46

Adaptive Mesh Refinement 法のライブラリ開発

Making of adaptive mesh refinement library

○山田 良透 (京大理)、宮下 尚 (京大理)

Yoshiyuki Yamada and MIYASHITA Hisashi

We make a library program by which we can easily apply adaptive mesh refinement technique to existing simulation codes based on finite difference or finite volume scheme. The library program is made independently of user defined time integration scheme. Thus the library program has wide applicability to simulation of various time evolution problems: viscous or inviscid fluids, MHD, combustion, and so on. To perform several accuracy tests for this code, we improved integration and interpolation algorithms. As an example, we investigate the possibility of a finite time blow-up in 2D ideal Boussinesq equations by the adaptive mesh code. We can use 2^{21} grids in each direction effectively. Our results suggest that the temperature gradient and vorticity diverge as $(t_0-t)^{-2}$ and $(t_0-t)^{-1}$, respectively. In the initial stage, temperature-vorticity sheets are formed which is known to grow exponentially at most. We can observe an instability of the sheets, which is considered as the source of a finite time blow-up.

1 Introduction

近年、Adaptive Mesh Refinement(以下 AMR) と呼ばれる方法によるシミュレーションが行なわれている。この技術は、偏微分方程式で記述される時間発展問題において、全体としては粒度の粗い格子で計算を進めることにより CPU 時間やメモリーといった計算機リソースの消費をおさえつつ、一方時間的及び空間的に必要なところに必要な解像度の格子を生成し、初期値及び境界値が整合するように粒度の粗い計算と同時時間発展計算を行なう事により、実質的に高い解像度の計算を行なったのと同じアウトプットを得ようとする目的で開発された手法である。我々のコードは、AMR ライブラリは、格子計算における汎用サポートライブラリとして開発されている。しかし、このコードはかなり複雑なデータ構造を要求するため、コードの構築は非常に大変である。

この手法自体は、1980 年代に Berger[4] らによって開発されたものであるが、当時の精度評価は必ずしも十分とはいえなかった。黎明期の精度評価は離散化度のレベルにして 3 程度でしか行なわれておらず、一方近年の AMR 法を用いた計算は離散化度のレベルで 10 を越える計算が行なわれている。そこで、我々は、衝撃波管問題や Bergers 方程式といった、流体分やでは良く知られた厳密解との比較を行ない、精度を再評価した。我々が、1980 年代に提唱された方法をそのまま実装して精度チェックを行なったところ、離散化度のレベルが 3 か 4 程度以上になると、それ以上の精度の改善が見られなくなることが分かった。そこで、十分な精度を得るために、二点の改良を行なった。一つは時間発展アルゴリズムの変更が必要であり、もう一つは誤差評価のアルゴリズムの改良である。これに伴い、データ構造の改良も必要となった。本講演では、我々の開発したライブラリとその応用について紹介する。

黎明期の AMR 法の提案には、具体的な時間発展アルゴリズムは書かれていない。従って、我々の行なった改良を新しいものと評価しないとする意見もある。しかし、その後最初の提案者も含め多くの著者が、この方法を実装して計算しており、この中では具体的なデータ構造も示されている。我々の改良は、これらが示すデータ構造では実現し得ないものであり、最初の提案者の提案は曖昧ではあるが、我々が今回行なったような改良法を含むと解釈することには無理があると、著者らは考えている。

2 Physics Simulation and AMR method

最初に、AMR 法の位置付けについて述べる。物理法則のシミュレーションを行う場合、典型的な場合は、以下の手順が踏まれることになる。

(1). 対象のモデル化に基づく、偏微分方程式の導出　この段階では、対象の研究手法を下に、偏微分方程式を記述する。例えば、物理学的な手法で対象をモデル化する場合は、物理学の理論に基づいた方程式が導出される。

(2). スキームの導出　偏微分方程式の、空間・時間の適切な離散化に基づく、計算スキームの導出を行なう。この段階では、数値計算の技術が要求される。与えられた連続的な偏微分方程式を、どのように空間的・時間的に離散化するか、また、時間積分などの方法はどのようにして行うかをこの段階で決定する。分解能、解像度が、数値計算結果に与える精度への影響は、この計算スキームが左右することになる。

