重力多体系の数値計算法

牧野淳一郎
東京大学理学系研究科天文学専攻

平成14年12月16日

何故数値計算するのかということ

いまさらかもしれないが、、、

- 解析的には解けない
- 衝突系でもフォッカー・プランク近似は正しいとは限らない
- 多次元ではフォッカー・プランク方程式は計算量的に解けない

で、多体問題を直接数値積分すれば、「原理的には」なんでもわかるはず。

現実的な効果

- 多くの系は年齢が熱力学的進化のタイムスケールよりはるかに若い
- 回転や速度の非等方性の効果は無視できない
- 構成要素である星の質量は等しくない
- 星は質点ではなく、大きさをもつ
- 星間ガスとかそういうものもないわけではない
- 相対論的効果とか、、、

といった効果も、多体計算なら比較的容易に取り込める。

数値計算の方法等

この種の問題：基本的に

- より大粒子数で
- より正確な

計算をすることで、「新しいことがわかる」（こともある）。

↓

どうやって今までより「良い（大きく、正確な）」計算ができるようにするかが問題

「良い」計算をする方法

- 計算法を改良する
- 速い計算機を買う
- 速い計算機を作る

以下、それぞれの話をする。

計算法

原理的には、多体シミュレーションはとっても単純：
運動方程式

$$\frac{d^2 x_i}{dt^2} = \sum_{j \neq i} G m_j \frac{x_j - x_i}{|x_j - x_i|^3}, \tag{1}$$

を数値積分するだけ。

右辺を計算するプログラム：2重ループで10行くらい
時間積分：なにかルンゲクッタとか適当なものを使えばいい

というだけで話が済めばいいけれど、もちろん世の中はそんなに簡単ではない。

何が問題か？

- 計算精度の問題： 2粒子の近接散乱、自己重力による構造形成 — 時間刻みをどんどん短くしないとちゃんと計算できなくなる。
  積分時間が長いので高精度の公式を使いたい。
- 計算量の問題： 右辺の計算量が $O(N^2)$ — N が少し大きくなるとすぐに計算時間が現実的ではなくなる

というわけで、以下、時間領域での計算法と空間領域のそれを概観しよう。

計算法 — 時間領域

算数としては、単に常微分方程式の初期値問題の数値解。まず、数値解法にはどういうものがあるかというのを概観しておく。

- ルンゲ・クッタ法
- 予測子・修正子法
- 外挿法

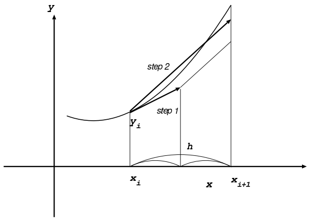
ルンゲ・クッタ

ルンゲ・クッタ法というのは、ある一般的な形に書ける公式のクラスであるが、ちょっとわかりにくいのでまず例から。

二次のルンゲ・クッタ法

以下のような計算法を考える

$$\begin{aligned} k_1 &= x_i + \frac{h}{2} f(x_i, t_i) \\ x_{i+1} &= x_i + h f(k_1, t_i + h/2) \end{aligned} \tag{2}$$



これは２次精度

別の二次のルンゲ・クッタ法

$$\begin{aligned} k_1 &= h f(x_i, t_i) \\ k_2 &= h f(x_i + k_1, t_{i+1}) \\ x_{i+1} &= x_i + \frac{k_1 + k_2}{2} \end{aligned} \tag{3}$$

さらに、これらの公式を含む一般的な公式を与えることもできる。

ルンゲ・クッタの一般形

$$\begin{aligned} x_{n+1} &= x_n + h \sum_{i=1}^{s} b_i k_i \\ k_i &= f(x_n + h \sum_{j=1}^{s} a_{ij} k_j, t_n + c_i h) \end{aligned} \tag{4}$$

s :段数（number of stages）という。

a_{ij}, b_i, c_i はパラメータ

a と c は普通

$$c_i = \sum_{j=1}^{s} a_{ij} \tag{5}$$

となるようにとる。

2段の公式

$s=2$の場合に書き下してみると

$$\begin{aligned} x_{n+1} &= x_n + h(k_1 b_1 + k_2 b_2) \\ k_1 &= f(x_n + h(a_{11}k_1 + a_{12}k_2), t_n + c_1 h) \\ k_2 &= f(x_n + h(a_{21}k_1 + a_{22}k_2), t_n + c_2 h) \end{aligned} \tag{6}$$

と、まあ、こんな感じ。

- $a_{ij} \quad (j \geq i)$ がすべて0ならば、k_1から順に計算していくことができる。つまり、「陽的」公式になっている。
- $a_{ij} \quad (j > i)$ が0のときは、各 k_i についての式に k_i だけが入ってくる。これを半陰的 (semi-implicit) 公式という。この場合には、まず k_1 についての方程式をといて、次に k_2 についてのを解いて、、、と順番に計算出来る。

陰公式

- 制約が全くない時は、「陰的」公式ということになる。このときは、すべての k_i に対する（一般には非線形な）方程式を一度に解く必要がある。

なぜ陰的公式といった面倒なものをつくるかというのはよくわからないかもしれない。これについてはあとで説明するとして、まず、陽的公式の例について述べる。

古典的 Runge-Kutta法

陽的ルンゲ・クッタ法のなかでもっとも広く使われているのが「古典的」といわれる公式である。書き下すと、

$$\begin{aligned} x_{n+1} &= x_n + h(k_1/6 + k_2/3 + k_3/3 + k_4/6) \\ k_1 &= f(x_n, t_n) \\ k_2 &= f(x_n + hk_1/2, t_n + h/2) \\ k_3 &= f(x_n + hk_2/2, t_n + h/2) \\ k_4 &= f(x_n + hk_3, t_n + h) \end{aligned} \tag{7}$$

というものである。これは、いろいろ良い性質をもつ。

古典的 Runge-Kutta法の特長

1. a_{ij} が $i-j=1$ 以外すべて0なので、右辺の計算が楽である。
2. 次数が4次であり、4段陽的公式で到達可能な最高次数を達成している
3. 係数が簡単な有理数なので、プログラムしやすい。また丸め誤差を小さくできる。

この公式が 4 次であることを示すのは、それほど簡単ではない。腕力に自信があるひとは挑戦してみて欲しい。

陽公式の到達可能次数

理論的、あるいは実用上重要なのは、段数を決めた時に、次数がいくつまで可能かということである。が、これは一般の場合には未解決であり、以下の結果しかわかっていない

s	1	2	3	4	5	6	7	8	9	10	$s \geq 10$
p	1	2	3	4	4	5	6	6	7	7	$p \leq s-2$

ある段数で、到達可能な最高次数を達成する公式というのは一通りではない。2段2次の場合でも、自由パラメータが一つあって「無限個」の公式があり得るわけである。段数が増えれば自由パラメータの数も増える。

どういう公式がいいか？

それではどういう公式を使うべきかというのは、精力的な研究が行なわれてきた。問題にもよるが、例えば

1. 局所離散化誤差の係数が小さい
2. 係数の数字が簡単な有理数になる
3. 係数に0が多い

というようなことはそれぞれ実用的には重要である。

Dormand と Prince は、段数／次数のさまざまな組合せについて、局所離散化誤差を非常に小さくした公式を求めた。

`http://www.unige.ch/math/folks/hairer/software.html`

から入手できる (Fotran と C がある)。

刻み幅調節

ある精度で計算したい: 要求精度に応じて刻み幅を変える必要あり

計算時間の無駄を減らす: 局所誤差が小さい（解が滑らかな）ところは刻みを大きく、逆に解が急にかわるところでは刻みを小さくする。

可変刻み (variable step) / 適応刻み (adaptive step)

どうやって刻み h を変えるか = どうやって誤差を推定するか

埋め込み型公式

RK 型の公式

$$x_{n+1} = x_n + \sum_{i=1}^{s} b_i k_i \tag{8}$$

k_i は全部そのまま使って、 b_i を別の b_i' に置き換える

$$x' = x_n + \sum_{i=1}^{s} b_i' k_i \tag{9}$$

x' が局所離散化誤差が元の公式よりも大きいが、むやみと大きくはない（例えば、一次次数が低い）なら、元の公式との差を誤差の推定に使うことができる。

この形の公式のことを埋め込み型 embedded 公式という。最初に提案した人の名前をとって Runge-Kutta-Fehlberg 公式ということも多い。Dormand-Prince はこの形である。

陰的ルンゲクッタ公式

陰的公式は、(理論的には) 素晴らしい性質をもつ。

「s 段陰的公式の到達可能次数は $2s$ である。」

これのため、いくらでも次数の高い公式をつくることができる。陽的な場合には8次よりも高次のものはほとんど知られていないのとは対照的である。

これは単なる存在証明ではなく、実際に $2s$ 次の公式が構築されている。これは実はユニークに決まり、「陰的ガウス公式」と呼ばれるものがそれ。

ガウス型数値積分

ガウス型公式について述べる前に、まずそれと非常に関係の深い数値積分公式について

数値積分とは、一変数であれば

$$y = f(x) \tag{10}$$

と、微分方程式ではなく単に関数として与えられたものに対して、ある固定区間 (t_s, t_e) での定積分を数値的に求めること。

数値積分公式

数値積分法として、すでに皆さんが知っているであろうものは、たとえば

- 台形公式
- シンプソン公式

といったものであろう。それぞれ、刻み幅に対して2次、4次の精度をもつ。

これに対して、もっとましな (高次の) 方法というのを考えられないであろうか？

補間多項式の利用

一つの方法は、区間の中の点をいくつかとって、それらを通る補間多項式を作り、それを積分すること

中の点を等間隔にとる:ニュートン・コーツ型の公式。

一般に補間するための点を p 個とった時、 $p-1$ 次の多項式を持ってくればそのすべての点を通るように出来る。

ニュートン・コーツ型の公式はそのような最小次数多項式を使うものである。例えばもとの関数が実際に $p-1$ 次以下の多項式ならば、厳密に正しいものが求まる。

補間多項式の構成 (ラグランジュ補間)

以下のような多項式を考える

$$p_i(t) = \frac{(x-x_1)(x-x_2)...(x-x_{i-1})(x-x_{i+1})...(x_i-x_n)}{(x_i-x_1)(x_i-x_2)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)} \tag{11}$$

この多項式は、 $x = x_i$ で 1、それ以外のすべての x_j で 0 になる。従って、補間多項式は

$$f_n(x) = \sum p_i(x) y_i \tag{12}$$

とすればいい。

数値積分に使うには、あらかじめ $p_i(t)$ の積分を求めておいて、それと y_i を掛けて和をとるだけである。

もっとうまい方法はないか

等間隔でなく点をとることで精度をあげることはできないかというのが、ここでの問題である。

答だけ書くと:

「ガウス・ルジャンドルの多項式の零点を点にとると、次数を最大にできる」

ガウス・ルジャンドルの多項式は、有限区間で帰納的に定義できる直交関数系である。つまり、最初の項を定数として、k 番目の項は k 次多項式であって、$k-1$ 次の任意の多項式と直交するようになっている。

証明とかはばっさり略。

微分方程式への積分公式の適用

微分方程式の数値解法と数値積分はもちろん非常に良く似たものである。が、大きな違いは、数値積分では被積分関数が

$$y = f(x) \tag{13}$$

と、独立変数の陽な関数として与えられていたのに対し、微分方程式では

$$\frac{dx}{dt} = f(x,t) \tag{14}$$

と積分された結果自体に被積分関数がよっていることである。このために、話がすこし複雑になる。

簡単な例

中点公式 (これもガウス型公式) の場合。数値積分: 単に

$$S = hf((x_0 + x_1)/2) \tag{15}$$

微分方程式の場合、 $(t_0 + t_1)/2$ での x の値がわからない。

近似多項式に戻って考えるて、f を定数で近似し、その結果解である x を一次式で近似する、つまり

$$\hat{x}(t) = x_0 + f(\hat{x}(t_0 + h/2), t_0 + h/2)(t - t_0) \tag{16}$$

という近似をする。問題の中点での値を $x_{1/2}$ と書くと

$$x_{1/2} = x_0 + f(x_{1/2}, t_0 + h/2)h/2 \tag{17}$$

と書けるわけで、これは $x_{1/2}$ に対する代数方程式になっている。これを解けばいい。

一般の場合

一般には、n 次のルジャンドル多項式から導かれる公式では、f の $n-1$ 次近似多項式を構成することになる。これを前節と同様 L_{n-1} とし、零点を t_k とすれば、t_k での「解」 x_k が

$$x_k = x_0 + \int_{t_s}^{t_k} L_{n-1}(t)dt, \quad k = 1, 2, ...n \tag{18}$$

がなり立ち、さらに

$$L_{n-1}(t_k) = f(x_k, t_k), \quad k = 1, 2, ...n \tag{19}$$

がなり立つように x_k と L を決め、それを積分して t_e での解が求まるということになる。

と書くと非常に大変そうだが、係数をあらかじめ計算しておけば上の手続きが陰的ルンゲクッタ公式として表現できる。これで、例えば 4 段で 8 次の公式が実現出来るわけである。

代数方程式の扱い

さて、上のようなわけで陰的公式というものが構成できるということはわかったが、これを実際に使うには、 x_k に対する代数方程式をとかないといけない。この解き方について簡単に説明しておこう。

代数方程式といっても、ここで扱うものはほとんど必ず

$$x = f(x) \tag{20}$$

の形、すなわち、ある関数の不動点を求めるという形になっている。この時は、以下のような反復を繰り返すことによって解くという方法が考えられる。

$$x_{new} = f(x_{old}) \tag{21}$$

これを直接代入法という。

直接代入法の問題(？)

直接代入法は容易にプログラム出来て、実際に広く使われる方法である。が、理論的にはいろいろ怪しい。つまり、

- 収束するかどうか
- 収束の速さ

というような観点からみて使いものになるのかどうかというのは明らかではない。

しかし、陰的ルンゲクッタから、というか、一般に常微分方程式の数値解法から出てくる方程式の場合、これで結構うまくいくことが多い。

直接代入の収束の速さ

方程式 (20) の解が x であるとき、 直接代入法での古い近似解と新しい近似解それぞれの真の解との差は

$$\begin{aligned} e_{old} &= x_{old} - x \\ e_{new} &= x_{new} - x \end{aligned} \tag{22}$$

となる。下の方を変形し、f を展開すれば

$$e_{new} = f(x_{old}) - f(x) \sim \frac{df}{dx}(x_{old} - x) \tag{23}$$

となる。従って、

- 収束するためには $|df/dx| < 1$ であればよい。
- $|df/dx|$ が小さいほど速く収束する。

ということがわかる。

陰的ガウス法への応用

もっとも簡単な中点公式（陰的中点公式）で考える。解くべき式は、$\xi = h/2$ として

$$x = x_0 + f(x, t_0 + \xi)\xi \tag{24}$$

と書ける。リプシッツ連続条件

$$|f(t, x) - f(t, x')| \leq L|x - x'| \tag{25}$$

がなり立つとしてよいので、中点公式 (24) の右辺の x による微分が $L\xi$ で押えられるということがわかる。つまり、刻み幅を $1/L$ に比べて小さくとればいいということになる。

なお、これでは済まないような場合もある。(いわゆる「硬い」方程式)

線形多段階法

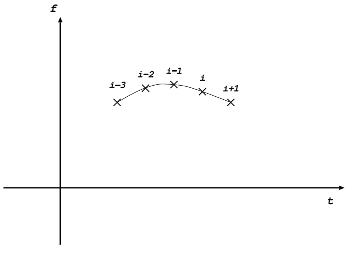
ルンゲ・クッタ法:「一段階法」t_i における解 x_i がわかっていれば、それだけから次の時刻 t_{i+1} での解 x_{i+1} が求まる。

