GRAPE Project
— 専用計算機による計算科学

牧野淳一郎
東京大学理学系研究科天文学専攻

この講義の(一応の予定)

1. 天文学における「シミュレーション」の役割
2. 重力多体系の進化
3. N体シミュレーション
4. 専用計算機によるN体シミュレーション

某所で3日使った話を1日でやろうというものなので、どうなるかお立ち合い。

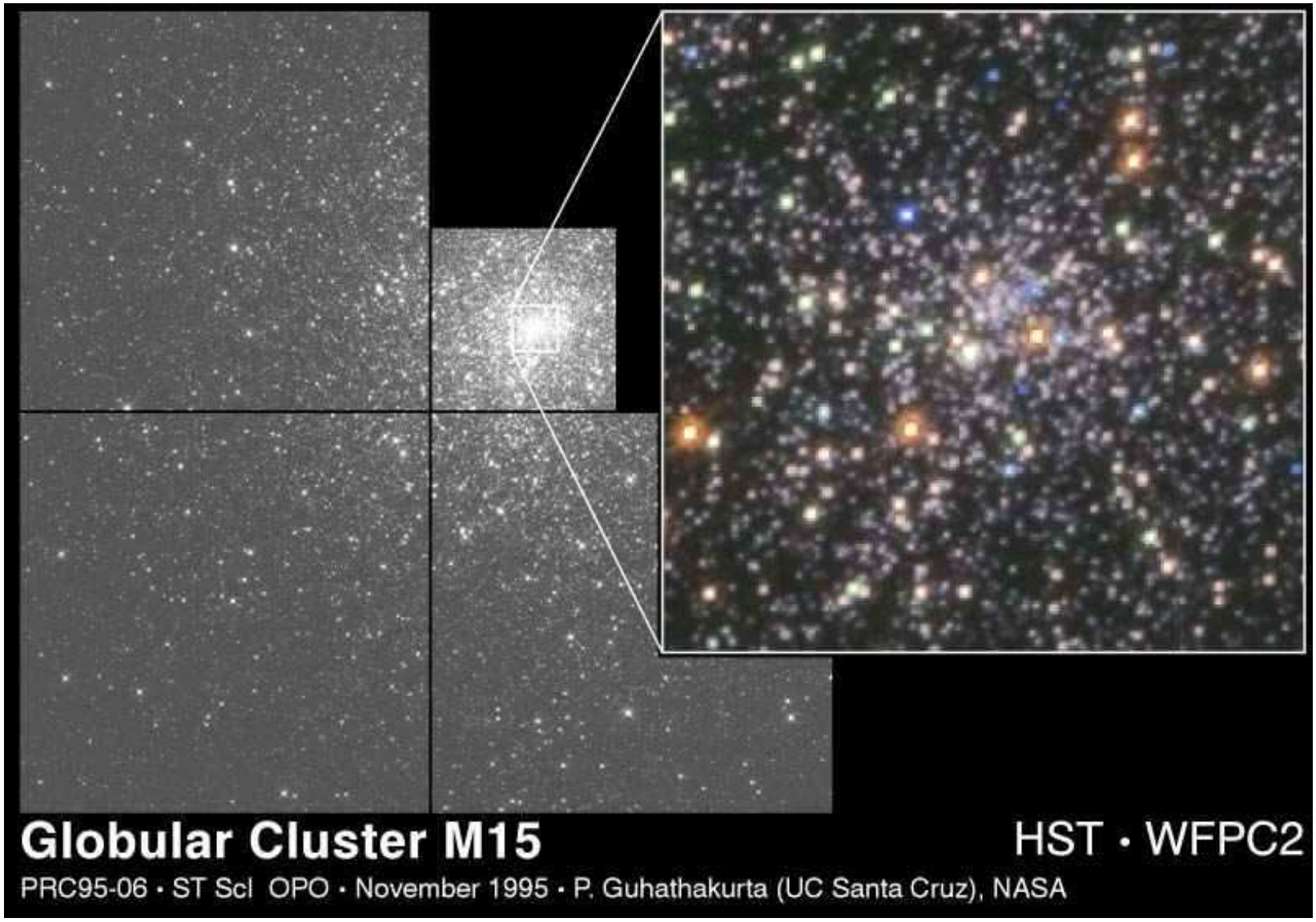
天文学における「シミュレーション」の役割

- 観測の解釈 ── 理論＋モデル計算
  知られている物理法則から天体現象を理解する
- もうちょっと原理的な問題
  カオス
  熱力学

観測の解釈の例

球状星団の中心にはなにがあるか？

- 非常に古い星の集団（100万個前後)
- 現在の形:ほぼ球形。全体としては回転していない
- 「熱力学的に緩和している」


Globular Cluster M15
HST • WFPC2
PRC95-06 • ST ScI OPO • November 1995 • P. Guhathakurta (UC Santa Cruz), NASA

球状星団 M15

- 「コア崩壊型」星団 — 星の数密度が中心までべき (半径の -1.8 乗) で増加
- 中心にブラックホールがある？

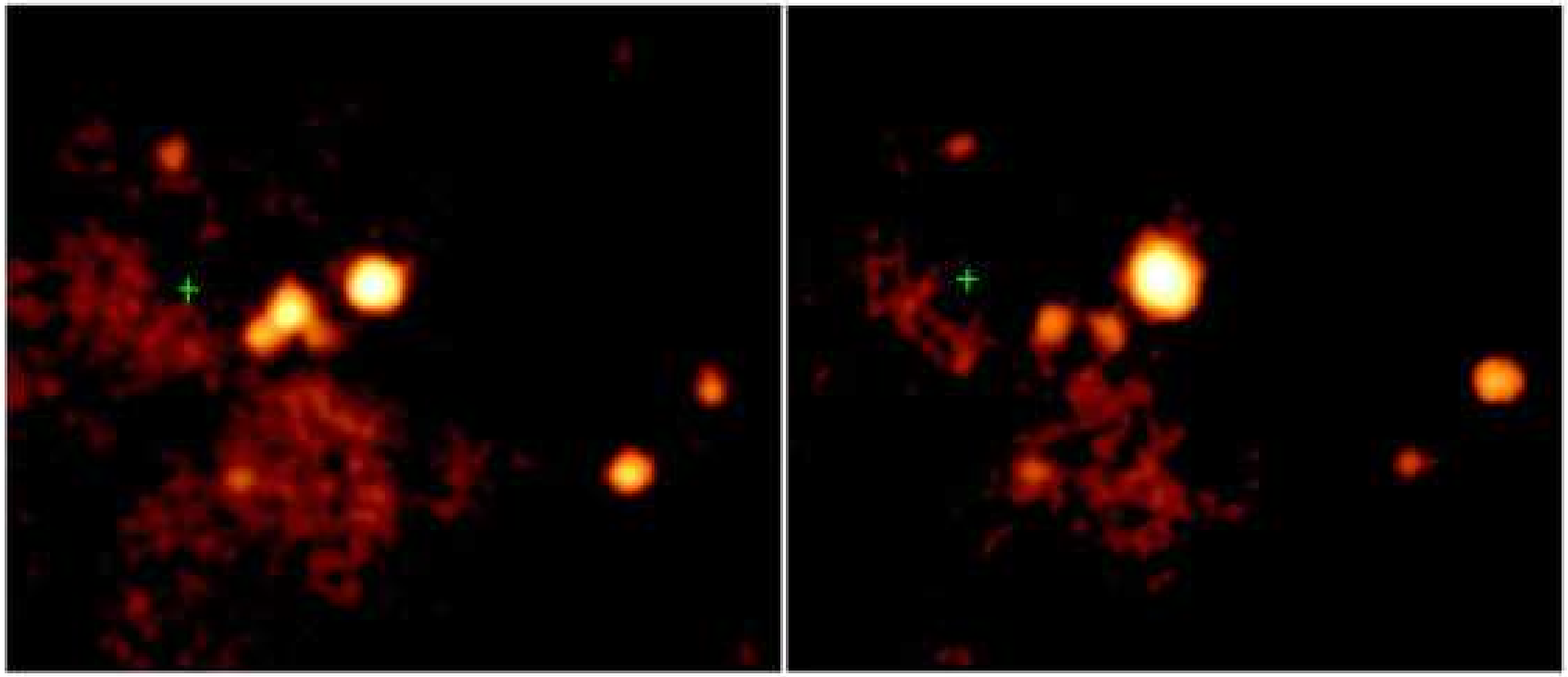
ブラックホールの見つけ方

ブラックホール — それ自体は光っていない。

- ブラックホールに落ち込むものが光る (主に X 線で)。それを見る (牧島さんあたり) 明るさに対応する質量があるはず
- ブラックホールの近くの星やガスの運動を見る。速度に対応するだけの質量があるはず

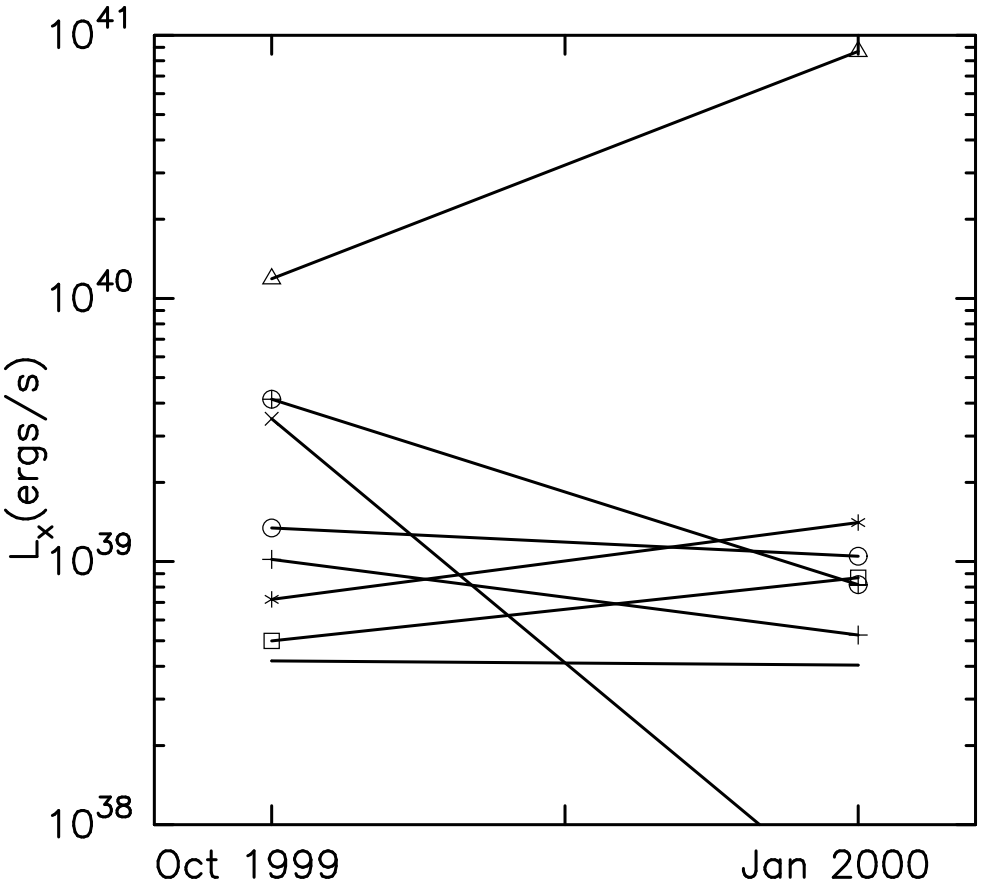
X線で光る

Matsumoto *et al.* ApJL 547, L25



Multiple point sources with high variability.

Variability



The strongest source shows significant variability.

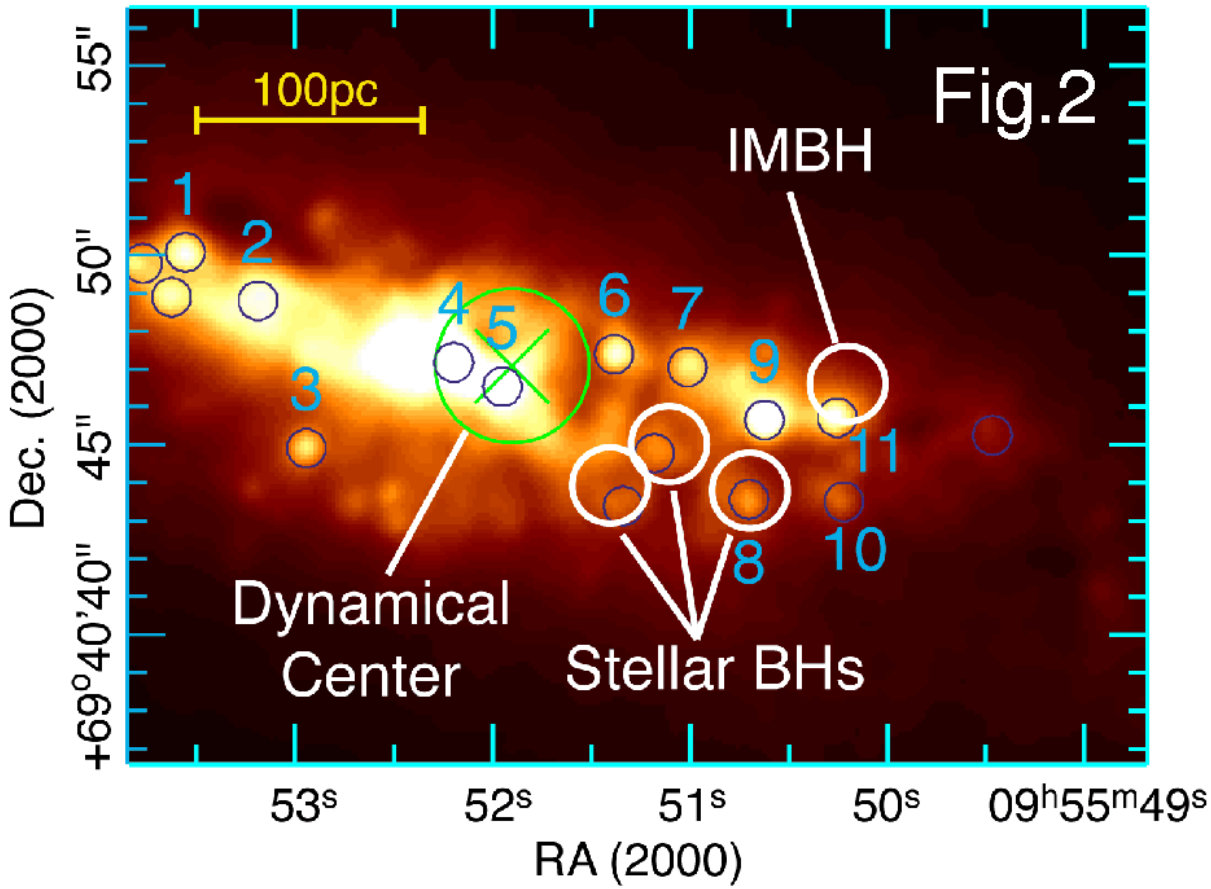
Implication of M82 IMBH (candidate)

- **First** candidate BH with mass $>>$ 10, but (most likely) $<<$ 10^6
- Eddington mass $\sim$ **700**$M_\odot$ = First IMBH (intermediate-mass BH).
- Off-center: Some 200 pcs away from the dynamical center of M82
- Center of the expanding shell of molecular gas (Matsushita *et al.* ApJ 2000, 545, L107)

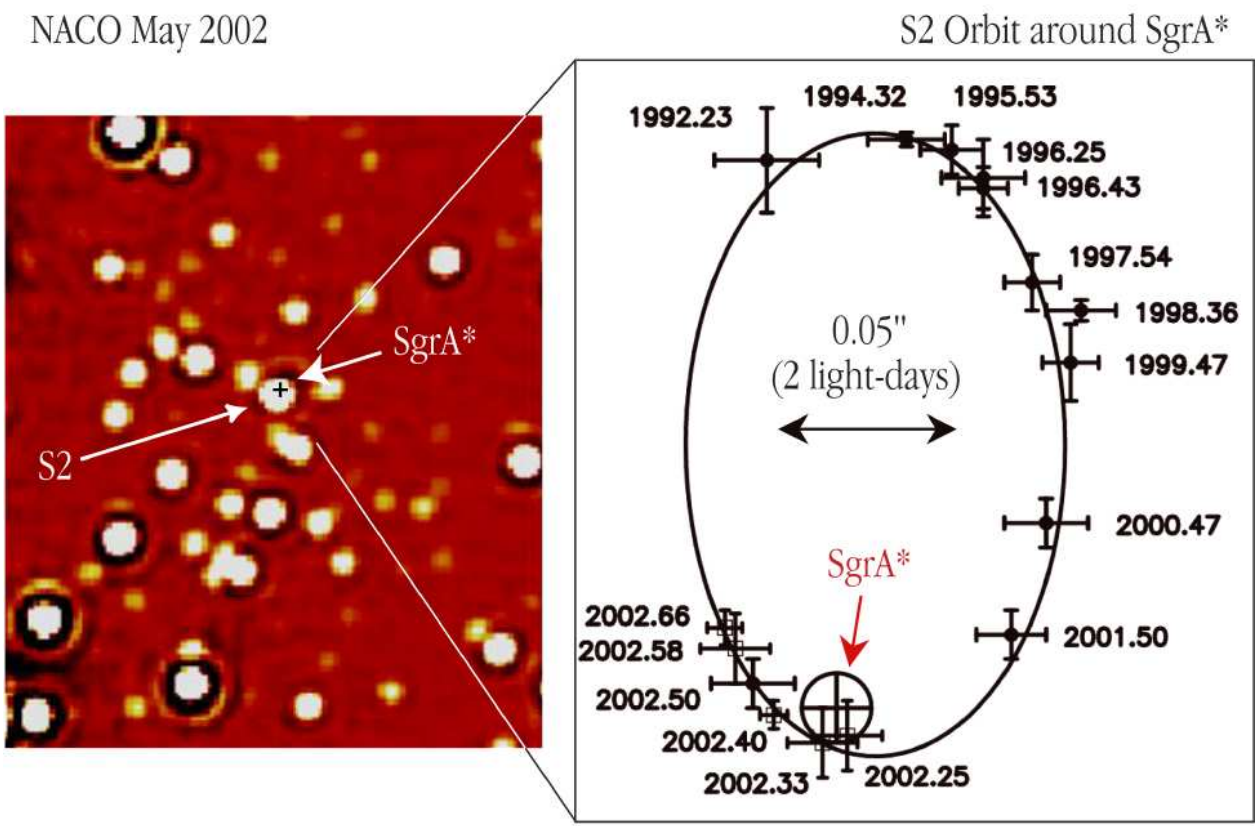
IR Counterpart

SUBARU IRCS view of the center of M82 (K' band)

still unpublished...



星の運動



The Motion of a Star around the Central Black Hole in the Milky Way

ESO PR Photo 23c/02 (9 October 2002)

© European Southern Observatory

理論/シミュレーションの役割

今までの例くらい確かなもの：あるかどうかの判定にはあんまり理屈はいらない

理屈が必要:

- もうちょっと怪しいもの
- どうやって出来たかを考える

怪しい例

昨年 9 月、ハッブル宇宙望遠鏡による「球状星団 M15 におけるブラックホールの発見」

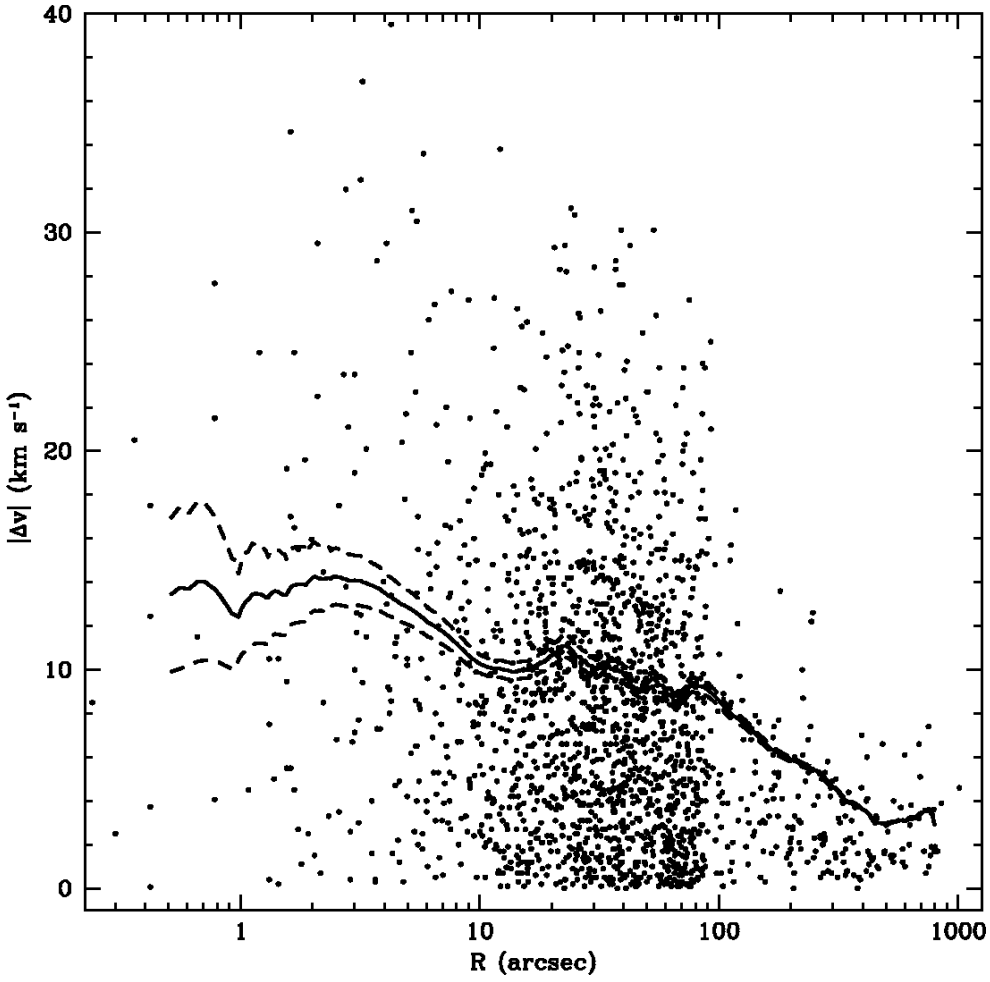
M15 — Claims

STScI Press release (Sept 18, 2002)

Van Der Marel led a team that uncovered a black hole in the center of the globular star cluster M15, 32,000 light-years away in the constellation Pegasus. His collaborator Joris Gerssen, also of the Space Telescope Science Institute, pinned down the black hole's mass at 4,000 times that of our Sun.

Previously, X-ray observations ..... have identified ultra-bright X-ray sources that could also be interpreted as intermediate-mass black holes in star-forming galaxies. However, alternative interpretations for these X-ray sources continue to exist. By contrast, Hubble's measurements are based on the velocities of stars .....which yield a direct measurement of the black hole masses.

