了値，増分値
  .......
  .......
enddo
```

このときカウンタ変数は初期値から開始して，“増分値”ずつ増加しながら do ブロック内の手順

を繰り返し，“カウンタ変数＞終了値”の時点で終了します．
　例えば，m が 10 以下の奇数の時のみ計算をしたい時は，

```
do m = 1, 10, 2
  .......
  .......
enddo
```

と書きます．この時の終了値は 10 ですが，10 は奇数ではないので計算しません．
　増分値は負数を指定することもできます．負数の時にはカウンタ変数が減少していくので，“カウンタ変数＜終了値”になった時点で終了します．例えば，100 から順に下って 1 まで繰り返す時には以下のように書きます．

```
do m = 100, 1, -1
  .......
  .......
enddo
```

増分値が負数の時には，“初期値＜終了値”ならば do ブロック内部は一度も実行されません．
　do 文のカウンタ変数を増加するタイミングは，enddo 文の実行時です．例えば，

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

という文は，

```
m = 1
 ［m＞3 の判定をする．満足しないので，do ブロックを実行］
a(m) = m**2
m = m + 1
 ［m＞3 の判定をする．満足しないので，do ブロックを実行］
a(m) = m**2
m = m + 1
 ［m＞3 の判定をする．満足しないので，do ブロックを実行］
a(m) = m**2
m = m + 1
 ［m＞3 の判定をする．満足するので，do ブロックを終了］
```

という動作になります．この展開からわかるように，do ブロックを終了した時点で，カウンタ変数には終了値より大きい値が代入されています．上例では，do ブロック終了後，m＝4 です．do ブロック終了時の m の値を利用する時は，このことを考慮しなければなりません．
　次のように，do ブロックの中に別の do ブロックを入れることもできます．ただし，カウンタ変数は異なるものを使わなければなりません．

```
do k = 1, 100
  a(k) = k**2
  do m = 1, 10
    b(m,k) = m*a(k)**3
    c(m,k) = b(m,k) + m*k
  enddo
  d(k) = a(k) + c(10,k)
enddo
```

よく使うので覚えておくと便利なのが，合計を計算する do 文です．例えば，n 要素の一次元配列，a(1)，a(2)，....，a(n)の中に数値データが入っている場合，これらの合計を求めるには do 文を使って以下のように書きます．

```
sum = 0
do m = 1, n
  sum = sum + a(m)
enddo
```

この文が合計を計算していることを理解するには，やはり以下のように手動展開してみると良いでしょう．

```
sum = 0
m = 1
sum = sum + a(m)
m = m + 1
sum = sum + a(m)
m = m + 1
sum = sum + a(m)
m = m + 1
   ......
```

ここで判定文は省略しました．このプログラムで重要なことは，do ブロックの前で変数 sum に 0 を代入していることです．これがないと正しい結果が得られないことがあります．このようなプログラムならではの書き方はパターンとして覚えておくと良いと思います．

5. 条件分岐 ― if 文

条件に応じて異なる手順を行わせることを“条件分岐”といい，if 文を使って指定します．もっとも単純に，一つの条件に応じて一つの文を実行するかしないかを決めるだけの時は単純 if 文を使います．単純 if 文は以下の形式です．

```
if（条件）実行文
```

単純 if 文はかっこ内の“条件”が満足されれば，その右の“実行文”を実行し，条件が満足されなければ何もしないで次の文を実行します．例えば，

```
a = 5
if (i < 0) a = 10
b = a**2
```

と書くと，i<0 の場合には a=10 となり，それ以外は a=5 のままなので，それに応じて b に代入される値が異なります．

しかし単純 if 文には実行文が一つしか書けないので,実行したい文が複数あるときには使えません．また，条件に合った時の動作指定しかできないので，合わなかった時の動作を別に指定したい時には不便です．

そこで，通常は単純 if 文ではなく，ブロック if 文を使います．ブロック if 文とは，if 文の“実行文”のところを then にした文のことで，以下のようにブロック if 文と endif 文で一連の実行文の範囲を指定します．

```
if（条件）then
  .......
  .......
endif
```

この指定された範囲を if ブロックといい，ブロック if 文に書かれた条件を満足した時のみ，if ブロック内の実行文が実行されます．例えば，

```
a = 5
b = 2
if (i < 0) then
   a = 10
   b = 6
endif
c = a*b
```

と書くと，i＜0 の場合には a=10，b=6 の代入文を実行してから c=a*b を計算しますが，それ以外，すなわち i≧0 の場合には if ブロック内を実行しないので，a も b も変化せず，a=5，b=2 のままで c=a*b を計算します．

さて，この例では，i＜0 という条件を満足しないときには，あらかじめ a=5，b=2 という代入をする必要がありません．このように“条件を満足しないとき”に別の動作をさせたいときには if ブロック内に else 文を挿入します．else 文を挿入すると，ブロック if 文で指定した条件を満足しない場合に，else 文から endif 文までの実行文が実行されます．例えば上記のプログラムは，

```
if (i < 0) then
   a = 10
   b = 6
 else
   a = 5
   b = 2
endif
c = a*b
```

と書くことができます．この場合，i＜0 の場合には a=10，b=6 を実行し，“それ以外の場合”には a=5，b=2 を実行します．

また，条件を満足しない場合に，さらに別の条件を指定したいときには else if 文を使います．else if 文も条件はかっこで指定し，その後に then を書きます．