多段法: t_i での情報だけでなく、その前のステップでの情報（導関数の値や解そのものの値）を記憶しておいて、それを使う。

これにより、余計な計算をしないで計算精度をあげたい。

アダムス法

原理: いくつかのステップでの導関数（微分方程式の右辺）f の値を憶えておいて、それを通る補間多項式を作り、それを積分して解を求める。



左の図に概念を示す。ここでは、積分公式の時と同様にラグランジュ補間（ニュートン補間）をして多項式を作る。で、その作った多項式を積分する。

アダムス法の実際

例えば、点 i から $i+1$ まで積分するのに、点 $i-p$ から i までの関数値を使うとすれば、p 次の多項式で

$$P(t_j) = f_j = f(x_j, t_j) \quad (i - p \leq j \leq i) \tag{26}$$

を満たすものを作る。で、$i+1$ での解 x_{i+1} は

$$x_{i+1} = x_i + \int_{t_i}^{t_{i+1}} P(t)dt \tag{27}$$

で与えられる。刻み h が定数であるとすれば、p を決めれば上の式を

$$x_{i+1} = x_i + h\sum_{l=0}^{p} a_{pj} f_{i-l} \tag{28}$$

の形に書き直せる。

2次の公式

簡単な例として、$p = 1$の場合を考えてみよう。この時、補間多項式は一次であって

$$P(t) = f_i - \frac{f_{i-1} - f_i}{h}(t - t_i) \tag{29}$$

となって、これを積分すれば、結局

$$x_{i+1} = x_i + \frac{h}{2}(3f_i - f_{i-1}) \tag{30}$$

となる。

一般に、アダムス法では任意段数の公式が構成でき、その次数は段数に等しいことがわかっている。これは、ルンゲ・クッタなどに比べればはるかによい性質をもっているということでもある。

出発公式

アダムス法はいくらでも高次の公式が作れ、計算量もあまり多くないということがわかっているが、必ずしも広く使われているというわけでもない。その理由はいろいろあるが、一つは、

「どうやって計算を始めるべきかよくわからない」

ということ。

初期値問題としてはもちろん t_0 におけるx_0しか知らない

多段階法ではその前の時刻での解が必要

なんらかの方法で与える:面倒

陰的アダムス法

陰的な補間多項式、つまり t_{i+1}での関数の値を使った公式を考える。

刻み hが定数であるとすれば、pを決めれば前と同様に

$$x_{i+1} = x_i + h\sum_{l=-1}^{p-1} b_{pj} f_{i-l} \tag{31}$$

の形に書けることになる。また手をぬいて $p = 1$の場合を考えれば、これは単に台形公式

$$x_{i+1} = x_i + \frac{h}{2}(f_i + f_{i+1}) \tag{32}$$

となる。

代数方程式の解きかた

通常の方法

- 初期値として同じ次数の陽的アダムス法の解を持ちいる。
- そのあとは直接代入法で反復する。

というものである。

なお、このやり方を、予測子・修正子法と呼ぶ。線形多段階法はほとんどこの形で使われるため、線形多段階法のことをさして予測子・修正子法と呼ぶ人もいる。

反復回数

収束まで繰り返す必要はないことに注意。

例えば、予測-加速度計算-修正(PEC) とするだけでも、予測-加速度計算(PE) だけよりも精度が上がる。それ以上反復を繰り返しても大して精度は上がらない。

場合によっては P(EC)^2 や PE(CE)^2 といった方法も使われる。

時間刻みを変える時

時間刻み一定:あらかじめ係数を求めておいて、前の関数値の線形結合で計算できる。

刻みを変えたい場合:

もっとも一般的で強力な方法は、補間多項式を計算し、さらにそれを現在の時間刻みで積分した形の係数を求める、すなわち、毎回積分公式を作り直すという方法。

大変そうに聞こえるけどそれほどでもない。細かい話は省略。

計算中の誤差評価

誤差評価はルンゲ・クッタに比べて容易。

予測子・修正子型の公式の場合、この2つの間の差がかなり良い誤差の推定値（もちろん大きめだが）になっているので、これを使ってやればいい。

ルンゲ・クッタとの比較

必要な精度を決めた時、ほとんどどんな場合でも、線形多段階法のほうがルンゲ・クッタ法よりも少ない計算量で答を得ることが出来る

(実験的にも、また誤差の係数をつかった理論的な解析からも)

ルンゲ・クッタ公式が優れているのは、基本的にはその使いやすさにおいてのみ

使いやすさというのは実際的にはなかなか重要

2階の方程式向けの方法

ハミルトン系のような2階の方程式の場合:

アダムス型公式をそのまま使うのは賢くない

アダムス法の場合、加速度に対する補間多項式を2回積分すれば位置の変化が求まる。

速度についての補間多項式を構成したり、速度の昔の情報をとっておく必要はない。

必要なメモリ、計算量の両方が減る。

エルミート公式

良く研究されている解法:

微分方程式を与えられたものと考え、その右辺値、つまり導関数だけを使って積分公式を構成する。

ある意味素直な方法:高次の導関数も使う。

どんどん次数をあげると計算量が指数関数的に上昇するのであまり現実的な方法ではない

次の導関数くらいならそれほど大変ではない

重力の場合

2体間の重力

$$f_{ij} = -Gm_j \frac{r_i - r_j}{r_{ij}^3} \tag{33}$$

なら、その一階導関数は

$$j_{ij} = -Gm_j \left[\frac{\mathrm{v}_{ij}}{r_{ij}^3} - \frac{3(\mathrm{v}_{ij} \cdot \mathrm{r}_{ij})\mathrm{r}_{ij}}{r_{ij}^5} \right], \tag{34}$$

である。ここで

$$r_{ij} = x_j - x_i, \tag{35}$$

$$v_{ij} = v_j - v_i. \tag{36}$$

であり、 x_i、v_i はそれぞれ粒子 i の位置、速度、G は重力定数、m_j は粒子 j の質量である。

一階導関数の利用

アダムス法を拡張して、補間多項式を導関数も使って構成

この型の補間自体: エルミート補間

積分公式は、人によってエルミート型、あるいはオブレヒホフ型の公式と呼ばれる。

利点:段数に対して達成出来る多項式の次数が2 倍。

特に、実質的には1段法である2段の陰的公式で4次精度が出来る。

さらに、予測子を加速度の導関数まででとめてしまえば予測子も1段法になって実装が簡単。(出発公式不要)

エルミート公式:予測子

予測子は単に以下のようなテイラー展開で与えられる。

$$x_{\rm p} = \frac{\Delta t^3}{6}\dot{a}_0 + \frac{\Delta t^2}{2}a_0 + \Delta t v_0 + x_0 \tag{37}$$

$$v_{\rm p} = \frac{\Delta t^2}{2}\dot{a}_0 + \Delta t a_0 + v_0, \tag{38}$$

ここで $x_{\rm p}$ と $v_{\rm p}$ はそれぞれ位置と速度の予測子であり、x_0, v_0, a_0 および $\dot{a}_0$ は前の時刻での位置、速度、加速度と加速度の一階時間導関数である。Δt は時間刻み $t_1 - t_0$ である。

エルミート公式:修正子

$$x_{\rm c} = x_0 + \frac{\Delta t}{2}(v_{\rm c} + v_0) - \frac{\Delta t^2}{12}(a_1 - a_0), \tag{39}$$

$$v_{\rm c} = v_0 + \frac{\Delta t}{2}(a_1 + a_0) - \frac{\Delta t^2}{12}(\dot{a}_1 - \dot{a}_0). \tag{40}$$

となって非常に簡単に4次精度の公式を作れる。

重力多体系の計算では割合一般的に使われている (と思う)。

外挿法

の話もしようかと思ったけど、とりあえず多体計算ではまず使わないので省略。

独立時間刻み

重力多体問題: 原理的には単に大規模な常微分方程式の初期値問題

ナイーブに考えると、今まで議論してきたいろんな公式がライブラリであるので、それを使えば済みそうな気がする。

それだけで済むならこの講義の半分はいらない。

済まない理由:

- 粒子によって非常に大きく軌道のタイムスケールが違うことがある
- 連星とかそういったものができる

軌道タイムスケールの問題

- 構造形成による効果
- 一様な系でも起きる問題

構造形成による効果

自己相似的なコラプス:

$$\begin{aligned} \rho &\propto r^{-\alpha} \\ \sigma &\propto r^{(2-\alpha)/2} \end{aligned} \tag{41}$$

軌道タイムスケール: $1/\sqrt{G\rho} \sim r^{\alpha/2}$

コアが収縮するにつれてその力学的タイムスケールはどんどん短くなる。

一様な系でも起きる問題

重力が引力であるために確率的には2つの粒子がひじょうに近付くような近接散乱が起こる

インパクトパラメータが 0 に近い2体衝突: 非常に短い時間刻みが必要

これは重力多体系特有の問題: 相互作用が引力で、しかも距離 0 で発散するため。

例えば分子動力学計算ではこういう問題はおこらない。

計算量への影響

単純な可変時間刻みでは計算量が大きくなる。

理由:
どちらの場合も、タイムステップの分布がべき乗的なテイルをもつようになる。

このため、粒子数が増えるに従って、タイムステップが短くなる。

構造形成の効果: 最悪 $O(N)$

2体衝突の効果: $O(N^{1/3})$ 程度

対応

- 粒子毎に時間刻みをバラバラに変化させる。(独立時間刻み)
- 2体衝突、連星は座標変換して扱う。

例

10 体問題の例 — デモ計算

独立時間刻みの原理

粒子毎にばらばらの時刻 t_i と時間ステップ Δt_i を与える

1. $t_i + \Delta t_i$ が最小の粒子を選ぶ。
2. その粒子の軌道を新しい時刻まで積分する。
3. その粒子の新しい時間刻みを決める。
4. ステップ 1 に戻る。

問題:ある粒子の時刻 $t_i + \Delta t_i$ で他の粒子の位置が必要

他の粒子の位置の計算

時間刻み可変の「予測子」を使えば問題ない

つまり、

- 各粒子の時間積分公式としては、可変ステップの線形多段階法を PEC モードで使う。
- ある粒子の新しい時刻での加速度を計算するには、他の粒子のその時刻での位置を予測子を使って予測する。

ということになる。

時間刻み自体の決定

伝統的には、以下のなんだか意味が良くわからない形の式が使われている。

$$\boldsymbol{\Delta t} = \sqrt{\eta \frac{|a_1||a_1^{(2)}| + |\dot{a}_1|^2}{|\dot{a}_1||a_1^{(3)}| + |a_1^{(2)}|^2}} \qquad (42)$$

これは、どういう理屈があって決まるのかといわれても困るような代物だが、経験的には非常にうまく動くことが知られている。以下、「うまく」動くとはどういうことかをもうちょっと考える。

タイムステップをどうやって決めるか？

時間ステップをどのように決めるべきかというのは結構良くわからない問題

普通の話:

位置なり速度の誤差の推定

→ 誤差をある値以下になるように調整

どうやって誤差を測るか？

- 絶対誤差
- 相対誤差

空間構造が発達し、速度に幅ができるとどちらも今一つ

誤差の測りかたが明確な例

太陽系の惑星の運動の長時間積分:

最終的に問題になるのは例えば 10 億年積分したあとでのそれぞれの惑星の位置そのもの

宇宙の構造形成:

構造は自由落下のタイムスケールで出来る

つまり: 共動座標でみて粒子はあまり動いていない

位置、速度そのものを見ていい。

良くわからない例

大抵の場合はそう。

シミュレーションの最終状態でのそれぞれの粒子の位置は原理的にほとんど意味を持たない。

理由:

各粒子の軌道そのものは、リヤプノフ指数が正。

時間スケールが力学的タイムスケール (典型的な軌道周期) の数分の一程度であるという意味でかなり強くカオス的であるからである。

カオス的って?

定義は難しい、、、

とりあえず、

ある N 体系と、それと同じだけど粒子一つだけほんの少し違う初期条件にあるもう一つの系で、時間積分を進めていく時にその対応する粒子の軌道が時間の指数関数で離れていく

ということをさす。

まあ、 系の $6N$ 次元の位相空間での2つの軌道が、その距離が小さいうちは指数関数的に離れていくといういいかたをしてもおなじ。

タイムスケール

距離が e 倍になるのが軌道周期の数分の1くらい。

粒子数に弱く依存 ($\log N$ という説と $\log\log N$ という説あり)

これは、一見致命的

有効数字が軌道周期あたり1桁程度減っていく

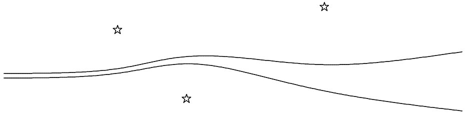
10 周期回ると誤差が 100億倍に増幅される

何故ずれが成長するか

難しげにいうと:$\varGamma$ 空間（$6N$ 次元位相空間）での軌道が双曲的であるから

(指数関数的に広がるというのの言い替え) でしかない。

具体的にミクロスコピックに見る:



他の粒子と近接遭遇した時に軌道間の距離が拡大される

(本当は1次ではなく2次の効果が効く)

指数的成長

というわけで、初期条件が少しでも違うと、必ず違いは指数関数的に大きくなる

これは、力学系自体の性質:

どれほど数値積分を正確に行なっても、やはり広がる

つまり: 多体シミュレーションでは必ず個々の粒子の位置は「正しくない」

指数的成長と熱力学的緩和

軌道が広がること = 初期条件を忘れること

と思うと、指数的成長のタイムスケールと緩和のタイムスケールに関係があるような気がする。そういう主張の論文も沢山ある (Gurzadian and Savvidy 1986 とか)

でも、それは気のせいなので、以下、そういう話。

復習:タイムスケール

指数的成長のタイムスケール: 力学的 $t_d \sim 1/\sqrt{G\rho}$

緩和: $t_r \sim t_d N/\log N$

緩和時間はずーーーーっと長い。

メカニズム

指数関数的成長: 近くの粒子との相互作用が効く (距離 $R/\sqrt{N}$ くらいのもの)

緩和: R/N から R までの全ての粒子

ポテンシャルの形を少し変えると、緩和時間はあまり変えないで指数的成長は遅くするとかできる。

そうはいっても、なにか関係ないのか？という気も、、、

指数的成長と緩和

理想化された問題として、例によって一様 (ランダム) に粒子が分布 (止まっている) する中を動くテスト粒子 (の集団) を考える。

熱力学的緩和: 集団がある程度大きく広がる

指数的成長: 実は、大きさが $R/\sqrt{N}$ くらいで「非常に遅く」なる:散乱がコヒーレントでなくなるため。

コヒーレントでない: (大雑把にいうと)「ランダム」な変化=拡散=緩和

つまり、熱力学的な緩和は、指数的成長の先にあるものということになる。

注意

とはいえ、「粒子の位置がどこかわからなくなる」のはやはりその通り。

で、これは数値計算の精度の問題ではなく、物理系自体の性質。

「数値計算で何が正しく求まるか」という問題は依然ある。

なにを「正しく」できるか

優等生な答:
個々の粒子の位置には左右されないような統計的な性質

今までの話では:
重力熱力学的カタストロフィー
自己相似的進化
といったもの。

統計的な性質って?