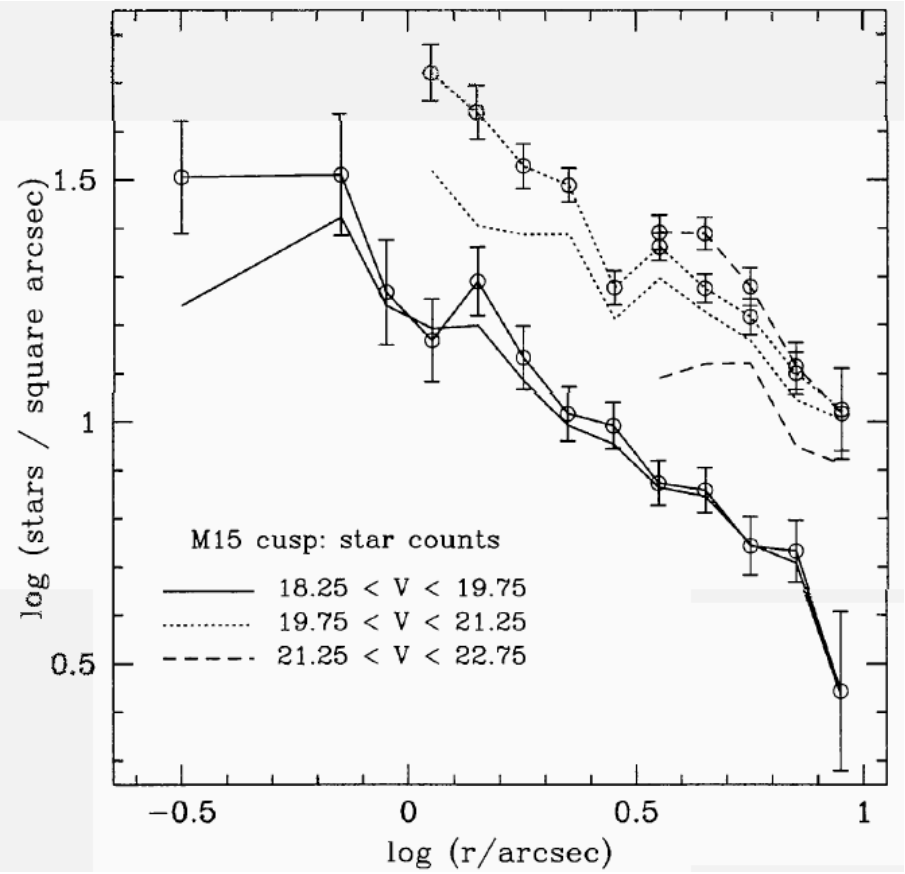
M15 — Observation

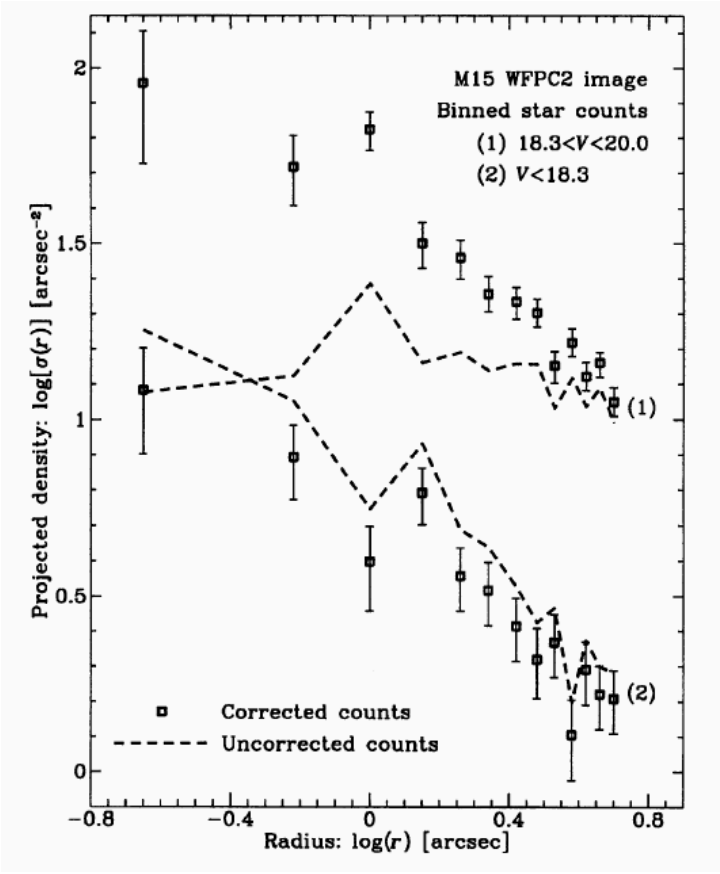


RMS velocity profile by Gerssen *et al.* (astro-ph/02090315).

HST STIS, CFHT and other data combined.

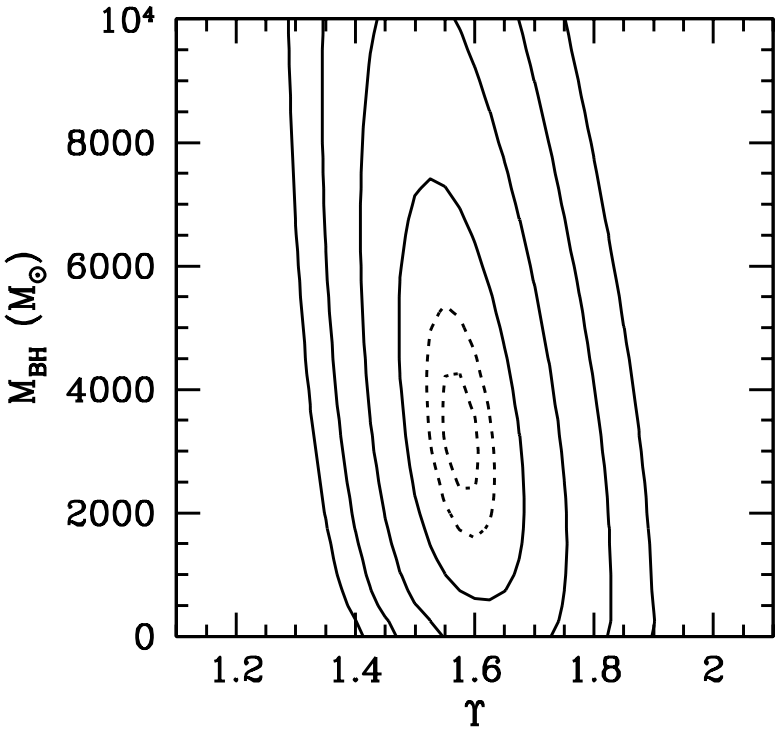
Surface density

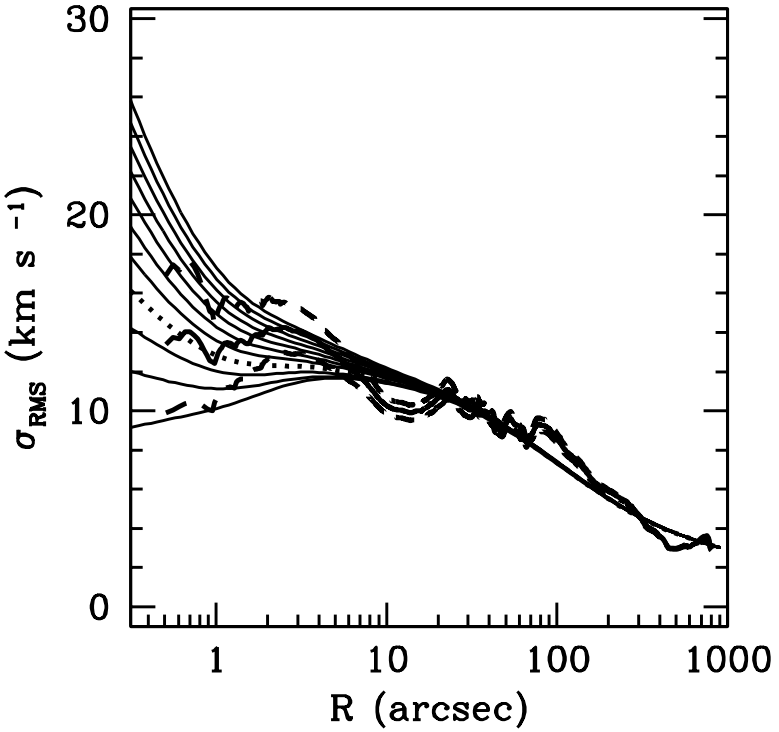




Left: Sosin & King 1997, Right: Guhathakurta *et al.* **1996.**

The black hole found?





3000 $M_\odot$ black hole!

何故ブラックホール?

実際に観測された速度分散は、中心5秒角の範囲で (わずかではあるが) 中心にいくに従って増加

中心5秒角内の表面輝度はスロープ -0.8 のべき (カスプ)

↓

質量と光が比例してれば、速度分散も弱いべきではあるが中心に向かって下がるはず

なので、ブラックホールがある

明らかな問題:

質量と光は比例するのか?

速度分散は等方的なのか?

質量と光の関係

常識的な理解: 球状星団では質量と光の関係は「単純ではない」

球状星団の年齢: 熱力学的な緩和時間よりずっと長い
重い星は、エネルギーを失って中心に集まることができる

この時、十分に進化が進むと中心部はほとんど重い星だけになり、そこの分布はべき乗 $\rho \sim r^{-2.23}$ になる。

球状星団のもっとも重い星: 中性子星 $(1.4\ \mathrm{M}_{\odot})$、重い白色矮星 (見えない)
観測している星: 赤色巨星 $(0.8\ \mathrm{M}_{\odot})$

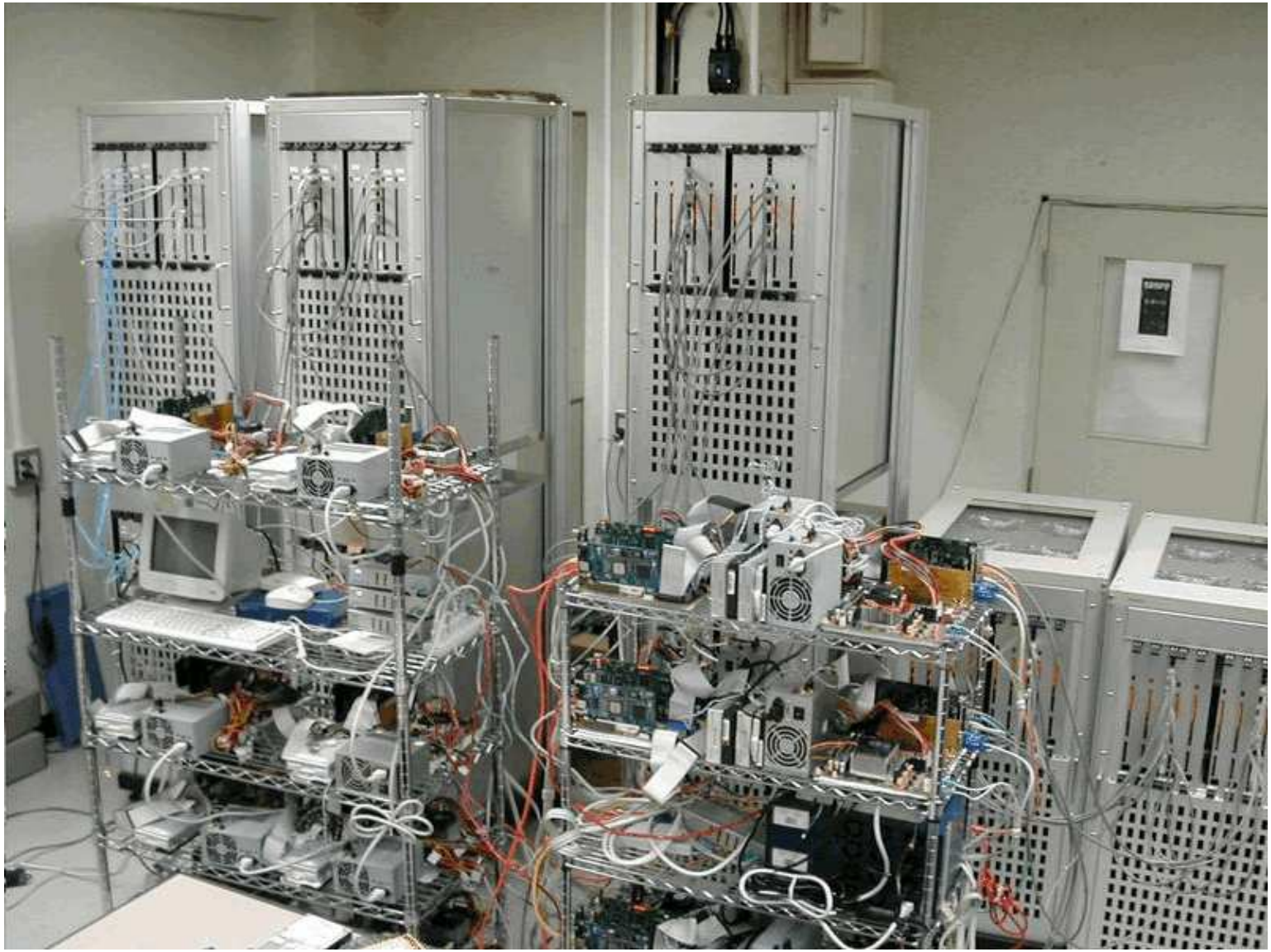
数値シミュレーションでもうちょっと細かくみてみる。

N-body simulation

Baumgardt and Makino (MNRAS accepted)

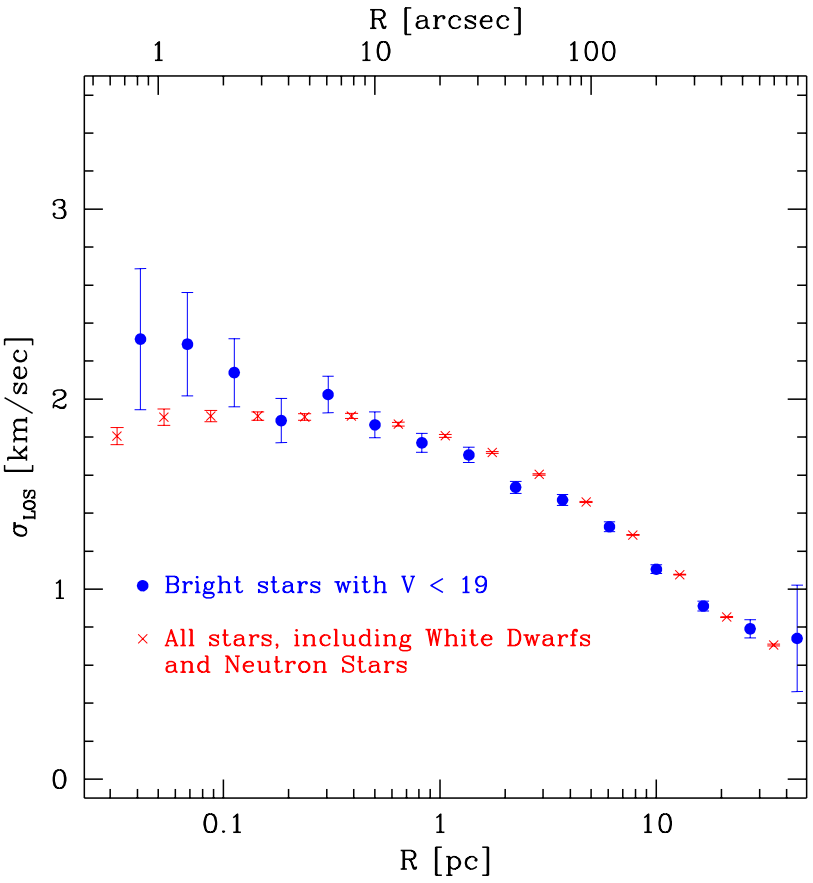
- **N=128k (mass at core collapse = $2\times10^4 M_\odot$ ($\sim$1/40 of M15)**
- **Kroupa (2001) IMF (15 $M_\odot$ cutoff)**
- **Hurley *et al.* (2000) Stellar evolution**
- **NBODY4 on GRAPE-6**
- **$W_o = 7$ King model at 8.5kpc from GC**
- **reached core collapse at 12.6 Gyrs**
- **40 days on a 4-Tflops GRAPE-6.**

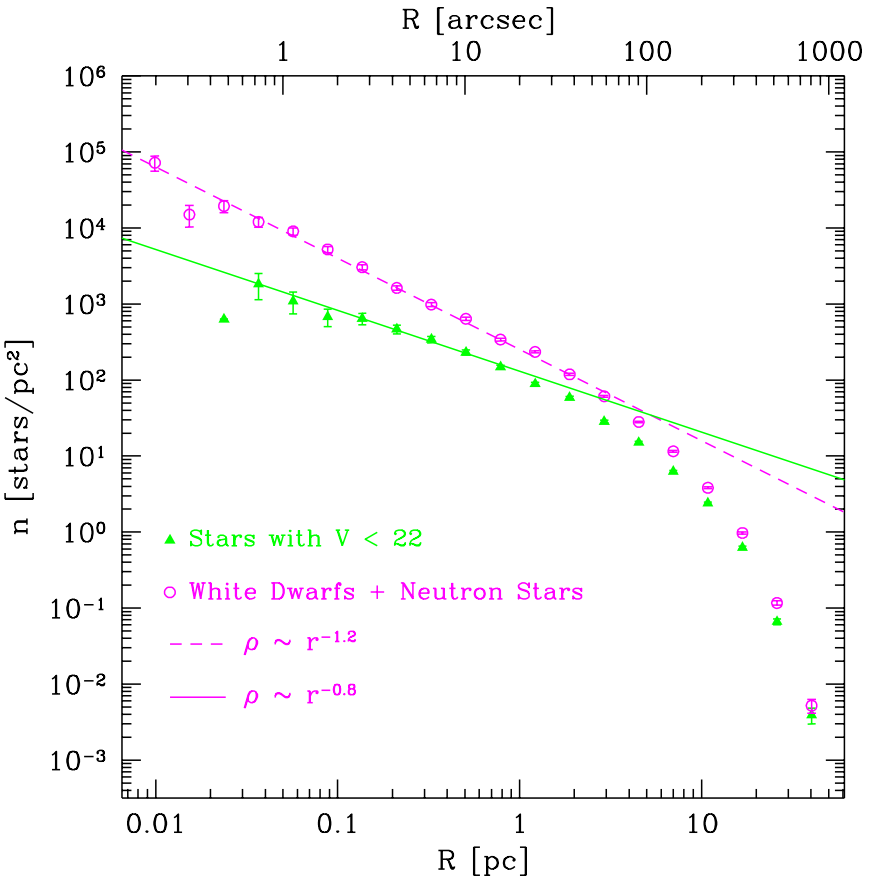
GRAPE-6



Results

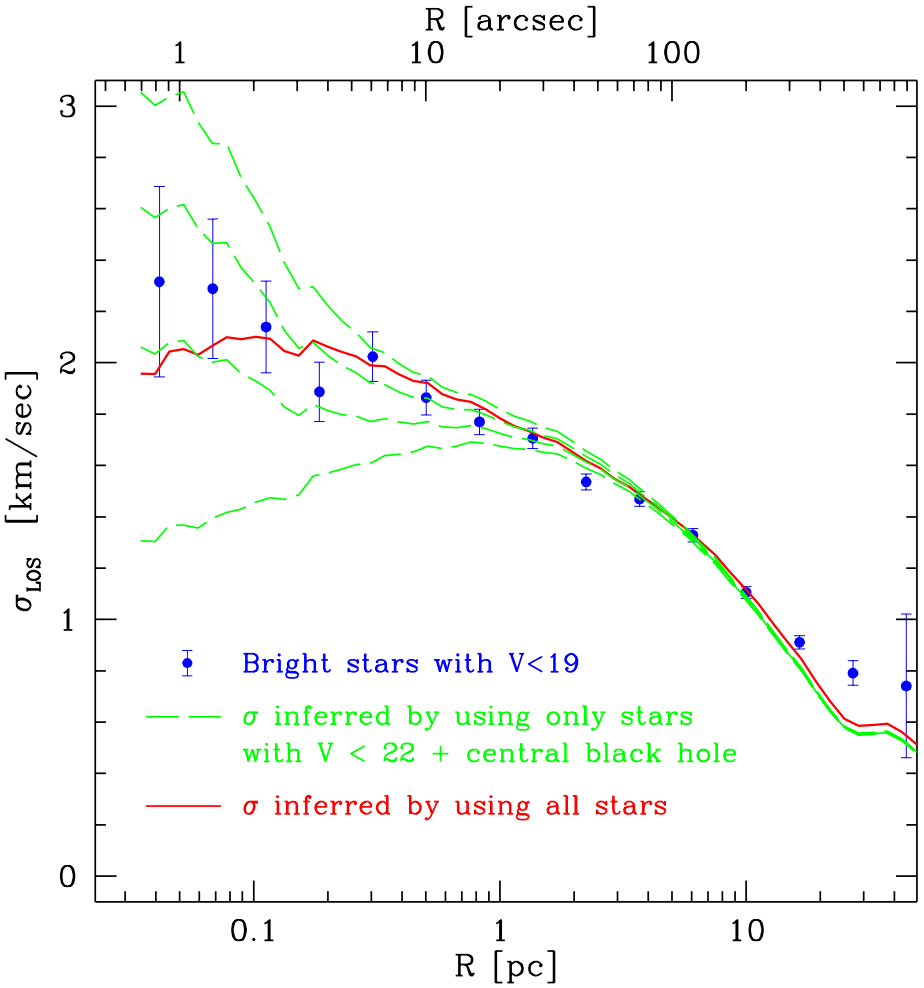
astro-ph/0210133, ApJ accepted.





Left: velocity dispersion; Right: Surface density.

And we "found" BH...



Inversion of surface number density of bright stars gives too small central velocity dispersion without central BH.

Black hole mass $= 80 M_{\odot}$.

If scaled to M15,

$\sim 3 \times 10^3 M_{\odot}$ **(Gerssen** *et al.*: $\sim 3 \times 10^3 M_{\odot}$**)**

Why we "found" BH?

We "found" BH because we assumed constant M/L.

If we use the true M/L, of course we do not need BH.