```
if (i < 0) then
   a = 10
   b = 6
else if (i < 5) then
   a = 4
   b = 7
else
   a = 5
   b = 2
endif
c = a*b
```

この場合，i＜0 の場合には a=10，b=6 を実行，0≦i＜5 の場合には a=4，b=7 を実行し，それ以外（i≧5）の場合は a=5，b=2 を実行，となります．else if 文による新たな条件は if ブロック内にいくつ入れてもかまいません．その場合は，“その else if 文より以前の条件を全て満足しない場合に，その条件を満足すれば”という意味になります．これに対し，else 文は最後の一回しか使えません．

数値計算でよく使う代表的な比較条件の書き方を表 3 に示します．

表３．比較条件の書き方

比較条件記号	記号の意味	使用例
==	左辺と右辺が等しいとき	x == 10
/=	左辺と右辺が等しくないとき	x+10 /= y-5
>	左辺が右辺より大きいとき	2*x > 1000
>=	左辺が右辺以上のとき	3*x+1 >= a(10)**2
<	左辺が右辺より小さいとき	sin(x+10) < 0.5
<=	左辺が右辺以下のとき	tan(x)+5 <= log(y)

使用例のように，比較条件を指定する場合には両辺どちらにも計算式を書くことが可能です．

さらに表４の論理演算記号を使えば，これらの条件を論理的につないだ条件や，否定した条件を与えることもできます．

表４．論理式の書き方

論理演算記号	演算の意味	使用例
“条件１”.and.“条件２”	“条件１”かつ“条件２”のとき	x > 10 .and. 2*x <= 50
“条件１”.or.“条件２”	“条件１”または“条件２”のとき	x > 0 .or. y > 0
.not.“条件”	“条件”を満足しないとき	.not.(x < 0 .and. y > 0)

例えば，

```
if (i > 0 .and. i <= 5) then
   a = 10.0
 else
   a = 0.0
endif
```

と書くと，i が 0 より大きく“かつ”5 以下の時，すなわち，$0 < i \leqq 5$ のときに a=10，それ以外は a=0 となります．なお，横着してこの if 文を，

```
if (0 < i <= 5) then        ! これはエラー
```

と書くことはできないので注意しましょう．

6．無条件ジャンプ ― goto 文，exit 文，cycle 文

プログラムというのは基本的に上から順番に実行していくものです．do 文を使えば，do ブロックで指定した範囲の実行文を所定の回数繰り返すことができますが，繰り返す範囲は固定されているし，繰り返しを終了する条件はカウンタ変数の増加で決まります．

これに対し，より一般的にプログラムの流れを変えたい時，例えば途中で計算を中断してプログラムの最初からやり直す，とか，最後の文に一気に移動して終了する，とかいう時には goto 文を使います．goto 文を使えば指定した行へ強制的に移動することができます．計算機的には，これを“ジャンプする”といいます．goto 文とは以下のような，goto の後に整数値を指定した形の文です．

```
goto 整数値
```

この goto の後の整数値がどの行にジャンプするかを指定するための数値で，“文番号”と呼ばれています．文番号はジャンプ先の行に書かれた実行文の前に，スペースを１個以上空けて書きます．例えば，

```
      cd = 10
      goto 11
      cd = 50
      ab = 20
      ij = 1
   11  ab = 1000
```

と書きます．最後の行で ab=1000 の前の 11 が文番号です．この例では，最初の cd=10 の実行後，cd=50 から ij=1 までの文は実行されず，直ちに ab=1000 が実行されます．すなわち，cd=50, ab=20, ij=1 の文は書いていないのと等価です．このように有無をいわさずジャンプすることを“無条件ジャンプ”といいます．

goto 文でバックすることも可能です．この場合，指定した文番号の行と goto 文の間の動作を繰り返し実行します．

```
      cd = 50
   22  ab = 200
      cd = cd + ab - ef
      ef = 10
      goto 22
      ij = 1
```

もっとも，この例ではいつまでたっても goto 文の次の ij=1 は実行されません．こういうのは“無限ループ”と呼ばれ，プログラムエラーの一つです．計算結果に応じて条件分岐し，goto 文より下の行へジャンプする別の goto 文や，プログラム自体を終了させる stop 文を挿入しなければ，プログラムは永遠に終了しません．

文番号は行の指定に使うだけなので，重複さえしなければどの行にどんな数値を付けても良いのですが，なるべく下に行くほど大きくなるように数値を選びます．また，文番号を特定の実行文に付けると，変更する時に付け替えが面倒だと思う時には continue 文を挿入します．continue 文に動作は無く，文番号で位置を指定するためだけに用います．例えば，上記のプログラムは，

```
      cd = 50
   22  continue
      ab = 200
      cd = cd + ab - ef
      ef = 10
      goto 22
      ij = 1
```

と等価です．

goto 文は，多用するとプログラムの流れがわかりにくくなるし，無限ループに陥る可能性もあるので使用はできるだけ避けるべきです．基本的な繰り返しや条件に応じたジャンプは do ブロックと if ブロックでほとんどすべて書くことができます．

do ブロックを使って繰り返し計算をする時，条件によって途中で繰り返しを終了したい場合があります．この場合，原理的には goto 文を使わなければなりませんが，goto 文の使用を極力避けるという方針から exit 文が用意されています．例えば，

```
      do m = 1, n
        sum = sum + a(m)
        if (sum > 100) exit
      enddo
      sum = sum/n
```

のように書いたプログラムは，次の goto 文を使ったプログラムと同じです．

```
      do m = 1, n
        sum = sum + a(m)
        if (sum > 100) goto 10
      enddo
   10 sum = sum/n
```

すなわち，exit 文を実行すると do ブロックの外に飛び出して enddo 文の直後から実行を開始します．なお，do ブロックの中から外へのジャンプはできますが，外から中に入るジャンプは禁止されています．

ただし，上記のプログラムを変更して，以下のように n ではなく，do ブロックを終了した時点の m で平均を計算する時には注意が必要です．

