地球磁気圏数値シミュレーションにおける計算効率改良

Improvement of Efficiency in Numerical Simulation using MHD

吉浪　颯　　　（共著者：才田 聡子）
So YOSHINAMI

In recent years, magnetohydrodynamic (MHD below) simulations have been performed on supercomputers using Fortran to reproduce interactions between the Earth's magnetosphere and the solar wind. The Reproduce Plasma Universe (REPPU) code for modeling complex systems of a magnetohydrodynamics (MHD below) model and a magnetosphere-ionosphere interaction model. The REPPU code becomes larger, the more costs for developments and operations we will need. Furthermore, only certain people can develop and execute scientific calculations with a supercomputer. For giving new knowledge, we have to allow the verification of scientific claims, by allowing others to look at the reproducibility of results, and by integrating data from many simulation codes. Therefore, we should rewrite the REPPU code into C so that it can be executed on PCs in the future. To enhance maintenance performance, we propose to partially rewrite and to modularize the source code so that the development cost and the calculating delay time will be reduced. In this study, we rewrite some part of the REPPU simulation code including conditional branch instructions and repeat instructions into C. In addition to that, the computational efficiency depends on how the grid is constructed in the simulation. Therefore, we rewrite the grid construction modules of the code in order to improve the computational efficiency.

Key Words : *Substorm, Aurora, Computer simulation, Magnetosphere, Fortran*

1. 緒　　言

地球磁気圏とは，宇宙空間の中で地球の磁場が届く領域のことである．地球には地球の南極をN極・北極をS極とする1本の巨大な棒磁石があるかのように磁場が形成されている．この地球磁気圏に対して太陽からは太陽風と呼ばれる高速かつ高温のプラズマ流体が吹いている．そのため図1のように，地球磁気圏の太陽側は圧縮され，太陽とは反対側の領域は太陽風が流れていく方向に引き延ばされた形をしている．また，地球の大気の上部は電気を帯びたプラズマ状態になっており，電離圏を形成している．地球磁気圏シミュレーションを行うことにより，太陽風の変化によって地球周辺に生じる電磁気現象を予想できると考えられる(1), (2)．

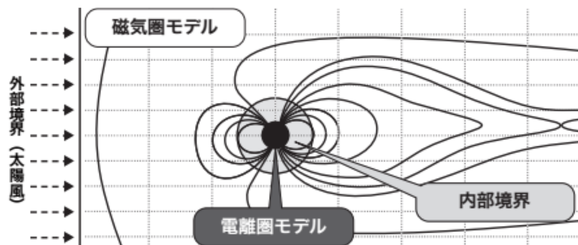

Fig. 1 The magnetic field around the Earth(2)

現在，地球磁気圏シミュレーションによって，上流の太陽風や枠星間磁場（IMF）の変化に対する磁気圏・電離圏での大きな擾乱現象であるサブストームや磁気嵐を数値シミュレーションから調べようとする試みも行われるようになった(3)．

現代社会におけるGPS測位や衛星放送などのように，人類の生活基盤は宇宙空間にも存在している．しかし，太陽活動の影響による激しい変動が生じると，宇宙空間に滞在する人工衛星の損壊や宇宙飛行士の被曝，通信障害などが起きる．このような被害は，人類の活動に大きな打撃である．この打撃を最小限に止めるために，宇宙空間で起きる現象の解明と現象の数値シミュレーションによる再現が重要になる．サブストームは，宇宙空間で起きる爆発現象の典型例であるといえる．そのため，サブストームの解明は，宇宙で起きる同様の爆発現象の解明や普遍的な宇宙プラズマ現象の理解につながる．このような観点から，サブストームの研究は重要である(4)．