もうちょっと原理的な問題

重力だけで相互作用する質点の集まりには一体なにが起きるのか？

粒子数 $N = 2$: ニュートンが解決
$N > 2$ ポアンカレが「一般には解析解はない」ことを示した

球状星団：$N \sim 10^6$
銀河: $N \sim 10^{11}$
なにが起きるか？
数値計算なら、初期条件を与えれば答はでる。

計算すればなんでもわかるか？

原理的には答は **YES**。
現実の問題としては **NO**。

- 計算機の能力
- 素過程の理解

例:銀河の進化

典型的な銀河: 1000 億個くらいの星、それと同程度の質量のガス、その数倍の質量の「ダークマター」からなる (ということになっている)

- 1000 億体問題は今の計算機では全く無理
- 1000 億体問題が解けても、それだけでは駄目
  - ガスから星ができる過程
  - 超新星爆発等で星がガスに戻る過程
  - ダークマターはどうするか

重力多体系

- 基礎方程式
- 平衡解
- 熱力学的な進化

重力多体系の基礎方程式

もとの方程式自体はもちろん、各粒子の運動方程式

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

数値計算にはもちろんこれを使うわけだが、理論的な扱いには不便

というわけで、しばらくは (1 粒子) 分布関数 $f(x, v, t)$ で話をする。

この時の基礎方程式:(無衝突) ボルツマン方程式

(無衝突) ボルツマン方程式

$$\frac{\partial f}{\partial t} + v \cdot \nabla f - \nabla \Phi \cdot \frac{\partial f}{\partial v} = 0, \tag{2}$$

f:6 次元位相空間での分布関数 Φ :重力ポテンシャル, 以下のポアソン方程式の解

$$\nabla^2 \phi = -4\pi G \rho. \tag{3}$$

ここで、 G は重力定数であり、 ρ は空間での質量密度

$$\rho = m \int dv f, \tag{4}$$

なお、以下の議論では (当分) m のことは忘れて、その代わり f が個数密度ではなくて質量分布であるということにしておく。

ボルツマン方程式の導出

$\mathbf{BBGKY}$ とか使った導出はなんか見れば書いてあるし、良く知らないのでパス。

直観的な意味:

$$\frac{\partial f}{\partial t} + v \cdot \nabla f - \nabla \Phi \cdot \frac{\partial f}{\partial v}$$

要するに 6 次元位相空間でのラグランジュ微分 Df/Dt で、これが $= 0$: 非圧縮での連続の式

力学平衡

「力学平衡」$=$ 上の無衝突ボルツマン方程式とポアソン方程式を連立させたものの定常解

ある分布関数 f が力学平衡にある$=$ それによって決まるポテンシャル Φ を固定して考えた時に、f の時間微分が 0 になるということである。

普通の流体なら静水圧平衡にあたる。

当然であるが熱平衡とは限らない

運動の積分

ポテンシャル Φ のもとで、ある x, v の関数 I が運動の積分であるとは、その上で

$$\frac{d}{dt}I(x,v) = 0, \tag{5}$$

がなり立つこと。

つまり、実際にすべての粒子の軌道について、その上でその量が変化しないということ

ちょっと変形すれば

$$v \cdot \nabla I - \nabla\Phi \cdot \frac{\partial I}{\partial v} = 0 \tag{6}$$

無衝突ボルツマン方程式の定常解と同じ式

ジーンズの定理

I について以下の「ジーンズの定理」がなり立つ。

ジーンズの定理 任意の無衝突ボルツマン方程式の定常解は、運動の積分を通してのみ位相空間座標に依存する。逆に、任意の運動の積分の関数は定常解を与える。

いいかえると、分布関数 f が定常であるためには、運動の積分 $I_1, I_2, ..., I_m$ があって $f = f(I_1, I_2, ..., I_m)$ の形で書けることが必要十分ということ。

球対称の場合

球対称の場合、運動の積分はエネルギーと角運動量の 3 成分で 4 つ。
一般にはもう一つあるが、これは特別な場合を除いてあまり意味がないので、定常な分布関数はエネルギーと角運動量だけで書けると思っていい。
意味がある特別な場合: ケプラー軌道のような、軌道が閉じる場合。
この時には、エネルギーと角運動量の他に、軌道全体の向きを表す量 (近点経度) が保存する。これはちゃんと保存量になっている。

$f(E, J)$

f は E と $\mathbf{J}$ によるということにしたわけだが、いま球対称な場合ということなので $\mathbf{J}$ の方向にではなく、絶対値だけに依存するのでないといけない。したがって、実は球対称の分布関数は一般に $f(E, J)$ と書けるということになる。
我々が扱いたいのは自己重力系なので、実際にポアソン方程式を球対称の場合に書き下してみると

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Phi}{dr}\right) = 4\pi G \int f\left(\frac{1}{2}v^2 + \Phi, |r \times v|\right) dv, \qquad (7)$$

てな感じになる。

$f(E)$ の場合

$f(E, J)$ の場合でもまだちょっと大変なので、さらに単純化してとりあえず J にもよらない場合というのを考えてみる。

この場合空間上の各点で速度分散が等方的

球対称な分布関数の例

ここであげるのはあくまでも例であるが、さまざまな理由から
その性質がよく調べられているものである。

- ポリトロープ
- 等温解

ポリトロープとプラマーモデル

ある意味でもっとも簡単な分布関数の例は、$\mathcal{E}$の冪乗(パワー)で書けるものである。例えば

$$f(\mathcal{E}) = \begin{cases} F\mathcal{E}^{n-3/2} & (\mathcal{E} > 0) \\ 0 & \mathrm{otherwize} \end{cases} \tag{8}$$

これから、まず密度を Ψ の関数として求められる ($v^2 = 2\Psi\cos\theta$ なる変数変換のあと)。で、答えは

$$\rho = c_n \Psi^n \quad (\Psi > 0) \tag{9}$$

となる。ただし、 c_n が有限になるためには $n > 1/2$ でないといけない。

上を使ってポアソン方程式から ρ を消去すると

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Psi}{dr}\right) + 4\pi G c_n \Psi^n = 0 \tag{10}$$

変数を適当にスケーリングして

$$\frac{1}{s^2}\frac{d}{ds}\left(s^2\frac{d\psi}{ds}\right)+\psi^n=0 \tag{11}$$

としたものを **Lane-Emden** 方程式と呼ぶ。

実際には、上の **Lane-Emden** 方程式を解かないとポテンシャルや密度がどうなっているかはよくわからない。

プラマーモデル

一般の n ではこの方程式には初等的な解はないが、 $n = 5$ の場合には解があることが古くから知られている。これは

$$\phi = \frac{1}{\sqrt{1 + \frac{1}{3}s^2}} \tag{12}$$

の形をしている。密度は $c_5\phi^5$ で与えられることになる。

密度が $r = 0$ で有限で、$r \to \infty$ で $1/r^3$ より速く落ちるので、質量は有限である。

これは、天文学的になにか素晴らしいものであるというわけではないが、球状星団のうち中心密度が低いものにはまあまあ似ていなくもない。とりあえず、これの意味は、解析関数で簡単に書ける自己重力系の self-consistent なモデルであるということである。

等温モデル

熱平衡状態では(古典統計なので)分布関数はマックスウェル–ボルツマン分布、すなわち

$$f(\mathcal{E}) = \frac{\rho_1}{(2\pi\sigma^2)^{3/2}} e^{\mathcal{E}/\sigma^2} = \frac{\rho_1}{(2\pi\sigma^2)^{3/2}} \exp\left(\frac{\Psi - v^2/2}{\sigma^2}\right) \tag{13}$$

で与えられなければならない。

等温モデル(続き)

まず、例によって分布関数を速度空間で積分して密度をポテンシャルの関数として表す。この時に誤差関数についての

$$\frac{2}{\sqrt{\pi}}\int_0^\infty e^{-x^2}dx = 1 \tag{14}$$

を使うと、

$$\rho = \rho_1 e^{\Psi/\sigma^2} \tag{15}$$

ポアソン方程式にこれを入れると

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Psi}{dr}\right) = -4\pi G\rho \tag{16}$$

従って、

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\log\rho}{dr}\right) = -4\pi G\sigma^2\rho \tag{17}$$

等温モデル(続き2)

まず、一つ特別な解があるということを指摘しておく

$$\rho = \frac{\sigma^2}{2\pi G r^2} \tag{18}$$

は、解の一つとなっている。(singular isothermal sphere)

これは self consistent ではない: 質量が $M_r \propto r$ となって発散

渦巻銀河の、「回転速度が中心からの距離に(あまり)依存しない」(いわゆる flat rotation curve)という性質: $\rho \sim 1/r^2$ のダークハローを示唆(?)

特別ではない解は、中心密度を有限にして中心から外側に向かって解いていけばいい。この時でも、 $r \to \infty$ の極限では singular isothermal に近付く。

熱平衡状態の非存在

重力多体系に統計力学 / 熱力学を適用する際の基本的な問題 :
「重力多体系には熱平衡状態がない」ということ

「熱平衡状態がない」というのはそもそもどういうことか？

示し方はいろいろ、、、以下は１例

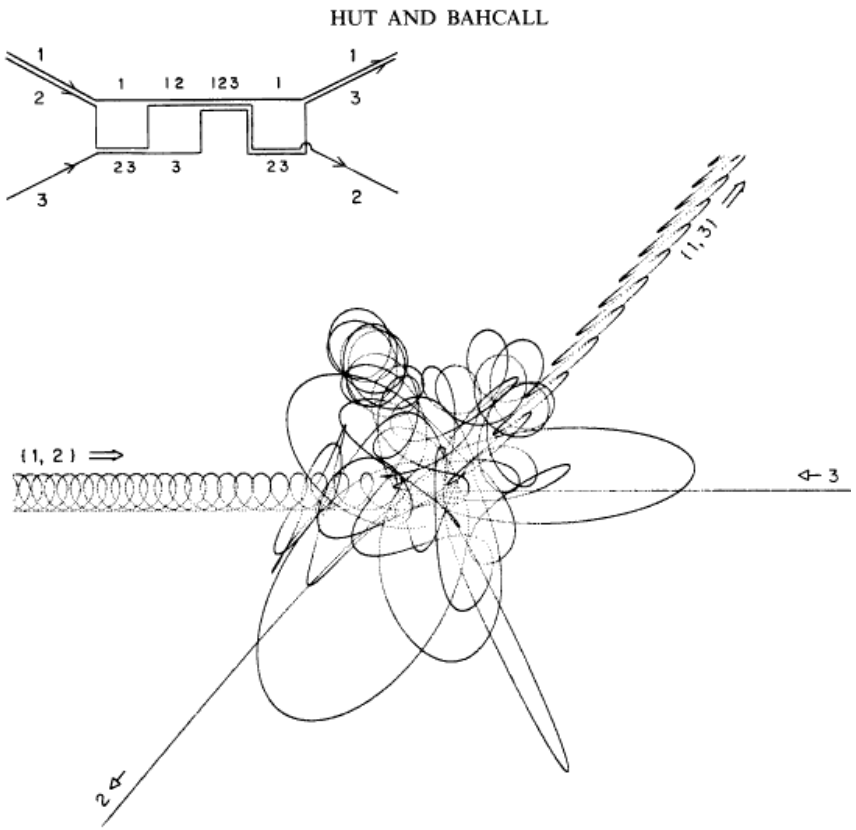
熱平衡状態の非存在(続き)

古典統計である
したがって、速度分布関数はマックスウェル・ボルツマンでなければならない
これは不可能
自己重力系にある粒子のエネルギーは、定義からして無限遠での重力ポテンシャルの値を超えない
したがって、ある空間のある点での速度分布には必ず上限がある。
言い換えると、上限を超えた粒子は逃げていってしまう、、、

例 — 重力3体問題

2体問題では周期解になるだけで何も起きないので、3体問題。

重力3体問題の解の一例



重力3体問題(2)

全系のエネルギーが負で、全体としては自己重力で集まっている適当な初期条件(前の例では2つは連星というちょっと特別な例)から運動させる

しばらくは3つがそれぞれお互いの回りを運動する

そのうちに、粒子のうちの2つが強く結合した状態になり3つめがその反作用ではね飛ばされるという現象が起きる。

3体問題としては、これが「最終状態」

重力多体系の最終状態

粒子数が多い系でも、本質的には同じことが起きる。

つまり、粒子同士の散乱の結果、高エネルギーの粒子が作られるとそれは系から逃げていってしまうのである。

というわけで、統計力学からいえること：

重力多体系は、十分長い時間が立てば蒸発してしまう

しかし、これは間違いではないけど、でも、現実に存在している重力多体系の理解には役に立たない。

それでも、熱平衡状態について

まあ、それでも、やっぱり、熱平衡というものを考えたい

- いきなり非平衡とかいわれてもわけがわからない
- ある種の重力多体系では、熱平衡からそんなに遠くない状態が実現されている

というわけで、熱平衡状態がどういうものかを理解しておくことは極めて重要。

とはいえ、厳密には熱平衡はない。

熱平衡状態を作る

問題 : どうやって、「本質」を失わないで熱平衡がある系に持っていけるか？

- 空間の次元を落す。
  空間 1 次元にすると、熱平衡がある。(粒子間ポテンシャルが無限大で発散するので)
- 空間は 3 次元のままで、エネルギー無限大でも逃げないようにする (壁をつける)

どちらもいろいろな人がいろいろなことをしている。

熱平衡状態を作る (2)

1 次元系の話はいろいろ面白いが、今日は省略。興味がある人は、
Tsuchiya, Konishi and Gouda, PR, E50, 1996, 2210
Makino, J., astro-ph/9903238
などを見ること。

今日 1 次元系の話をしないのは、結局 1 次元系では 3 次元系で起きるいろいろな重要な物理が失われるから。

壁をつけるのは、「熱平衡がある」という他はあまり大きく現実と変わらないので、以下こっちの話。で、さらに話を簡単にするために壁は球対称とする。

熱平衡分布の熱力学的安定性

以下、熱平衡分布が安定かどうかを考える。
で、とりあえず流体の場合を考える。流体だと普通に温度や圧力、エントロピーとか考えられて、熱力学的な取り扱いができるから。
何に対して安定か : 熱 (エネルギー) の再分配を与えるような摂動に対して。
普通の統計力学では、平衡状態は安定に決まっている。
どこかから熱をとって、別のところに与える → とられたところは温度が下がり、もらったところは温度が上がる → とられたところには回りから熱が流れ込む、もらったところは流し出す → 等温に戻る。

自己重力系の熱力学的安定性

重力が働くと、そうとは限らない：

ここまでで求めた等温平衡状態：中心では重力をささえるために密度が高い。外側にいくほど密度が下がる。

↓

熱を奪われると、圧力が下がって少し収縮する

↓

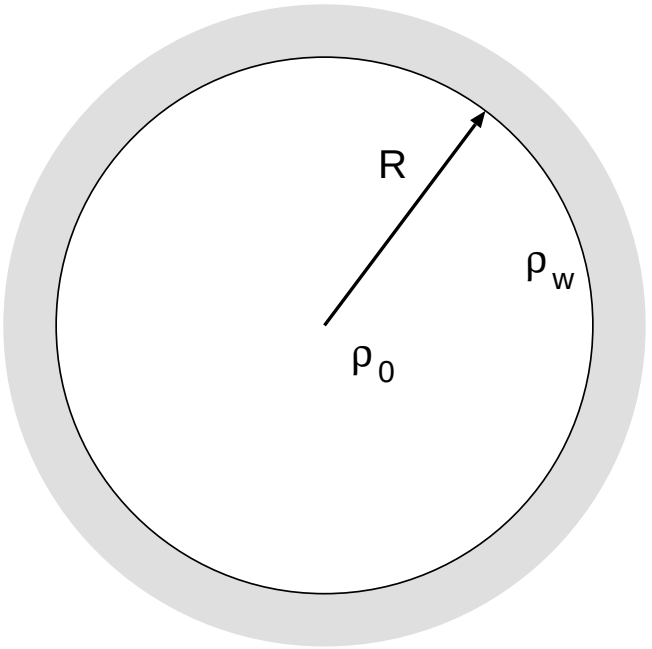
重力が強くなるので、もうちょっと収縮する

↓

回りより温度が上がる

ということが起こり得る。

熱力学的安定性（続き）



熱をとられて温度が下がる
＝「見かけの」比熱が負である

以下、線形安定性解析の結果を
見ていくことにする。
改めて、絵を書いておく。
半径 R の中に質量 M のガスがあり、中心密度 ρ_0 と壁のところでの密度 ρ_w の比が D であるとする。
$D = 1$: 重力なし（温度無限大）
$D = \infty$: 特異解 $\rho \propto r^{-2}$

熱力学的安定性 (続き２)

熱力学的安定性の解析方法はいろいろ：

- **Linear series analysis (Chandrasekhar, Lynden-Bell and Wood 1968)**：D を変えた時のエネルギーの変化を見る。極小値があれば、そこから先は熱力学的に不安定。
- 摂動を与えて、適当な制約のもとでエントロピーの２次の変分を見る **(Hachisu and Sugimoto 1978)**
- 摂動を与えて、その時間発展を見る (普通の線形解析, **Makino and Hut 1991**)

以下、３番目の方法に沿って話す。

定式化

熱伝導の式：

$$K\frac{\partial T}{\partial r} = -\frac{L}{4\pi r^2} \tag{19}$$

$L(r)$：半径 r のところでの熱流束

K ：熱伝導の係数

仮定：

$$K = \rho^{\alpha} \tag{20}$$

ここで、一応重力多体系に似せる：$\alpha = 1$

エントロピーの式など

エントロピーについての式：

$$\frac{\partial L}{\partial r} = -4\pi r^2 \rho T \frac{\partial s}{\partial t}|_M. \tag{21}$$

境界条件：

$$L = 0 \quad \text{for} \quad M = 0 \quad \text{and} \quad M = 1. \tag{22}$$

線形化した方程式

微小な摂動に δ をつけることにして、線形化した方程式は

$$\frac{d\delta \ln p}{dM} = \frac{M}{4\pi p r^4}(\delta \ln p + 4\delta \ln r), \tag{23}$$

$$\frac{d\delta \ln r}{dM} = -\frac{1}{4\pi r^3 \rho}(3\delta \ln r + \delta \ln \rho), \tag{24}$$

$$\delta \ln p = \delta \ln \rho + \delta \ln T, \tag{25}$$

$$\delta s = \frac{3}{2}\delta \ln T - \delta \ln \rho, \tag{26}$$

$$\tag{27}$$

境界条件は

$$3\delta \ln r + \delta \ln \rho = 0 \quad \text{for} \quad M = 0 \tag{28}$$

$$\delta \ln r = 0 \quad \text{for} \quad M = 1 \tag{29}$$

線形化した方程式(2)

熱流束とエントロピーの変化については、もとの式が線形なのでそのまま使える。この方程式系に対して、以下の形をした解を捜す。

$$
\begin{aligned}
\delta \ln p &= \delta \ln p_0 e^{\lambda t}, \\
\delta \ln r &= \delta \ln r_0 e^{\lambda t}, \\
\delta \ln \rho &= \delta \ln \rho_0 e^{\lambda t}, \\
\delta \ln T &= \delta \ln T_0 e^{\lambda t}, \\
\delta L &= \delta L_0 e^{\lambda t}, \\
\delta s &= \delta s_0 e^{\lambda t}.
\end{aligned}
\tag{30}
$$

添字 0 がついたものは時間発展解の空間依存性を表す。

固有値問題

$$\frac{d\delta \ln p_0}{dM} = \frac{M}{4\pi p r^4}(\delta \ln p_0 + 4\delta \ln r_0) \quad , \tag{31}$$

$$\frac{d\delta \ln r_0}{dM} = -\frac{1}{4\pi r^3 \rho}(3\delta \ln r_0 + \delta \ln \rho_0) \quad , \tag{32}$$

$$KT\frac{d\delta \ln T_0}{dM} = -\frac{\delta L_0}{(4\pi r^2)^2 \rho_0} \quad , \frac{d\delta L_0}{dM} = -\lambda T \delta s_0 \tag{33}$$

$$\delta \ln p_0 = \delta \ln \rho_0 + \delta \ln T_0 \quad , \tag{34}$$

$$\delta s_0 = \frac{3}{2}\delta \ln T_0 - \delta \ln \rho_0 \quad . \tag{35}$$

が出てくる。

境界条件

$$3\delta \ln r_0 + \delta \ln \rho_0 = 0 \quad \text{for} \quad M = 0 \tag{36}$$

$$\delta \ln r_0 = 0 \quad \text{for} \quad M = 1 \tag{37}$$

$$\delta L = 0 \quad \text{for} \quad M = 0 \tag{38}$$

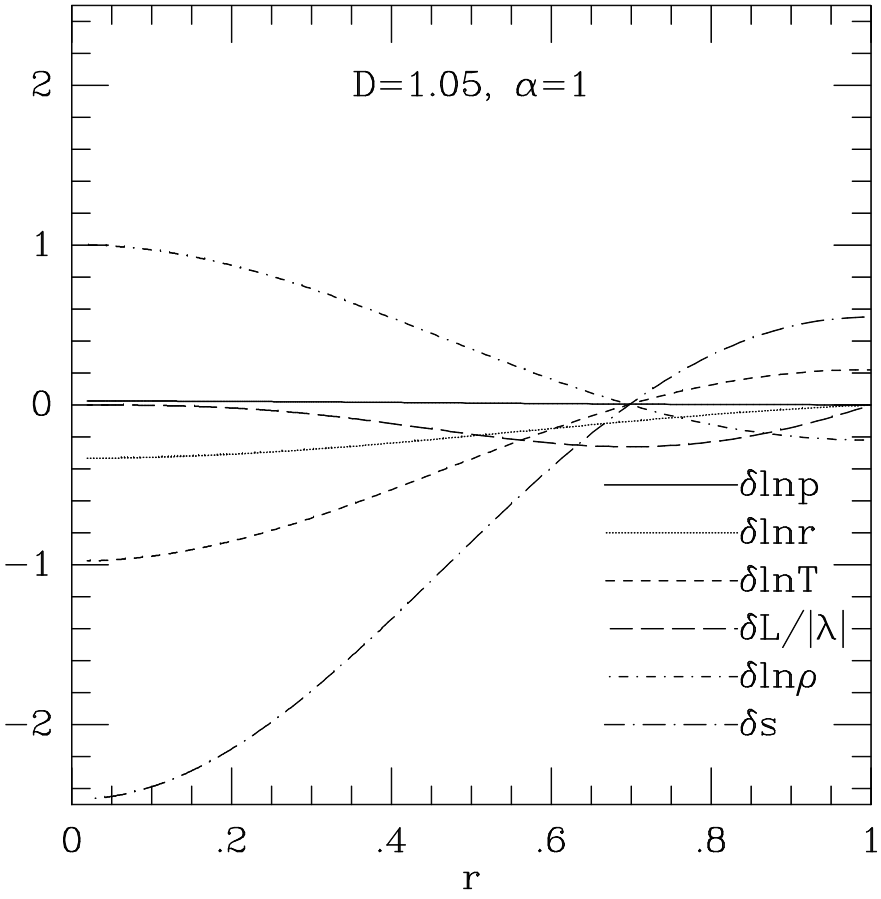
$$\delta L = 0 \quad \text{for} \quad M = 1 \tag{39}$$