```
      do m = 1, n
        sum = sum + a(m)
        if (sum > 100) exit
      enddo
      sum = sum/m
```

なぜなら，m＝n で do ブロックを実行中に，sum＞100 の条件を満足して do ブロックの外に出た時は m＝n のままですが，条件を一回も満足せずに do ブロックを終了した時には m＝n+1 になるからです．後者の場合を考慮したプログラムにするには，do ブロックを終了した時点で m>n かどうかを判定し，必要に応じて m を修正しなければなりません．

do ブロックの途中で実行を中断し，残りの部分をスキップしてカウンタ変数を進める時には cycle 文を使います．例えば，

```
      do m = 1, n
        sum = sum + a(m)
        if (sum > 100) cycle
        sum = sum*2
      enddo
```

のように書いたプログラムは，次の goto 文を使ったものと同じです．

```
      do m = 1, n
        sum = sum + a(m)
        if (sum > 100) goto 10
        sum = sum*2
   10 enddo
```

すなわち，cycle 文を実行するのは enddo 文にジャンプするのと等価です．enddo 文にジャンプすれば，カウンタ変数 m を増加して，m>n かどうかをチェックした後，条件を満足しなければ再び do ブロックを最初から実行することになります．ただし，cycle 文は do ブロック内部の制御なので次のように if ブロックを使って同じ動作をさせることができます．

```
      do m = 1, n
        sum = sum + a(m)
        if (sum <= 100) then
          sum = sum*2
        endif
      enddo
```

この方が，条件に応じた動作を明示している点で良いでしょう．goto 文を多用しない方がよい，という意味合いと同じで，cycle 文を使うとプログラムがわかりにくくなるのであまり使わない

方が良いと思います．

なお，do ブロックによる繰り返しの終了を，ブロック内部の条件分岐だけで行う場合にはカウンタ変数を書かない do 文を使うことができます．例えば，

```
do
  sum = sum + x**2
  if (sum > 100) exit
  x = 1.2*x + 0.5
enddo
```

と書くと，sum が 100 を超えるまで計算を続けます．カウンタ変数なしの do ブロックは無限ループになるわけです．無限ループですから，何らかの条件分岐により外に出る記述がないと永遠に止まらないので注意して下さい．

7. プログラムのチューンアップ

計算過程が複雑になったり大量のデータを処理するようになると，数式を単純にプログラムに置きかえるのではなく，計算機の特性を考慮したプログラムにすることで計算効率や精度を高めることができます．ここでは，このような“スマートな書き方”をいくつか紹介します．

7.1 演算の速度を考える

コンピュータの計算動作は a+b のような加算が基本です．a-b のような減算は b を負数に変換して加算するだけなので加算とそれほど実行時間は変わりませんが，a*b のような乗算は加算の繰り返し動作ですから，加減算よりかなり遅いです．a/b のような除算にいたっては，減算の繰り返しを条件付きで行うので，さらに時間がかかります．この演算速度の比較を書けば次のようになります．

加減算　≫　乗算　⋙　除算

このため，割り算ができるだけ少なくなるような計算手順にします．例えば，

```
x = a/b/c
```

と書くより，

```
x = a/(b*c)
```

と書く方が速くなります．最近の計算機はかなり高速に計算することができるので，数回の計算だけならこのような書き換えはあまり意味がありませんが，do ブロック内で大量に処理をする時には価値があります．

また，do ブロック内で何度も同じ割り算をするときには，逆数を掛けるように書き換えると速くなります．例えば，

```
do i = 1, 10000
  a(i) = b(i)/c
  x(i) = a(i)/10.0
enddo
```

と書くより，

```
d = 1.0/c
do i = 1, 10000
  a(i) = b(i)*d
  x(i) = a(i)*0.1
enddo
```

と書く方が速くなります．もっとも，プログラムがわかりにくくなるという欠点もあるので，割り算の箇所が少なく，繰り返し回数がさほど多くない時にはそれほど神経質に変形する必要はないでしょう．

べき乗算はもっと遅いので，2 乗や 3 乗程度のときは掛け算にする方が速くなります．例えば，

```
x = a**2 + b**3
```

と書くより，

```
x = a*a + b*b*b
```

と書く方が速くなります．ただし，指数が大きいときには意味がないし，プログラムもわかりにくくなるので 3 乗程度まで で良いと思います．

べき乗の中で，1/2 乗に関しては，組み込み関数 sqrt を利用する方が速いです．例えば，

```
y = x**0.5
z = y**1.5
```

と書くより，

```
y = sqrt(x)
z = y*sqrt(y)
```

と書く方が速くなります．

7.2　多項式を計算する手法

多項式を計算するときは，掛け算の回数ができるだけ少なくなるように考えます．一般的に，一番良い計算手法は horner 法です．例えば，

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

を，乗算を明示してプログラムにすると，

```
y = a0 + a1*x + a2*x*x + a3*x*x*x + a4*x*x*x*x
```

のようになり，10 回の乗算が必要です．ところが，これを変形して，

```
y = a0 + (a1 + (a2 + (a3 + a4*x)*x)*x)*x
```

と書けば，4 回の乗算で計算できます．これが horner 法です．一般的に，horner 法は 0 の係数が少ない多項式の計算に有効です．もし n 次の多項式，

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

において，係数 $a_0, a_1, a_2, ..., a_n$ が，a(0), a(1), a(2), . . . , a(n) という配列に入っている場合には，次のような do 文を使って計算することができます．

```
y = a(n)
do i = n-1, 0, -1
  y = a(i) + x*y
enddo
```

7.3　同じ計算は繰り返さない

do ブロック内部で同じ計算を繰り返す時には，あらかじめ計算をしておきます．例えば，

```
do i = 1, 10000
  a(i) = b(i)*c*f
  b(i) = sin(x)*a(i)
enddo
```

と書くと，繰り返しごとに c*f や sin(x)を計算することになりますが，これらは do ブロック内部では変化しないのですから，あらかじめ計算しておくと速くなります．例えば,