現在，電磁流体シミュレーションは，Fortranという言語を用いてスーパーコンピュータで実行されている．しかし，スーパーコンピュータでは開発費や運用費が大幅に必要になる．また，Fortranもメジャーな言語とは言えず，多くの人が様々な環境で実行できるようにするためには，シミュレーションコードを改良する必要がある．そのため，将来多くの人がシミュレーションを実行できるように，C言語に書き直した方が良い．よって本研究では，Fortranで書かれた電磁流体シミュレーションコードをC言語に書き換える．また，シミュレーションで構築するグリッドによって計算効率が変わる．そのため，グリッド構築の部分のコードを書き換え，計算効率の向上を目指す．シミュレーションコードはReproduce Plasma Universe（REPPU）コードを使用する(5)．

2. 実験

2・1 実験手法

本研究ではREPPUコードを実行し12面体分割三角格子による，ブロック並列化M-I結合系シミュレーションを行う．このシステムは磁気圏だけでなく電離圏も詳細に計算し，地磁気変動やオーロラ活動などの細部まで再現することを目指している．この点に関して特に注意を払って格子が設計されているのがREPPUコードの特徴である．その結果，FACと電離圏擾乱を詳細に解析する点では，今までのシミュレーションとは比較にならない高精度を達成している．これにより，電離圏で解と観測とを比較することができ，解がどの程度観測結果を再現しているかを検証できる(6)．

REPPUコードは，いくつかのステップに分けて実行される．プログラムは前準備，本体，後処理に分かれており，本体と前準備の一部はスーパーコンピュータで実行される．前準備では，12面体分割格子の生成や座標格子の並列化，磁気圏初期値の設定，電離圏システムの生成などを行う．本体では，座標データ，初期値データ，電離圏データ，太陽風データを入力し，磁気圏・電離圏の計算を進め，結果をファイルに出力する．本体を実行する際，前準備の一部のプログラムも共に実行する．後処理では，磁気圏可視化データの生成，電離圏可視化データの生成などを行う．後処理のプログラムによって，本体で出力した結果を可視化する．以上がREPPUコード実行の概要である．

書き換えを行うのは，12面体分割格子の生成のプログラムである．このプログラムは図2に示すように，12面体の頂点（20点）面芯（12点）からなる格子（計32点）を1次分割とし，以下1つの三角形を4つに分割する操作を加え，2次，3次...と高次の格子を作成する．磁気圏計算のためには，原点近くで球座標，遠方で円筒座標に類似した配置に漸近を行う．座標データは1ファイルに出力する．以上が12面体分割格子の生成プログラムの概要である．

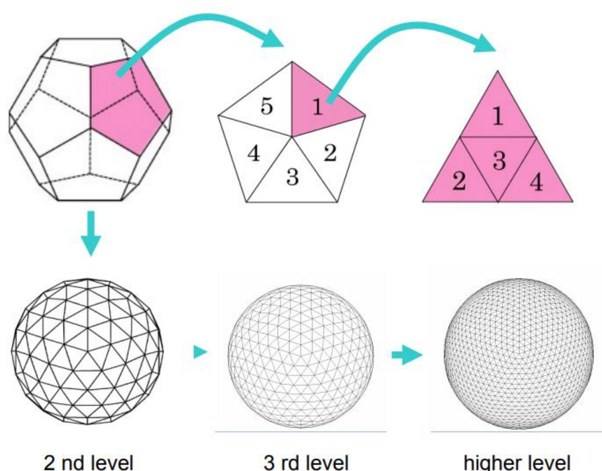

Fig. 2 Generating a dodecahedral triangular lattice(5)

2・2 手法

12面体分割格子の生成プログラムは，18個の関数で書かれたプログラムである．これらの関数のうち，10個の関数をC言語に書き換えた．C言語に書き換えたプログラムと，それに該当するFortranプログラムについて，計算速度を比較する．

計算速度上昇のために，コードの最適化を行い，計算速度を比較する．最適化とは，ソースからオブジェクトコードを生成する過程において，効率の良いオブジェクトコードを生成する手法である．最適化によって，コードのパフォーマンスを向上できる．最適化をすると，コンパイラは，コンパイル時間や場合によってはプログラムのデバッグ能力を犠牲にして，パフォーマンスやコードサイズの改善を試みる．使用した最適化オプションについては2.3節で記述する．