ということになる。

解析結果 1

以下、結果。

安定な場合



λ: 固有値

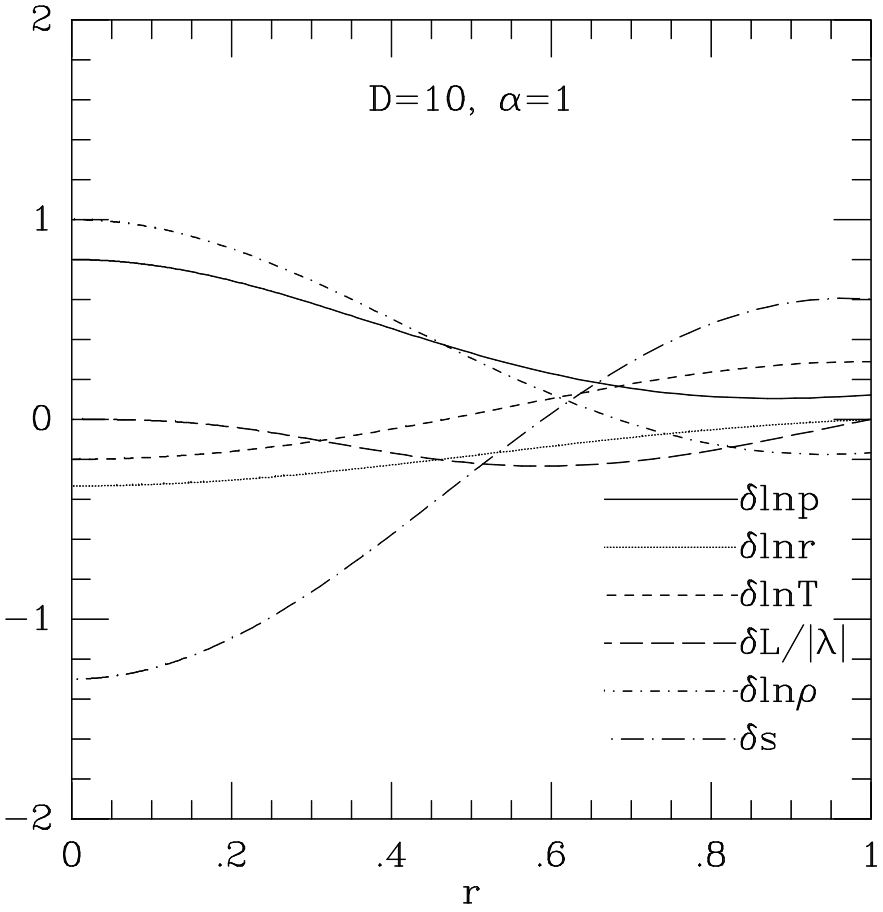
$D = 1.05$: 重力が無視できる場合。

- 圧力は変化しない
- エントロピーと温度が比例

要するに、普通の断熱容器のなかのガス。

解析結果 2

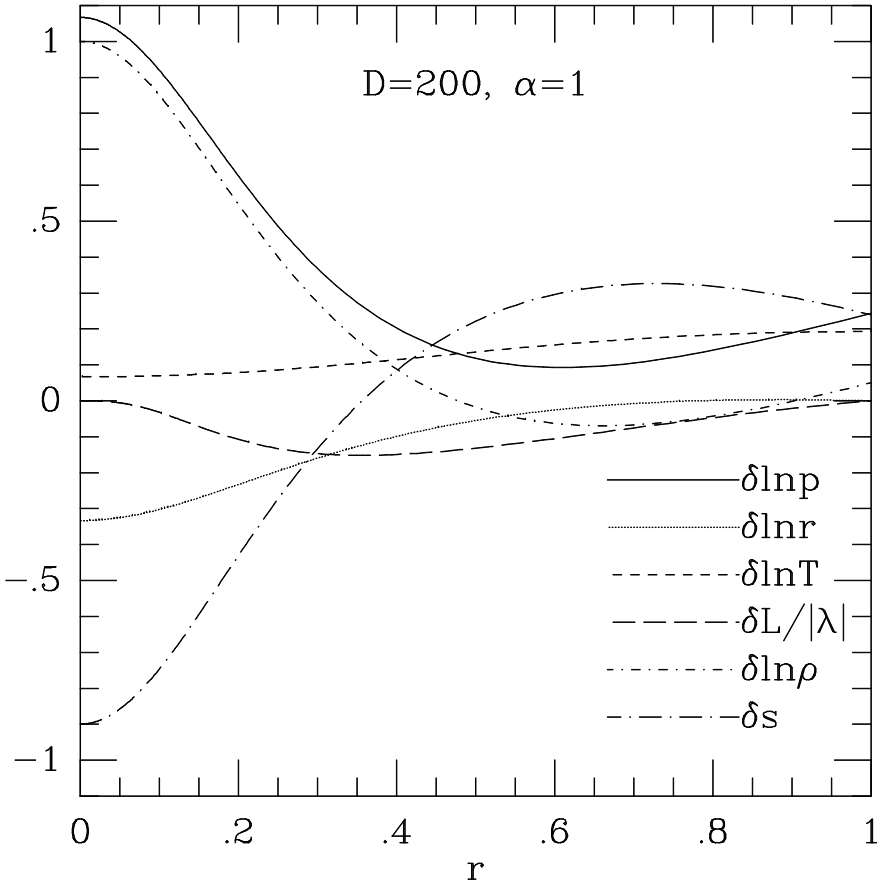
安定な場合 (2), $D = 10$



- 中心で圧力が上がる
- 温度は断熱変化の影響も受けるので、エントロピーとずれる

解析結果 3

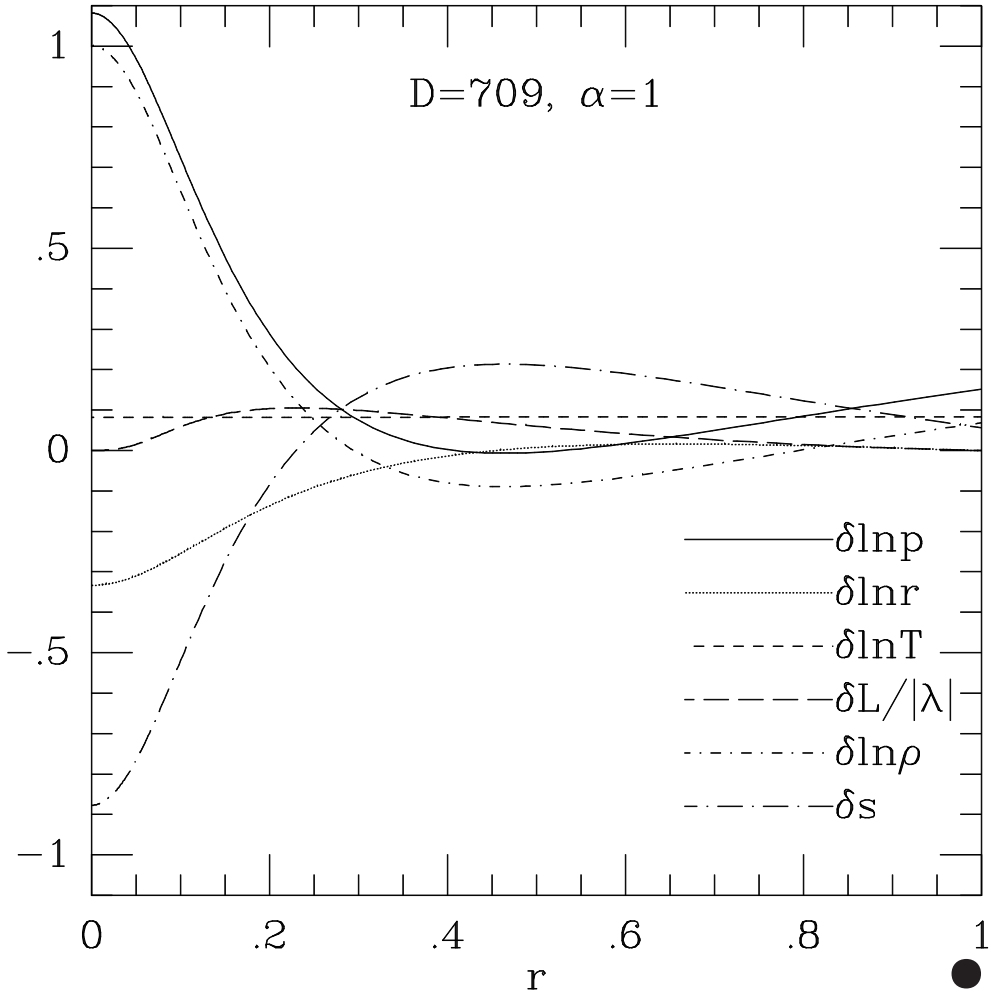
安定な場合 (3), $D = 100$



- 中心で温度も上がる
- 温度勾配はエントロピー変化を減らす向き (この場合中心の方が低温)
- 熱力学的には安定

解析結果 4

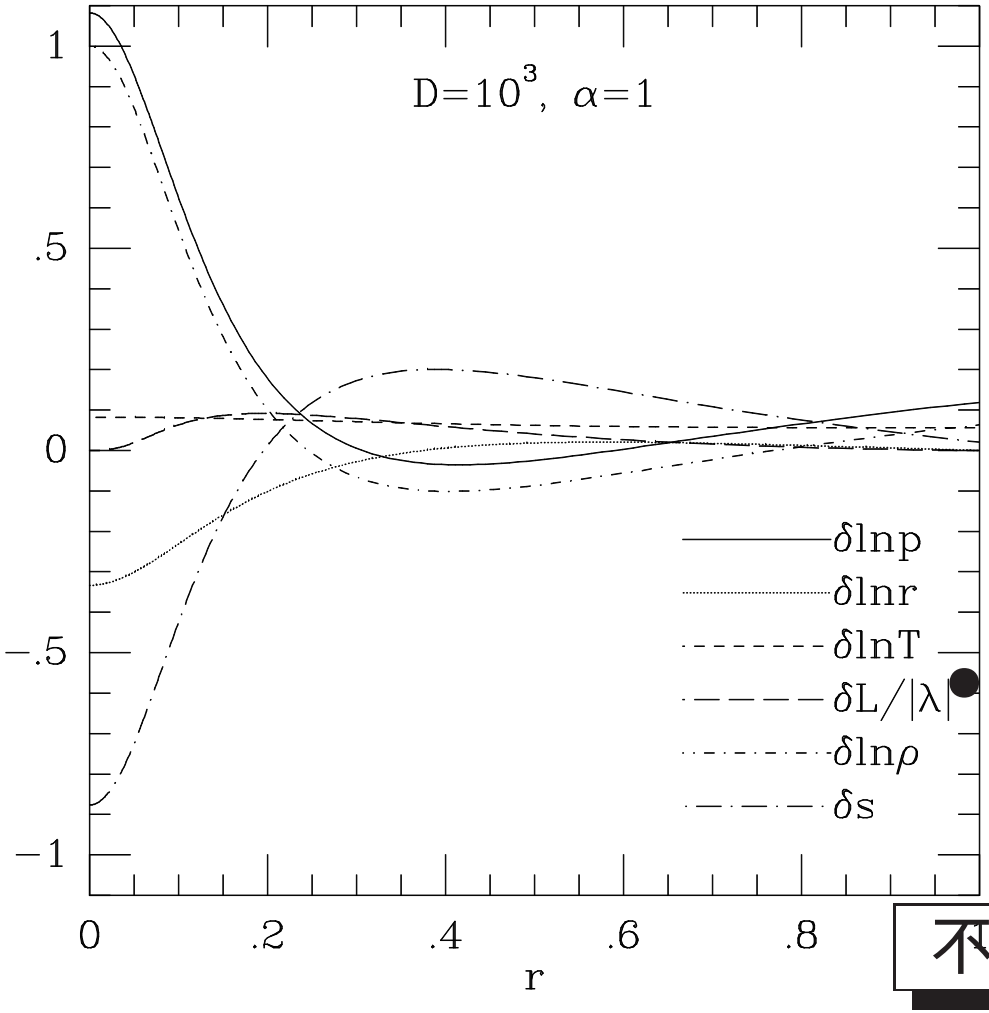
中立安定, $D = 709$



- 温度勾配ができない

解析結果 5

不安定, $D = 1000$



- 中心のほうが温度上昇が大きい

不安定になっている

結果のまとめ

というわけで、線形解析の結果：

> 断熱壁をつけて等温の平衡状態を作っても、重力が効いていると熱力学的に不安定

一応、「重力熱力学的不安定性」 **gravothermal instavility** という名前がついている。

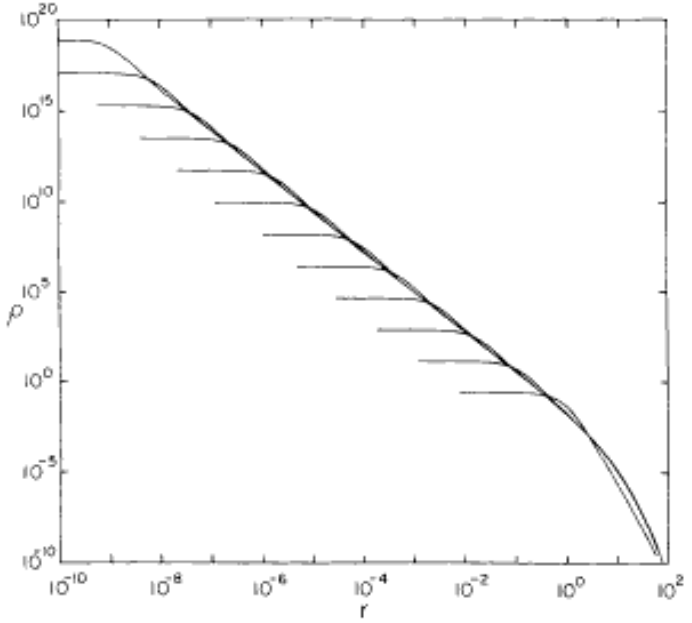
有限振幅の進化

摂動が有限振幅まで成長したあとの進化：数値計算で調べる。

Hachisu *et al.* **(1978)**：自己重力流体について数値計算した。

Lynden-Bell & Eggleton (1980)：流体の自己相似解を求めた

Cohn (1980)：流体近似を使わない軌道平均フォッカー・プランク方程式の数値積分から、自己相似解が実現していることを示した。



自己相似解の後の進化

自己相似解：有限時間で中心密度が発散
「その後」どうなるか？が問題

- 球状星団
- 矮小銀河の中心部
- 普通の銀河の中心部（？）

今までに密度が発散しているはずのものもある。

- 途中でブラックホールに、、、(普通の球状星団では難しい)
- 連星によるエネルギー供給で反転

連星

連星： 3 体散乱で確率的にできる (3 つ粒子があれば、そのうち 2 つを連星にして重力エネルギーをとり出した方がエントロピー的には得なので)。

普通の星の中心での核融合と同様にエネルギー供給として働く

→ 収縮を止めて、安定な定常状態になる？

数値計算

Henon (エノン・ハイレスのエノンと同一人物) **(1975)**: モンテカルロ法 (といっても、メトロポリス法みたいなのとは全然違う、、、詳しくは後述) での計算にエネルギー供給を人為的に入れて、ホモロガスに膨張する解を得た。

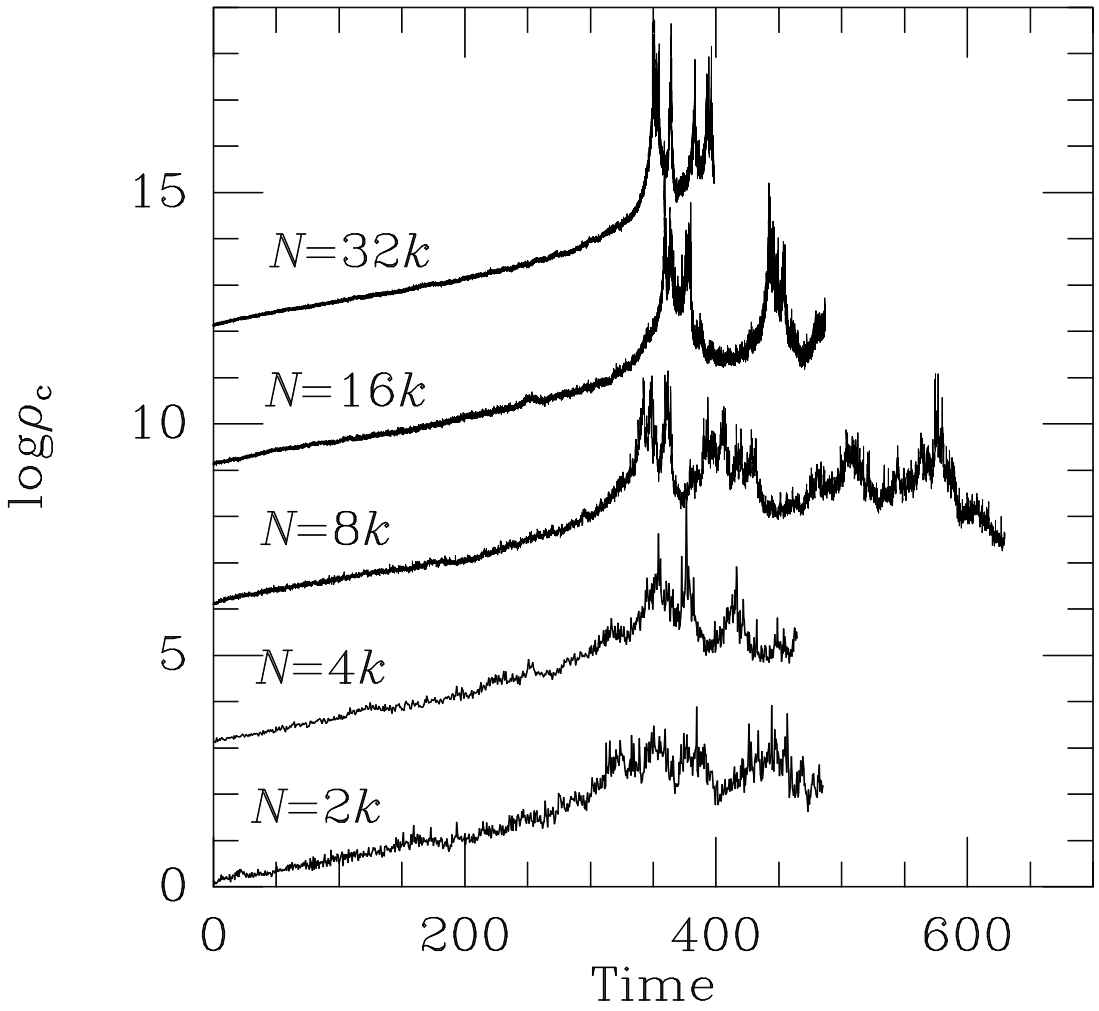
Heggie 1984, Goodman 1984: 同様な結果

Sugimoto and Bettwiser 1983: ホモロガスに膨張が実は不安定であることを、流体近似の数値計算から「発見」

1985 年には他のグループも確認、 **1986** 年にはフォッカープランクでも確認。

N 体計算

1996 年：N 体計算でも確認



まとめ

あんまりまとめではないですが、、、一応、ここまでで、重力多体系の進化の話はおしまい。ここから数値計算法の話にはいることにします。

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

$$\frac{d^2 x_i}{dt^2} = \sum_{j \neq i} G m_j \frac{x_j - x_i}{|x_j - x_i|^3}, \tag{40}$$

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。

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 $\mathbf{PEC}$ 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。

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

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。

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、 $r \longrightarrow 0$ 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

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

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

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。

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

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

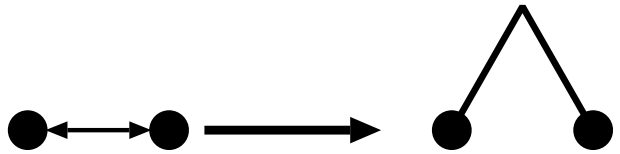
2体以上の系の扱い

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？

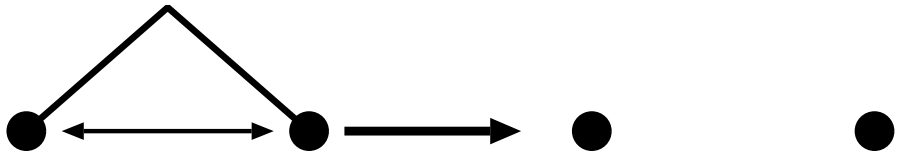
- 「3体」として認識、それ用の正則化
- 2体+1体としては認識、再帰的に扱う