```
cf = c*f
sx = sin(x)
do i = 1, 10000
  a(i) = b(i)*cf
  b(i) = sx*a(i)
enddo
```

のように書き換えると速くなります．

7.4　do 文を多重にするときの順序

配列を使った繰り返し計算をするときは，メモリをできるだけ連続的に読み書きする方が速くなります．このため，２次元以上の配列計算をするときには左の要素から順に進めるようにします．例えば,

```
real b(10,100)
integer m,n
do n = 1, 100
  do m = 1, 10
    b(m,n) = m*n
  enddo
enddo
```

のように，２次元配列 b(m,n)に計算結果を代入するときは，左側の要素 m に関する do ブロックを右側の要素 n の do ブロックの内側に持ってくる方が高速です．これは，内側の do ブロックのカウンタ変数が先に進むので，b(1,1),b(2,1),b(3,1)…のように，メモリの並んでいる順に格納していくからです．これを,

```
real b(10,100)
integer m,n
do m = 1, 10
  do n = 1, 100
    b(m,n) = m*n
  enddo
enddo
```

のように書くと,b(1,1),b(1,2),b(1,3)…のように飛び飛びに格納していくので効率が悪くなり,スピードダウンします.

7.5　桁落ちに気を付ける

実数型数の有効数字は倍精度でも 15 桁程度です．よって，値の接近した 2 個の実数の引き算をするときは気を付けなければなりません．例えば,

2000.06 - 2000.00 = 0.06

ですが，計算結果 0.06 は元の 2000.06 と比べると有効数字が 5 桁も少なくなっています．これを“桁落ち”といいます．桁落ちするような引き算はできるだけ避けねばなりません．

例えば，２次方程式 $ax^2 + bx + c = 0$ の１根は,

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$$

ですが，$b>0$ かつ $b^2 \gg |4ac|$ の時には，b と $\sqrt{b^2-4ac}$ がほぼ等しくなるので，$-b+\sqrt{b^2-4ac}$ の

計算をすると桁落ちする可能性があります．そこでこの根を計算する時は，分子の有理化をします．すなわち，分母分子に $-b-\sqrt{b^2-4ac}$ を掛けると，

$$x_1=\frac{\left(-b+\sqrt{b^2-4ac}\right)\left(-b-\sqrt{b^2-4ac}\right)}{2a\left(-b-\sqrt{b^2-4ac}\right)}=\frac{2c}{-b-\sqrt{b^2-4ac}}$$

となりますが，最後の公式を使えば桁落ちする心配がありません．2 根とも計算する時には，2 根の積が c/a であることを利用して，まず桁落ちしない方の根 x_1 を計算した後で，もう一方の根を $c/(ax_1)$ で計算すると良いでしょう．

大きな数に小さい数を加えると，小さい数が桁落ちして情報が失われる可能性があります．例えば，１次元配列 a(i)の合計 s を計算するプログラムは，

```
s = 0
do i = 1, n
  s = s + a(i)
enddo
```

で良いのですが，a(i)が全て正数で，n が非常に大きい場合には，合計 s がだんだん大きくなって，後の方で加えた a(i)の情報が失われる可能性があります．倍精度実数を使うことを推奨している理由の一つがこれです．この桁落ちを防ぐには，例えば，2 個ずつ加えてそれらを別の配列に入れ，その後それらを同様に 2 個ずつ加えていって，... とするのが良いのですが，プログラムが複雑になってしまいます．

ひとつの比較的簡単な解決法は，補助変数 r を使って，誤差を別に評価しておく方法です[2]．

```
s = 0
r = 0
do i = 1, n
  r = r + a(i)
  t = s
  s = s + r
  t = s - t
  r = r - t
enddo
```

この方法では，桁落ち分を変数 r に保存して別途加えるので，計算精度が上がります．倍精度計算をしていれば必ずしもこの方法にする必要はありませんが，精度の良い合計計算が必要になった時のために覚えておくと良い方法です．

桁落ちしそうな引き算は可能な限り避けるようにします．例えば，小さい正数の dt(i)という１次元配列が与えられていて，これから，

```
t(1) = 0.0
do i = 2, n
  t(i) = t(i-1) + dt(i)
enddo
```

により t(i)という１次元配列を作ったとします. この t(i)を使って t(i+1)-t(i)や t(i+1)-t(i-1)の値が必要な時には, dt(i)も保存しておいて dt(i)や dt(i)+dt(i-1)を使って評価するべきです.

8. 読みやすいプログラムを書くには

ここではプログラムを読みやすくするためのヒントをいくつか紹介します．プログラムは，１文字書き間違えただけでも結果が全く異なることがあり，書くことよりも誤りがないことをチェックする方が大変です．“読みやすいプログラム”というのは，チェックがしやすいプログラムのことです．読みやすく書くように心がけていれば，書き間違いをしても気がつきやすいし，実行時にエラーが出ても早期に間違っている箇所を発見することができます．以下のヒントはプログラミングの初心者にはその重要性がわからず，“細かいことだから無視してもいいや”と感じるかもしれません．しかし，読みやすいプログラムにすることは，プログラムが長くなるほど意義が出ますし，慣れればそれほどの手間ではありません．面倒がらずに心がけましょう．

8.1　コメント文を多用しよう

Fortran では“!”の後に続く文字列は全て無視されます．すなわち何を書いても実行とは無関係です．これをコメント文といいます．コメント文を機会あるごとにプログラム中に入れて，書かれている内容を表示するように心がけましょう．さもなくば，プログラムが長くなるにつれて，自分でも何を書いたのか忘れてしまいます．例えば，