2・3 最適化オプション

最適化オプションについて記述する．

・O1

最適化には多少の時間がかかり，大きな関数の場合には多くのメモリが必要となる．このオプションを使用すると，コンパイラは，コンパイルに多大な時間を要する最適化を行わず，コードサイズと実行時間の短縮を図る．

・O2

このオプションはO1の最適化に加え，さらに最適化を行う．空間速度のトレードオフを伴わない，サポートされているほぼ全ての最適化を実行する．コンパイラは，ループアンローリングと関数のインライン化を実行しない．ループアンローリング，関数のインライン化，及び，レジスタのリネーム化を除く全ての最適化オプションを実行する．また，全てのマシンでfforce-menオプションをオンにし，デバッグを妨げないマシン上でフレームポインタを消去する．このオプションは，O1と比較して，コンパイル時間と生成されたコードの性能の両方を向上させる．

・O3

このオプションは，O2の最適化に加え，さらに表1の最適化を行う．O2と比べ，ベクトライズ化やインライン展開などのさらなる積極的な最適化が行われる．環境やコードによっては実行効率が良くなるが，逆に遅くなることや不

Table 1 Optimization Option O3

fgcse-after-reload	fsplit-paths
fipa-cp-clone	ftree-loop-distribution
floop-interchange	ftree-partial-pre
floop-unroll-and-jam	funswitch-loops
fpeel-loops	fvect-cost-model=dynamic
fpredictive-commoning	fversion-loops-for-strides
fsplit-loops	

具合が発生することもある．

・OO

オプションを指定しない場合，このオプションがデフォルトとなる．コンパイラはコンパイルのコストを下げ，デバッグで期待通りの結果が得られる．ステートメントは独立しているため，ステートメント間にブレークポイントを設けてプログラムを停止し，変数に新しい値を代入したり，関数内の他のステートメントのプログラムカウンターを変更することができ，ソースコードから期待される結果と全く同じものを得られる．

・Os

このオプションはサイズを最適化する．O2 の最適化から，表 2 のコードサイズを増加させるものを除き，全ての O2 の最適化を行う．また，finline-functions を有効にし，コンパイラが実行速度よりもコードサイズを優先して調整するようにする．さらに，コードサイズ削減のための最適化を行う．

Table 2 Optimization Option Os

falign-functions	falign-loops
falign-jumps	fprefetch-loop-arrays
falign-labels	freorder-blocks-algorithm=stc

・ffloat-store

このオプションは，浮動小数点型変数をレジスタに格納せず，浮動小数点値をレジスタとメモリのどちらから取得するかを変更する可能性のある他のオプションを禁止する．このオプションは，浮動レジスタが，double が持つはずの精度よりも高い精度を保つ，という望ましくない過剰精度を防ぐ．ほとんどのプログラムでは，過剰な精度による実害は少ないが，一部のプログラムでは，IEEE 浮動小数点の正確な定義に依存している．そのようなプログラムでは，関連するすべての中間計算を変数に格納するようにプログラムを修正した後，このオプションを使用する．

・fno-defer-pop

このオプションは，関数呼び出し後に引数をポップする必要があるマシンでは，各関数が戻り次第，常に引数をポップするようにする．O1 以上のレベルでは，fdefer-pop がデフォルトである．これにより，コンパイラは関数呼び出しのために引数をスタックに蓄積させ，それら全てを一度にポップさせることができる．