なにが統計的な性質でなにがそうでないかというのはそんなに自明ではない。

- 実際の系でも系を構成している粒子の数はそれほど多くはない
- 粒子数が小さなサブシステムが勝手にできてきて、それがどうなるかが系の進化を決めることがある

例:重力熱力学的振動

収縮から膨張へのスイッチングを起こすのはコアでの連星系からのエネルギー供給

連星系の形成: 3 体衝突による (高い密度が必要)

自己相似的成長: 密度が高い $=$ コアの質量 (粒子数) 小さい
連星ができる時のコア粒子数: ≤ 100
同時に存在する連星の数: 数個
連星系からのエネルギー供給は非常に大きく揺らぐ

重力熱力学的振動の「連続極限」

エネルギー供給に揺らぎがあっても膨張・収縮は熱力学的に起こるのであるから、多少の揺らぎはそのなかで吸収されるのではないか?

実際は逆

逆にエネルギー供給を決定論的に与えた流体計算でも振動はカオス的

決定論的カオス $+$ small-N ゆらぎ
これが「正しい」かどうかどうやって分かるか?

本質的には、自由度が小さい系でのカオスと同じ問題。

実際的にはどうしているか

- エネルギー保存をみる
- 常識的な範囲ならいいことにする

常識的な範囲、、、、、って、、、、
結局:
「みたい答が計算精度に依存しないところまで精度をあげる」
ということが必要。

時間刻みの決定

時間刻みに話を戻す

- 軌道の時間積分の誤差をみてそれを評価したい
- パラメータは「無次元」量であってほしい。(例えば系が膨張したらパラメータを変える必要があるようでは困る)
- ガリレイ変換に対して不変でないと困る

重力加速度と、その高次の時間導関数だけを使って構成され、時間の次元を持つような量
だといい。

実際に使うには

例えば
予測子と修正子の差の絶対値を加速度の絶対値で割ったもの
でもいいが、特別な場合にうまくいかないことがある。
前に書いた妙な式は、4 次精度の公式では割合うまくいく。

と、これで時間刻みの話は一応終わり。

特異点と正則化

重力ポテンシャルは距離の逆数つまり $1/r$ で、 $r \rightarrow 0$ で発散する。しかも引力

質点系のシミュレーションでは2粒子間のポテンシャルが実際に発散し得る。
対策:

- 多少物理に影響するかもしれないがポテンシャルをいじって発散をなくす
- 数値計算法を工夫することで対応

ポテンシャル・ソフトニング

r が小さいところで重力の形を変えてしまって、発散が起きないようにする。
物理には影響しないか?という問題
「問題によっては」十分に影響範囲が小さいなら大丈夫。
もっとも広く使われているのは

$$\phi_{\rm soft} = -\frac{1}{\sqrt{r^2+\epsilon^2}} \tag{43}$$

プラマーモデル ($n = 5$ のポリトロープ) のポテンシャルに対応

2体緩和への影響

- 2体緩和は、近接散乱よりも主に遠くの粒子との相互作用が効く
- 問題によってはそもそも2体緩和自体正しく表現する必要はない、
- むしろ積極的に押えられるものならば押えたい。

例えば無衝突系とみなせるものを少なく粒子数で表現する時

プラマーソフトニングの性質

利点:

- 滑らか (無限回微分可能)
- 計算が速い

みかけの欠点 — 収束が遅い

収束が速いもののほうが精度が高いかどうかは自明ではない問題。

ソフトニングと時間刻み

ソフトニング ϵ が R/N より十分大きい
↓
タイムステップに実質的に下限ができる

この時、単独の2体散乱では軌道はほとんど変わらない
→ 時間刻みは ϵ/v の数分の1程度にとってよいのは明らか
本当はもっと長くてもいい: 加速度の時間変化の高周波成分をいい加減に積分してもその影響は小さい。
但し、時間刻みを可変にすると高周波成分の影響を受ける。長くとれるのは一定刻みの場合のみ。

ポテンシャルをいじらない場合

ソフトニングが使えない場合:例えば

- 連星形成まで扱う
- 物理的な衝突を扱う

2粒子が非常に近付いた時でもうまく計算できるような方法が必要

原理的には “straightforward”: 2体が十分近づいたら、解析解(またはそれ+摂動)でおきかえればいい。

具体的には

- 2つの粒子が近付いたかどうかをチェックする
- 近付いていたら、その2粒子の重心および相対位置（内部運動）を新しい変数として時間積分する
- 相対運動に対して、回りの粒子からの摂動が十分に小さければ数値積分しないで解析的に解く
- 2つの粒子が十分離れたらその重心と相対位置から元の粒子を復元する

正則化

正則化の基本的な考えは、座標と時間を適当に変換して数値解の性質をよくしようというもの

2次元の例。 2次元座標$\mathrm{r} = (x, y)$を、

$$x = u_1^2 - u_2^2 \tag{44}$$

$$y = 2u_1u_2 \tag{45}$$

という関係を満たす新しい変数$\mathrm{u} = (u_1, u_2)$に変換し、さらに独立変数を時間tから $rds = dt$ という変数変換で得られる変数sに置き換える。

正則化された方程式

すると2体問題の運動方程式

$$\frac{d^2\mathrm{r}}{dt^2} = -\frac{\mathrm{r}}{r^3} \tag{46}$$

が単振動の方程式

$$\frac{d^2\mathrm{u}}{ds^2} = \frac{E}{2}\mathrm{u} \tag{47}$$

に変換される。ここでEはシステムの全エネルギー。

これは、軌道角運動量が 0 の直線解も扱える。

解析解でつなぐ必要はない。

歴史

Levi Civita (1905) が2次元の変換を提案

3次元で摂動がある場合に拡張: Kustaanheimo and Stiefel (1965)

2次元の場合の本質: 位置ベクトルを複素数とみなしてその平方根をとる

Kustaanheimo and Stiefel はスピノルを使って4次元に軌道を拡張

ハミルトンの4元数を使っても書けるということが数年前に示された。

連星と計算量

連星の軌道周期: 短い

連星の熱力学: ほぼ「等温」のバックグラウンドに埋めこまれた自己重力系

一旦周りよりも「温度」が上がると、周りに熱を与えて無制限に温度が上がる。

(3体相互作用作用でエネルギーを与える)

周りと温度が等しい $= a \sim R/N$

軌道周期が他の星の $1/N$

軌道周期が他の星の $1/1000N$ くらいまでは系内にいられる。

こんなのを計算してたら日が暮れるどころではない。

連星の扱い

十分コンパクトな連星で、回りの粒子からの摂動が無視できるほど小さい場合: 要するに単に摂動を無視すればよい

どれくらい摂動が小さいなら無視していいか? —目的による
エネルギー: 断熱不変量
角運動量: そうではない

角運動量について正確な結果を得るのは現状では難しい

弱い摂動の扱い

まともにやるなら、軌道要素で表して、外力を軌道平均して、、、

画期的に安直な方法

- 摂動論的扱いができる範囲内で連星の軌道周期を遅くする
- 摂動項は、実時間での軌道要素の変化率が同じになるようにスケールして運動方程式に入れる。

摂動論の範囲内では摂動論と同じ結果を、軌道を直接数値積分して得ることができる。
(平均運動共鳴は扱えない)

うまくいかないケース

階層的な3重星: 連星の外側をもう一つ星が回っているようなもの

- 安定条件は数値的にしかもとまっていない。
- 共鳴が効くので時間のスケーリングはできない

いろいろ研究中

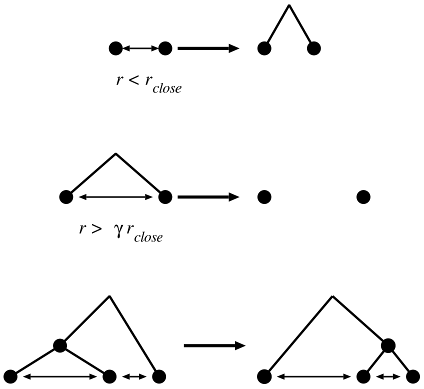
2体以上の系の扱い

例えば3体の系を重心運動と内部運動に分離するにはどうするか？

- 「3体」として認識、それ用の正則化
- 2体+1体としては認識、再帰的に扱う

今のところ広く使われているコードは前者
後者のコードもそれなりに動くようになってきている

構造の組み換え



積分公式再考

連星、階層3重星のような長時間積分が必要な系: 特殊な公式を使ってでも計算時間を短くしたい。そういうものはないか?

- シンプレクティック型公式
- 対称型公式

ハミルトン系/周期軌道に対して特別に良い性質をもつ

「良い」公式の古典: 2 次 leap frog

昔からいろんな業界で使われていて、名前が違う、、、

- leapfrog
- Verlet
- 2nd order ABM
- 2nd order Stömer-Cowell

他にも4個くらい名前があったような。

別の表現

$$x_{i+1} = x_i + \Delta t v_i + \Delta t^2 a(x_i)/2 \quad (52)$$

$$v_{i+1} = v_i + \Delta t[a(x_i) + a(x_{i+1})]/2 \quad (53)$$

こう書くと PC法みたいに見える。また、以下のようにも書ける

$$v_{i+1/2} = v_i + \Delta t a(x_i)/2 \quad (54)$$

$$x_{i+1} = x_i + \Delta t v_{i+1/2} \quad (55)$$

$$v_{i+1} = v_{i+1/2} + \Delta t a(x_{i+1})/2 \quad (56)$$

これは一見なんだか良くわからない。さらにまた、速度を消去して x_{i-1}, x_i, x_{i+1} の関係式の形で書いてあることもあるかもしれない。

シンプレクティック公式の性質

1. シンプレクティック公式は、すくなくともある種のハミルトニアンに対して使った場合に、それに近い別のハミルトン系に対する厳密解を与えることがある。
2. 周期解を持つハミルトン系に対して使った場合に、どんな量でも誤差が最悪で時間に比例してしか増えない。
3. 時間刻を変えると上のようなことは成り立たなくなる

leap frog 公式

$$v_{i+1/2} = v_{i-1/2} + \Delta t a(x_i) \quad (48)$$

$$x_{i+1} = x_i + \Delta t v_{i+1/2} \quad (49)$$

ここで、添字はステップである。 ±1/2 は中間点での値ということになる。

出発用公式として

$$v_{1/2} = v + \Delta t a(x_0)/2 \quad (50)$$

を使い、さらに終了用公式として

$$v_i = v_{i-1/2} + \Delta t a(x_i)/2 \quad (51)$$

を使う

leap frog の性質

普通にいう2次精度である。

局所誤差という観点からはこれは決して良い公式というわけではないが、現実にはこの公式は非常に広く使われている。

実際、非常に「良い」例えばエネルギーや角運動量のような保存量が非常によく保存するということである。これらの保存量については多くの場合に誤差がある程度以上増えない。

1. リープフロッグはシンプレクティック公式 のもっとも簡単なものの一つである。
2. リープフロッグは対称型公式 のもっとも簡単なものの一つである。

対称型公式の性質

1. 周期解を持つ時間対称な系に対して使った場合に、どんな量でも誤差が最悪で時間に比例してしか増えない。
2. 時間刻を変えてもうまくいくようにすることも出来る。

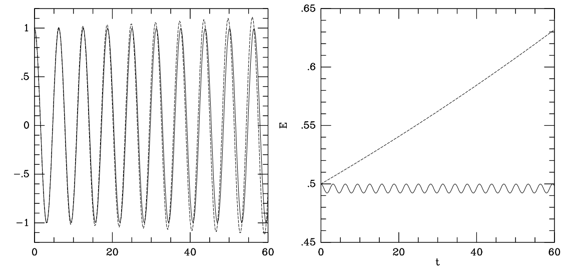
数値例

調和振動子

$$\frac{d^2x}{dt^2} = -x \tag{57}$$

を リープフロッグ と 2階のルンゲクッタで解いてみる。初期条件は $(x, v) = (1, 0)$ で、時間刻みは $1/4$ である。

軌道とエネルギー



保存量

この調和振動子の場合には、リープフロッグ公式は以下の量

$$H' = \frac{1}{2}(x^2 + v^2) - \frac{h^2}{4}x^2 \tag{58}$$

を保存する。

つまり、微妙に浅いポテンシャルでの調和振動子になっている。

$h \geq 2$ になるとポテンシャルが下に凸でなくなる。

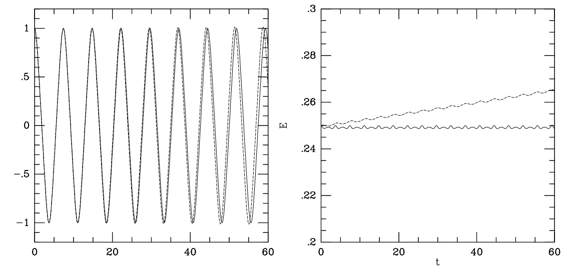
これは、安定でなくなる (差分方程式の固有の絶対値が 1 でなくなる) のと一致している。

非線形振動

$$\frac{d^2x}{dt^2} = -x^3 \tag{59}$$

をリープフロッグと 2階のルンゲクッタで解いてみる。初期条件は $(x, v) = (1, 0)$ で、時間刻みは $1/8$ である。

軌道とエネルギー



シンプレクティック公式

リープフロッグ 公式は、良いけど低次。

もっと高次の方法は?

そういうのを作る一つのアプローチ:シンプレクティック公式

シンプレクティック写像:要するに正準変換

リープフロッグ公式はシンプレクティック性を満たしている

これを時間刻みを変えて組合せたものもシンプレクティック性を満たしている

陽解法

吉田や鈴木による作用素分解に基づく公式が良く知られていて、広く使われている。

これらの方法の原理:

- リープフロッグをタイムステップを変えていくつか組合わせる。
- うまくタイムステップを組み合わせると誤差の高次の項を消すことができる

7段6次や15段8次の公式等が吉田によって導かれている。

陽解法はハミルトニアンが $T(p) + V(q)$ の形の場合にしか使えない

陰解法

遠い昔にでてきた陰的ガウスはシンプレクティック。

シンプレクティック公式の意味

シンプレクティックであるということと、「良い」ということの間の関係はなにか?