(3). 計算スキームに基づいて、実際のコードを作成。(2) で、決定したスキームを実際の FORTRAN や C 等の計算機言語で記述する。

(4). コードの実行 計算結果の収集　作成したコードを実行し、計算結果を取得する。計算の量や質に応じた計算機リソースが要求される。

(5). 計算結果の誤差評価　計算結果の誤差は、要求される精度の範囲内に収まっているかを検証する。

(6). 計算結果の解釈　もともとのモデルの観点から、計算結果を解釈する。例えば、物理学においては、物理学的な観点から、計算結果を捉えなおす。

一般にシミュレーションを行うためには、(2) で示したとおり、空間、時間の離散化が不可欠であり、精度を向上させるためには、その分解能および解像度を上げなくてはならない。しかし、時間分解能を a 倍に向上させると、当然、その計算量も a 倍に増えてしまう。また、空間解像度を b 倍に向上させると、b^d(d は次元数) 倍で計算

This document is provided by JAXA.

量及び、要求記憶量が増してしまう。しかも、近年、計算機の高速化に伴い、(2) を出来るだけ自然な形で実装し、導出された偏微分方程式を、そのまま解く事例、及び、要求が多くなっている。例えば、従来であれば、計算量を劇的に減らすために空間の対称性を仮定して次元数を減らすなどの細工を必要とした。しかし、このような手法は一般的なものではなく、多様化していくシミュレーションの要求に答えきれなくなっている。具体的に例をあげると、空間の次元が3次元である場合、シミュレーションの解像度を $b=2$ 倍高めようとすれば、$b^3=8$ 倍のメモリーが要求される。時間発展計算が必要な場合は時間解像度も関係するから、計算時間では少なくとも $b^{3+1}=16$ 倍必要となる。亜音速気体や電磁場、自己重力を含む系などでは、Poisson 方程式を解かなければならない。Poisson 方程式は、通常緩和法で解かれるが、一般に全体の格子数が増えれば、必要な反復回数も増える。従って、解像度 b に対して要求される計算量 b^r のべき r は、次元数 d に対して上の $d+1$ より大きくなる。つまり、空間解像度を2倍を高めようと思っても計算量の増加は16倍ではすまず、数十倍から100倍以上になることもある。つまり、多次元計算で十分な解像度のシミュレーションを行うことは容易なことではないのである。

AMR 法は、このような問題を解決するために、近年広く用いられている。しかし、その実装は非常に複雑であるため、我々は AMR 特有の部分をライブラリ化した。

3 Library construction

我々のコード開発には、特に以下の点が重要視されている。

(A) 解くべき基礎偏微分方程式、また、そのための計算スキームから、ライブラリは完全に独立した設計にする。依存部が不可欠である時でも、適切な抽象化層 (abstraction layer) を用意し、上位層の依存を避けるようにする。

(B) ライブラリの使用者は、使用者の問題に関して選択された計算スキームに基づく実際のコード化の作業だけに集中できるようにする。つまり、計算スキーム及び、離散化の手法、変数の性質に基づく平均化の部分の実装だけが必要とされるように、充分なサービスを、一般性を失わない範囲で提供する。

(C) 性能、および、その拡張性 (scalability) を重視する。各 component は、出来るだけ効率の良い実装方法を追求する。また、並列化についても重視して設計しており、各 object の相互依存性を出来るだけ排除するように設計している。実際、現状で、POSIX thread library に基づく、並列化が実装されている。また、thread model に基づく並列化だけでなく、message passing model に基づく並列化も考慮中である。

(D) (C) に悪影響を及ぼさない範囲で、可搬性を重視する。中枢部のコードは、ANSI C++ および、ANSI C の標準を逸脱しないように書かれており、特定の machine architecture への依存は完全に排除されている。また、コンパイル作業も GNU autoconf を用いて可搬性を高めており、一般的な POSIX system には、容易に移植することが出来るようになっている。また、Windows 上でも Visual C++で動作するようになっている。

(A) の独立設計のため、流体力学、宇宙物理学、相転移問題、燃焼問題など、さまざまなスキームに同じプログラムが利用可能である。しかし、(A) の設計方針は、応用可能性を目的としたものではなく、応用可能性は副産物としてついてくるものである。このような、高度なプログラミングにおいては、分離可能なものは極力分離し、抽象的な形で実現しておかなければ、コードのバグを発見することも著しく困難になってしまうのである。