・fforce-addr

このオプションは，メモリアドレス定数を演算する前に，強制的にレジスタにコピーする．これは，fforce-mem と同様に，より良いコードを生成する可能性がある．

・fomit-frame-pointer

このオプションは，O1 以上ではデフォルトで有効である．フレームポインタを必要としない関数では，フレームポインタを保持しないようにする．これにより，フレームポインタの保存や設定，復元を行う命令が不要になり，多くのマシンで余分なレジスタも利用可能になる．一部のマシンでは，標準の呼び出しシーケンスが常にフレームポインタを使用するため，このオプションは効果がない．なぜなら，標準的な呼び出しシーケンスは自動的にフレームポインタを処理し，それが存在しないように見せかけても何も保存されないからである．このオプションは，全ての関数でフレームポインタが使用されることを保証するものではない．ターゲットによっては，リーフ関数で常にフレームポインタが省略される．

・fno-inline

このオプションは，最適化しない場合デフォルトである．always_inline 属性でマークされた関数以外をインライン展開しない．単一の関数は，noinline 属性でマークすることにより，インライン化から除外できる．

・finline-functions

このオプションは，O2，O3，Os で有効である．また，fprofile-use および fauto-profile でも有効である．インライン化が宣言されていない関数も含めて，全ての関数をインライン化の対象とする．コンパイラは，この方法で統合する価値のある関数を決定する．ある関数への全ての呼び出しが統合され，その関数が静的に宣言されている場合，通常その関数はそれ自体ではアセンブラコードとして出力されない．

・fkeep-inline-functions

C 言語では，オブジェクトファイルにインライン宣言された静的関数は，その関数の呼び出し元すべてにインライン化されている場合でもエミュレートする．C++では，あらゆるインライン関数をオブジェクトファイルに出力する．

・fno-function-cse

このオプションは，関数のアドレスをレジスタに格納せず，定数関数を呼び出す各命令に関数のアドレスを明示的に格納させる．このオプションはコードの効率を低下させるが，アセンブラの出力を変更する一部の不正なハッキングは，このオプションを使用しない場合に実行される最適化によって混乱する可能性がある．このオプションは，ffunction-cse でデフォルトである．

・ffast-math

このオプションは，fno-math-errno，funsafe-math-optimizations，ffinite-math-only，fno-rounding-math，fno-signaling-nans，fcx-limited-range 及び，fexcess-precision=fast オプションを設定する．このオプションにより，プリプロセッサ __FAST_MATH__ が定義される．このオプションは，Ofast 以外のオプションでは有効にならない．これは，数学関数に関する IEEE または ISO 仕様の正確な実装に依存するプログラムでは，不正な出力になる可能性がある

ためである．しかし，これらの仕様の保証を必要としないプログラムでは，より高速なコードが得られる可能性がある．

2・4　計算速度比較

12 面体分割格子の生成プログラムの CPU 時間を，Fortran と C 言語で比較したものを表 4 に示す．12 面体分割格子の生成プログラムの CPU 時間のうち，最適化オプション O2 が最も高速であった．最も高速であった O2 について，Fortran と C 言語で計算速度を比較する．値を比較したものを表 3 に，グラフで比較したものを図 3 に示す．

Table 3 O2 CPU time comparison

オプション		Level-2	Level-3	Level-4	Level-5
O2	Fortran	0.071875	0.0859375	0.13125	0.671875
	C	0.0120805	0.015746	0.0465927	0.4904406

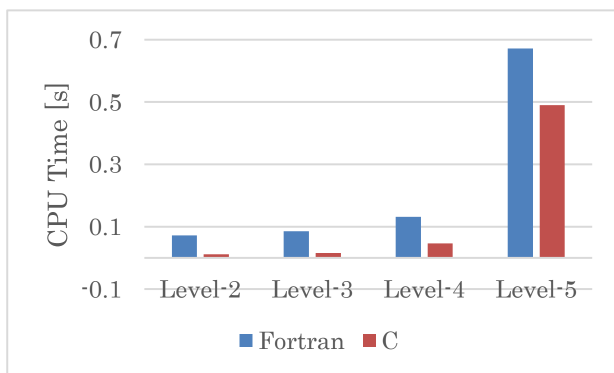

Fig. 3 O2 CPU time comparison graph

比較した結果，Lelvel2～Level5 の全ての場合で，Fortran よりも C 言語の方が，CPU 時間が早い結果となった．これは表 1 で示した全ての最適化においても，同じように Fortran よりも C 言語の方が早い結果となった．その原因としては，12 面体分割格子の生成プログラムでは，配列演算に置き換えることのできない条件分岐や繰り返し処理が多かったことが考えられる．また，Level を上げると配列が増えるため，Level を上げるごとに Fortran の計算速度が上がることが考えられる．