あるハミルトニアン H で表される系をシンプレクティックな p 次の公式を使って積分した解は、別のハミルトニアン H' で与えられるシステムの厳密解になっていて、H と H' の間に

$$H = H' + h^{p+1}H_p + O(h^{p+2}) \tag{60}$$

という関係がある（H' を求める数列が収束すれば）ということがわかっている。

(収束するかどうかは一般には明らかではない)

シンプレクティック公式と可変時間刻み

適当に時間刻みを変えると上手く働いてくれない(エネルギーとか保存しなくなる)

以下のような考えかたでの研究が進められている。ハミルトニアン H を

$$H = H_1 + H_2 + \cdots \tag{61}$$

と複数の項の和にわけ、それぞれについて違う時間刻みを与えることで実効的に時間刻みを変える。

局所誤差

シンプレクティック公式は1段法:局所誤差に対する計算量という観点からは線形多段階法に比べて必ず悪い

普通の(誤差を小さくした)ルンゲ・クッタに比べても驚くほど悪い

例:8 次の公式

15段の吉田による公式の誤差の係数は、2階の問題に最適化された非シンプレクティックな Dormand らによる9段公式（実質 8 段 8 次）の 10^4 倍

同じ計算時間での局所誤差は100万倍

局所誤差の小さい公式

陰的ガウスはいろいろな意味で「最適」な公式であり、局所誤差も極めて小さい。

直接代入で収束させても、他の方法より速いという実験結果もある。

MVS公式等

「ハミルトニアンの分離」の考え方を、主要なハミルトニアン+摂動の系 (例えば摂動があるケプラー問題) に適用する。

リープフロッグとの基本的な考え:

$$H = T(p) + V(q) \tag{62}$$

に対して、pに対する以下の変換

$$p \leftarrow p - \Delta t \frac{\partial H}{\partial q} \tag{63}$$

と q に対する同様な変換がそれぞれシンプレクティックなので、それらを順番に適用したものもシンプレクティックになるというもの。

摂動ケプラー問題

摂動をうけたケプラー問題のハミルトニアン:

$$H = T(p) + V_1(q) + V_2(q) \tag{64}$$

惑星系なら V_1 が太陽からの重力で、 V_2 は自分以外の惑星からの寄与。

このとき

$$H_1 = T(p) + V_1(q) \tag{65}$$

の解はそれ自体シンプレクティックであり、また V_2 だけを考えた

$$p \leftarrow p - \Delta t \frac{\partial V_2}{\partial q} \tag{66}$$

というマッピングもシンプレクティックである。従って、この2つを組み合わせて積分公式をつくることができる。

解釈

リープフロッグ: $x \leftarrow x + \Delta t v$ と直線で動かす

MVS: ポテンシャル V_1 に沿って動かす

ケプラー問題は解析的に解けるので、このようなやり方で太陽中心の軌道を極めて高精度に積分できる。

実用的にこの方法を使うためには、ケプラー問題を高速に解く方法が必要になる。これは、実空間でまともに解く方法と、KS 変換した座標系で解く方法のそれぞれで極めて高速に解ける方法が提案されている。

対称型公式

(ここでの) 時間反転に対する対称性とは、常微分方程式

$$\frac{dy}{dx} = f(x, y) \tag{67}$$

に対する一段法

$$y_{i+1} = F(x_i, y_i, f, \Delta x) \tag{68}$$

が、

$$y_i = F(x_{i+1}, y_{i+1}, -f, \Delta x) \tag{69}$$

を満たすこと、つまり、直観的には、ある微分方程式系があって、それを1ステップ分数値的に軌道を進めたとする。で、そこから逆に戻ると厳密に元の値に戻るということ。

具体例

前進オイラー法は対称型ではない。

その逆写像である後退オイラー法も対称型ではない。

台形公式

$$y_{i+1} = y_i + \frac{h}{2}(f_i + f_{i+1}) \tag{70}$$

は対称。

対称形公式の利点

- ハミルトン系に対して対称型公式を使った場合、エネルギーや角運動量などの保存量の誤差がある範囲内にとどまる
- 時間刻みを変えても上の性質を保つことが出来る

前のほうの性質はシンプレクティック公式と同じであるが、後の方の性質はシンプレクティック公式よりもある意味でよいものである。

陰公式か陽公式か

一階の方程式に対する一段法という制限をつけた場合:対称な公式はかならず陰的公式

陰的ガウス法は対称型公式でもある。
一階の系に対する通常の一段法では、これよりよい方法は多分ない。それでは、

- ハミルトン系専用の解法ではどうだろうか
- 多段法ではどうなっているのだろう
- それ以外の方法はないのだろうか

以下、順番に考えていくことにする。

ハミルトン系用の陽的対称型RK公式

リープフロッグ公式は対称型でありしかも陽公式である。で、リープフロッグの組合わせで作られる公式も、実はすべて対称型になっている。

対称型線形多段法

一段法という枠を外してみる=線型多段法を考える。
線形多段階法の対称性を以下のように定義しておく。
2階の方程式用の線形多段階法の一般形:

$$\boldsymbol{\alpha_0 x_{i+p} + \cdots \alpha_p x_i = h^2(\beta_0 f_{i+p} + \cdots \beta_p f_i)} \tag{71}$$

$\beta_0 = 0$ なら陽的、そうでなければ陰的公式
対称型であるとは、係数が

$$\boldsymbol{\alpha_i = \alpha_{p-i}}, \quad \boldsymbol{\beta_i = \beta_{p-i}} \tag{72}$$

を満たすということ。
リープフロッグをこの形にしたものは、

$$\boldsymbol{x_{i+1} - 2x_i + x_{i-1} = h^2 f_i} \tag{73}$$

注意

係数が対称という意味で対称な公式は、実際に時間反転に対して対称といえる。
$(x_i, ...x_{i+p-1})$ から上の式で x_{i+p} を求めたとする。時間を逆転させた解を考えて、$(x_{i+p}, ...x_{i+1})$ から x_i を求めるということを考えても、使う式が時間が逆転していないものと同じになっているのである。

歴史

公式が存在することは 1970 年代から知られていた
注目されるようになったのはシンプレクティック法の研究が盛んになった 1990 年代に入ってから。特にトロント大の Quinlan と Tremaine が14次までの対称型公式を導いた。
さらに最近になって、国立天文台の福島によって、Quinlan & Tremaine によるものよりもはるかに良い性質を持つものが導かれている。

問題点

これらの公式に共通する欠陥: 6次以上の公式は線形解析では安定であっても、非線形問題(例えば、単純なケプラー問題)に適用すると共鳴により不安定な振動を起こす。

このために6次以上の公式は現在のところ実用になっていない。

時間刻みの対称化

対称型公式でも、普通に時間刻みを変えると上手くいかない。

普通に (適当に) 時間刻みを変える

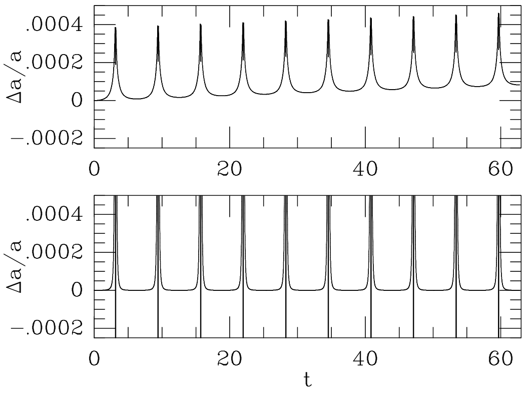
↓

行きと帰りでステップサイズが変わる

↓

元に戻らない

例—ケプラー問題、$e = 0.9$



時間刻み対称化

時間刻みが対称でないというのが永年誤差がでる原因だとすれば、対称にしてしまえば永年誤差がなくなるかも知れない。

時刻 t_0 で計算した時間刻み Δt_0 と 逆に戻る時にt_1 で計算した時間刻み Δt_1 が等しくなるようにする。

一つの可能性は以下のようなもの。

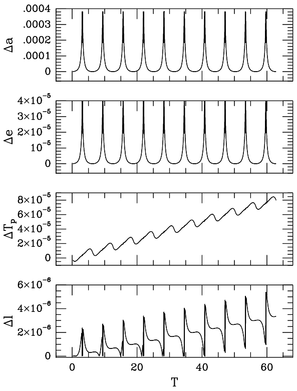
$$\Delta t = \frac{1}{2}[h(\xi_0) + h(\xi_1)] \tag{74}$$

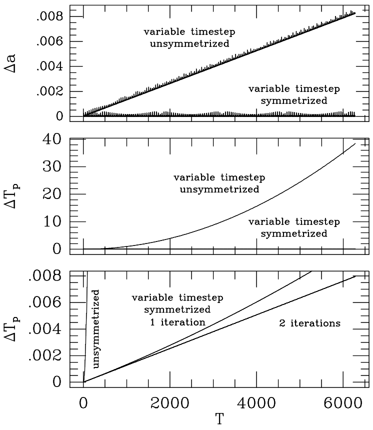
ここで $h(\xi)$ は適当な時間刻みを与える関数

積分の始点と終点で時間刻みをそれぞれ計算して、その平均を取る

ξ_1 は Δt の関数であるので、これは時間刻み自体が陰的に決まるということ。

数値実験





理論的にわかっていること

Cano と **Sanz-Serna** は、周期運動するハミルトン系について、

(a) 時間刻み一定のシンプレクテック公式,

(b) 時間ステップ対称化した対称型公式、

(c) ハミルトニアンを保存するように構成された公式

のどれでも、任意の量の誤差が最大でも時間の１次でしか増大しないということを証明した。

これは時間刻み対称化にある程度の理論的裏付けを与える。

他の重要かもしれない方法

Mikkola and Aarseth の最近のいくつかの論文では、正則化と実質同じ効果をより簡単に実現するいくつかの方法が提案されている。

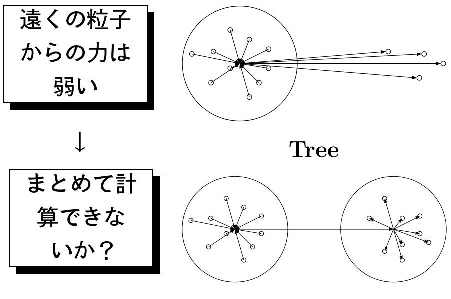
計算法 — 空間領域

運動方程式の右辺をどうやって評価するか？という問題。

以下、 独立時間刻みのことはとりあえず棚上げにして話をすすめる。
(後で2つの組合せについては議論する)

広く使われている方法： Barnes-Hut treecode
有名な方法： 高速多重極法

ツリー法、FMMの基本的発想



遠くの粒子からの力は弱い

↓

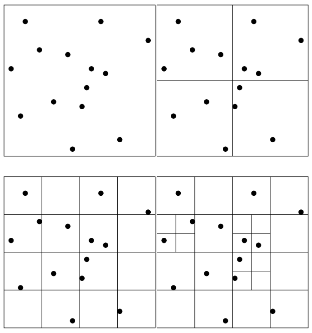
まとめて計算できないか？

Tree

FMM

- ツリー：力を及ぼすほうだけをまとめて評価
- FMM：力を受けるほうもまとめて評価

どうやってまとめるか? — ツリー法の場合

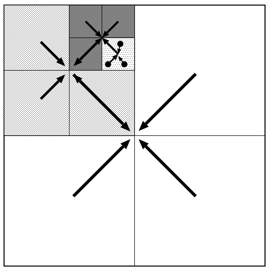


階層的なツリー構造を使う。

- まず、全体が入るセルを作る
- それを再帰的に 8 (2次元なら4) 分割する
- 中の粒子がある数以下になったら止める（上の例では1個）

多重極展開の構成

まず、ツリーの各セルのなかの粒子がつくるポテンシャルの多重極展開を計算する。

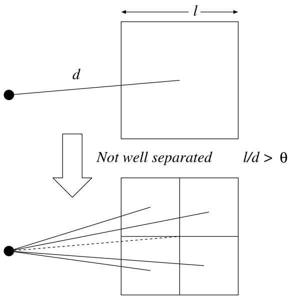


- 最下層のセル:そのなかの粒子が作るポテンシャルを多重極展開
- それ以外:子セルの多重極展開の展開中心をシフトして加算

下から順に計算していけばよい。
計算量は $O(N)$。展開をシフトする式はかなり複雑。

ツリー法での力の計算

再帰的な形に表現すると格好がいい。



- 十分に離れている:重心（あるいは多重極展開）からの力
- そうでない：子ノードからの力の合計

系全体からの力 ＝ ルートからの力

ツリー法の計算量と計算精度

簡単なオーダー見積り:

$$誤差 \propto \theta^{(p+1)}$$
$$計算量 \propto \theta^{-3}p^2N\log N$$

p:多重極展開の次数

$\log N$: ツリーの階層数
θ^3 ある階層で、相互作用を計算するセルの数

実際には、、、

実際の振舞い：複雑。

- 精度は上の見積りよりよい（θ のべきが大きい）
- 計算量はそれほど増えない

ことが多い。（粒子の分布による）

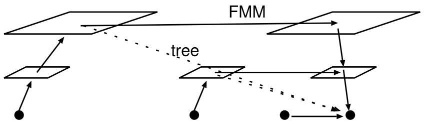
高速多重極展開法(FMM)

- ツリー：力を及ぼすほうだけをまとめて評価
- FMM：力を受けるほうもまとめて評価

ツリーに比べたメリット：

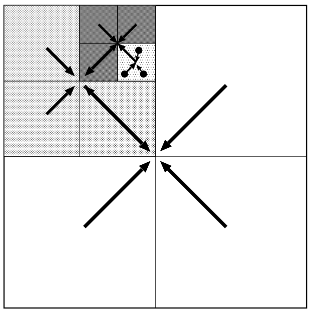
計算量が $O(N)$ になる！

粒子間相互作用の計算されかた



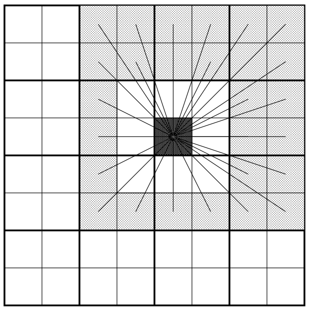
- 近くの粒子は直接計算
- 遠くにいくほど、上のセルを通って相互作用が計算される。

FMMの計算手順 - (1) M2M



各セルが外に作るポテンシャルの球面調和関数展開（多重極展開）を、自分の下のセルの多重極展開の展開中心をシフトして求める。(M2M)
(ここはツリーと同じ)

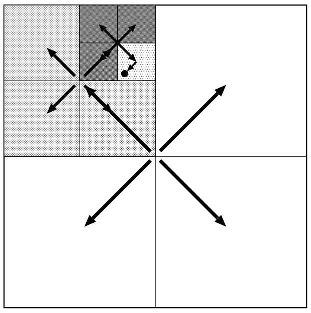
FMMの計算手順 - (2) M2L



- 自分と同じレベルで
- 親の隣接セルの子で
- 自分の隣接セルでない

セルによるポテンシャルの和を自分の中心での調和関数展開にしておく。(M2L)

FMMの計算手順 - (3) L2L



一番上のレベルから順に、自分の受けているポテンシャルを子セルに伝える（局所展開の中心をシフトする, L2L）
最終的には各粒子が自分の入っているセルの中心での局所展開を評価する。

FMMにおける計算精度と計算量

簡単なオーダー見積り：

$$\text{誤差} \propto \theta^{(p+1)}, \qquad \text{計算量} \propto \theta^{-3} p^4 N$$

計算量は減らせる：ツリーの最下層セルにはいっている粒子の数をnとすれば、計算量のオーダーは

$$Np^4/n + Nn$$

なので、

$$n \propto p^2$$

としておけば

$$\text{計算量} \propto \theta^{-3} p^2 N$$

アンダーソンの方法とPseudoparticle method

FMM、高精度ツリーの「実用上の欠点」

- 数学が結構ややこしい：実装、デバッグが大変、高速化も面倒
(球面調和関数の計算：漸化式なので速くするのが難しい)

もう少し簡単に書けないか？
→ FMM without Multipoles

- アンダーソンの方法 (Anderson 1992)
- 疑似粒子法 (Makino 1998)

アンダーソンの方法

多重極展開を球面上でのポテンシャルの値で表す。

球面を境界条件とするラプラス方程式 → ポアソンの公式

ポアソンの公式 (外部解と内部解)

$$\Psi(x) = \frac{1}{4\pi}\int_S \left[\sum_{n=0}^{\infty}(2n+1)\left(\frac{a}{r}\right)^{n+1} \mathrm{P}_n(s\cdot x/|x|)\right]\Psi(as)ds,$$

$$\Psi(x) = \frac{1}{4\pi}\int_S \left[\sum_{n=0}^{\infty}(2n+1)\left(\frac{r}{a}\right)^{n} \mathrm{P}_n(s\cdot x/|x|)\right]\Psi(as)ds,$$

積分を球面上の適当な標本点をつかった数値積分に置き換える

ポアソンの公式

調和関数展開が球面上のサンプル点でのポテンシャルの値に置き換わる。

→ もともとのFMMの長い式の代わりに、上の式2本で全部済む。

標本点のとりかた

2次元の場合： 単に等間隔でいい。

調和関数展開（フーリエ展開）の直交性がなり立つ。

3次元： 球面調和関数の直交性を保つ配置をする必要がある

そのような配置： “spherical t-design”

球面上の点が spherical t-design をなす
= t次までの任意の多項式の球面上での積分が、それらの点上での多項式の値の和 (の $4\pi/N$ 倍) に等しい。

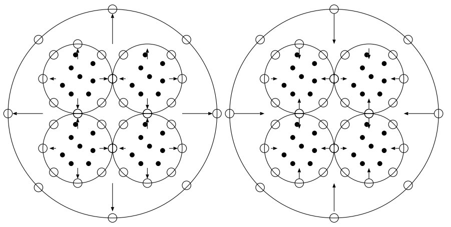
$t/2$ 次までの球面調和関数の直交性が保てることになる。

例：正4面体の頂点：1-design, 正20面体の頂点：4-design。

20-design（216点）といったものも作られている。

アンダーソンの方法の実際 (M2M, L2L)

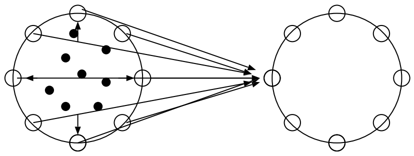
多重極展開（左）と局所展開（右）のシフト



（本当は球の半径はもっと大きい、、、立方体セルを完全に含まないといけないので）

アンダーソンの方法の実際 (M2L)

多重極展開から局所展開への変換



M2M, M2L, L2L のどの場合でも、単に各点のポテンシャルの値をポアソンの公式に従って数値積分するだけ。

アンダーソンの方法の利点と欠点

利点

- プログラムが非常に簡単になる
- 次数等が同じなら、実測では普通の FMM より速いようである

但し、演算数を数えると普通の FMM より増えているはずなので、なぜ速くなっているのか (適切な評価かどうか) は疑問。

欠点

- 誤差、収束性が怪しい
- (後で述べる) 専用計算機GRAPE はそのままでは使えない、、、

疑似粒子法

基本的な考え：

球面調和関数展開：多数の粒子が作るポテンシャル場の近似

↓

逆に球面調和関数展開を粒子分布で表現できないか?