(C) の設計はバランス問題である。コードを洗練されたものにすることにより、実行効率が犠牲になるようなことは、極力避けている。そのことが、データ構造にある種の制限を加えてしまう場合もある。しかし、効率が必要とされ、かつ効率に非常に関わる部分では、それもやむを得ない。また、実際に計算を行った結果、並列化効率も非常に良いことが分かった。ただ、POSIX thread library に基づく並列化を行っているため、thread の同期に対するオーバーヘッドが大きく、共有メモリー型の Simmetric Multi Processing(SMP) 型の計算機以外での効率は、今のところ評価していない。

(D) の設計手法のおかげで、現在 Linux、FreeBSD、IRIX、Digital Unix 上で動作している。スーパーコンピューターの利用目的のため、ほかの特殊な OS を使わなければならないとしても、標準的な POSIX 環境であれば、移植性はかなり高い。autoconf には頼っていないが、現在独立に Windows 系の OS の上で Visual C++が動作する環境であれば、そのための Makefile も同時に構築している。この他、UML による分析や cvs によるコード管理など、ソフトウエア業界では既に標準となりつつあり、近年科学計算の分野でも採り入れられ始めているソフトウエア工学の様々な手法を利用して、開発にあたっている。

4 Algorithms and Specification

AMR 法は、格子生成法の一つと見ることも出来る。その意味では、航空宇宙業界で用いられている解適合格子の一つである。しかし、通常の解適合格子は、解適合プロセスを一連のシミュレーションの中で数回用いるに過ぎないが、この方法では毎時間発展毎に解適合のためのプロセスを実行している。我々は、計算座標上は等間隔の n 次元格子を利用する。これは、実際に物理的な意味で等間隔の格子を使うことは意味していない。計算座標として、何らかの関数形で表される適切な座標系に移っていれば、物理的には不等間隔・非直交でも計算座標では等間隔で表現される場合がある。そのような座標系を利用することが適切と判断される問題については、ユーザーはそのような選択を行なう余地がある。しかし、基本的には二次元あるいは三次元の直交等間隔格子で計算を行ない、解像度が不足すれば解適合プロセスにより細かい格子を生成すると言うことで十分である。

図1で、我々が利用する格子の概念図を示す。この図で示したように、我々が採用した Mesh Base の AMR 法は、複数の長方形格子の同時時間発展を行う方法である。それぞれの格子は、異なる解像度 Δx を持つ。解像度は、非負の整数 ℓ によって

$$\Delta x_\ell = \Delta x_0 u^{-\ell} \tag{1}$$

と書かれる。ここで、u は一つの計算の間には定数で、2以上の整数である。この u を、upper level factor と呼ぶことにする。研究者によっては refinement ratio という呼び方を採用している場合もある。この式 (1) の ℓ は各メッシュの離散化度を表すもので、レベルと呼ばれる。レベルは、原理的に無限に増えてよいのだが、計算リソースの制限から無限に増えると計算が困難である。そこで、ℓ の最大値 $\ell_{\max}$ というパラメーターを用意しておくこと

This document is provided by JAXA.

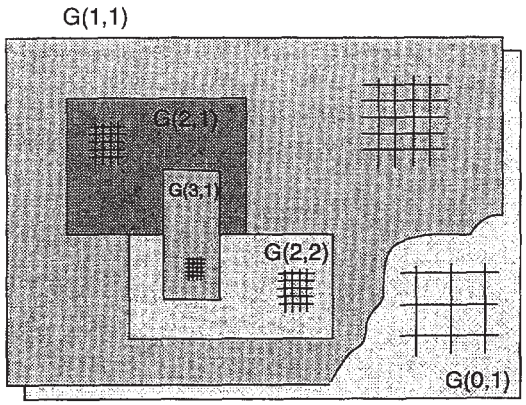

Fig. 1: メッシュベースの Adaptive Mesh Refinement 法の概念図。各格子は $G(\ell,i)$ と記され、level ℓ と同じレベルの格子の index i の二つの index で区別される。G(0,1) と G(1,1) は、計算領域全体を覆うメッシュで、計算初期から最後まで変更されることなく存在する。隣接する二メッシュの比較により、必要な部分に通常のメッシュを生成する。