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

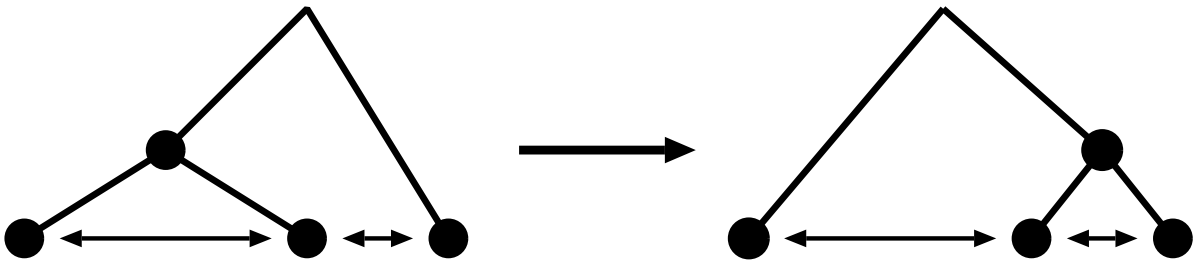
構造の組み換え



$r < r_{close}$



$r > \gamma r_{close}$



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**

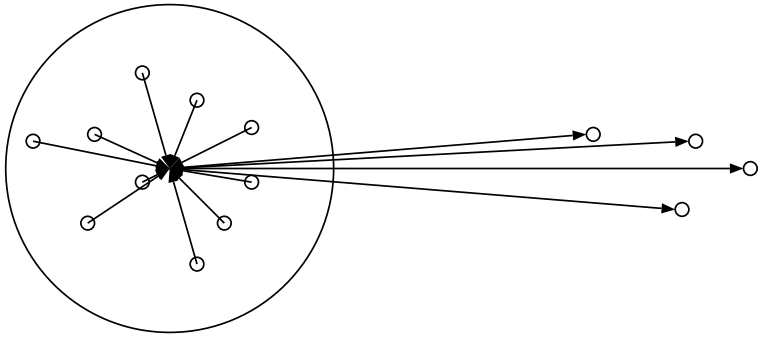
有名な方法： 高速多重極法

ツリー法、FMMの基本的発想

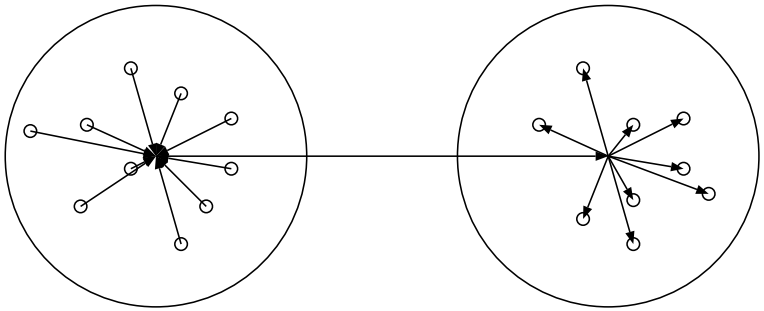
遠くの粒子からの力は弱い

↓

まとめて計算できないか？



Tree



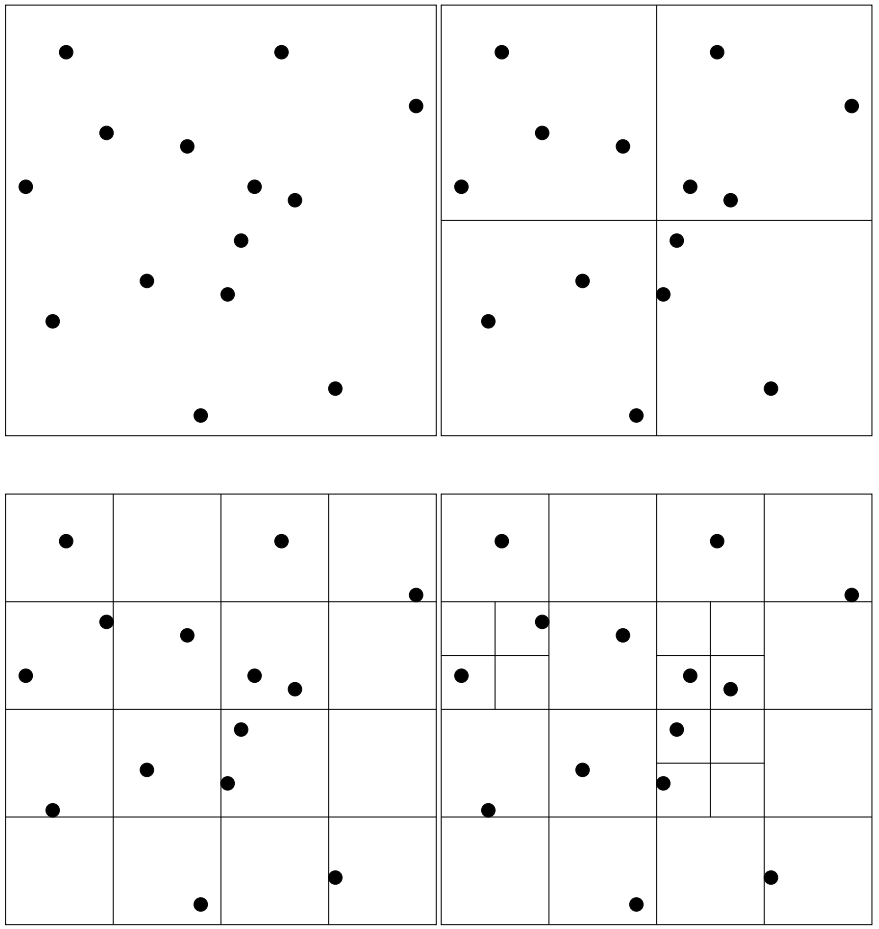
FMM

- 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

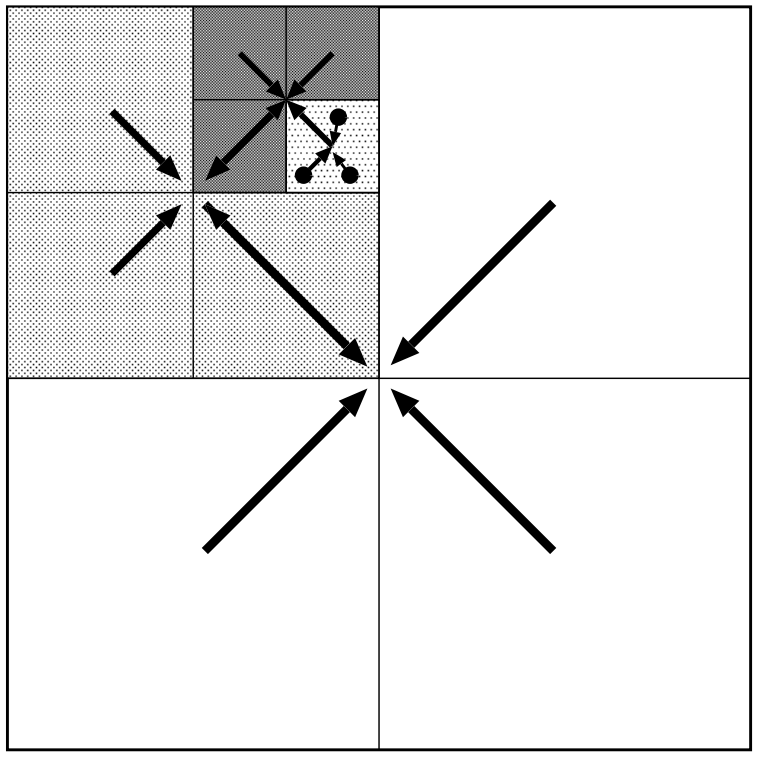
階層的なツリー構造を使う。

- 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。

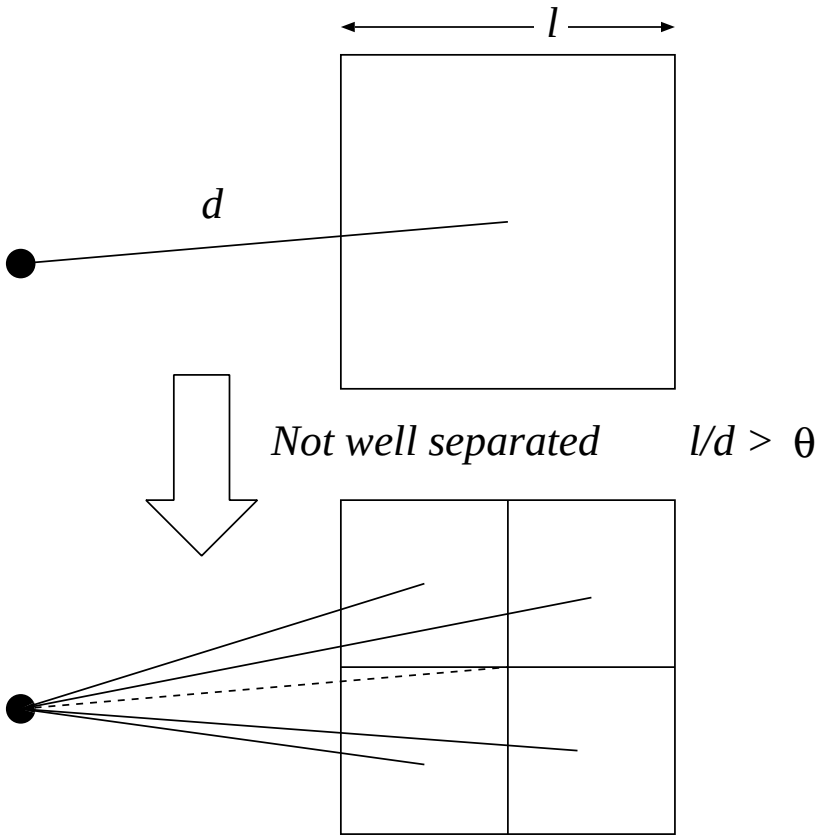


- 最下層のセル:そのなかの粒子が作るポテンシャルを多重極展開
- それ以外 : 子セルの多重極展開の展開中心をシフトして加算

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:

$$\text{誤差} \propto \theta^{(p+1)}$$

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

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)

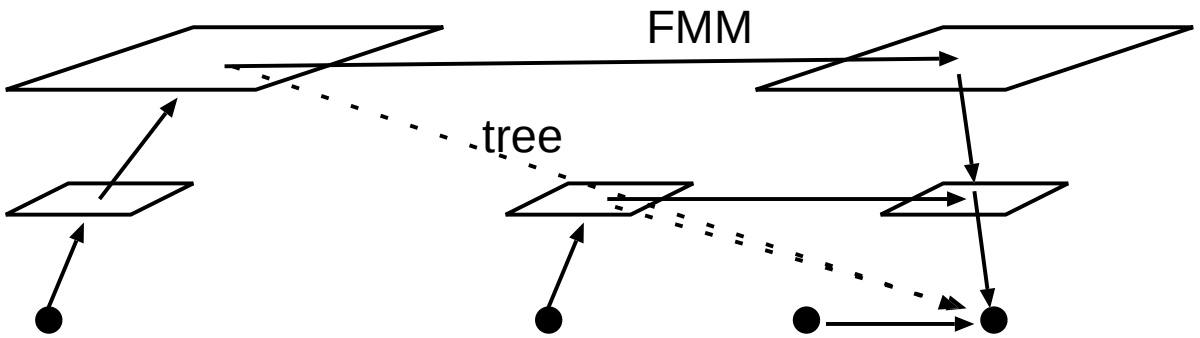
高速多重極展開法 (FMM)

- 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！

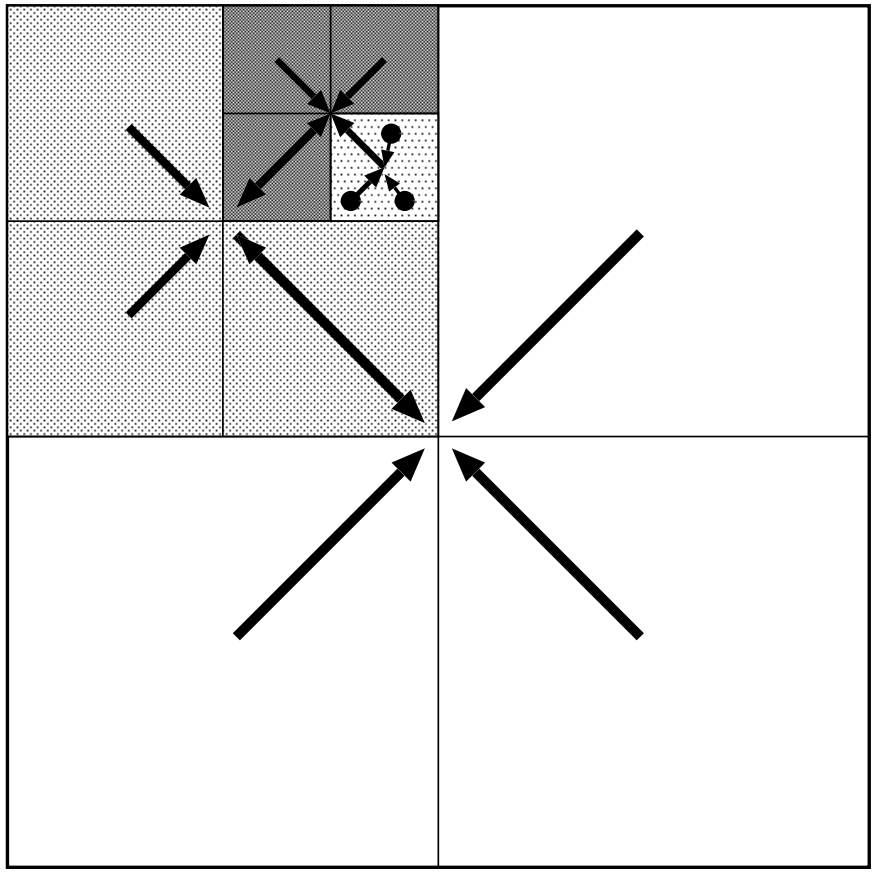
粒子間相互作用の計算されかた



- 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

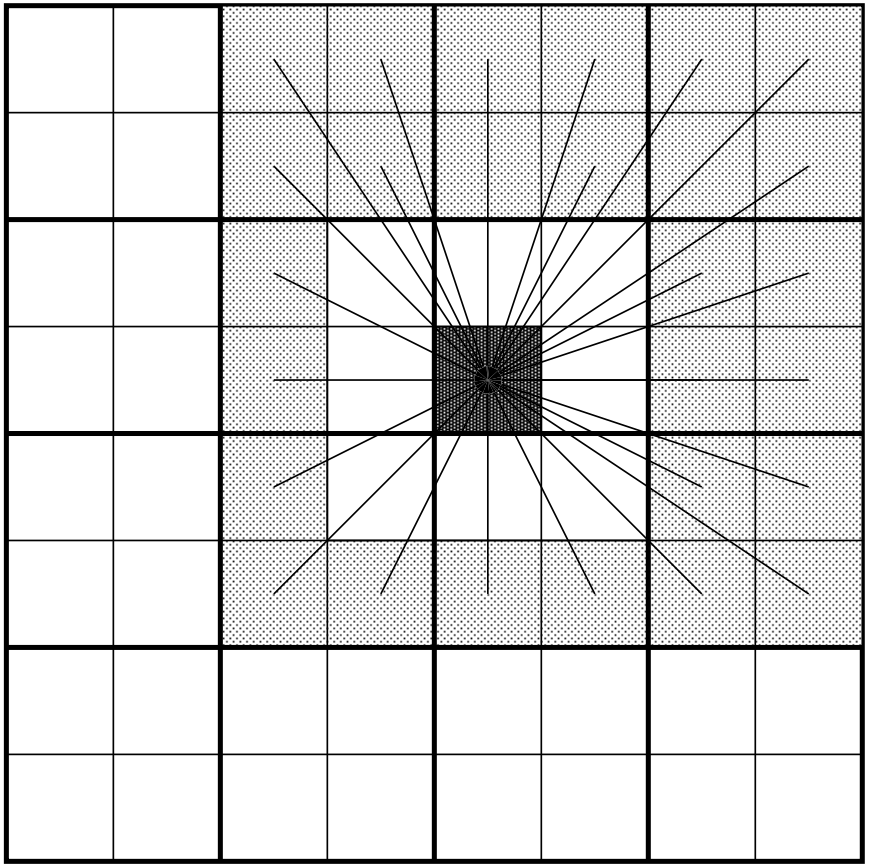
各セルが外に作るポテンシャルの球面調和関数展開 (多重極展開) を、自分の下のセルの多重極展開の展開中心をシフトして求める。(M2M)
(ここはツリーと同じ)



FMMの計算手順 - (2) M2L

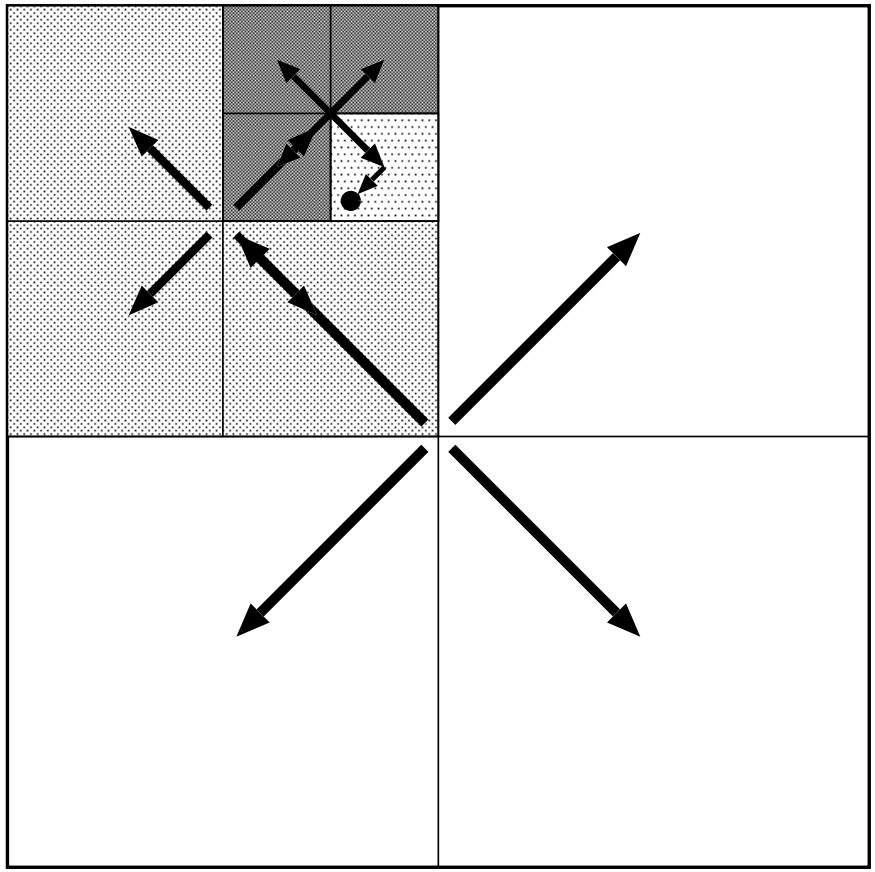
- 自分と同じレベルで
- 親の隣接セルの子で
- 自分の隣接セルでない

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 $\longrightarrow$ ポアソンの公式

ポアソンの公式 (外部解と内部解)

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

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

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。

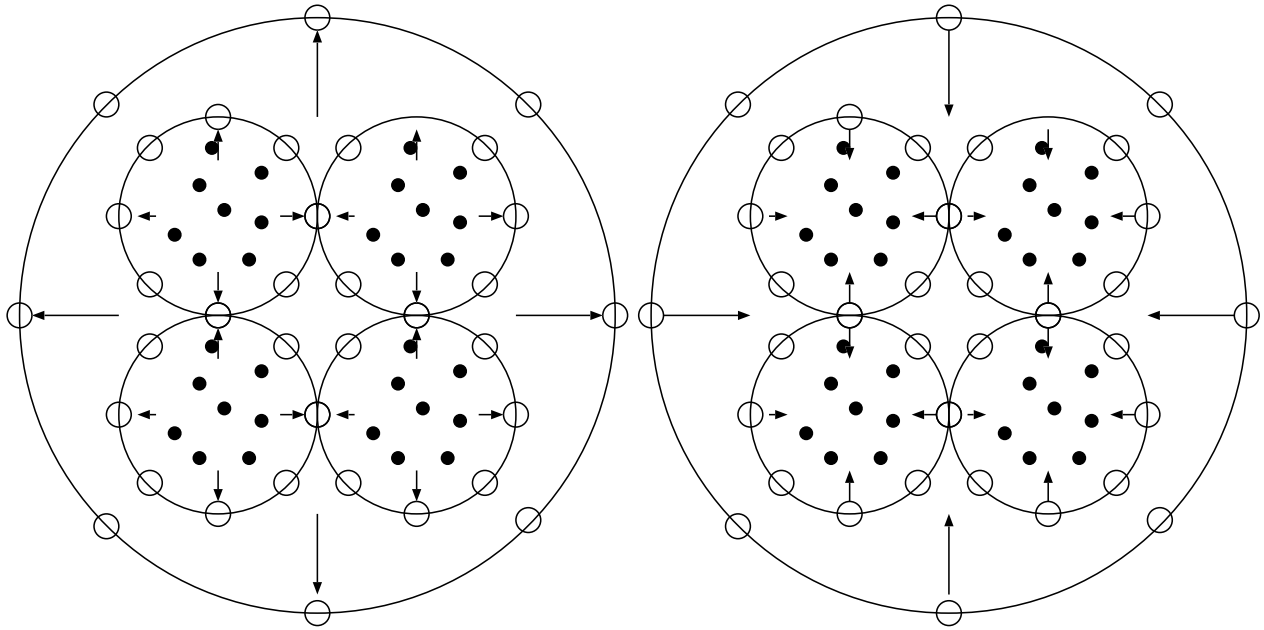
$t/2$ 次までの球面調和関数の直交性が保てることになる。

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)

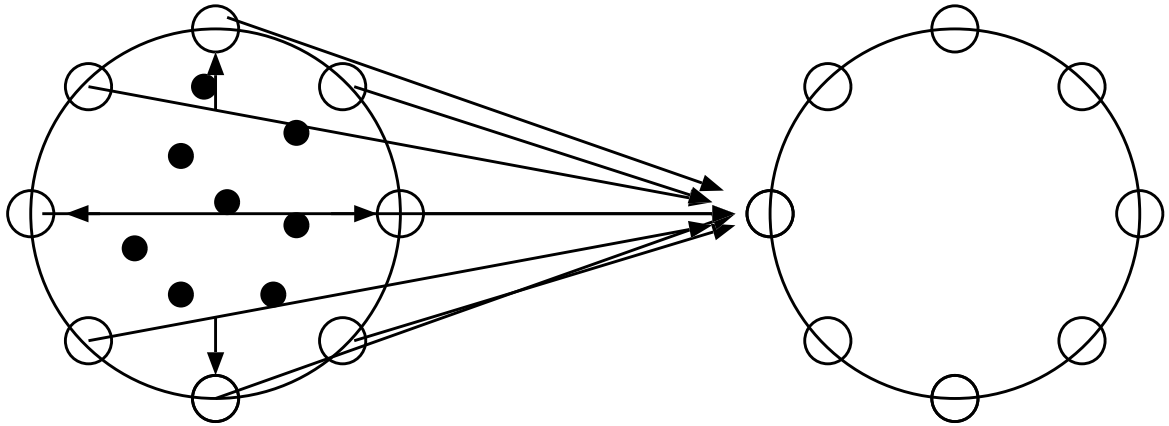
多重極展開 (左) と局所展開 (右) のシフト



(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：$(p+1)^2$

  $\longrightarrow$ 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

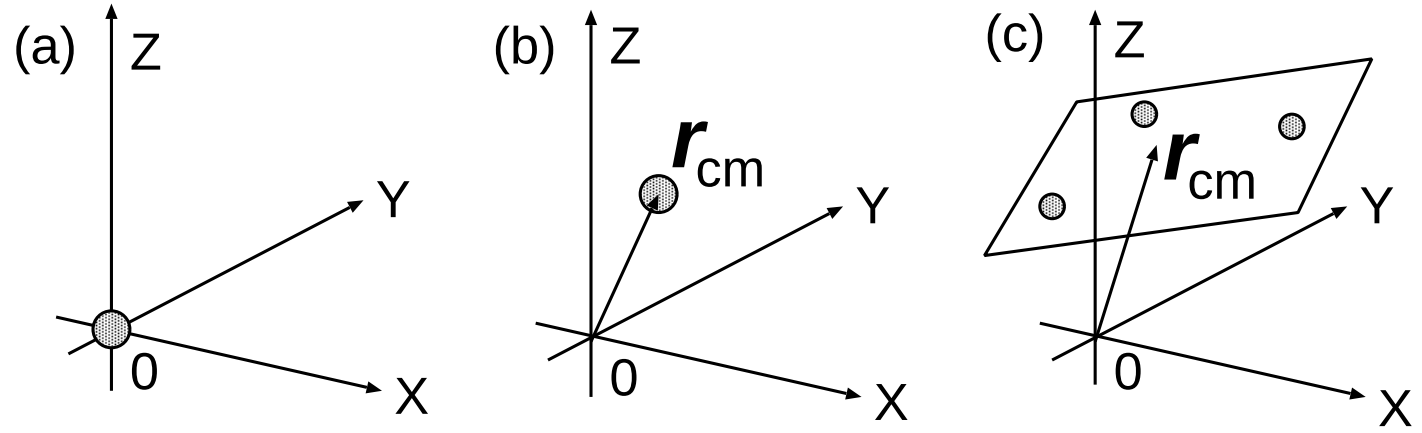
  - （欠点）必要とする粒子の数が増える
  - （利点）いつでも線形計算で配置を決めることが出来る