こういうことを考える理由

- 意味がわかりやすい
- 多体問題専用計算機 (GRAPE) が有効に使える

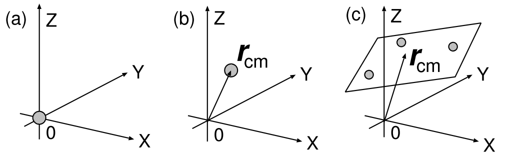
問題：どうやって配置を決めるか？

粒子配置の考え方

- 自由度の数から考える
  粒子1つの自由度：4個
  p次までの球面調和関数の全自由度：$(p+1)^2$
  → 例えば3個の粒子で $p=2$ を表現出来る。
  - 非線形の連立方程式を解かないといけない
  - 有意味な解があるかどうか良くわからない
- 粒子の空間配置は固定して、質量だけ変える
  - （欠点）必要とする粒子の数が増える
  - （利点）いつでも線形計算で配置を決めることが出来る

最小粒子数での実現

$p=0,1,2$ では可能。
$p=1$: 単に重心
$p=2$ (Kawai and Makino 1999)



以下では、粒子を置く座標はあらかじめ決めておくやりかたについて考える。

疑似粒子法の数学 (M2M)

疑似粒子のおき方：アンダーソンの方法と同じく球面上とする。最終的な式：

$$m_j = \frac{2l+1}{K}\sum_{i=1}^{N} m_i \sum_{l=0}^{p} (r_i/r)^l P_l(\cos\gamma)$$

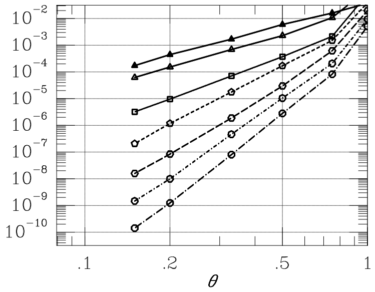
ここで γ は粒子の位置と疑似粒子の位置の間の角度。

- 上の式は、実粒子から仮想粒子への変換と仮想粒子から上位セルの仮想粒子への変換の 両方に使える。
- 収束性を考えると、球の半径をセルの大きさより 少し小さい程度にとっておくのが良い。

ツリー法との組合せ

計算精度の例
粒子分布は球内一様、26万体。

次数は上の2つが一次（定式化が違う）その後順に 2–6次。



ツリー／FMMの効率化・高速化

2方向のアプローチ

- 実装上の効率化
  - ベクトル化、並列化

  この話は後でまとめて。
- アルゴリズムの改良による計算量の減少
  - FFT, 回転等

アルゴリズムの「改良」

ツリー法は通常4重極まで → アルゴリズムをいじる余地少ない。

天文ではもっと高精度が必要な場合には依然直接計算が主流。（独立時間刻みとの組み合わせが困難なため）

FMM： 多数提案がある。

ほとんどのものが、なんらかの方法で M2L の計算量のオーダーを下げようというもの（ここと近くの直接計算の計算量が主要）

FMM の「高速化」

・FFT を利用するもの

M2L の計算は、基本的には多項式の原点移動であり、係数のコンボリューションで表現できる

→ FFT を使って、 $O(p^2 \log p)$ で実行可能

現実には、 $p = 8$ でM2Lの速度が2倍程度。

・ 回転を利用するもの

シフトするまえに、座標軸（z軸）を移動方向に向けておく

→ M2L の計算量が、 p^4 から p^3 に減少（角度方向が分離）

現実の効果はまだ未知数。その他にもいくつか提案はある。（理論家のオモチャ？）

独立時間刻みとツリー、FMM

- 組み合わせることは可能 (McMillan and Aarseth 1993)
- 高精度なもののMPP への実装は存在しない
- GRAPE への実装もない

実装が困難な根本的な理由：計算量が減った分 並列度が下がっているため

今後の大きな課題といえる。

高速化の第二の方法: 速い計算機を使う

速い計算機を使えば速く計算できるというのはなんかトートロジーみたいで当たり前という気もするが、実はそうでもない。

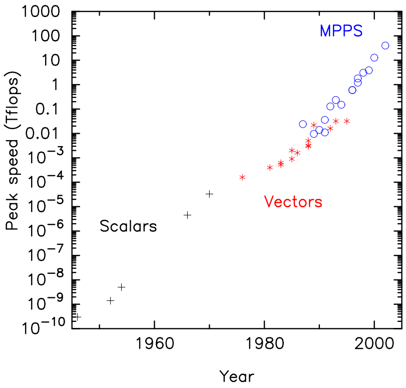
根本的な理由:

最近 30 年間の計算機の「発展」のおかげで、あるアルゴリズムを速い計算機でちゃんと性能がでるように動かすのはどんどん大変になってきた。

計算機の発展

速度の進化: 50 年で 10^{10} 倍

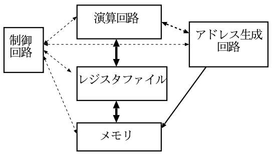
ほぼ時間の指数関数



指数関数的進化を可能にしたもの

1. ムーアの法則:トランジスタの大きさ: 3 年で半分
    - トランジスタの数: 4 倍
    - 速度: 2 倍
2. 計算アーキテクチャの革命
   スカラー計算機 → ベクトル → 並列

スカラー計算機: 1946–1975



要するに普通に「計算機」だと思うもの。

「プログラム」があって、それを 1 命令づつ実行する。

「フォン・ノイマン」アーキテクチャ

命令は基本的なもの:

load: メモリからレジスタへ

store: レジスタからメモリへ

演算: レジスタからデータを演算器に、結果をまたレジスタに

分岐: 次に実行する命令のアドレスを (条件により) 変更

スカラー計算機を速くする方法

- クロックを速くする
    - トランジスタの動作速度
    - 配線遅延
- 命令実行に必要なクロック数を減らす
- 複数の命令を同時に実行する
    - パイプライン化
    - 「スーパースカラー」

必要クロック数を減らす

(数値計算には) もっとも重要なのは乗算

計算機の内部で乗算する方法: 基本的には人間が筆算するのと同じ

```
       101
     x 101
-------------
       101
     101
------------
     11001
```

2 進数で計算

かけられる数と、かける数の各ビットとの積を作って、シフトして足していく。

乗算を速くする

普通の (昔の計算機): 32 ビットなら、32 ビットの加算器を 1 つ使って、32 回足し算をする。

速くする: 32 個の部分積を一度に作って、「一度に」足す (トーナメント方式で足す)。

演算器は 32 (細かくいうともうちょっと多い) 個

全部加算するのにかかる時間: $\log_2 32$ くらい。

だいぶ速くなる

パイプライン化

「一つの命令を実行」:実際にはいろんなことをしている。例えば:

命令読み込み (フェッチ F)

命令解釈 (デコード D)

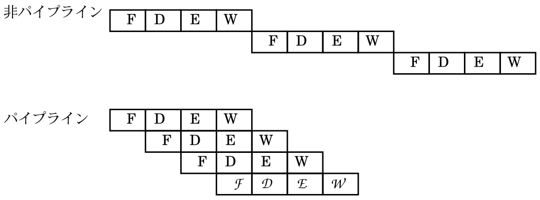
命令実行 E

結果の書き戻し W

それぞれに 1 クロック使うと 4 クロックかかる。

これらはそれぞれ別の回路: 一つの命令を始めた次のクロックで次の命令を始めれば速くなる。

パイプライン化


非パイプライン
F D E W
F D E W
F D E W
パイプライン
F D E W
F D E W
F D E W
F D E W

分岐するとややこしい

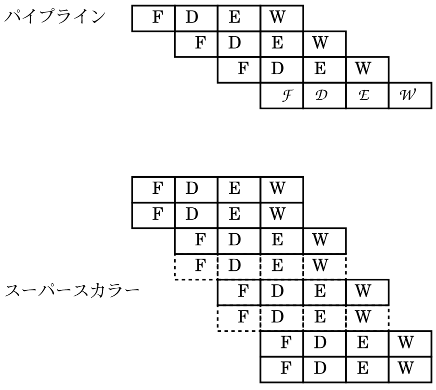
スーパースカラー

フェッチ・デコード: 複数命令を同時に処理

実行ユニットは複数準備

「同時に実行できる命令があれば」同時に実行

スーパースカラー


パイプライン
F D E W
F D E W
F D E W
F D E W
スーパースカラー
F D E W
F D E W
F D E W
F D E W
F D E W
F D E W
F D E W
F D E W

もうちょっと速くする

- パイプラインの段数を増やして、その分クロック周波数をあげる
  - Intel P4 では 20 段
- 命令を実行する順番を、意味が変わらない範囲で実行中に適当に変更
  Out-of-order execution
- 命令自体を再解釈して、パイプライン化/並列実行しやすくする

実行速度の制限要因: memory wall

今までの話: 演算命令とかの実行は速くできる
速くするのが難しい: メモリからの読み込み (load)

- DRAM のサイクルタイムはあまり速くない
- CPU とメモリの間のデータ幅 (信号線の数) はあまり多くできない

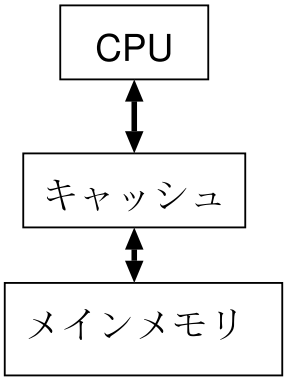
例:

Intel P4 データ 32 bit, クロック 1066 MHz (DRAM チップ内部はもっと遅い)

AMD K7 データ 64 bit, クロック 333 MHz

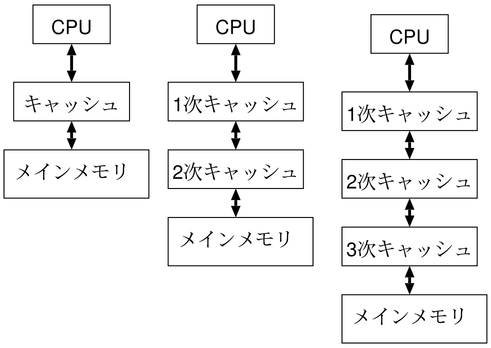
CPU のクロックに比べると 5-10 分の 1

姑息な対応: キャッシュメモリ



CPU とメモリの間に、容量は小さくけど速いメモリを置く (キャッシュメモリ)。
適当なアルゴリズムで、最近使ったデータをキャッシュに残す。
データがキャッシュにあればそっちを使う。

キャッシュの「進化」



なんだか分からない。

キャッシュによる対応の限界

本当にメインメモリにアクセスする必要があればどうしようもない。

```
do i=1,10000000
      a(i) = b(i) + c(i)
enddo
```

こういうのを速く処理するにはメモリを速くするしかない。

いわゆる「スーパーコンピュータ」:この問題があらわになったのは1970年前後 (CDC 7600)

ちょっと余談

最近のマイクロプロセッサの「新技術」:大抵は、いわゆる「スーパーコンピュータ」に 30年くらい前に導入されたもの

- スーパースカラー CDC 6600
- load-store アーキテクチャ (CDC)
- キャッシュ IBM 360/85
- OOO 実行 CDC 6600

真似っこというわけではなく (そういう面もあるが)、技術的な要請が同じようなものであるため

匡体一台 — チップ一つ

という大きさの変化だけ

メモリを速くする

遅いものは速くできないので、沢山並べて

- 順番にアクセス (マルチバンク)
- データ幅を広くする

「ベクトル計算機」の考え方:沢山並べて、並列にアクセスする: ずーっと先まで次に何をするかがわかっていれば簡単

キャッシュでは具合の悪い

```
do i=1,10000000
      a(i) = b(i) + c(i)
enddo
```

みたいなのが理想的

ベクトル計算機

単純なレジスタの代わりに「ベクトルレジスタ」
例えば要素数 64 のベクトルを 8 本とかしまう。

- メモリ—ベクトルレジスタ転送
- 演算

の両方を、「ベクトル」単位で実行 = ベクトル命令
ベクトル命令が使えるようにプログラムを書くことが出来れば速い

スカラー、ベクトル計算機の発展

これまでみた計算機の構成の発展: ハードウェアがどんどん大がかりになる方向

何故そんなことが出来たか: ムーアの法則のおかげで使えるトランジスタ数がむやみに増えたため。

8086 : 10^4 トランジスタくらい。

P4 : 10^8 に近づく。

しかし、いろいろ限界、、、、

限界の具体現れ: 並列計算機の発展

ベクトル計算機の高速化

最初の実用ベクトル計算機: Cray-1 (1976)
使えるトランジスタはもっとどんどん増える。

- ベクトル演算パイプラインを沢山付ける
  CDC Cyber 205, 初期の日本の機械
- CPU+ベクトル演算パイプライン の組合せ自体を沢山付ける
  Cray-2, Cray-(X/Y)MP

メモリ構成

前のどちらの方向も「共有メモリ」の構成をとった。

メモリのどこにあるデータも。どの CPU (あるいはパイプライン) からでも同じ速さで読み書きできる。

言い換えると: メモリと演算器がクロスバースイッチみたいなものでつながる。

- 長い配線が避けられない
- スイッチのハードウェア規模、遅延が大きくなる

性能をあげるのが難しくなる

共有メモリ: 16-32 CPU くらいが物理的な限界?