にする。同じレベルの格子は一つとは限らない。そこで、それぞれの格子は、レベル ℓ と、同じレベルの格子を区別するための index i により

$$G_{\ell,i}$$

と書くことにする。また、同じレベルの格子の全体を、

$$G_\ell = \cup_k G_{\ell,k}$$

と書くことにする。

計算を開始するときには、格子は $G_{0,1}, G_{1,1}$ の二つしかない。これらの格子は、計算の全領域を D として、その全体をカバーする。

$$G_1 = G_{1,1} = G_0 = G_{0,1} = D$$

時間発展の途中では、たくさんの格子が存在する。実際、我々の計算でも 200 以上の格子が同時に時間発展している。隣接 2 層 (ℓ と $\ell+1$) で時刻が同期する毎に、結果を比較して精度評価を行う。この結果、精度が不足しているなら $G_{\ell+2,i}$ を生成、精度が十分なら $G_{\ell+1,i}$ を消去してメモリーを節約する。この格子生成消滅の操作が終了した後、$G_{\ell+1,i}$ の結果を $G_{\ell,i}$ に上書きに上書きし、次の時間ステップの計算をはじめる。

点 $(x,y) \in D$ は、複数のメッシュに含まれることがある。この時、最終的な解は、この点を含む最も細かい格子からとられる。すなわち、最も細かいメッシュの解は、再計算されることがなければ求める誤差範囲で正しい解であるが、粗いメッシュでの解は求める誤差範囲を超えている場合があるので、採用されることの内容に注意を払うことが必要である。

計算領域 D の外の情報はユーザーが与える必要がある。メッシュ$G_{0,1}, G_{1,1}$ および、D の境界 ∂D に接したレベル 2 以上のメッシュに境界条件を与える手段は、一般のシミュレーション同様ユーザーが提供する必要がある。

計算領域の境界 ∂D に接していない、レベル $\ell > 1$ のメッシュ$G_{\ell,i}$ の境界 $\partial G_{\ell,i}$ の情報は、レベル ℓ の物理量自身から、あるいは必要ならば粗いグリッドの情報から線形に内挿して得る。そのためのもっとも単純な手続きは、ライブラリーが提供している。

計算精度の関係で、複雑な内挿式を用いる必要がある場合は、この手続きをユーザーが書き換えることができる。内挿方法は多種多様なので、ライブラリー設計としては、ユーザーが書き換える手段を提供する以上のことはできない。

各時間ステップの発展における初期条件は、メッシュの形が変わらない場合は、前の時間発展の結果を用いればよい。ただし、より細かいメッシュがある場合は、時間発展を開始する前に、細かいメッシュでの時間発展の結果を、適当な粗視化を行って上書きしておかなければならない。一般的な誤差評価方法は、一時間発展での誤差の程度を見積もるものである。最終結果としてより細かいメッシュの結果を保持しているとしても、誤差評価を適切に行うためには、毎時間発展ごとに粗いメッシュと細かいメッシュの間で結果の同期をとっておく必要がある。

新たに細かいメッシュが生成された場合は、適当な内挿手続きを用いて、現在ある粗いメッシュの結果を細かいメッシュに書き写す必要がある。この内挿手続きは、境界条件の場合と同様、ユーザーが提供すべきである。

レベル ℓ の長方形格子の境界条件と初期条件を決める手続きをまとめると、

(i) 境界セルを包含するレベル $\ell-1$ の格子上の値から空間内挿して、レベル ℓ 相当に離散化された値を得る。
(ii) 必要ならば、適切な時間内挿を行い境界の解を得る
(iii) レベル ℓ に値を供給するさらに細かい格子があれば、(ii) から得られた値を上書きする。

ということになる。

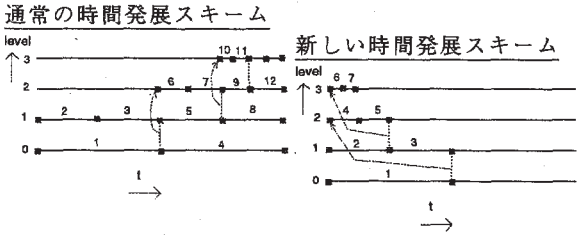

Fig. 2: rewind の概念図：左図で 0 と 1 のレベル間で、誤差が閾値を越えるとレベル 2 を遡って積み上げる。中央の図の様に新たに各レベルを時間発展させる。更に、1 と 2 のレベル間で誤差が閾値を越えると、更に遡ってレベル 3 を積み上げる。このようにして、誤差が閾値を越えないように時間発展を行う。各図中の 1, 2, 3, ... は、実際のステップ発展の順序を表している。上層のレベル程時間刻が短い事に注意。