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：

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

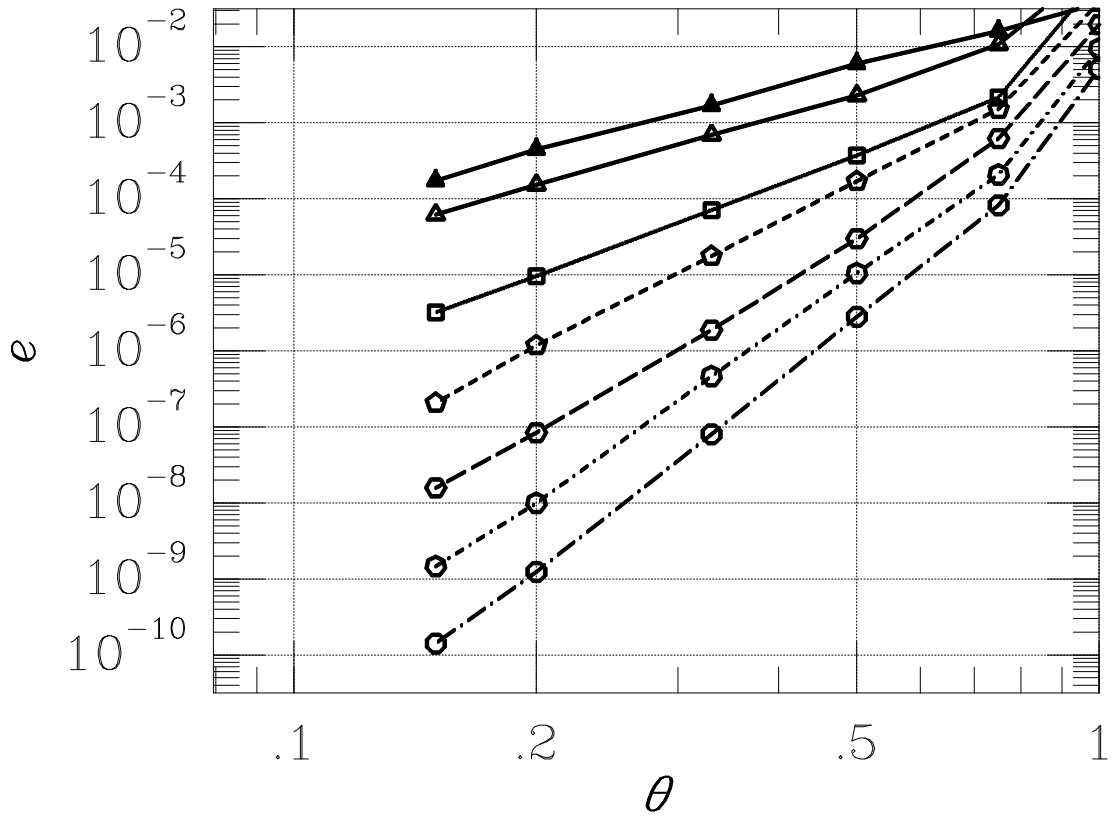
ここで γ は粒子の位置と疑似粒子の位置の間の角度。

- 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 $\rightarrow$　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

$\longrightarrow$ 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

$\longrightarrow$ 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

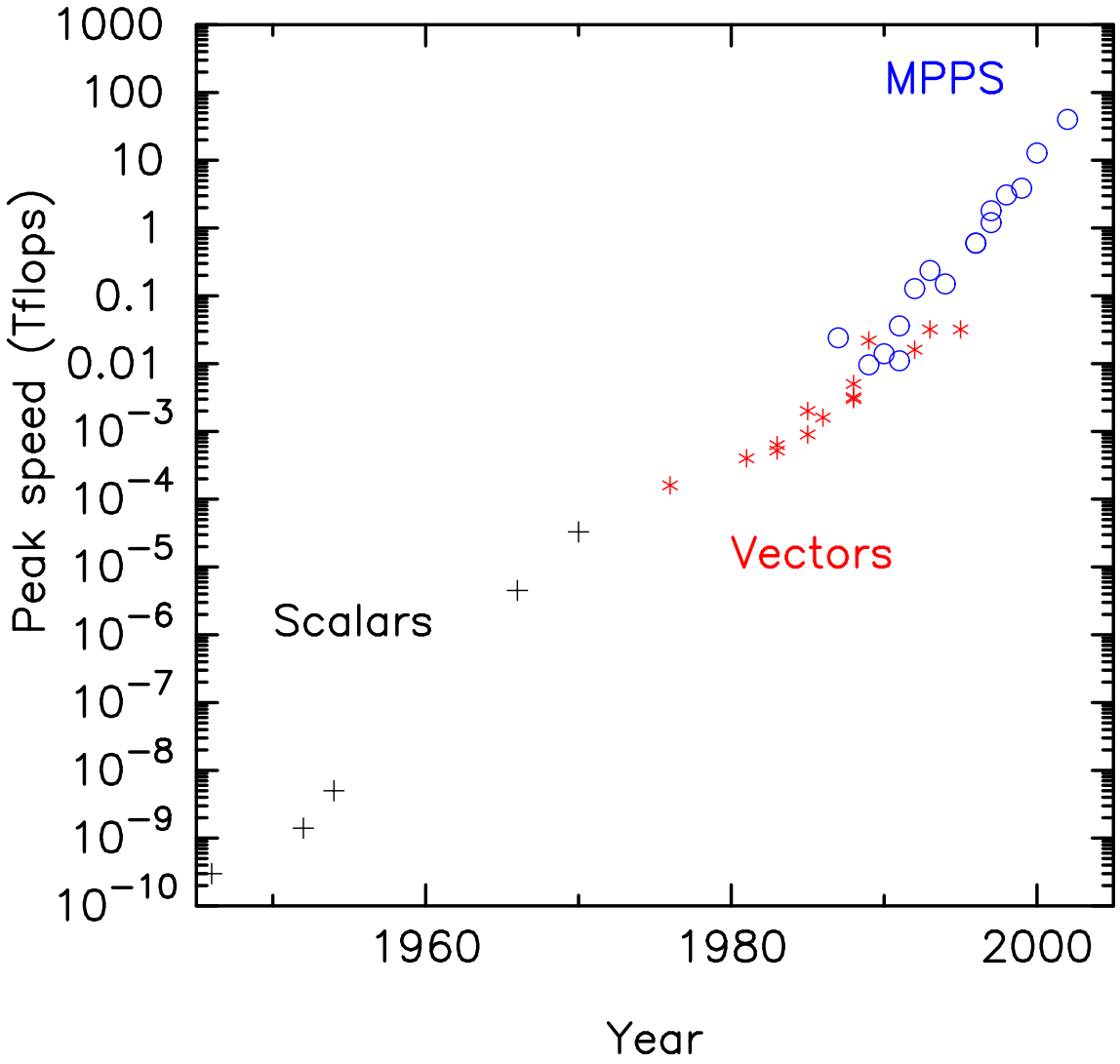
速い計算機を使えば速く計算できるというのはなんかトートロジーみたいで当たり前という気もするが、実はそうでもない。

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

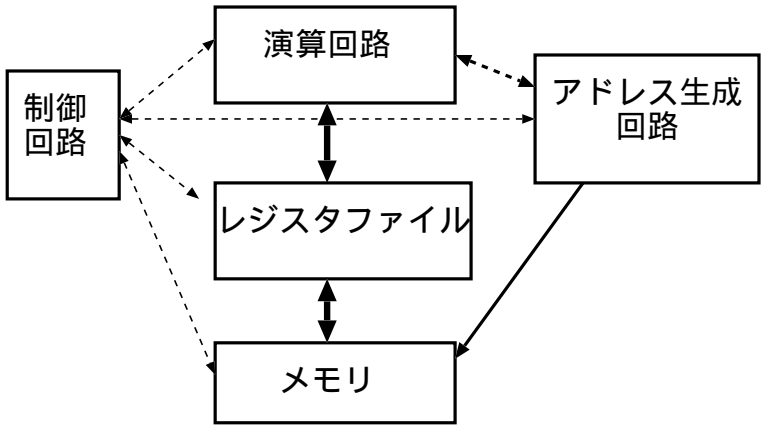
ほぼ時間の指数関数



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 $\rightarrow$ ベクトル $\rightarrow$ 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」

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

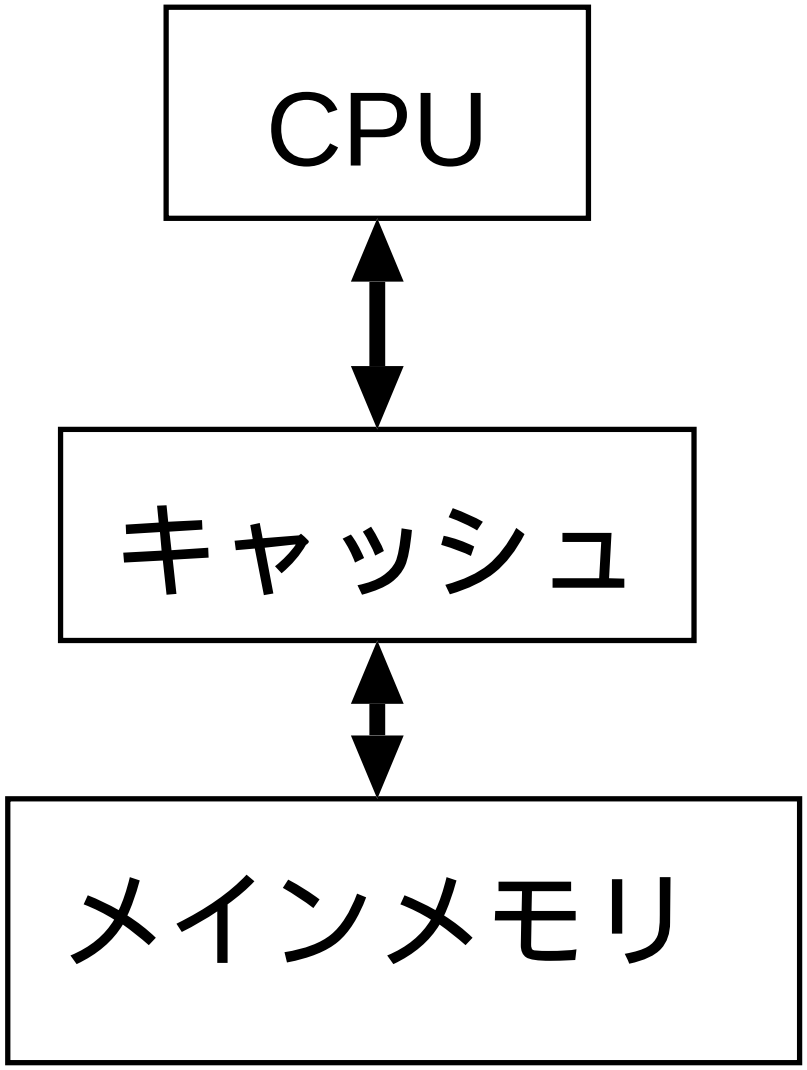
実行速度の制限要因: memory wall

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

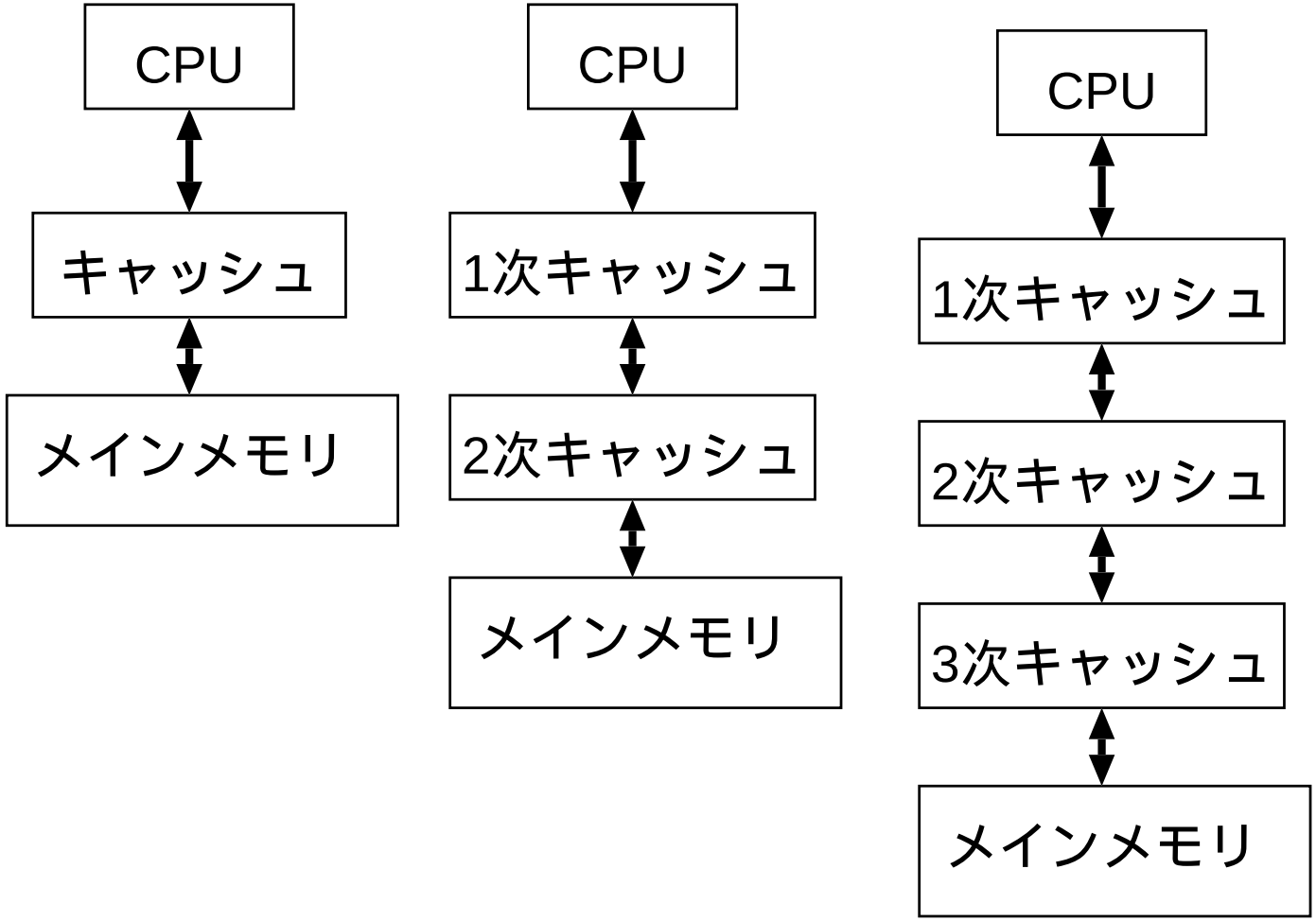
例:
Intel P4 データ 32 bit, クロック 1066 MHz (DRAM チップ内部はもっと遅い)
AMD K7 データ 64 bit, クロック 333 MHz
CPU のクロックに比べると 5-10 分の 1

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 $(\mathrm{CDC}\ 7600)$

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 $\rightarrow$ 問題が (とりあえず) 回避される。
つまり: $1\ \mathrm{CPU} +$ 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

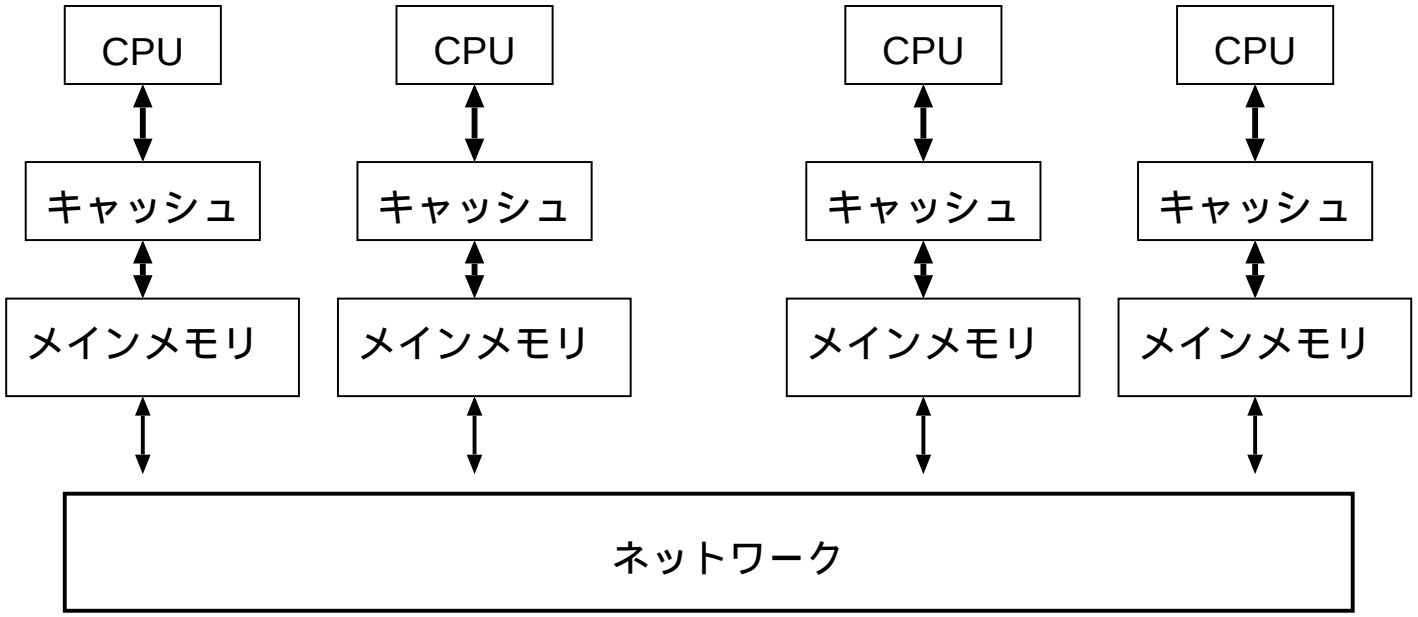
分散メモリ型並列計算機

(数値計算業界からみて) 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)

分散メモリ型計算機の (偏った) 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 $\rightarrow$ 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？

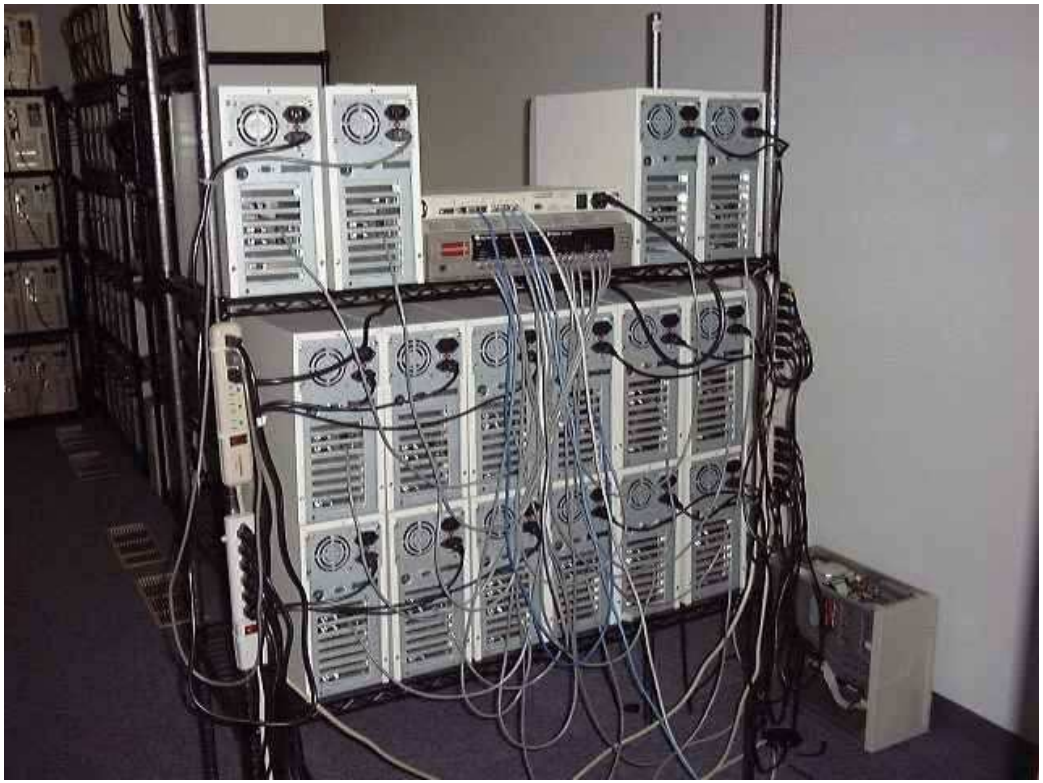
- なるべく価格性能比のいいパソコンを沢山買う
- それを必要最小限の性能のイーサネットを使った普通のネットワークでつなぐ

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

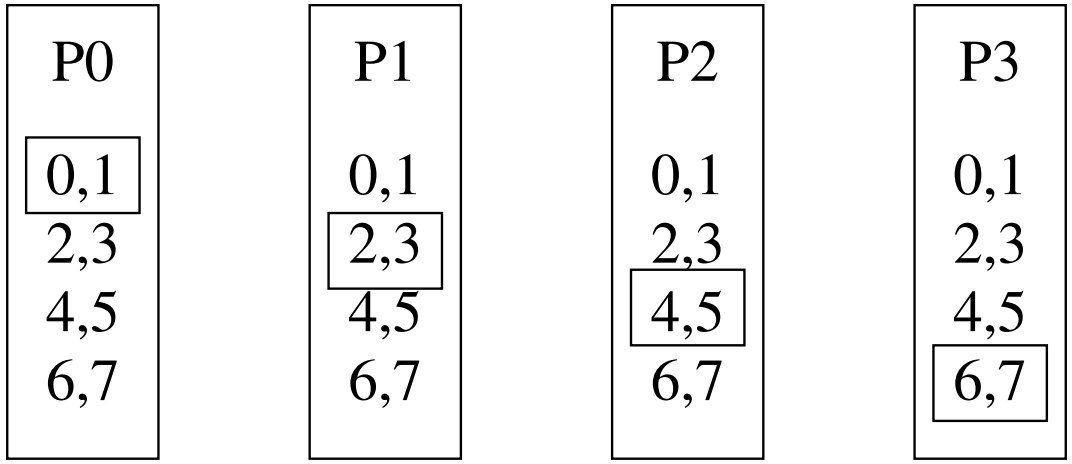
並列化の 2 方法

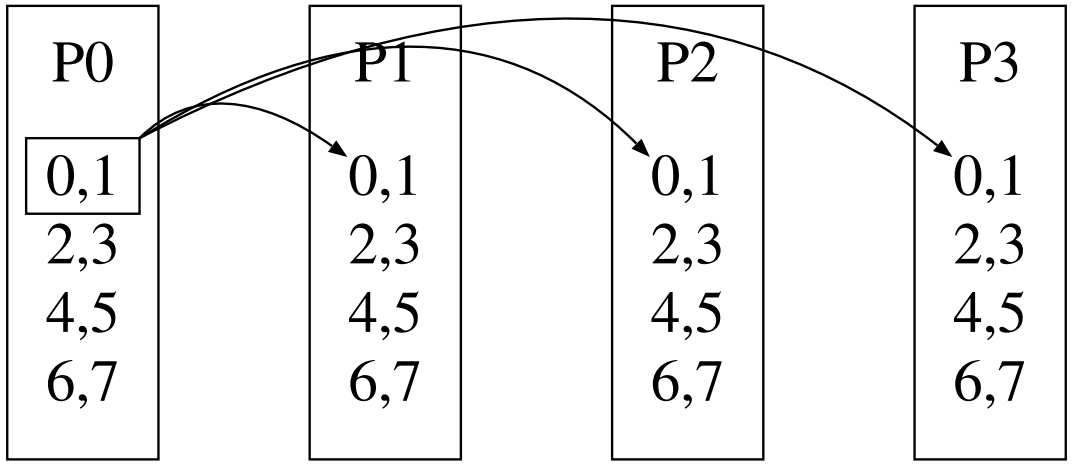
- 各ノードが全粒子のコピーを持って、重力計算、時間積分は分担する
- 各ノードが自分の分担の分だけを持つ。重力計算のためには他のノードから粒子を貰ってくる。