1980年代後半から 90年代になって、ベクトル計算機の性能向上は遅くなった。

並列計算機

並列計算機といっても、共有メモリならベクトル計算機と同じ問題が発生。

共有メモリにしない→ 問題が (とりあえず) 回避される。

つまり: 1 CPU + メモリの簡単な計算機を構成要素にして、それを沢山並べる。

その間は、なんか適当なネットワークとかでつながる

という方向だと、ややこしい問題を (計算機を作る人は) 考えなくてもいい。

=分散メモリ型並列計算機

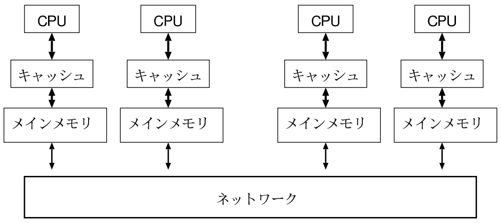
分散メモリ型並列計算機

開発の重要な源流は

- QCD 計算のための計算機 (Caltech Hypercube)
- 星野-川合らによる PACS/PAX
- イリノイ大学の ILLIAC IV

どれも、割合単純な連続系 (偏微分方程式系) を解くことだけを目標に設計・製作された。

分散メモリ型計算機の構成



どれもおなじようなもの。ILLIAC IV は特殊: SIMD 型(全部の CPU が同じ演算を実行)。

あとの流れは MIMD (みんなバラバラに計算する)

分散メモリ型の中でも特に台数が多いもの: Massively-Parallel とか Highly-Parallel と呼んでいた。

分散メモリ型計算機の歴史

1970年代: 汎用マイクロプロセッサを使った計算ノード

1980年代: 多様

- 汎用マイクロプロセッサ/専用プロセッサ
- SIMD/MIMD
- 分散メモリ/分散共有メモリ/AllCache/...

分散メモリ型計算機の歴史(2)

1990年代: 淘汰

- ベクトル型分散メモリ— 富士通、NEC
- Cray T3x — Alpha
- Beowulf — Intel x86

2000年代: ???

- Beowulf、SMP クラスタ以外は死滅?

死滅した理由

- 価格性能比
- プログラミングモデル

価格性能比

消費者マーケットに出るもの: 生産数が何桁も違う。
それだけ、開発費を投入して量産コストを下げることができる。

2002年時点で:

生産数 $< 10^4$ 個のカスタム LSI: 開発費 数億、量産コストチップあたり5-10万動作クロック 300-500 MHz

Intel Pentium 4: 開発費 ??? 売値: 安いものなら1万、動作クロック: 2GHz 以上

同じようなものを作ると、価格性能比で 10 倍は損

プログラミングモデル

多様なアーキテクチャ → プログラミングモデルへの要求:

- 移植性
- 性能の高さ

生き残ったモデル: message-passing

要するに: 各ノードでは普通にプログラムが走る。必要に応じて、通信ライブラリを使って通信する。

自動並列化とか並列処理言語とかそういったものには頼らない。

もっとも安直なハードウェアに有利な状況

Beowulf とは？

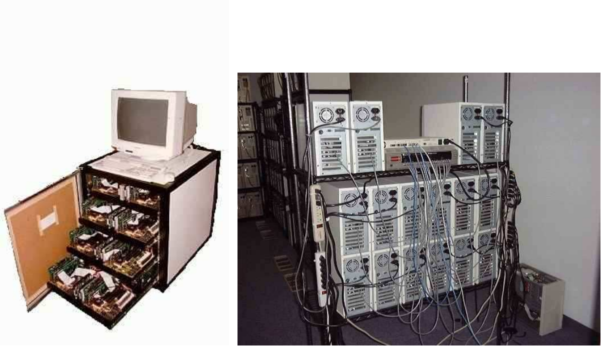
- なるべく価格性能比のいいパソコンを沢山買う
- それを必要最小限の性能のイーサネットを使った普通のネットワークでつなぐ

それだけ。

安くあげる: OS, コンパイラにお金を使わない: Linux, gcc, MPICH

Early Beowulfs

`http://beowulf.gsfc.nasa.gov/overview.html`



Beowulf 上の並列プログラム

境界条件

- メッセージパッシングによるプログラム
- ネットワークを通したデータ転送
  - レイテンシは(特殊なハード、ソフトを使わないと)遅い。 $> 100\mu$s
    「遅い」主記憶に比べてもさらに3桁遅い
  - バンド幅はどうやっても細い。今なら GbE で 100MB/s
    主記憶に比べてもさらに数十倍遅い

アルゴリズムに要求される条件

通信回数、通信量が十分に少ないこと。

多体シミュレーションの並列化

- 独立時間刻み
- ツリー・FMM

独立時間刻みの並列化

実はそれ以前の問題: 「全粒子から全粒子への力の計算」の並列化

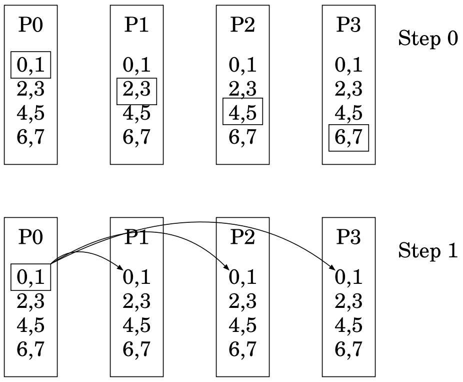
並列化の 2 方法

- 各ノードが全粒子のコピーを持って、重力計算、時間積分は分担する
- 各ノードが自分の分担の分だけを持つ。重力計算のためには他のノードから粒子を貰ってくる。

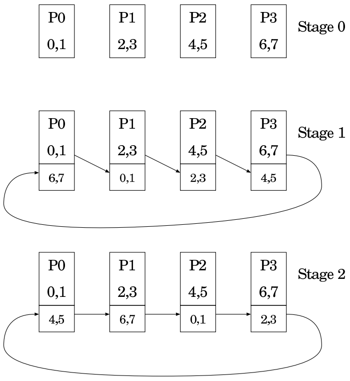
どちらにしても、各ノードが全粒子の情報を使って力を計算する。

ノード一つの通信バンド幅でスケーラビリティが制限される。

The copy algorithm


P0
P1
P2
P3
Step 0
0,1
2,3
4,5
6,7
Step 1

The ring algorithm


P0
P1
P2
P3
Stage 0
0,1
2,3
4,5
6,7
Stage 1
6,7
0,1
2,3
4,5
Stage 2
4,5
6,7
0,1
2,3

計算時間のスケーリング

N 粒子、p ノード

$$T_{\rm ring} = C_f N^2/p + C_c N + C_s p. \tag{75}$$

C_f	力の計算にかかる時間
C_c	1粒子を転送するのにかかる時間
C_s	通信ライブラリのオーバーヘッド

通信時間は p によらない $\rightarrow$ $O(N)$ ノードしか使えない。

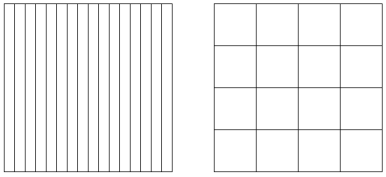
独立時間刻みの場合

- 1ステップで動かす粒子数が減る
- 通信ライブラリのオーバーヘッドが無視できなくなる

スケーラビリティが一層悪くなる

もうすこし違う考え方はないか？

近接相互作用の時（2次元の例）

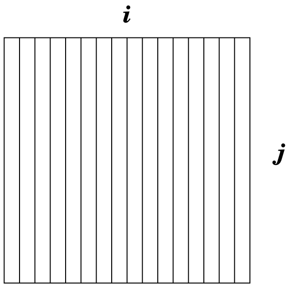


1-D decomposition　　2-D decomposition

1次元空間分割：スケーラブルでない
ノード数 $\propto$ 1次元方向のメッシュ数

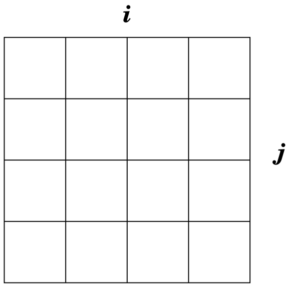
2次元空間分割：スケーラブル
ノード数 $\propto$ 全体のメッシュ数

遠距離相互作用 — 相互作用行列を考える



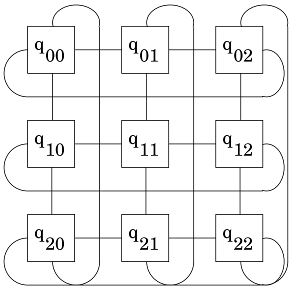
- 相互作用の行列を見ると、普通のアルゴリズムは1次元分割になっている。
- これがスケーラブルでない理由
- 2次元分割は可能か？

相互作用の2次元分割



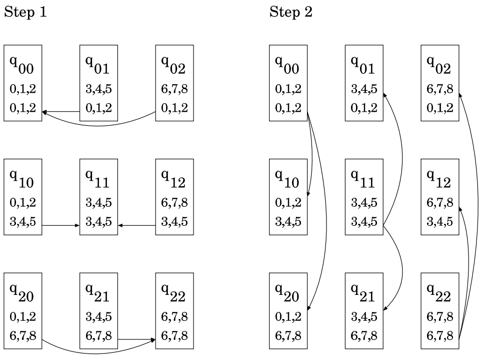
- 形式的には左のような絵を書ける。
- 実際にはこれはなにを意味しているのか？

ハードウェアとしては、、、


q_{00}
q_{01}
q_{02}
q_{10}
q_{11}
q_{12}
q_{20}
q_{21}
q_{22}

- 分割した相互作用行列にプロセッサを対応させる。
- トポロジ：2D トーラスなり「ハイパークロスバー」なり、、、

計算手順


Step 1
Step 2
q_{00} 0,1,2 0,1,2
q_{01} 3,4,5 0,1,2
q_{02} 6,7,8 0,1,2
q_{10} 0,1,2 3,4,5
q_{11} 3,4,5 3,4,5
q_{12} 6,7,8 3,4,5
q_{20} 0,1,2 6,7,8
q_{21} 3,4,5 6,7,8
q_{22} 6,7,8 6,7,8

横に総和、縦に放送

スケーリング（理論）

r^2 プロセッサ

$$T_{\rm 2Dbcast} = C_f N^2/r^2 + 2(C_c N/r + C_s \log_2 r). \tag{76}$$

計算時間は $(N/r)^2$、通信は N/r になる。従って、 $r \sim N$、つまりプロセッサ数を N^2 にとれる。(スタートアップが無視できれば)

使えるプロセッサ数

・効率が 50 ％ になるプロセッサ数

$$p_{\rm half,2Dbcast} \sim (NC_f/2C_c)^2 \tag{77}$$

(通信のスタートアップが無視できる場合) ここから先でも速度は向上する（プロセッサ数の平方根に比例）

独立時間刻みの時

$$p_{\rm half,ind,2D} \sim N^{5/3} C_f/[C_s \log_2(N^{5/3} C_f/2C_s)] \tag{78}$$

大抵の場合にスタートアップ時間がスケーラビリティを制限する。

N^2 まではいかないが、 1 次元分割に比べれば圧倒的に良い。

こんなのは昔から知られてないか？

- N^2 個の（仮想）プロセッサを使うアルゴリズム
  Barnes and Hillis (1987?) に触れられている。
- Hyper-systolic algorithm (Lippert *et al.* 1998)
  1 次元リング＋ non-unit-stride communication
  なんだか難しい数学を使って通信パターンを最適化

同様の考え方自体は知られてはいた。
スケーラビリティ等は明確には議論されていない。

ツリー法のベクトル化、並列化

ベクトル化、並列化の技法はツリー法も FMM も同じ。

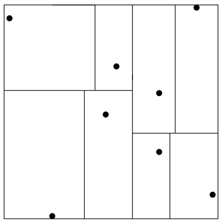
- 共有メモリの機械では容易（アルゴリズムの並列度は実際上 N）
- メッセージパッシングの機械：効率の良い空間分割が必要。Salmon & Warren が様々な方法をテストした。プログラムは面倒だが、作ってしまえば効率はよい。

従来の並列化アルゴリズム

実用になっている並列化法は以下の 2 つ。どちらももと Caltech Hypercube のグループの Salmon と Warren によるもの

- Orthogonal Recursive Bysection (ORB)
- Hashed Oct Tree (HOT)

ORB



- まず、粒子が半分ずつにわかれるように x 軸に垂直な平面で切る
- 切ったそれぞれについて、また半分になるように y 軸に垂直な平面で切る 3 次元なら次に z、2 次元なら次は x 軸になる

というのを、プロセッサ数になるまで繰り返す

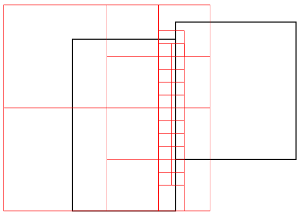
ORB 空間分割の実装

ORB も実装はいろいろ。良く知られているのは Dubinski の論文によるもの。粒子はすでに各プロセッサに適当に分配されているとする。

1. まず 2 分法で分割点を捜す。各プロセッサで粒子を数えた後、関係するプロセッサ全部での合計をとって新しい分割点を決める。
2. 各プロセッサで分割点の右と左に粒子を振り分ける
3. 各プロセッサは、 ORB ツリーの現在の階層で自分とペアになるプロセッサと粒子を交換する

というのを、階層分繰り返す。
ツリーはとりあえず各プロセッサでバラバラに作る。

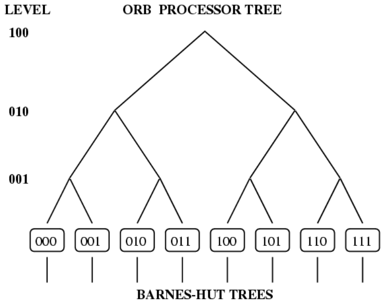
他のプロセッサにある粒子からの力の計算



他のプロセッサから、「必要ないところをカットした」ツリーをもらってくる (local essential tree, LET)
それらと自分のところを適当にまとめて全体ツリーを作る

「適当にまとめる」とは？

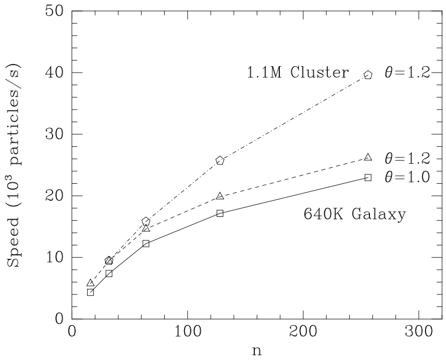
Dubinski の実装：この上の階層は ORB ツリーそのもの


LEVEL
ORB PROCESSOR TREE
100
010
001
000
001
010
011
100
101
110
111
BARNES-HUT TREES

ORB tree の問題

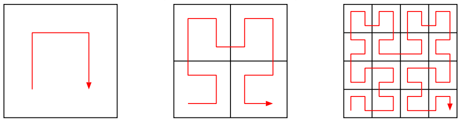
- 実装が面倒
  - ORB tree のところと local tree のところでツリーの構造が違う
  - LET はツリー構造（ポインタとか）もちゃんと保持して送る必要あり
- スケーラビリティが悪い
  - 通信回数がプロセッサ数に比例
  - 空間分割の形が悪くなると通信量も計算量も増える
- シリアルコードと結果が丸め以上に違う