図 2 に、従来の時間発展法と我々の改良した時間発展方を示す。図中の数字は、時間発展手続きが行われる順序である。従来法では、1, 2, 3 の時間発展を行った後に誤差評価をし、精度が不足していれば、1 および 3 の計算が終わった時刻で細かいメッシュを生成し、ここから計算をはじめる。この場合、2, 3 の計算を行った離散化度では精度が不十分である。そして、6 の計算を開始するために用意される初期条件は、3 の終了時のデータから得られる。つまり、6 の計算の初期条件にはすでに大きな誤差が含まれている。したがって、この部分の誤差はそのあとの計算で改善される可能性は、偶然正しい解に向かう以外にはあり得ない。そこで我々は、誤差評価の結果メッシュの再構築が必要であると判断した場合には、それぞれの計算が誤差評価を行った時刻の一つ前に同期した時点にさかのぼって、計算を再度行うような改良を

This document is provided by JAXA.

行うことにした。これを、rewindと名づける。

Berger & Oligerの論文示されているアルゴリズムは不明瞭で、どちらをともとれるのだが、Berger & Oligerの論文にはデータ構造が示されており、このデータ構造で実装可能なアルゴリズムは、我々が図中で従来法としてしめしたものだけがである。Berger & Oligerの論文だけ引用して我々の改善が新しくないと主張するのは、誤りである。実際、多くの研究者がこのあとの計算で具体的なアルゴリズムを示しており、すべて我々が従来法として示したものになっている。

もう一つの改良は、誤差評価法である。Berger & Oligerの論文では、ことなる離散化度の計算結果の誤差評価にRichardson外挿が用いられている。この外挿手続きを行なうため、粗い格子の計算結果と、細かい格子の計算結果を粗い格子へ平均化したものとの比較が行なわれる。しかし、平均化は、細かい格子を生成するための情報を落している。そして、誤差評価方法と細かい格子の生成法の整合性は、誤差評価を行なう上で非常に重要である。そこで、われわれは、解適合プロセスのたびに、細かい格子を生成し、粗い格子の計算結果から生成された細かい格子と実際に細かい格子で行なわれた計算結果とを比較することで、誤差評価を行なうことにした。

5 Performance

次に、我々の改良コードの性能を評価する。

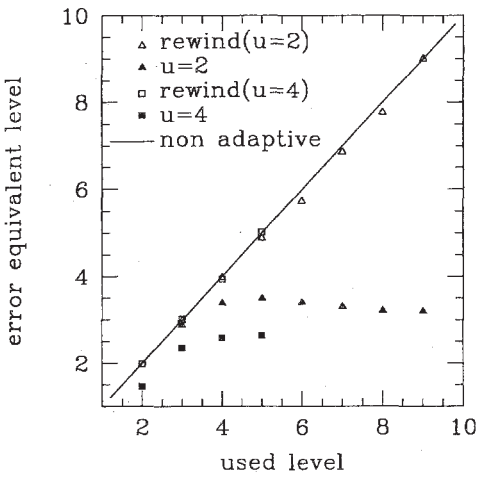

Fig. 3: 高いレベルまで精度が直線的に伸びる事が分かる。我々の計算は白抜き、従来の方法は黒い図形で表している

誤差評価には、厳密解が知られているものを用いている。具体的には一次元の衝撃波管問題と、二次元のBurgers方程式である。これらの問題を、適切な初期条件のもとで、AMR法を用いた場合とAMR法を用いずに等間隔で大きなメッシュを使った場合の結果を比較したのが、図3である。

この図では、誤差をerror equivalent levelという形で表示している。これは、非AMR計算において同じ誤差を与える計算の格子間隔Δxを、AMR計算で最も離散化度の細かい格子の格子間隔に対応する数字に焼き直したものである。定義により、非AMR計算では*used level*は*error equivalent level*に一致する。我々が改良した時間発展アルゴリズムであるrewind法を用いた場合、結果はこの直線にほぼのって降り、期待する精度を得たと結論でいる。一方従来提案されている時間発展アルゴリズムでの計算結果は、離散化度3から4程度でさちっており、これ以上いくら離散化度の細かい格子を積み上げても、精度は改善しない。図の結果は衝撃波管の例だが、Burgers方程式でも結果は同じである。