```
!  area of circles
   s = pi*r*r
```

のように，１列目に ! を書けば，その行はコメント行となります．また，

```
   s = pi*r*r          !  円の面積
   v = 3*pi*r*r*r/4    !  球の体積
```

のように，実行文の末尾に書き込むこともできるし，日本語を書くこともできます．ただし，日本語環境によっては正しく認識できずに文字化けすることがあるので，可般性を考慮するならローマ字つづりでも良いから半角英数字だけでコメントを書いた方が無難です．

8.2　ブロックを明確にするために字下げする

今までの例でも示していますが，do や if などのブロック中では字下げ（インデント）をしましょう．これは，ブロックの範囲が一目でわかるからです．

例えば，if ブロック，

```
   if (n < 0) then
      x = 10.0
    else if (n < 10) then
      x = 1.0
    else
      x = 0.0
   endif
```

や do ブロック，

```
   do i = 1, n
     b(i) = a(i)
     c(i) = a(i)*sin(b(x))
   enddo
```

のように，ブロック内部の文をスペースを入れてずらせておくとブロックの範囲が一目で判断できます．通常，2～4 個のスペース程度で良いと思います．

最近のエディタには，オートインデントといって，改行するとその前の行と先頭がそろうようにカーソルが移動する機能があるので，インデントをするのはそれほどの手間ではありません．

8.3　文字間や行間は適当に空ける

文中の文字間は適当に空けたほうが読みやすくなります．筆者は“=”と“+”の両側に必ずスペースを入れることにしています．また，代入文が何行も続くときには“=”を上下にそろえると読みやすくなります．特に，同じパターンで少しずつ内容が違う文を並べるときには，パターンが並ぶようにスペースを入れることをお勧めします．以下は，筆者のシミュレーションプログラムの１部です．

```
wm( 1,m1) = (1-dx)*(1-dy)*(1-dz)
wm( 2,m1) =    dx *(1-dy)*(1-dz)
wm( 3,m1) = (1-dx)*   dy *(1-dz)
wm( 4,m1) =    dx *   dy *(1-dz)
wm(11,m1) = (1-dx)*(1-dy)*   dz
wm(12,m1) =    dx *(1-dy)*   dz
wm(13,m1) = (1-dx)*   dy *   dz
wm(14,m1) =    dx *   dy *   dz
```

これをスペース抜きで書くと，以下のようになります．

```
wm(1,m1)=(1-dx)*(1-dy)*(1-dz)
wm(2,m1)=dx*(1-dy)*(1-dz)
wm(3,m1)=(1-dx)*dy*(1-dz)
wm(4,m1)=dx*dy*(1-dz)
wm(11,m1)=(1-dx)*(1-dy)*dz
wm(12,m1)=dx*(1-dy)*dz
wm(13,m1)=(1-dx)*dy*dz
wm(14,m1)=dx*dy*dz
```

この二つは全く同じことが書いてあるのですから全く同じ結果を出します．しかし，パターンをそろえた前者は単なる美的趣味でスペースを入れたのではありません．この例では，どこか一箇所，例えば dx を dy に書き間違えただけでも正しい結果が得られません．しかも，dx と dy を書き間違えても文法的なミスにはならず，出力結果を見ても間違いに気づかない可能性があります．よって，できるだけプログラムを書いている段階でミスを取り除いておかねばなりません．

この時，前者のようにパターンをそろえておけば，横方向に一行一行チェックするだけでなく縦方向にも比較しながらチェックすることができます．色々な角度からチェックすることで，ミスのないプログラムにすることができるのです．

この他，文と文の間に適当なコメント文や，何も書いていない行を挿入すれば，プログラムの流れに区切りができてわかりやすくなります．これは文章を段落に分けるように，プログラムを区分けすると考えればいいでしょう．

8.4　意味不明の定数はできるだけ減らそう

数学的に決まっている単純な定数（2 倍するとか，1 を加えるとか，3 乗するとか）の場合以外は，できるだけ定数の使用量を減らすほうがいいと思います．プログラムを書いている時には理解していても時間が経ってから読んだ時に意味が分からなくなる可能性があるからです．

例えば，電子の質量 m_e（=9.1093897×10^{-31}kg）と光の速度 c（=2.99792458×10^{8}m/s）を使って計算すれば $m_e c^2$ = 8.187111168×10^{-14}（J）ですが，これを単位とするエネルギー，$energy/m_e c^2$ を計算するプログラムを，

```
ene = energy/8.187111168e-14
```

と書くと，後で 8.187111168e-14 という数字がどこから来たのか分からなくなる可能性があります．こういう時には多少面倒でも，

```
me  = 9.1093897e-31
cl  = 2.99792458e8
mc2 = me*cl**2
ene = energy/mc2
```

としておけば，わかりやすいしプログラム内容の記録にもなります．

また，do ブロックを 100 回繰り返す，とか，10 回ごとに出力する，とかいう制御数も，変数にしておけば数値の意味が明示されるし，変更もしやすくなります．例えば，

```
do i = 1, 100
  a(i) = b(i)**2
enddo
sum = 0
do i = 1, 100
  sum = sum + a(i)
enddo
```

という do ブロックにおいて，100 という数は配列要素数に依存する共通の数値ですから，以下のように変数にします．