Level を上げるごとに，Fortran の計算速度が上がる可能性があるため，Fortran から C 言語への時間短縮の割合を求める．オプション O2 での，Fortran から C 言語への時間短縮の割合を，図 4 に示す．Level-2 では 83%だが，徐々に下がっていき，Level-5 では 27%という結果となった．また，その他のオプションの場合では，Level-2 では 80%以上だったが，徐々に下がり，Level-5 では 20～30%という結果となった．このことから，Level が上がり配列が増えるほど，両者の差がなくなることが分かる．また，Level をさらに上げることができれば，C 言語の方が遅くなる可能性も考えられる．

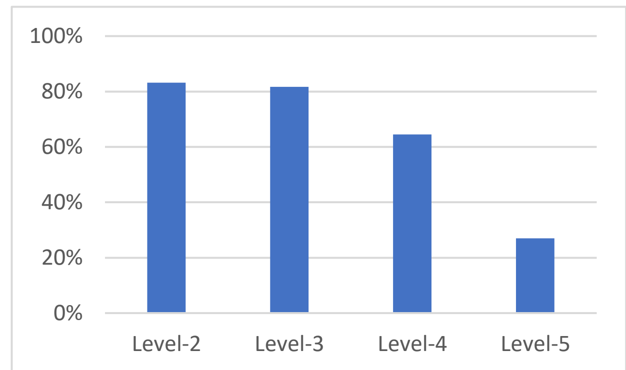

Fig. 4 Percentage of time saved from Fortran to C

3.　結言

コードの最適化を行った結果，オプション”O2”の計算速度が最も早かった．また，全ての場合において Fortran よりも C 言語の方が早い結果となった．その原因としては，シミュレーション準備段階である 12 面体分割格子の生成プログラムでは，配列演算よりもグリッド構築の処理が煩雑であることが挙げられる．Fortran は配列演算を早く処理できるが，今回のプログラムでは，配列演算に置き換えることのできない条件分岐や，繰り返し処理が多かった．そのため，それらの部分の処理速度が早い C 言語の方が早くなったと考えられる．

Level を上げると配列が増えるため，Level を上げるごとに Fortran の計算速度が上がる可能性がある．そのため Fortran から C 言語への時間短縮の割合を求めた．その結果，Level-2 では 80%以上だったが，Level-5 では 20%～30%という結果となった．このことから，Level が上がり配列が増えるほど，両者の差がなくなることが分かる．また，Level をさらに上げることができれば，C 言語の方が遅くなる可能性も考えられる．このことから，12 面体座標格子の生成プログラムでは，Level の高くないシミュレーションの場合，C 言語の方が，速度が早いことが分かった．そのため，高 Level でないシミュレーションでは，C 言語を用いることで計算効率の向上を図ることができる．

Fortran と C 言語の計算速度の比較結果から，プログラムによっては，C 言語に書き換えることで計算効率を向上できることが分かった．ここから，さらに計算効率を向上させるためには，Fortran と C 言語の使い分けが考えられる．関数群をモジュール化して，配列処理なら Fortran, 条件分岐や繰り返し処理なら C 言語化して Fortran から C を呼び出す，または C から Fortran を呼び出すことで，お互いの得意な処理を任せる形で処理を進めることができる．この場合，Fortran コンパイラを持たないコンピュータでの実行はできないが，計算効率を向上できる．また，C 言語はメジャーな言語であるため，メンテナンス性の向上も期待できる．さらに，データの可視化の部分のプログラムに，