また、我々はメモリー効率についても評価した。AMR法では、必要な粒度の細かいメッシュは動的に生成されるため、予め必要な数のメッシュを予測できない。同じ領域に粒度の細かいメッシュと粗いメッシュが共存していることにより、単一の粒度の細かいメッシュを利用する場合よりメモリー使用量が増大する可能性がある。計算領域全体にわたって細かい計算が必要な問題では、そのような事情は発生し得る。また、我々は精度の改善のためにrewindという手法を導入している。この手法では、すでに進行中の計算を必要な時点まで戻すことを可能にするため、過去の計算結果の一部を保持している。このことも、メモリー効率を落す一つの要因である。しかし、実際に計算してみると、一次元衝撃波管問題でもメモリー使用量は粒度の細かい単一メッシュを使う場合より若干ながら少なく、また二次元Burgers方程式では十分な改善が見られている。次元が増えれば、粒度の細かい計算が必要な領域の体積比は減ることが予想されるので、2次元、3次元問題ではrewindを使ったとしてもAMRを用いる効果は十分見られることが分かる。

6 Application to the fluid dynamics

また、このソフトウエアを応用した例として、現在我々は、Boussinesq近似方程式

$$\frac{\partial T}{\partial t} + \boldsymbol{v}\cdot\nabla T = \kappa\triangle T, \tag{2}$$

$$\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v}\cdot\nabla \boldsymbol{v} = -\nabla p - \alpha g T \boldsymbol{e}_y + \nu\triangle \boldsymbol{v}. \tag{3}$$

の特異性を持つ解の再現のための計算を進めているので、一例としてこれを紹介する。ここで、νとκはそれぞれ動粘性率と熱拡散係数で、α、gは体積膨張率と重力加速度の大きさを表し、$\boldsymbol{e}_y$をy軸方向の単位ベクトルとすると重力の方向はy軸下向きとなる。一般性を失うこと無くαgを1とできる。また、既に平均密度ρ_0は、1と置いている。我々は、非粘性の場合に滑らかな解から出発して有限時間発散をする場合があるかどうかということに注目している。この計算での格子間隔は、一次元方向に最大計算領域に対して2×10^6である。初期条件として、中央部が温度が高く、周囲で温度が低く、その境界が直線ではなく三角関数的な摂動が入っている場合をとる。境界条件はx、y方向とも周期2πの周期境界条件を用いた。重力は、y軸方向下向きに取っている。

各格子での時間発展スキームは、空間方向の差分にTVD条件を満たす2次のMUSCLE法[2]を、また時間発展には2次のルンゲ・クッタ法を用いた。ポアソン方程式の解法には、2次精度のSCG法を用いている。空間差分の精度を一致させるためには、高次差分を用いる必要がある。また、高いレベルの情報が低いレベルにフィードバックされる効果も採り入れる必要があるかもしれないので、Multi Grid Iterationと呼ばれる緩和プロセスを導入することで、Poisson方程式についてもAMR的に精度を向上させている。

This document is provided by JAXA.

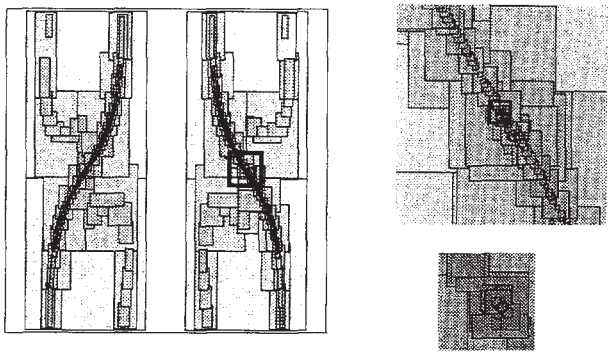

Fig. 4: レベル生成の一例：15段のレベルを積んだ状態。図中の太線で囲われた領域を拡大して表示している。強いフロントが形成されている所に細かいメッシュが集中している。