```
imax = 100
do i = 1, imax
  a(i) = b(i)**2
enddo
sum = 0
do i = 1, imax
  sum = sum + a(i)
enddo
```

こうしておけば意味がはっきりするし，100 を 200 に変更したい時には変更点が一箇所ですみます．このような変数化はプログラムが長くなるほど重要性が増します．

まとめと次回の予告

今回は，基本的なプログラムの書き方と，書く時の“こつ”について説明しました．今回説明した基本的文法を使うだけでも，様々な数値計算プログラムを書くことができます．昔から“習うより慣れろ”といいます．身近なパソコンを利用して色々な計算をしてみて下さい．

次回は，サブルーチンについて説明します．サブルーチンを使えば，一連の計算手順をブラックボックス化してその機能だけを使ったり，大きなプログラムを分割してメンテナンスのしやすいプログラムにすることができます．

また，数値を出力する際の出力形式を指定する方法や，データをファイルに保存したりファイルから読み込んだりする方法など，入出力文の詳細についても説明します．

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] 数値計算の常識，伊理正夫・藤野和建，共立出版，1985 年

[利用相談室便り]

利用相談について

今年度も５月よりサイバーサイエンスセンター本館利用相談室、弘前大学、秋田大学、山形大学で利用相談を行っています。日程等詳細は次頁をご覧ください。相談内容によってはメーカ等に問い合わせる場合や、時間を要する場合もありますが、利用者の問題解決にむけて努めております。直接面談のほかに、メールや電話での相談も受けておりますのでお気軽にご相談ください。

・プログラムを高速化するにはどうしたらいいの？
・プログラムを並列化してもっと速く計算したい！
・スパコンでプログラムを動かしても速さがPCと変わらないんだけど、どうして？
・研究室のコンピュータではメモリが足りない！
・研究室の電気代高騰で困っている。
・コンピュータの管理は面倒。研究に専念したい。
・サービスしているアプリケーションを研究室から利用するにはどうすればいいの？

このような、スーパーコンピュータ利用に関する疑問や問題をお持ちの方、これから利用してみたいとお考えの方、一度相談してみてはいかがでしょうか。

また、サイバーサイエンスセンター本館相談室には、各種マニュアル、書籍も多数揃えています。相談室での閲覧、貸し出し（一部の書籍、マニュアルを除く）も可能ですので是非ご活用ください。

《サイバーサイエンスセンター本館利用相談室》

Tel.022-795-6153　　　相談用メールアドレス　sodan05@isc.tohoku.ac.jp

サイバーサイエンスセンター本館

本館利用相談室

平成 24 年度利用相談日程と主な担当分野（本館）

曜日・時間		ﾃｸﾆｶﾙｱｼｽﾀﾝﾄ	主な担当分野
月	２－４時	佐々木大輔（情報基盤課共同研究支援係）	・スーパーコンピュータ ・大判プリンタ
火	２－４時	沢田　雅洋（理学研究科） 山下　毅　（情報基盤課共同利用支援係）	・並列コンピュータ ・スーパーコンピュータ ・アプリケーション全般
水	３－５時	山崎　馨　（理学研究科）	・アプリケーション（Gaussian） ・並列コンピュータ
木	３－５時	坂本　修一（電気通信研究所）	・アプリケーション（MATLAB） ・C/C++
金	２－４時	小松　一彦（サイバーサイエンスセンター） 森谷　友映（情報基盤課共同研究支援係）	・並列コンピュータ ・スーパーコンピュータ
＊上記以外の時間帯に面談・電話での相談を希望の方は、共同利用支援係（1 階窓口）まで相談内容をお申し出ください。センター内担当者に取り次ぎます。			

平成 24 年度利用相談日程と担当分野（他機関）

大学名	相談場所・日時	ﾃｸﾆｶﾙｱｼｽﾀﾝﾄ	相談分野
弘前大学	理工学部 1 号館 322 号室 在室中随時	佐藤　裕之	ｽｰﾊﾟｰｺﾝﾋﾟｭｰﾀ, 端末ﾛｸﾞｲﾝ,ﾌｧｲﾙ, Fortran, ﾍﾞｸﾄﾙ化, ASL
弘前大学	理工学部 2 号館 0404 室 月曜　16:00-18:00	宮本　　量	端末ﾛｸﾞｲﾝ, ﾌｧｲﾙ, Fortran, C/C++, Gaussian
秋田大学	工学資源学部 1 号館 337 室 在室中随時	田中　元志	ｽｰﾊﾟｰｺﾝﾋﾟｭｰﾀ, 端末･ﾛｸﾞｲﾝ, ﾌｧｲﾙ, ｼﾞｮﾌﾞ操作, Fortran, C/C++, MATLAB, 利用申請, ﾒｰﾙ
山形大学	学術情報基盤センター （小白川キャンパス） 金曜 10:00-12:00	板垣　幸由	端末･ﾛｸﾞｲﾝ, ﾌｧｲﾙ, ﾒｰﾙ, ｳｨﾙｽ対策ｿﾌﾄ, ｻｰﾊﾞ証明書
	工学部 7 号館 245 号室 火曜 14:30-16:30	高野　勝美	端末･ﾛｸﾞｲﾝ, ﾌｧｲﾙ, Fortran, MATLAB
	工学部学術情報基盤センター 在室中随時	鈴木　勝人	端末･ﾛｸﾞｲﾝ, Fortran, TOPIC/ｲﾝﾀｰﾈｯﾄ(組織間接続), ﾒｰﾙ, ｳｨﾙｽ対策ｿﾌﾄ

新テクニカルアシスタント自己紹介

《サイバーサイエンスセンター本館利用相談室》

小松　一彦（こまつ　かずひこ）
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タ研究部　助教