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。

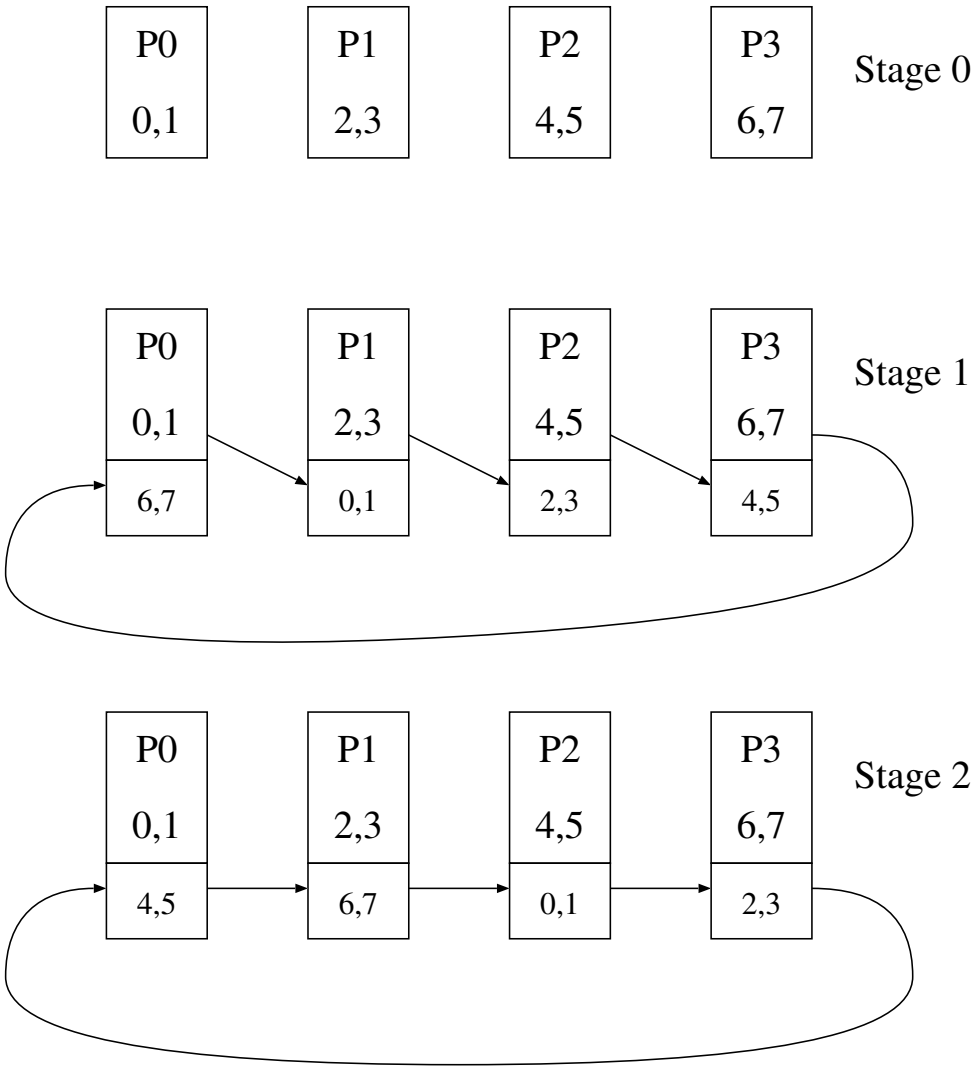
ノード一つの通信バンド幅でスケーラビリティが制限される。

The copy algorithm





The ring algorithm



計算時間のスケーリング

N 粒子、p ノード

$$T_{\mathrm{ring}} = C_f N^2/p + C_c N + C_s p. \tag{44}$$

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。

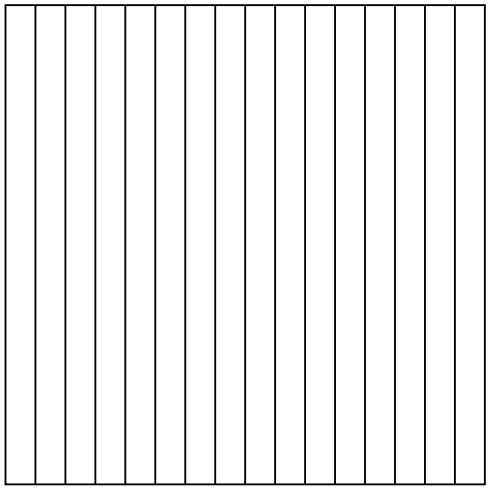
独立時間刻みの場合

- 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

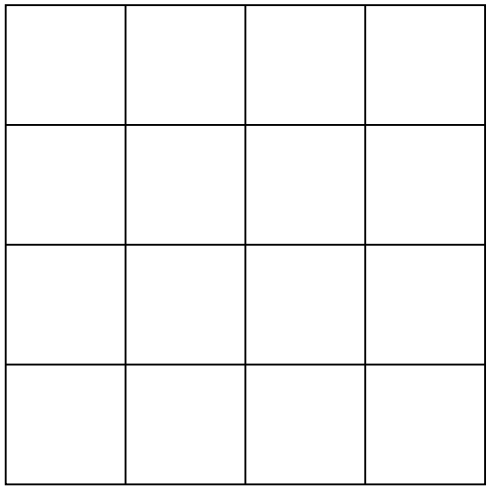
スケーラビリティが一層悪くなる

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

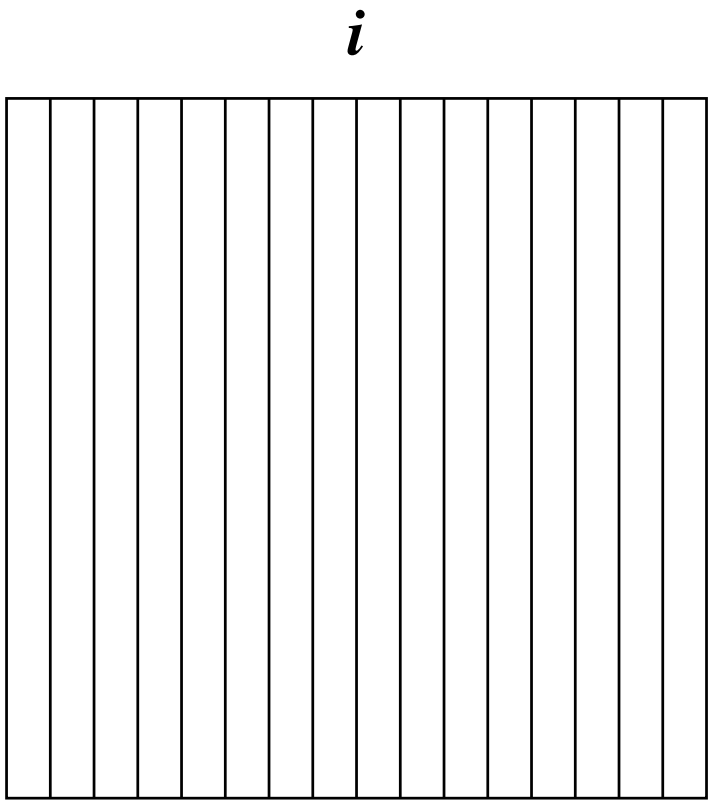
１次元空間分割：スケーラブルでない

ノード数 ∝ １次元方向のメッシュ数

２次元空間分割：スケーラブル

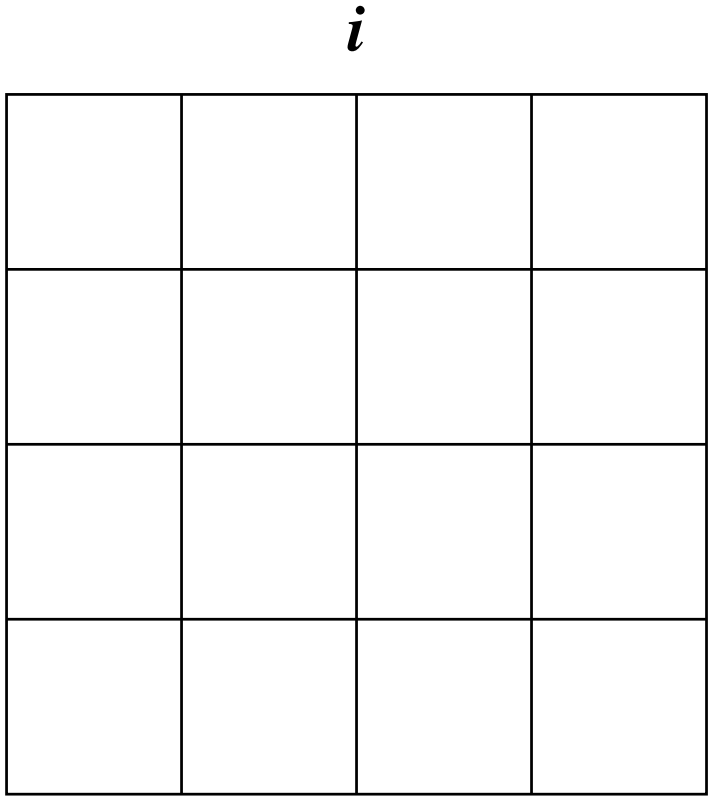
ノード数 ∝ 全体のメッシュ数

遠距離相互作用 ── 相互作用行列を考える



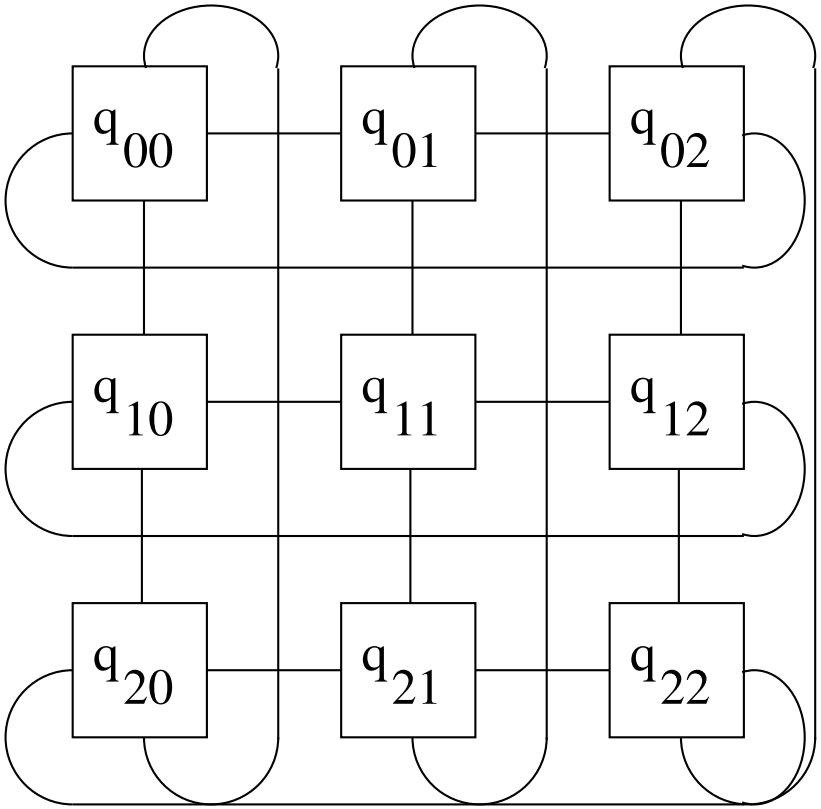
- 相互作用の行列を見ると、普通のアルゴリズムは1次元分割になっている。
- これがスケーラブルでない理由
- 2次元分割は可能か？

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？

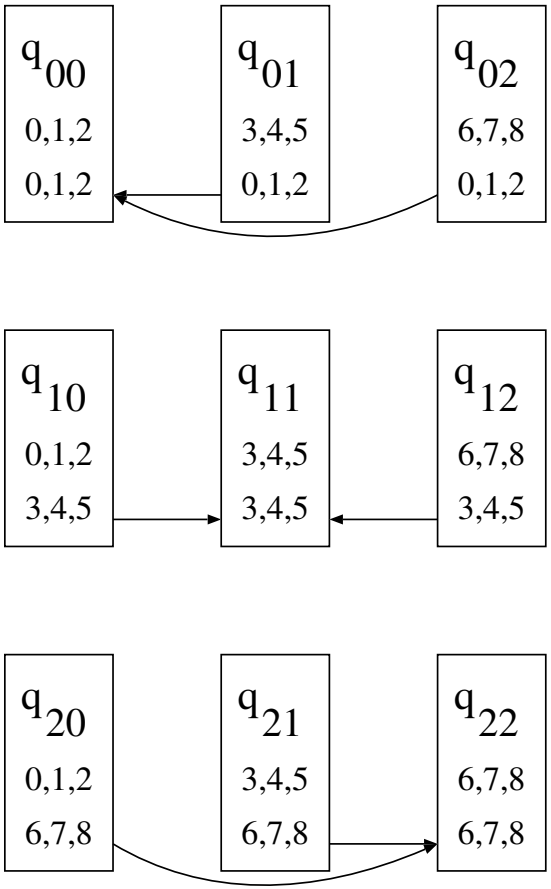
ハードウェアとしては、、、



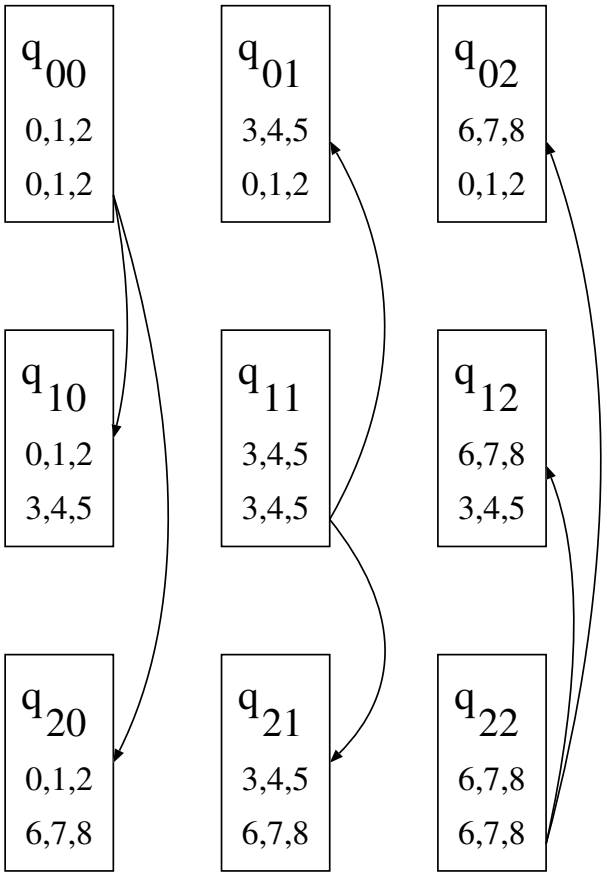
- 分割した相互作用行列にプロセッサを対応させる。
- トポロジ：2D トーラスなり「ハイパークロスバー」なり、、、

計算手順

Step 1



Step 2



横に総和、縦に放送

スケーリング (理論)

r^2 プロセッサ

$$T_{\mathrm{2Dbcast}} = C_f N^2/r^2 + 2(C_c N/r + C_s \log_2 r). \qquad (45)$$

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

$$p_{\mathrm{half,2Dbcast}} \sim (NC_f/2C_c)^2 \tag{46}$$

(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

$$p_{\mathrm{half,ind,2D}} \sim N^{5/3} C_f / [C_s \log_2 (N^{5/3} C_f / 2 C_s)] \qquad (47)$$

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。

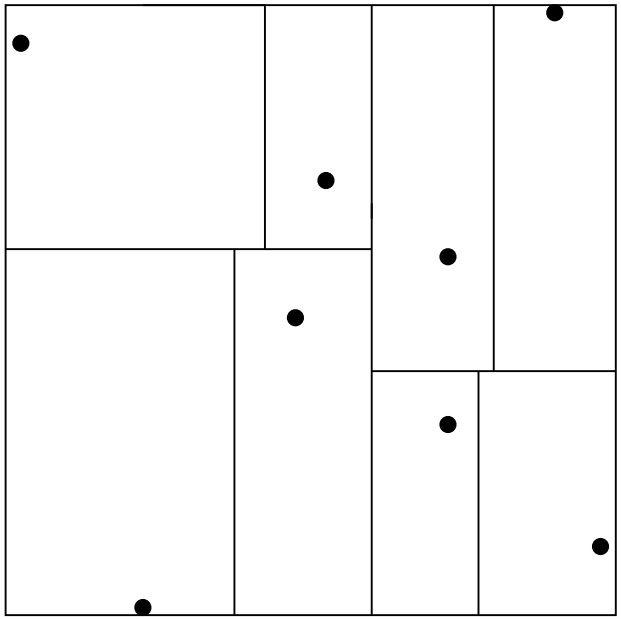
- 共有メモリの機械では容易 (アルゴリズムの並列度は実際上 N)
- メッセージパッシングの機械 : 効率の良い空間分割が必要。Salmon & Warren が様々な方法をテストした。プログラムは面倒だが、作ってしまえば効率はよい。