スケーリング



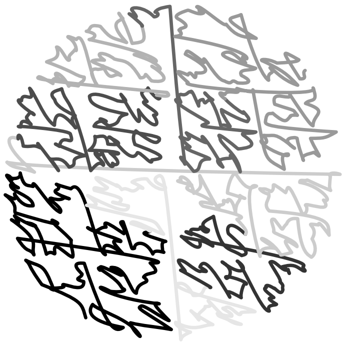
HOT

ペアノ曲線：N 次元空間を１次元にマッピング



- 粒子をペアノ曲線上の順番で並べる
- 各プロセッサに連続した粒子を割り当てる

HOT の例



ただしこれはペアノ曲線ではなくて Morton Ordering

HOT でのツリー構成・重力計算

ツリー構成

- ペアノ曲線を作るところでツリーが構成される
- 並列ソートが必要

重力計算

- LET のようなことは困難
- 計算中に必要になった情報を他のプロセッサにリクエストする。

メッセージコンバイニング、計算と通信の非同期実行などの複雑なテクニックが必要になる。

HOT の利点・欠点

利点

- 任意のプロセッサ数・プロセッサトポロジに適用可能
- プロセッサ数が増えても計算量は増えない
- シリアルコードと結果が一致

欠点

- 実装が非常に大変
- 速い通信ライブラリ and/or 通信を隠蔽する「何か」が必要

ORB と HOT

- ORB
  - 実装は比較的簡単。速度もまあまあ
  - スケーリングが悪い（計算量自体が増える）
  - プロセッサ数が 2^n でないといけない
- HOT
  - プロセッサ数は任意
  - 速度は？
  - 実装は大変

我々の使うことにした方法

ORB を多少変更

- 深さを 3 層までに制限
- 2分割以上も許す

例えば、 12 ノードなら $3 \times 2 \times 2$ とかいうふうに分ける。もちろん、 2, 4, 8 ノードでは従来の ORB と同じ。

ORB 空間分割の復習

ORBも実装はいろいろ。良く知られているのは Dubinski の論文によるもの。粒子はすでに各プロセッサに適当に分配されているとする。

1. まず2分法で分割点を捜す。各プロセッサで粒子を数えた後、関係するプロセッサ全部での合計をとって新しい分割点を決める。
2. 各プロセッサで分割点の右と左に粒子を振り分ける
3. 各プロセッサは、 ORB ツリーの現在の階層で自分とペアになるプロセッサと粒子を交換する

というのを、階層分繰り返す。

この方法の問題点：計算量、通信量（回数）ともに大きい。

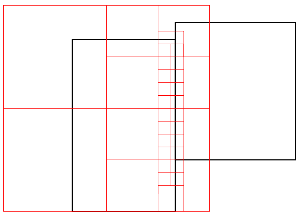
もうちょっと安易な方法

今回使った方法： Brackston & Suel によるものを踏襲

- 各プロセッサは適当に選んだサンプル粒子のセットをノード 0 に送る。
- ノード 0 は、集めたサンプルセットを使って分割を生成する。できた分割は全ノードに放送する。
- 各プロセッサは、もらった分割を使って粒子を振り分けて、担当するプロセッサに送る。

分割に多少の誤差が入るが、計算時間としては問題にならない程度。
計算・通信時間は要求する分割精度に依存

力の計算（復習）



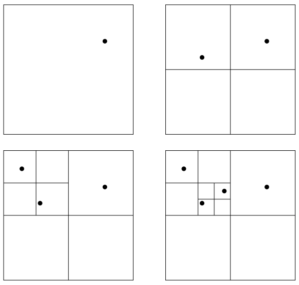
他のプロセッサから、「必要ないところをカットした」ツリーをもらってくる (local essential tree, LET)
それらと自分のところを適当にまとめて全体ツリーを作る

今回の実装

- LET ではなく、LET の最下層にある「粒子」だけを送る。
- 受けとった「粒子」を、自分のツリーに挿入する。

ツリーに粒子を挿入：もともとの Barnes のプログラムで使われている木の作り方。

粒子挿入によるツリー構成



粒子がどのセルに入っているか決める。

既に子セルがあればそのどれかに入れる

既に粒子が入っていれば分割する。

粒子挿入による全体ツリー構成

- 実装は簡単（もともとのツリーを作るルーチンがそのまま使い回せる）
- 通信量は理論的に可能な最小量
- ツリー構成の計算量は多少増える
- 重力計算の計算量はシリアルコードと同じ
- 結果もシリアルコードと一致

別のアプローチ — 計算機を作る

速い計算機を買ってきて動かすというのもなかなか大変である。

- 10年もたつと計算機アーキテクチャが変わる (かもしれない) ので昔頑張って作ったプログラムが水の泡になる (かもしれない)
- 最近考えないといけないことが増えた
  - 分散メモリでの並列化
  - キャッシュの有効利用による高速化
  - その他なんだか分からないテクニック

もうちょっと違う人生はないか?

一つの考え方: 計算機を自分で作る

人が作った計算機を苦労して使おうと思うから面倒。
ハードウェアから好きなように作ればむしろ簡単にならないか?

なぜ専用計算機を考えるのか

基本的には、汎用計算機に比べて、「速く、安く」できる（かもしれない）から。
なぜ「速く、安い」か？

- 問題自体の特性
- 技術的な要因
- 歴史的、経済的な要因

問題自体の特性

粒子系シミュレーションの特徴:一つの粒子が多数の粒子と相互作用する

- 計算量が多い（メモリ必要量に比べて）
- 計算が比較的単純な繰り返しである
- 通信パターンが規則的である

（独立時間刻み、ツリー法では細かい考慮が必要にはなる）

対象システムの分類

連続系（流体等）: 規則的、近傍通信、計算量小

粒子系：規則的（ $N \times N$ 通信)、計算量大

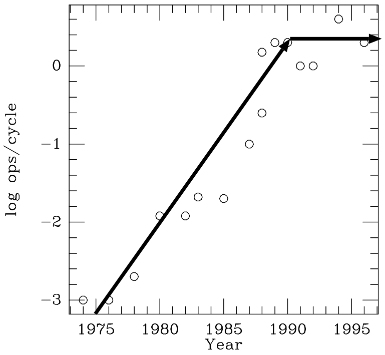
その他（離散不規則系）: 回路シミュレーション等？

規則的で計算量が大きい ＝ 専用ハードウェアに適している

技術的な要因

- 半導体製造技術の進歩 ＝ 多数の演算回路を集積する大規模回路が実現可能
- 半導体設計技術の進歩 ＝ 半導体技術の「素人」でもモジュールジェネレータ等で設計ができる (???)
- 汎用計算機の設計技術の限界 (?) ＝ トランジスタ利用効率の急速な低下

マイクロプロセッサの「進化」



代表的なマイクロプロセッサの、サイクル当たりの浮動小数点演算数

1 を超えてから、ほとんど止まっている ＝ 汎用の並列処理は難しい

マイクロプロセッサにおける並列処理

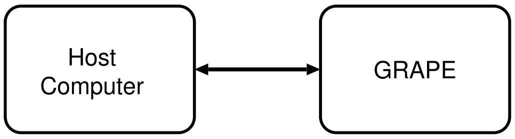
用途を制限すればある程度の並列化はできる

- ソニー PS2, 日立 SH4 : 3D 座標変換に特化したハードウェア
- Intel SSE, AMD 3DNow!, Motorola AltiVec: SIMD 並列処理。座標変換に重点

汎用の並列処理は難しい

- スーパースカラ
- VLIW

GRAPE の基本的考え


Host
Computer
GRAPE

Time integration etc.　Interaction calculation

専用ハード:　相互作用の計算
汎用ホスト:　他のすべての計算

専用ハードウェア

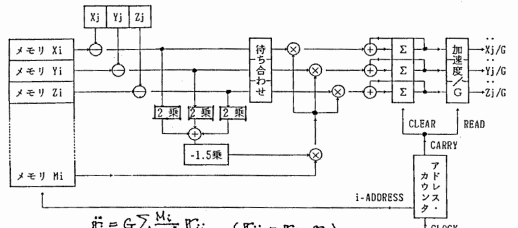
- 相互作用計算のための専用パイプラインプロセッサ
  - 多数の演算器を集積可能
  - すべての演算器が常時並列動作

  → 非常に高い性能
- すべてハードウェア → ソフトウェア不要

汎用ホスト計算機

- 高レベル言語 (Fortran, C, C++...)
- すでにあるプログラムが少しの変更で使える。
- 独立時間刻み、ツリー法等も使える

GRAPE パイプライン


Xj
Yj
Zj
メモリ Xi
メモリ Yi
メモリ Zi
メモリ Mi
2 乗
2 乗
2 乗
-1.5乗
待ち合わせ
Σ
Σ
Σ
加速度/G
$\ddot{X}_j/G$
$\ddot{Y}_j/G$
$\ddot{Z}_j/G$
CLEAR
READ
CARRY
アドレス・カウンタ
i-ADDRESS
CLOCK

$$\ddot{\mathbf{r}}_j = G\sum_i \frac{M_i}{r_{ij}^3}\mathbf{r}_{ij} \quad (\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j)$$

+, −, ×, 2乗は1 operation, -1.5乗は多項式近似でやるとして10operation 位に相当する.
総計24operation.
各operation の後にはレジスタがあって, 全体がpipelineになっているものとする.
「待ち合わせ」は2乗してMと掛け算する間の時間ズレを補正するためのFIFO(First-In First-Out memory).
「Σ」は足し込み用のレジスタ. N回足した後結果を右のレジスタに転送する.

図2. N体問題のj-体に働く重力加速度を計算する回路の概念図.

（近田 1988）

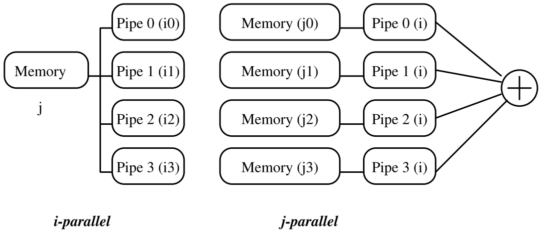
GRAPE における並列化

パイプライン１本が速くても、並列化できなければたいした意味はない。
ホストは１台として、 GRAPE の側の並列化を考える。
方針は 2 つ

- 複数の粒子への力を並列に計算する (i 並列)
- 複数の粒子からの力を並列に計算する (j 並列)

実現方法

i 並列: 一つのメモリからのデータを複数パイプラインに配る。
j 並列 : 複数の（メモリ＋パイプライン）に同じ粒子への力を計算させて、終ってから合計する。


Memory
j
Pipe 0 (i0)
Pipe 1 (i1)
Pipe 2 (i2)
Pipe 3 (i3)
Memory (j0)
Memory (j1)
Memory (j2)
Memory (j3)
Pipe 0 (i)
Pipe 1 (i)
Pipe 2 (i)
Pipe 3 (i)
+
i-parallel
j-parallel

並列パイプラインの実装

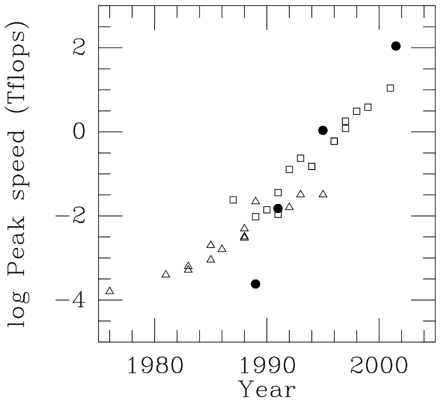
メモリを共有できる範囲（ボード／チップ）:
i 並列
それより上： j 並列
メモリデータの複製を許すなら i 並列も使える

GRAPE-4: ボード上 i 並列、ボード間 j 並列
GRAPE-6: チップ内 i 並列、ボード上 j 並列

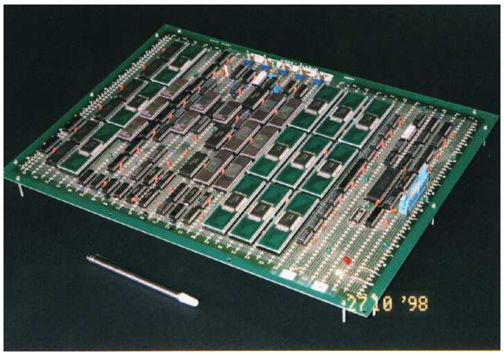
GRAPE アーキテクチャの発展

1989	GRAPE-1	低精度、EPROM で演算
1990	GRAPE-2	高精度、浮動小数点演算 LSI
1991	GRAPE-3	低精度、カスタム LSI
1995	GRAPE-4	高精度、カスタム LSI 、超並列
1998	GRAPE-5	低精度、複数パイプラインを集積
2001	GRAPE-6	高精度、複数パイプラインを集積

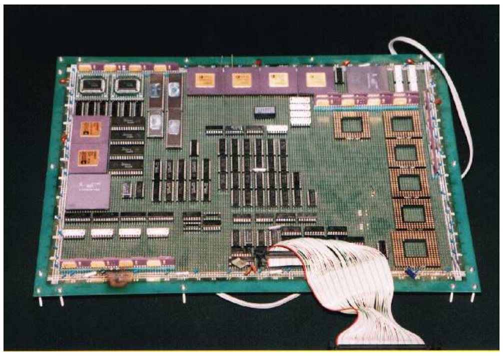
計算速度の発展



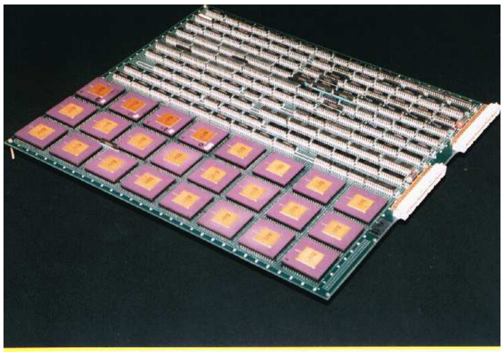
GRAPE-1



GRAPE-2

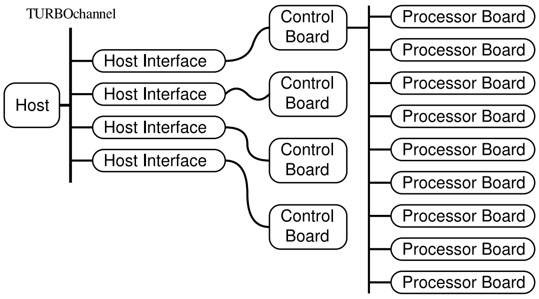


GRAPE-3

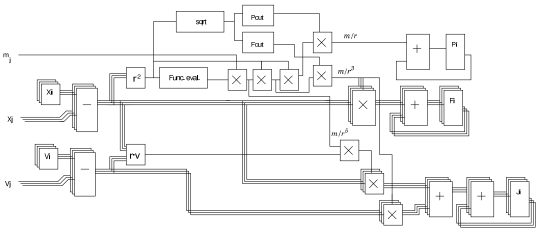


GRAPE-4

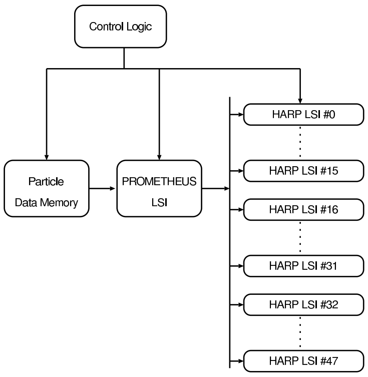
GRAPE-4 の構成


TURBOchannel
Host
Host Interface
Host Interface
Host Interface
Host Interface
Control Board
Control Board
Control Board
Control Board
Processor Board
Processor Board
Processor Board
Processor Board
Processor Board
Processor Board
Processor Board
Processor Board
Processor Board