本年度よりサイバーサイエンスセンター利用者相談室で利用相談員を担当させていただくことになりました。金曜日 14～16 時の担当で、担当分野はスーパーコンピュータ(SX-9)、並列コンピュータ(Express 5800)の利用方法全般、およびアプリケーションの高速化全般になります。

計算機アーキテクチャ・大規模並列計算を専門として研究に従事しており、様々な大規模計算環境におけるアプリケーションの最適化を通じて、次世代大規模計算機の要素技術を研究しております。このため、大規模計算機の計算機アーキテクチャやネットワークシステム構成などを考慮したアプリケーションの最適化・高速化をサポートさせて頂こうと考えております。利用相談員として、微力ながらみなさまのお力になれればと思います。よろしくお願いいたします。

森谷　友映（もりや　ともあき）
東北大学情報部情報基盤課共同研究支援係　技術一般職員

平成24年度から東北大学の技術職員として採用され、サイバーサイエンスセンター利用者相談室で金曜日(14～16時)の利用相談員を担当させて頂くことになりました、森谷と申します。

これまで民間企業で5年間働いてきましたが、以前からスーパコンピュータに興味があり、今年からスーパコンピュータ関連の仕事に従事するようになりました。1つ1つ仕事を学びながら、質の高いサービスを提供できるように心がけています。

サイバーサイエンスセンターでは、A0サイズ対応の大判カラープリンタの利用、各言語のプログラミング本、各種マニュアル、資料の閲覧も可能です。また、科学的、工学的分野に特化した様々なアプリケーションソフトの提供もしており、利用相談などを通じて、是非有効活用して、研究に役立てて欲しいと思っております。

利用相談員として、まだまだ未熟な部分もあり、ご迷惑もお掛けするかと思いますが、少しでもみなさまのお力になれればと思っています。よろしくお願いいたします。

*前年度以前よりテクニカルアシスタントをご担当頂いている皆様の自己紹介は、SENAC Vol.44,No.3 p.62-p.65(2011.7)をご覧ください。

［展示室便り⑤］

ACOS シリーズ

今回は、日本電気（株）製の ACOS シリーズです。センターで使用した計算機は、このシリーズの 700,900,1000,2020,3900(1976 年～1997 年)でした。オペレーティング・システム（OS）は ACOS-6 です。図 1 はこれらの計算機、約 20 年間の演算処理能力と主記憶容量を表したものです。20 年間で演算性能、メモリ容量ともに 250 倍となりました。

展示品 1　ACOS3900 の筐体

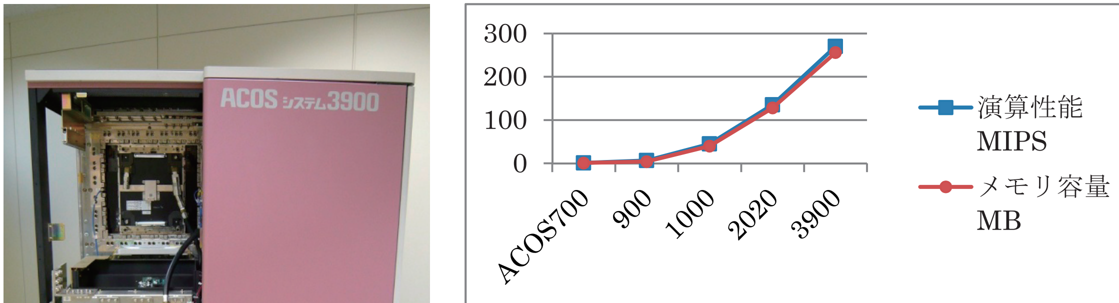
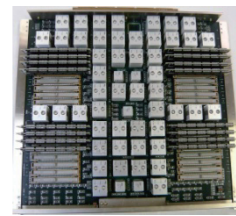


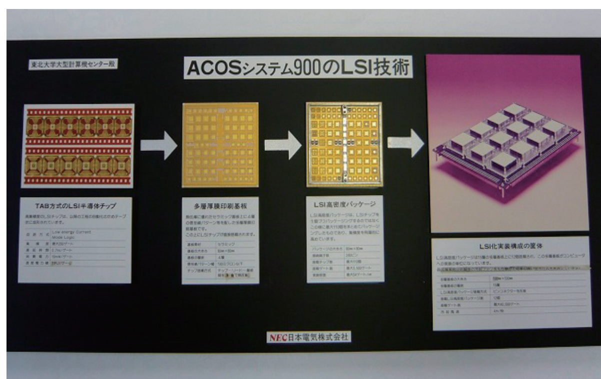
図 1　演算性能とメモリ容量

展示品 2　1CPU(表と裏)



展示品 3　メモリボード

展示品 1 は ACOS3900 の筐体の一部です。センターに設置していたときは全長 10m くらいありました。展示品は CPU、主記憶装置、信号ケーブル、電源ケーブル、CPU を冷やす水冷ケーブル、フレームなどから構成されています。展示品 2 は、ACOS3900 の 1 つの CPU です。この CPU は約 30×30cm の大きさで、水による冷却を行なっていました。裏面には水冷ケーブルの受け口があります。展示品 3 は、ACOS3900 の主記憶装置で CPU とほぼ同じ大きさです。このボード一つで 32MB を構成しています。現在センターで提供している並列コンピュータ Express5800 では、1 ボードあたり 64GB です。今皆さんご使用のノート PC でも数 GB は装備していると思われます。展示品 4 は「ACOS900 の LSI 技術」です。LSI チップから LSI 高密度パッケージを作る過程が示されています。



展示品 4　ACOS900 の LSI 技術

展示品 5　「TSS の使い方」説明書、ポータブルプリンタ、モデム装置

展示品 5 は左から「TSS の使い方」説明書、ポータブルプリンタ、モデム装置です。「TSS の使い方」はセンター教職員と利用者の協同により ACOS での TSS（タイムシェアリングシステム）の使い方について、初心者を対象に分かり易く書かれた説明書です。利用者から好評を得た説明書の一つでした。ACOS-6（OS）の TSS は非常に使い勝手がよく、また、そこで提供されていたテキストエディタの評判もよいものでした。中央はポータブルプリンタです。蓋をかぶせるとアタッシュケースのような型となり持ち運びができます。利用者はこの端末で研究室、あるいは自宅からも TSS を利用することができました。電話の受話器を端末の受け口（右側の黒いところ）にセット、電話機からセンターに設置してあるモデムの電話番号を回し、電話回線経由で端末とモデム装置を接続します。右側のモデム装置（変調復調装置）はセンター側に設置されていたもので、電話回線からのアナログ信号をデジタル信号に変換し ACOS に送信するものです。ポータブルプリンタから ACOS 利用のイメージは、図 2 のようになります。

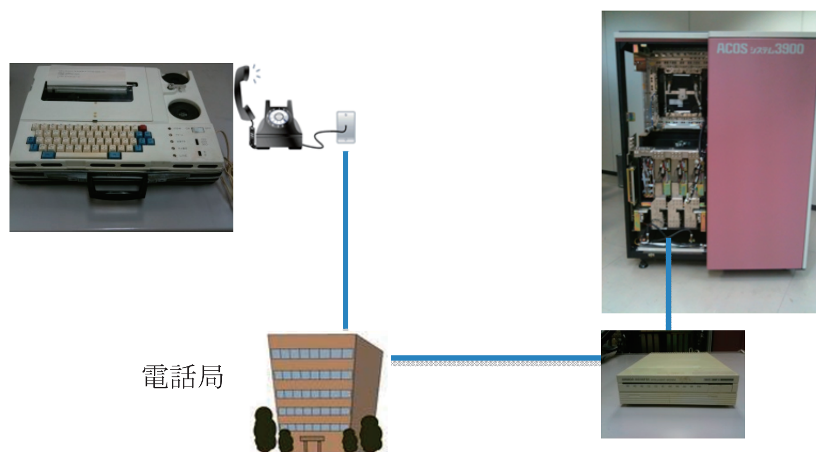


図 2　ポータブルプリンタから ACOS 利用