Table 4 CPU time comparison

オプション		Level-2	Level-3	Level-4	Level-5
O1	Fortran	0.076563	0.082813	0.132813	0.796875
O2		0.071875	0.085938	0.131250	0.671875
O3		0.075000	0.079688	0.126563	0.671875
OO		0.079688	0.089063	0.234375	2.278125
Os		0.068750	0.082813	0.126563	0.721875
ffloat-store		0.076563	0.092188	0.232813	2.284375
fno-defer-pop		0.078125	0.089063	0.234375	2.285938
fforce-addr		0.076563	0.087500	0.228125	2.278125
fomit-frame-pointer		0.070313	0.090625	0.234375	2.448438
fno-inline		0.078125	0.090625	0.229688	2.282813
finline-functions		0.078125	0.090625	0.231250	2.281250
fkeep-inline-functions		0.076563	0.090625	0.229688	2.281250
fno-function-cse		0.071875	0.092188	0.229688	2.284375
ffast-math		0.078125	0.089063	0.232813	2.285938
オプション	C	Level-2	Level-3	Level-4	Level-5
O1		0.012109	0.014878	0.049457	0.530291
O2		0.012081	0.015746	0.046593	0.490441
O3		0.011909	0.014615	0.045762	0.491096
OO		0.012851	0.021772	0.130471	1.779455
Os		0.012326	0.015405	0.049712	0.536237
ffloat-store		0.012692	0.022499	0.132194	1.784917
fno-defer-pop		0.012484	0.021198	0.129711	1.779390
fforce-addr		0.012215	0.021174	0.129511	1.779619
fomit-frame-pointer		0.011915	0.021140	0.129542	1.783634
fno-inline		0.011628	0.021479	0.130300	1.779856
finline-functions		0.012094	0.021072	0.129468	1.780051
fkeep-inline-functions		0.011849	0.021587	0.130102	1.780077
fno-function-cse		0.012374	0.021462	0.130522	1.779790
ffast-math		0.012168	0.021542	0.130130	1.779640

OpenGL などを用いるなど，他の処理もそれを得意とする言語で行うことで，計算効率とメンテナンス性の向上が期待できる．

汎用性の面を考えた場合，本研究でプログラムを C 言語に書き換えたように，Fortran から他のメジャーな言語に書き換えることが重要である．元のプログラムの中に，Fortran でしか読むことができない形式のファイルを扱う部分があった．そのファイルを利用するためには，Fortran を扱えなければならない．多くの人が利用できるようにするためには，Fortran 以外の言語でもそのファイルを利用できるようにした方が良い．そのため，多くの人が利用できるようにするためには，他のメジャーな言語に書き換えることが重要である．他のメジャーな言語に書き換える場合，汎用性の向上は期待できるが，欠点として，複数の言語に書き換える場合，開発・運用コストが上がってしまうことが挙げられる．

文　献

(1) 海老原祐輔，”オーロラ爆発のしくみ”，混相流 33(3), 2019，pp267-274

(2) 才田聡子他，“グローバル MHD シミュレーションモデルの磁気圏-電離圏境界条件における境界パラメータ感受性の調査”，第 63 回理論応用力学講演会, OS06/OS14-06-04, 2014 年

(3) Ebihara, Y. Simulation study of near-Earth space disturbances: 2. Auroral substorms. Prog Earth Planet Sci 6, 24 (2019).

(4) 柴田 一成，上出 洋介，総説・宇宙天気，京都大学学術出版会, 2011 年.

(5) 田中高史，”次世代 M—I 結合シミュレーション REPPU（Reproduce Plasma Universe）”，2017.5.27.

(6) Tanaka, T., Substorm Auroral Dynamics Reproduced by Advanced Global Magnetosphere–Ionosphere (M-I) Coupling Simulation, in Auroral Dynamics and Space Weather, Hoboken, NJ: John Wiley & Sons, Inc, 177–190 (2015).