GRAPE-4 パイプライン


sqrt
Pcut
Fcut
m/r
m/r^3
m/r^5
Pi
Fi
Ji
m_j
Xi
Xj
Vi
Vj
r^2
Func. eval.
r·v

GRAPE-4 演算ボード

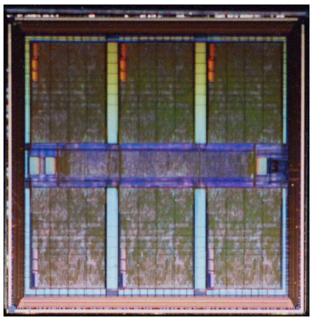

Control Logic
Particle Data Memory
PROMETHEUS LSI
HARP LSI #0
HARP LSI #15
HARP LSI #16
HARP LSI #31
HARP LSI #32
HARP LSI #47

多数の パイプライン
LSI がメモリユニット
を共有
↓
ボードが単純になり、
集積度をあげられる

GRAPE-6 について

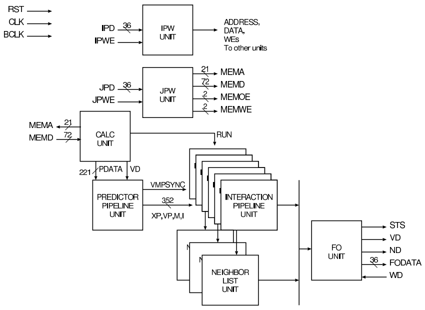
- プロセッサチップ
- プロセッサモジュール
- プロセッサボード
- ネットワークボード
- 開発の現状

パイプライン LSI



- 0.25 μm ルール
  (東芝　TC-240,　1.8M ゲート)
- 90 MHz 動作
- 6 パイプラインを集積
- チップあたり 31 Gflops

パイプライン LSI詳細

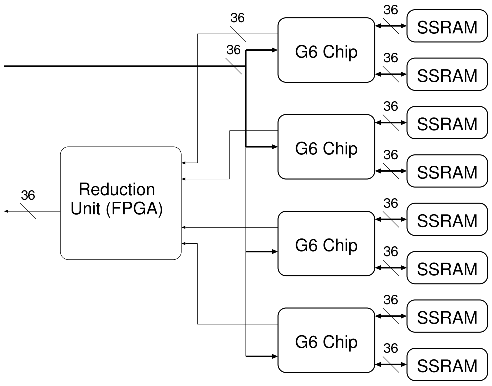

RST
CLK
BCLK
IPD
36
IPWE
IPW UNIT
ADDRESS,
DATA,
WEs
To other units
JPD
36
JPWE
JPW UNIT
21
MEMA
72
MEMD
2
MEMOE
2
MEMWE
MEMA
21
MEMD
72
CALC UNIT
RUN
221
PDATA
VD
PREDICTOR PIPELINE UNIT
VMPSYNC
352
XP,VP,M,I
INTERACTION PIPELINE UNIT
NEIGHBOR LIST UNIT
FO UNIT
STS
VD
ND
36
FODATA
WD

GRAPE-4 プロセッサボードの全機能を集積

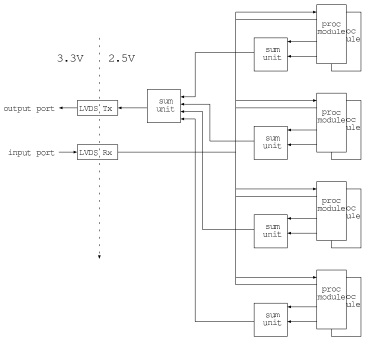
- ホスト IF
- メモリ IF
- パイプライン本体
- 制御回路

GRAPE-6 processor module

GRAPE-6 processor module

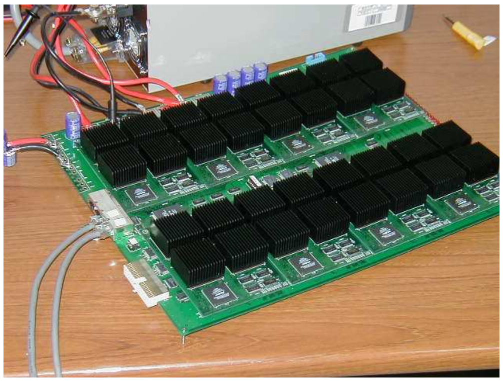

36
36
G6 Chip
G6 Chip
G6 Chip
G6 Chip
36
SSRAM
36
SSRAM
36
SSRAM
36
SSRAM
36
SSRAM
36
SSRAM
36
SSRAM
36
SSRAM
36
Reduction Unit (FPGA)

GRAPE-6 Processor board

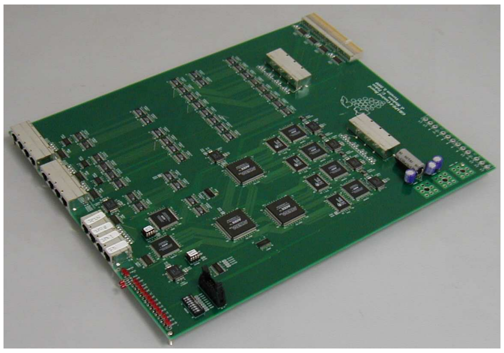

3.3V
2.5V
output port
input port
LVDS Tx
LVDS Rx
sum unit
sum unit
sum unit
sum unit
sum unit
proc module
proc module
proc module
proc module

- ボード 1 枚に 32 チップ
- セミシリアル (LVDS) インターフェース (350MHz clock, 4 wires)

GRAPE-6 processor board

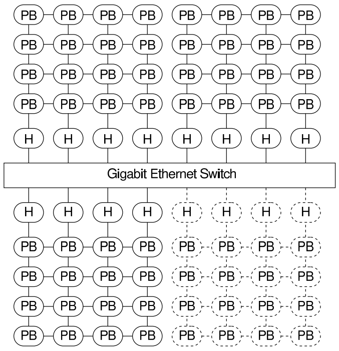


GRAPE-6 network board



GRAPE-6 host interface board

The full 64 Tflops GRAPE-6 system



- 4-host, 16-board “block” with dedicated network
- 4 (currently 3) “blocks” connected through GbE network

Combination of host network solution and dedicated network solution.

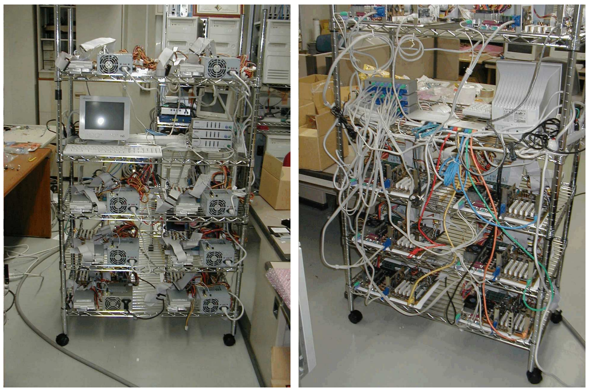
The 48-Tflops GRAPE-6 system

Present 48-Tflops system.

Three blocks with 16 host computers.

The fourth block will be operational by the end of this month.

The host “PC Cluster”



専用計算機開発の損得

13年ばかりやってきてどうだったか？ということ。

もともとターゲットとした問題（天文の多体問題）に対しては、予想以上の成果をあげたといって嘘にはならないと思う。

GRAPE は100枚以上のコピーが世界中の 20 以上の研究機関で使われ、惑星形成や星団の進化のシミュレーションのほとんど、また銀河形成や宇宙論シミュレーションについてもかなりの割合が GRAPE を使って行なわれている。

というあたりが「公式見解」ということで。

個人的な感想

GRAPE のようなものを「うまく作る」のは結構大変である。

- 対象となる問題
- アルゴリズム・要求精度
- 計算機アーキテクチャ
- 論理設計
- 実装
- OS、デバイスドライバ等のソフトウェア

の全般についてまあまあの理解が統合されていないと全体設計が出来ない。

普通の計算機を作るのに比べると

普通の計算機では

- 対象となる問題、アルゴリズム、要求精度

「必要ない」（本当？）

- それ以外

「まあまあ」ではまあまあの計算機しか出来ない。

他の誰かと同じようなことをすれば済むという面もある（1番になれるか？という問題はある）

Case Study： GRAPE-6 の場合

演算方式、論理設計： GRAPE-4 のものを多少改良／変更。大きな問題はなかった。

実装：テクノロジ微細化、低電圧化のためにいろいろ問題が起きた。

- レイアウトが収束しない（タイミング要求を満たせない）。
- チップ内で電源電圧が下がる。
- 負荷変動に電源が対応できない。

レイアウトが収束しない

（最近はましになってきてるかもしれないけど）ディープサブミクロン設計でありがちな問題。

配線遅延 >> スイッチング遅延
→ レイアウトしないと遅延時間がわからない

従来の論理設計段階では仮配線長を仮定するやりかたでは駄目。

本来、レイアウトと論理設計を並行して進めるような手法が必要

GRAPE での配線遅延

GRAPE では

- 長い配線はパイプラインとインターフェース回路の間くらいしかない
- そこのタイミング要求はあんまり厳しくない（マルチサイクルでいいのが多い）

ので比較的楽で、大幅なレイアウト修正なしでなんとか誤魔化していた、、、

実際、ほぼ同じ時期に同じ会社が作っていた高速マイクロプロセッサに比べれば半分以下の期間でレイアウトが終った。

チップ内で電源電圧が下がる。

最初のチップでは

- パッケージの電源ピンからダイのパッドまでの配線抵抗
- ダイ上の電源供給ネットの抵抗

が大きかったために、内部で電圧が下がってまともに動かなかった。

計算中と非動作時では2倍程消費電力が違うので、電圧もその分変わる。

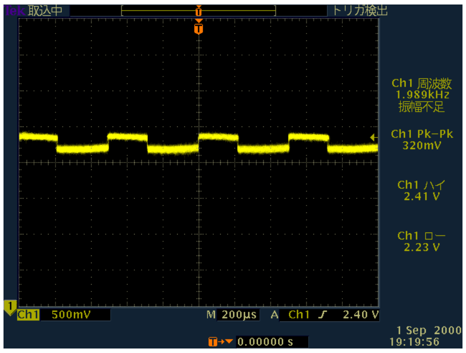
形式的な原因：チップ製造を担当した会社の設計ガイドラインに、コアロジックの消費電力から電源ピン数等への制約がなかった（らしい）

負荷変動に電源が対応できない。

チップが動作時と非動作時で2倍くらい消費電力が違う。動作時間は $10\mu s$ から 1ms 程度。ボード1枚では 70A と 140A の間で変わる

- 普通のスイッチング電源では電流変化に応答できない
- 大容量の電解コンデンサをつけてもほとんど気休めにしかならない

電源波形



負荷変動への対応

今から設計するなら

- スイッチング電源をフィードフォワード制御する
- 非動作時でも消費電力が下がらないように設計する

現在の対応

- ある程度応答の速い電源に変える
- 内部抵抗の小さい電解コンデンサを大量につける

Case Study からの教訓

結局は：

専用計算機も大きくなると大規模システムで起きるようなあらゆる問題が起きる。

というだけ。

あとは、、、

担当エンジニアのいうことは正しいとは限らない

↓

ボードやチップは専門家にまかせておけば出来るというものでもない

とはいえ、これはあまり専用計算機固有の問題というわけではない。

専用計算機は難しい？

「成功」といえるようなものはそんなに多くない。

多体系に話を限るとアプローチは2つ。どちらも結構いろいろある。

専用パイプライン

- DMDP (Delft)
- FASTRUN
- GRAPE
- MD-Engine
- MDM

並列計算機

- Digital Orrery
- Transputer-based projects...
- HaMM
- 他にも沢山

あんまりうまくいかないのはなぜ？

どれが成功でどれが失敗かというのはさておき、、、

基本的な理由は 2 つ。

1. 出来たものが実は使えなかった。
2. 出来た時には速度が汎用計算機より速くなかった。

理由 (1) は専用計算機に固有だが、例は少ない（公表されていないのがあるのかもしれないが）

理由 (2) は普通の計算機と同じ。開発に時間がかかるとムーアの法則の分相対的に遅くなる。

開発期間の問題

大抵の人は（私を含めて）見積もりが甘いというのは確か。

本質的な問題：

専用計算機の場合、開発期間が多少（1-2 年）延びたくらいで意味がなくなるようなプロジェクトは、そもそもあまり意味がない（出来てからのハードウェアの寿命がどうせ短い）。

大雑把には、計画時点で

- 価格性能比が 1000 倍 — 問題なし
- 価格性能比が 100 倍 — ちょっと苦しい
- 価格性能比が 10 倍 — やめたほうがいい

別の方向

開発に時間がかからないなら話は別: Beowulf

安い DSP を使うのはやはりそれなりに時間がかかる (QCDSP)

Dreamcast, PS2 ???

価格性能比 100 倍は可能か？

あくまでも「計画時点」。とはいえ、、、

半導体技術は 1-2 年先を見込める: 1.5-2 倍得

大量生産されるマイクロプロセッサほどクロックはあげられない: 3-5 倍損

結局、

- 演算数が同じならチップの値段を 1/100 にする
- 値段が同じなら演算数を 100 倍にする

必要がある。前者は多分無理なので、演算数を増やす以外の方法は多分ない。

100 個の演算器を 1 チップに

マイクロプロセッサはクロックあたり 1 演算程度しかしない

→ 100 個の演算器が 1 チップに入れば OK。

入れるだけならなんということはない。 10^7 トランジスタもあれば十分。

どうやって使うか

- チップ外への通信速度
- 演算器間の通信速度
- 内部メモリ、レジスタと演算器の間の通信速度

GRAPE の場合

- 実／仮想マルチパイプラインによるチップ外通信速度の削減
- ハードワイヤドパイプラインによる最小コストでの演算器間接続
- 同じくハードワイヤドのため内部メモリ不要

通常 on-chip multiprocessor で起きる問題をほぼ完全に回避している。

基本的にはそういうことが出来る問題だから。

逆にいえば、そういうことが出来る問題（アルゴリズム）でないと専用計算機はうまくいかない。

専用計算機の今後

GRAPE アーキテクチャ：汎用計算機に比べた相対的優位はだんだん大きくなる

ムーアの法則でトランジスタは増える

汎用計算機では増えたトランジスタの全部を演算器には使ってない

GRAPE は全部使える

ムーアの法則が成り立たなくなる頃 ???

FPGA はどうか

小規模な機械：短い開発期間で出来る（ソフトウェアを別にすれば）

が、カスタムに比べれば遅くて回路規模が小さい

大規模な機械：開発に時間がかかるという問題が復活する

FPGA を（数年に渡って）更新可能な設計をすれば、、、

汎用計算機の今後 ???

本当はこっちが問題。

GRAPE: 汎用計算機につける「アクセラレータ」

5 年後、10 年後の（科学技術計算用）汎用計算機はどんなものか？（そんなものがあるのか？）

専用計算機についてのまとめ

- GRAPE では、もっとも演算量の多い相互作用の計算だけを専用化し、他の計算は汎用計算機に回すことで高い性能と柔軟性を両立させている。
- この分担のためにハード開発は比較的容易である。
- このような分担をうまく機能させるためには、採用するアルゴリズムがそういうふうにできている必要がある。可能ならアルゴリズムの変更も考えるべき。
- 5-10 年先でも GRAPE の方法は多分有効
- 重力多体系以外にこういう方法が有効かどうかはいまだよくわからない。