[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/

負担金の支払い費目について（No. 135）

次回の利用負担金請求（平成24年4月1日から6月30日までの利用分）は7月初旬に行います。

負担金の支払費目の指定に関して、これまで shiharai コマンドで指定頂いておりましたが、今年度より廃止いたしました。学内の方については事前に費目の指定は必要ありません（請求金額確定後、センター会計係より各部局の会計担当を通して照会いたします）。

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

（共同利用支援係，会計係）

利用負担金額の表示コマンドについて　（No. 135）

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
kakin	利用者ごとの利用額を各システム、月ごとに表示
skakin	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）

― 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

編 集 後 記

SENAC の本号の編集中に、スーパーコンピュータの世界ランキングで有名な TOP500 の最新版(June 2012)が発表になり、理化学研究所と富士通のスパコン「京」が第2位を獲得したというニュースが届きました。「京」は昨年11月のランキングでは第1位で、今回連勝を逃したとは言え、世界の熾烈なスパコン開発競争の中で半年後にまだ2位に位置しているのは、大変立派な成果でしょう。

ところで、このようにずば抜けた演算性能を有するスパコンがあっても、やはりそれを使って得られる成果こそが、真に重要なものであると言えます。私たちのセンターにある SX-9 システムは、「京」と比較するとはるかに小さなものですが、それでも研究室とは比較にならない圧倒的な演算能力と巨大なメモリを利用することができ、国際的に競争力のある研究に大いに役立っています。SENAC では定期的に利用者の研究成果を紹介しておりますが、本号では 2 件の共同研究成果報告をお届けします。スパコンを利用した研究成果がどのように我々の社会や科学技術の進歩に役立っているのか、時々SENAC を手にとって、それらを感じ取っていただけると幸いです。

(後藤英昭)

センターで提供しているスーパーコンピュータ、並列コンピュータは、防災・減災に関する研究分野、気象・地球シミレーションに関する研究分野、医療に関する研究分野、ものづくりなど様々な研究分野で活用されています。また、センターで実行されているこれらのシミュレーションプログラムのソースコードを見ますと、まだまだ圧倒的に Fortran プログラムが多い状況となっております。利用者からの質問、講習会のテキストなども、Fortran ベースのものが多い状況です。

今回発行の SENAC より、「Fortran スマートプログラミング」と題しまして、3回にわたり Fortran プログラムの基本的な書き方を紹介していく予定でおります。すでに Fortran プログラムをご利用の方は、昔を懐かしみ復習の意味を込めて、またこれから新たにプログラムを作成される方は、プログラム開発の第一歩として今号の SENAC を手にとって頂ければうれしく思います。ぜひ、研究のツールとしてセンターのスーパーコンピュータをお役立てください。(S.O)

サイバーサイエンスセンター前
整備中の青葉山新キャンパス

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

平成24年7月発行
編集・発行　東北大学
サイバーサイエンスセンター
仙台市青葉区荒巻字青葉 6-3
郵便番号　980-8578
印　刷　東北大学生活協同組合
プリントコープ

システム一覧

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

サービス時間

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

ジョブクラスと制限値

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

目次

東北大学サイバーサイエンスセンター
大規模科学計算システム広報　Vol.45 No.3 2012-7

［共同研究成果］
「かぐや」月レーダサウンダが見た月の海 …… 小林敬生 1
加藤雄人
熊本篤志
小野高幸

金属ナノ構造を含む一般フォトニック結晶の
光学応答計算コードMPI化による高速化 …… 岩長祐伸 9

［研究成果］
大規模計算システムにおけるBCMの性能評価 …… 小松一彦 17
曽我　隆
江川隆輔
滝沢寛之
小林広明

［資料］
Fortranスマートプログラミング
― 第1回　基本的なプログラムの書き方 ― …… 田口俊弘 27

［利用相談室便り］
利用相談について …… 51
新テクニカルアシスタント自己紹介 …… 53

［展示室便り⑤］
ACOSシリーズ …… 54

［Web版大規模科学計算システムニュース］より
負担金の支払い費目について（No.135） …… 56
利用負担金額の表示コマンドについて（No.135） …… 56

執筆要項 …… 57

編集後記 …… 58

VEGETABLE OIL INK