平成 18 年 1 月 4 日

研究内容紹介

Curant(クーラン) 条件と Debug

1. 数値的安定条件

PIC-Simulation を実現するにあたり、計算コスト削減のために時間・空間格子の導入を行ったが、それでも膨大な計算時間を必要とする。時間・空間格子の導入は、計算を効率化する一方で、スキーム上の安定条件を満たさなければならない。これを Courant-Friedrich-Lewy(CFL またはクーラン) 条件という。これは、

$$\nu = (c\Delta t)^2 \cdot \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2} \right) \tag{1}$$

と表される。$|\nu| < 1$ の条件を満たすとき、計算を安定的に実行できる。逆にこの条件が満たされないと、数値振動や非物理的な現象が励起される可能性がある。

規格化された刻み幅 $\Delta\hat{t}, \Delta\hat{x}, \Delta\hat{y}, \Delta\hat{z}$ を用いて (1) 式を書き直す ($c = 1$) と

$$\frac{1}{\Delta\hat{t}^2} > \frac{1}{\Delta\hat{x}^2} + \frac{1}{\Delta\hat{y}^2} + \frac{1}{\Delta\hat{z}^2} \tag{2}$$

となる。例えば、$\Delta\hat{t} = 0.1, \Delta\hat{x} = \hat{y} = \hat{z} = 0.3$ の場合は、

$$100 > 33.333\cdots \tag{3}$$

となり、クーラン条件を満たす。しかし、$\Delta\hat{t} = 0.1, \Delta\hat{x} = \hat{y} = \hat{z} = 0.1$ の場合

$$100 \ngeq 300 \tag{4}$$

となり、クーラン条件を満たさない。したがって、数値振動や非物理的な現象の励起が懸念されることになる。クーラン条件の意味は、時間発展 $\Delta\hat{t}$ のうちに波 (情報) が 1 セル以上伝わってはいけないことを意味している。(4) 式の場合は、$\Delta\hat{t}$ のうちに周囲 1 セルを越える範囲からも波 (情報) が伝搬し、計算を安定的に進めることができない。

2. Debug

次に少し立ち入ったプログラムのデバッグ作業 (バグ取り) について述べます。コード開発を自身で行う場合はもちろん、既存のプログラムに何らかの機能を追加する場合もプログラムを**見やすく書く**ことが重要です。自身にではなく、他人が見ても分かるように書くことが、バグを減らすことに繋がるだけでなく、拡張性や信頼性をも向上させます。

PIC-Simulation コードにおいて、よくあるバグとその原因を紹介していきます。

- **計算開始後に nan を吐いて止まる**
  nan とは not a number の略で例外的な演算をした場合に発生します。nan には演算に用いられるとその結果も nan になるという特性があり、最初に nan が発生した箇所を発見することが、重要になります。計算開始直後に nan が発生した場合は、物理量の初期値が適切に設定されていない可能性が高いです。
- **電磁場が時間と共に急激に成長し inf になる**
  inf とは infinity の略で無限を意味しますが、ここではハードウェア的に取り扱い不能な大きな数値が発生したことを意味します。境界付近で大きな成長が見られる場合は、境界条件が適切に与えられていない場合が考えられます。計算領域全体で成長している場合は、用いている計算式や PIC 法等が原因になっている可能性があります。原因となっている物理量や計算式が特定できない場合は、一度電磁場 (FDTD) だけのコードを組むといいでしょう。
- **粒子の速度が光速を超える**
  適切な相対論補正が設定されていない可能性が考えられます。適切な補正が設定されていても、初期値で 1 を超える値が与えられていると、計算過程で inf が発生し、その次の計算で nan を生成し、結局意味をなさない計算となってしまう場合もあります。初期値の設定 (速度だけでなく) には十分な注意が必要です。
- **計算式はあっているが変数値が異常値をとる**
  型と型の整合がとれていない可能性が考えられます。double 型の計算式中に int 型の割算などが入って来ると、結果がすべて 0 になるケースがあります。型が混在する計算を行う場合は、適切な型変換を行うことが重要です。また、define で何らかの計算式を定義する場合は、定義した計算しきを括弧 ( ) でくくると良いでしょう。

define は自己定義関数と異なり、単語をそのまま置き換える特性がありますので、足し算や掛け算が混在した式中に括弧 ( ) で括らないで用いると計算順序が狂い正しい結果が得られないことがあります。

- **セグメンテーション違反**
  セグメンテーション違反とは、配列要素に定義した領域異常の値を代入した場合や何らかの不正値 (例 : -1) を入れた場合に発生します。セグメン違反は、コンパイル自体は正常終了している場合がほとんどで、配列要素に不適切な値を代入する可能性のある箇所を自力で見つけなければなりません。その際は、debug ツールが役に立ちます。また、厄介なことにマシン環境によってはセグメンエラーが発生しないケースもあり、注意が必要です。
- **変数値が大きな値をとる**
  計算式や関数は適切に設定されているはずなのに、変数値が有限な異常値をとる場合、変数が初期化されていない可能性が考えられます。宣言した変数には、ランダムな値が代入されており、初期化しないでカウンター (例 : i++) などに用いると、乱数に値が加算されていき異常な値を返します。宣言した変数は、必ず初期化を心がけてください。

3. gcc コンパイルオプション

- -Wformat
  printf や scanf の変換指定子のチェック
- -Wuninitialiezed
  変数の初期化チェック
- -Wall
  warning 用のオプションすべて。コード開発時は常にこのオプションを用いること。
- -o
  コンパイル結果 (モジュール) を指定したファイルに出力する。
- -O
  処理最適化のオプション。-O2 あたりが妥当。
- -v
  コンパイル時のエラーメッセージを表示する。
- -p
  解析プログラム prof に適するプロファイル情報を出力するための追加的なコードを生成します。
- -pg
  解析プログラム gprof に適するプロファイル情報を出力するための追加的なコードを生成します。

4. コードの妥当性

シミュレーションコードが完成し、エラーも発生せず、それらしい値が出力されるようになると、そのコードが本当に合っているかが疑問になります。そのような場合、検証する方法として

- よく知られたシミュレーションを行い、理論値とシミュレーション値を比較する。
- TotalEnergy が一定かどうかを見る。
- 物理的な振る舞いを観察し、妥当かどうかを判断する。
- 他の人の同シミュレーション結果と比較する。
- 論文との比較

等が挙げられます。しかし、例え完全なコードが完成していたとしてもシミュレーション条件によっては、おかしな振る舞いをしたり、適切な数値計算が行われない場合も多々あります。それは PIC-Simulation 自体の限界だったり、不適切な条件であったりと原因は様々です。取り扱いたい現象があったなら、それに合ったスキームを採用を採用しコード開発を行うことが重要です。また、シミュレーションだけに囚われないように注意してください。シミュレーションは道具 (ツール) であり、決して完全なモノではありません。常に多角的に現象を見ることが大事です。