図4は、計算途中でのメッシュの生成の様子を示したものである。ちょうど温度勾配がきつい部分で細かい格子を沢山生成していることが分かる。

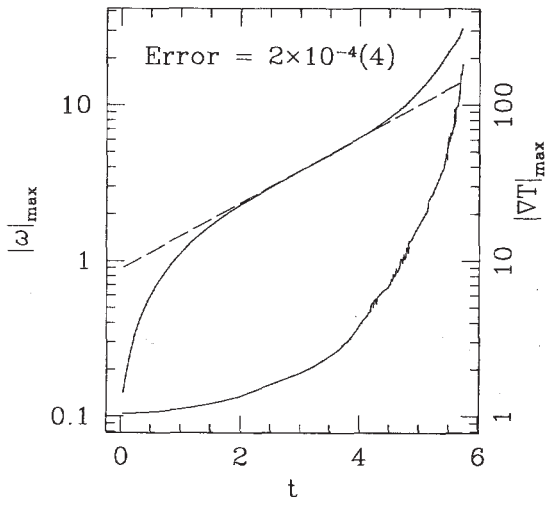

Fig. 5: 温度勾配および渦度の発散の様子

図5は、温度勾配及び渦度の発散の様子である。この図では、時間が線形、物理量が指数メモリで書かれているので、このグラフで直線に乗れば指数的な増大を示す。計算結果から、明らかに指数的な増大から離れていることが見える。指数関数的な増大は、物理量が無限になるのに無限時間を良うするので、ここで問題にしている発散にはあたらない。これから離れている・が、有限時間発散を示唆している。有限時間発散の場合は、ある発散時間 t_0 があって、物理量は $(t_0-t)^{-\alpha}$ の形で変化する。この $\alpha>0$ を数値的に決めることは難しい。というのは、t_0 と α を同時に決める必要があるからである。しかし、発散付近では十分発散・に近くまでデータを得るとは出来ないため、このフィッティングには多少任意性が残る。しかしながら、我々の計算では

$$|\omega|_{\max} \propto (t_0-t)^{-1}$$

$$|\nabla T|_{\max} \propto (t_0-t)^{-2}$$

の結果が得られている。有限時間発散が起こるとすれば、渦度の勾配は -1 より小さく、温度の勾配は -2 より小さく、その差は1であることが数学的に示されているので、ほぼこれに準じた結果を得たと考えている。

現象が発散に近付く、あるいはある種の不安定性が増大するような問題においては、発散や不安定の増大が計算上のものなのか、物理的に起こり得るものなのかを判断することは非常に難しい。これは、AMR法固有の問題ではなく、数値計算全般に言えることである。厳密解が知られている問題に関して精度を評価することは、計算コードの正しさを示す一つの証拠となる。また、例えば上のように数学的に予想される発散のしかたとの一致を調べることも、一つの証拠を示したものとなる。スペクトル法では、初期のエネルギー分布を高波数側で十分に落しておき、各段階でエネルギー分布をモニターすることで、計算の信頼性があるかどうかのチェックを行なうことが出来る。そこで、今後、スペクトル法なども併用して、この問題をよりクリアーにしてゆく予定である。

7 Future works

一般に、物理の方程式は、発展方程式と制限方程式(拘束方程式)の組合せで書くことが出来る。電磁場でも相対性理論でも、数学的にそのような構造になっている。ソフトウエアの応用可能性として、我々はすでに時間発展型の双曲型偏微分方程式と、Poisson型の制限方程式の組合せを解くことが出来るようになっているので、必要な計算スキームを組み込むことで、このコードはより多くの問題を解くことが出来ると期待できる。

双曲型方程式だけで閉じる例として、我々は相転移モデル、圧縮性流体、Burgers方程式をチェックしている。拘束方程式としては、静電場のチェック問題を用いて精度を評価している。上で例示したBoussinesq方程式は、これらの組合せである。一般に非圧縮性流体は発展方程式と拘束方程式の組合せとなる。

今後、航空宇宙業界のような実用分野でこのコードを利用するには、計算領域内部にある境界相の扱いを実装する必要がある。これについては、Zeeuらが行なっている方法があるので、これを応用する予定である。また、スキームとしても、燃焼問題などより複雑なものへ応用してゆきたいと思っている。

参考文献

[1] 山田良透、宮下尚、物性研究
[2] van Leer, B.: *J. Comput. Phys.* bf 32 (1979) 101
[3] Pumir, A. and Siggia, E.D. : *Phys. Fluids*, **A4** (1992) 1472
[4] Berger, M. J. and Oliger, J. *J. Comput. Phys.*, **53** (1984) 484-512.

This document is provided by JAXA.