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

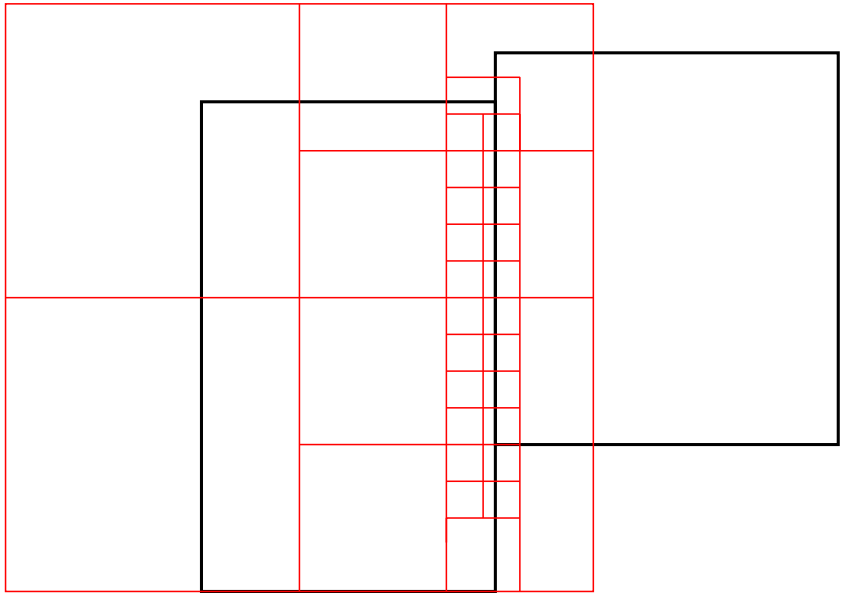
ORB 空間分割の実装

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。

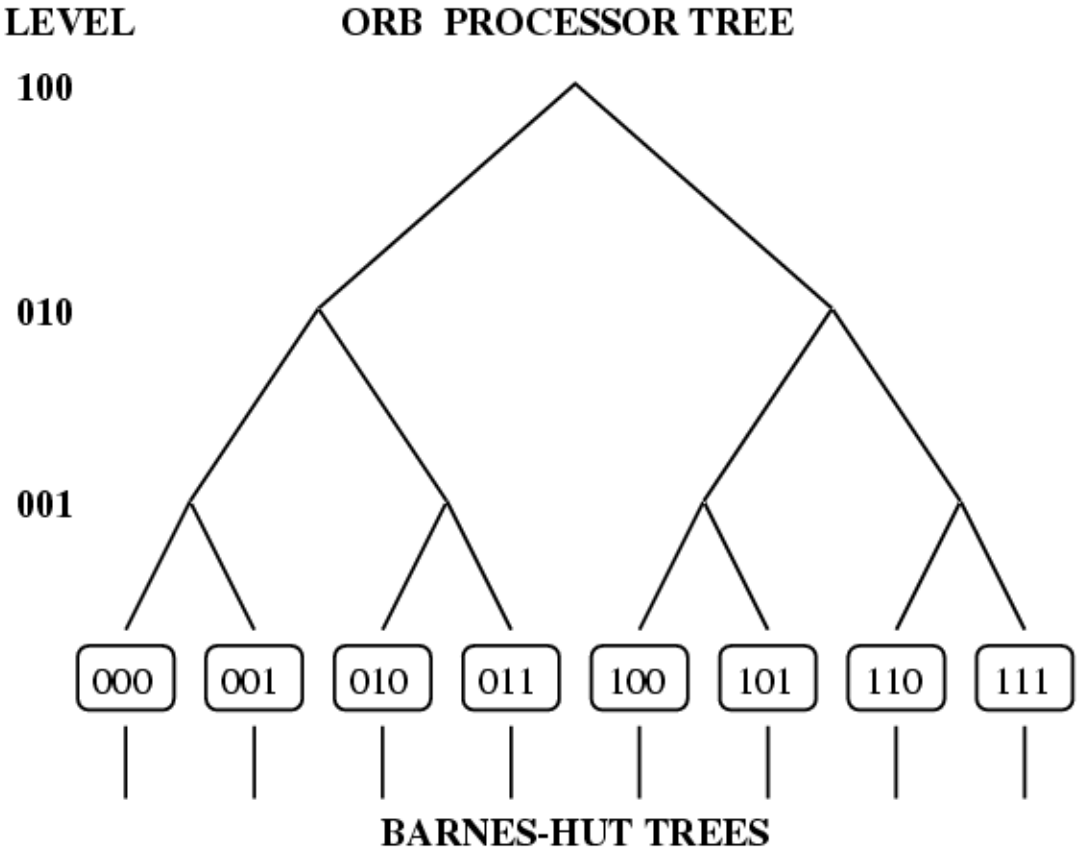
他のプロセッサにある粒子からの力の計算



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？

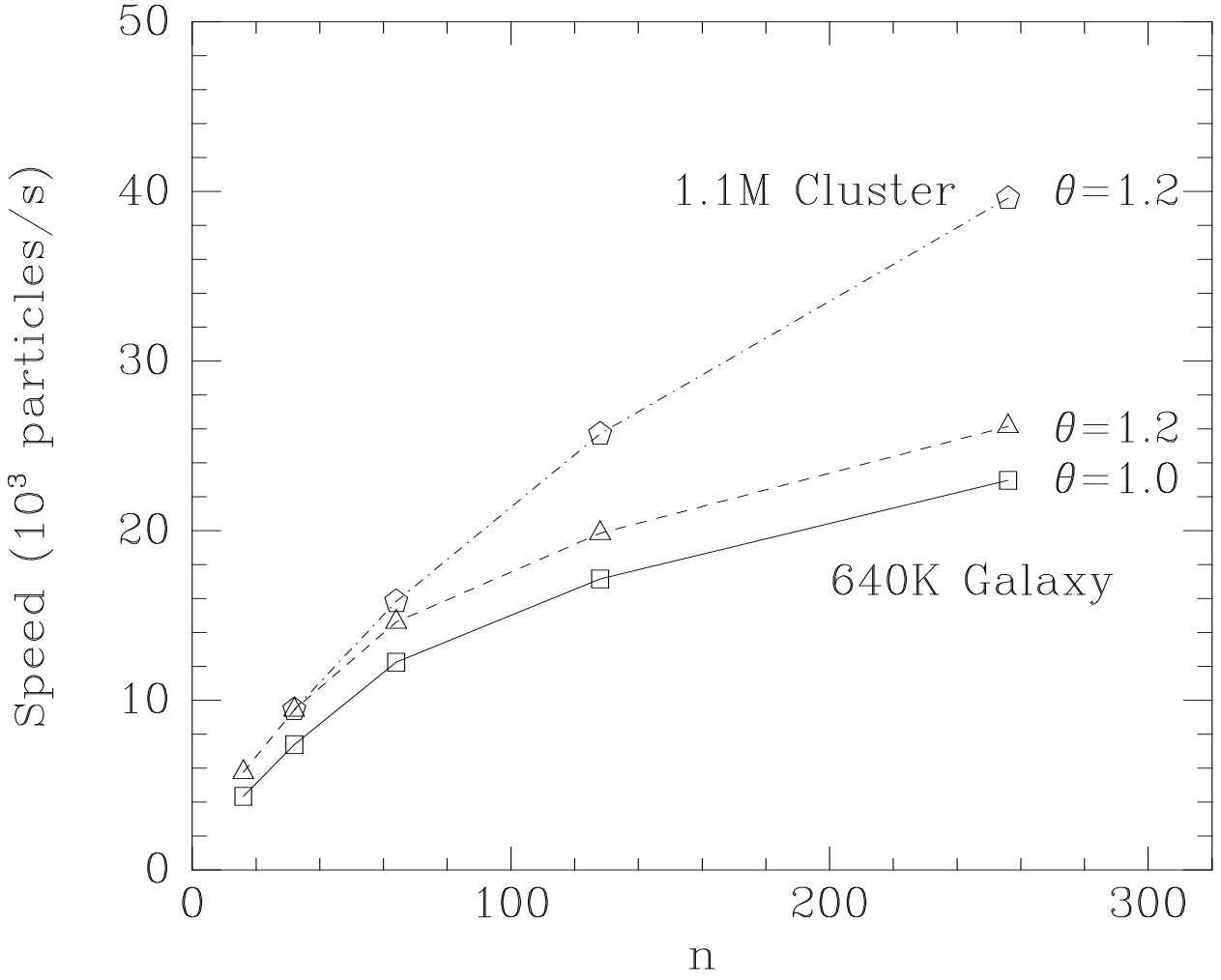
Dubinski の実装：この上の階層は ORB ツリーそのもの



ORB tree の問題

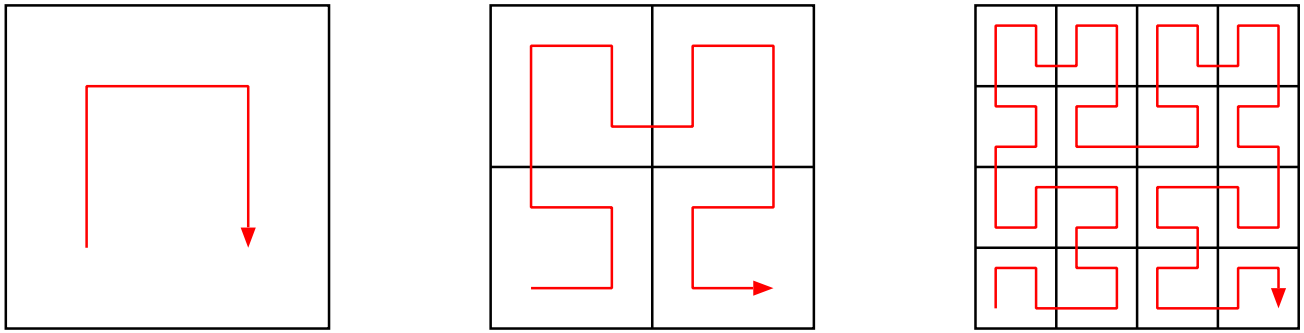
- 実装が面倒
  - ORB tree のところと local tree のところでツリーの構造が違う
  - LET はツリー構造 (ポインタとか) もちゃんと保持して送る必要あり
- スケーラビリティが悪い
  - 通信回数がプロセッサ数に比例
  - 空間分割の形が悪くなると通信量も計算量も増える
- シリアルコードと結果が丸め以上に違う

スケーリング


50
40
30
20
10
0
0
100
200
300
n
Speed (10^3 particles/s)
1.1M Cluster
θ=1.2
θ=1.2
θ=1.0
640K Galaxy

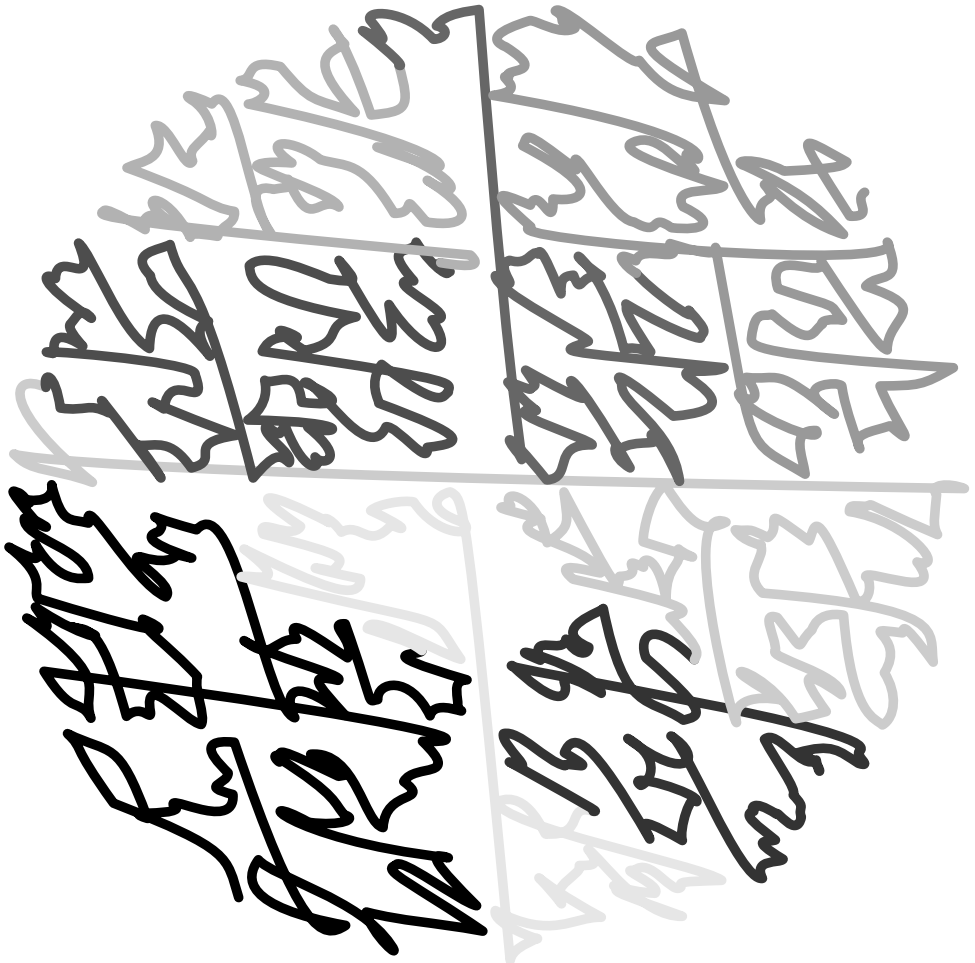
HOT

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 $\mathbf{and/or}$ 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。

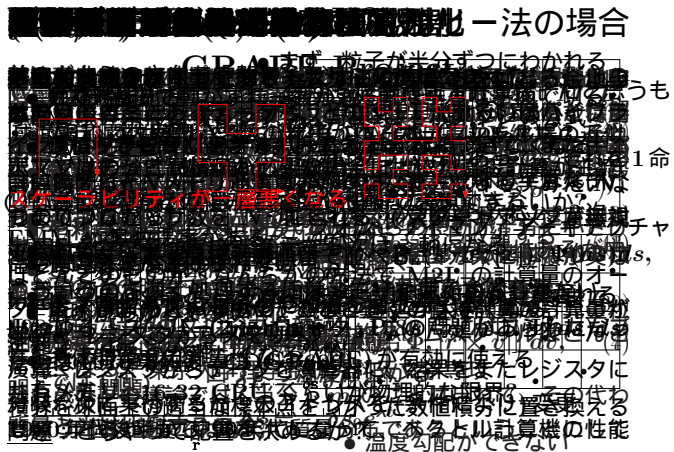
もうちょっと安易な方法

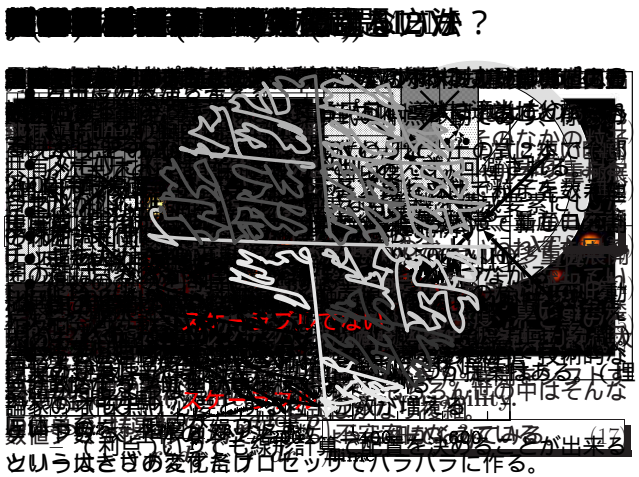
今回使った方法： Brackston & Suel によるものを踏襲

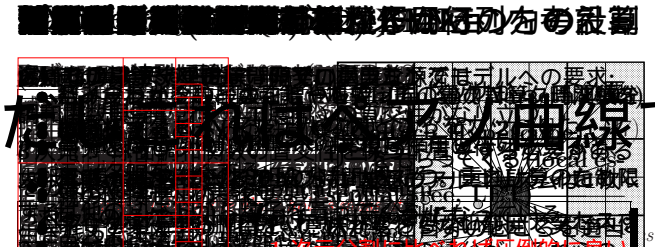
- 各プロセッサは適当に選んだサンプル粒子のセットをノード 0 に送る。
- ノード 0 は、集めたサンプルセットを使って分割を生成する。できた分割は全ノードに放送する。
- 各プロセッサは、もらった分割を使って粒子を振り分けて、担当するプロセッサに送る。

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）







たたみ方のレベル曲線ではなくて 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 $\mathbf{and/or}$ 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)

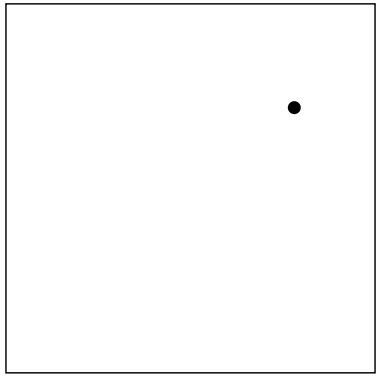
他のプロセッサから、「必要ないところをカットした」ツリーをもらってくる (**local essential tree, LET**)
それらと自分のところを適当にまとめて全体ツリーを作る

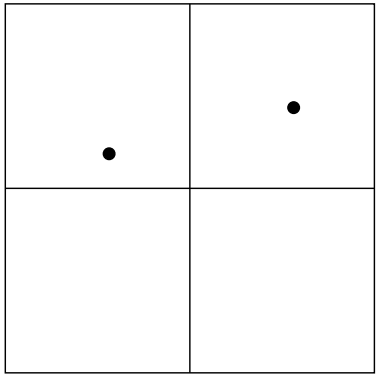
今回の実装

- 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。

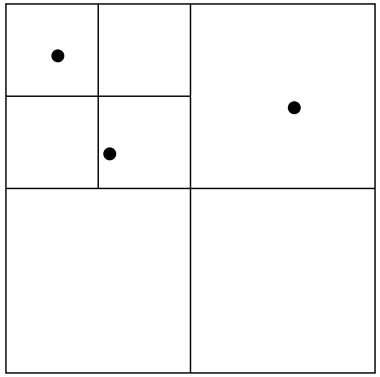
ツリーに粒子を挿入：もともとの Barnes のプログラムで使われている木の作り方。

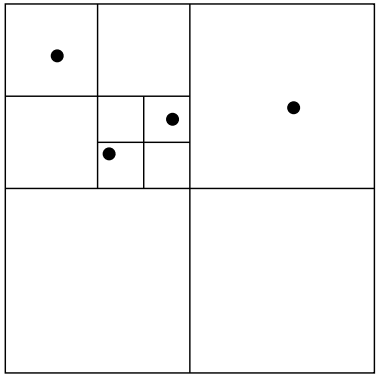
粒子挿入によるツリー構成





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

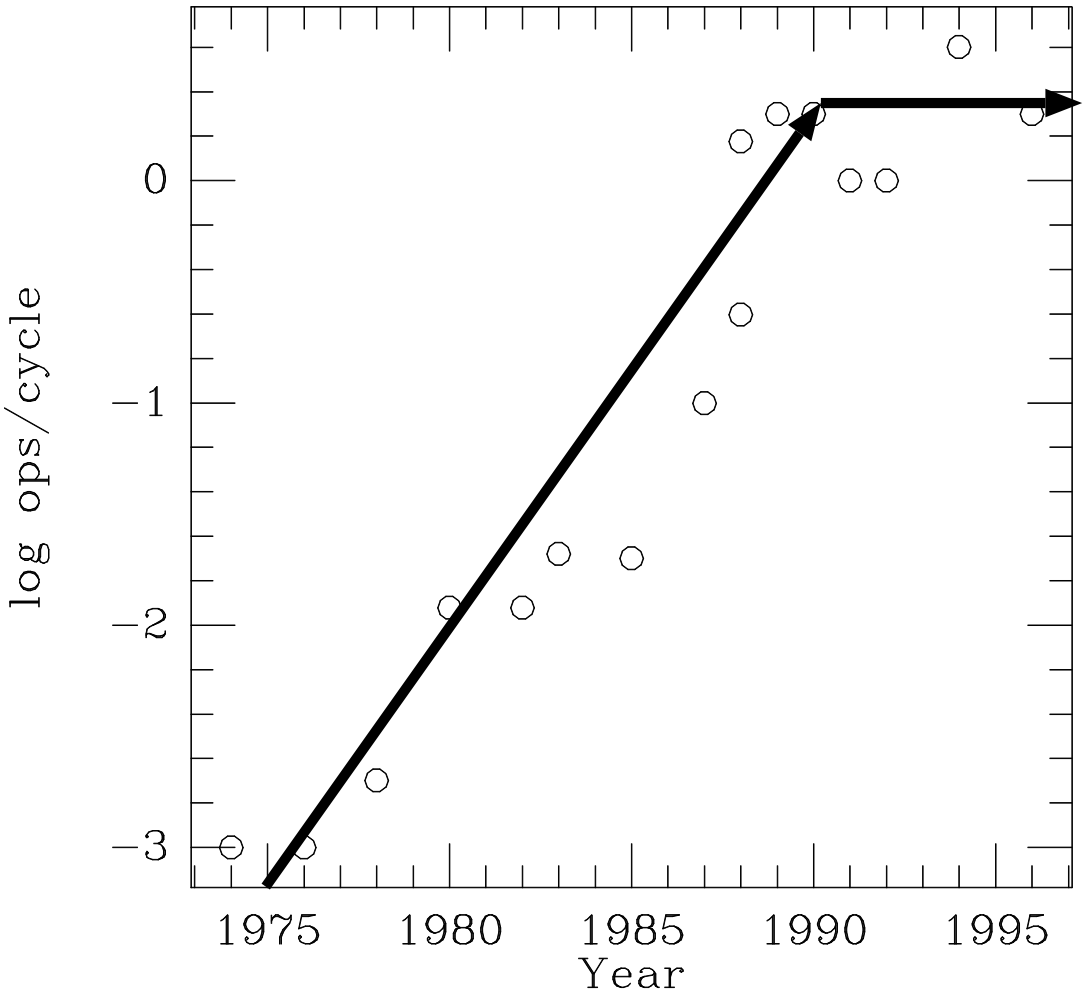
その他 (離散不規則系): 回路シミュレーション等？

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

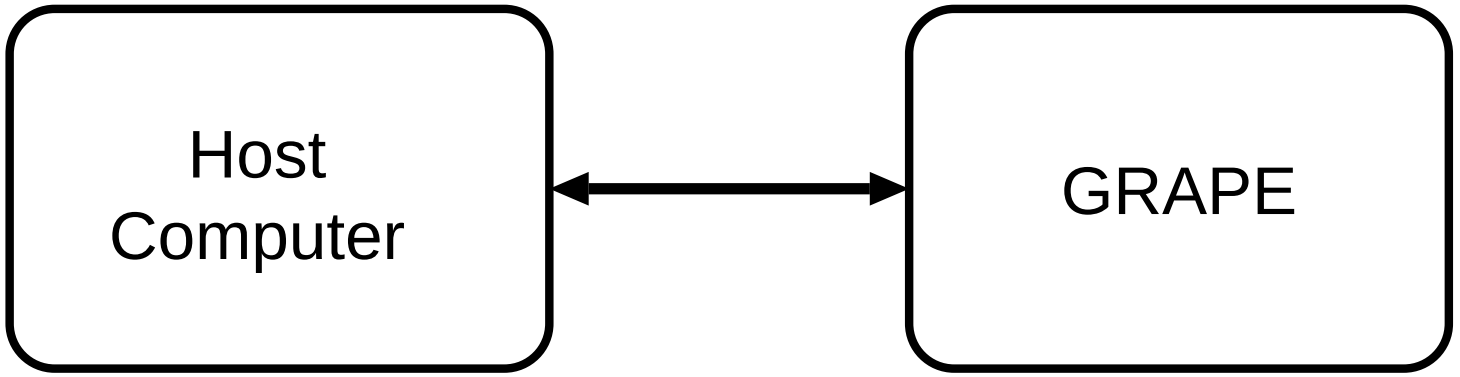
用途を制限すればある程度の並列化はできる

- 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



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

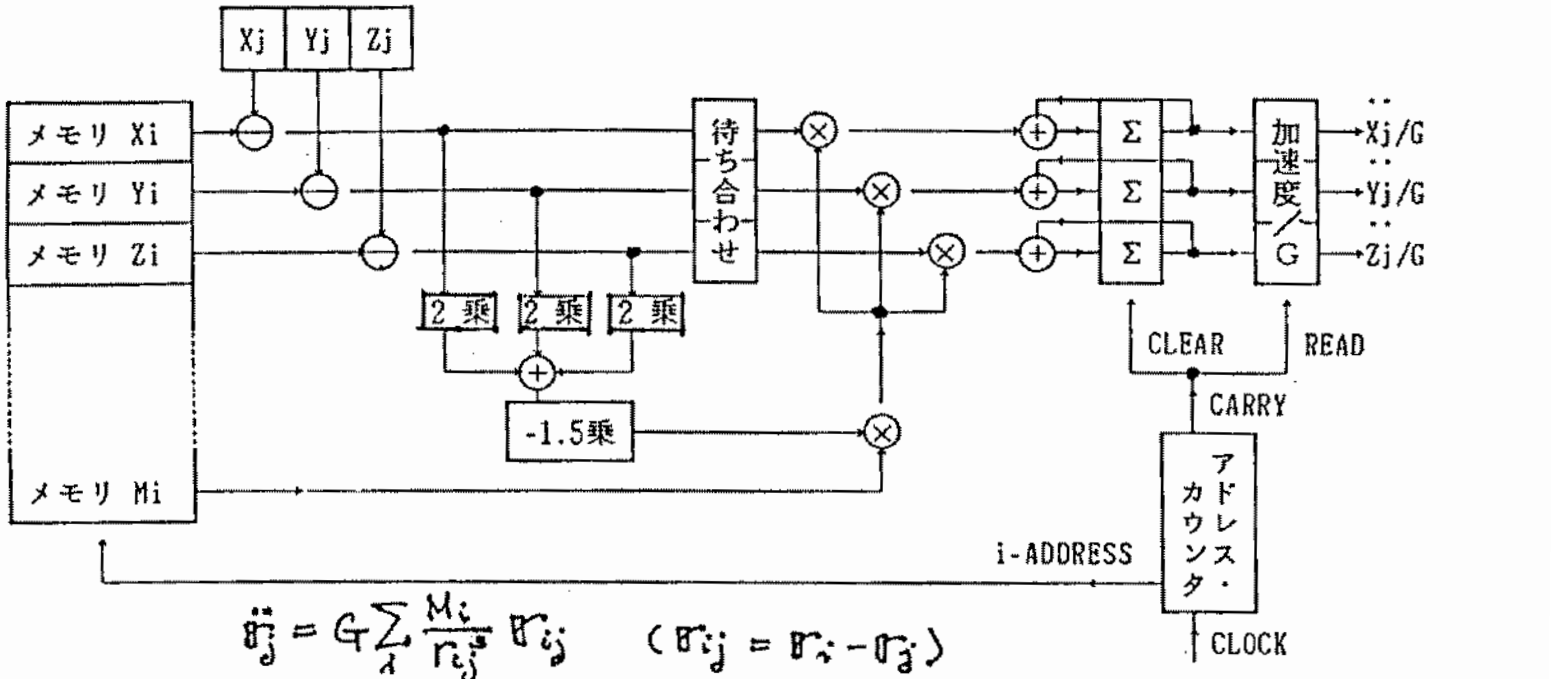
  $\longrightarrow$ 非常に高い性能

重要な条件: **Memory wall** の回避

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



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

+, −, ×, 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。

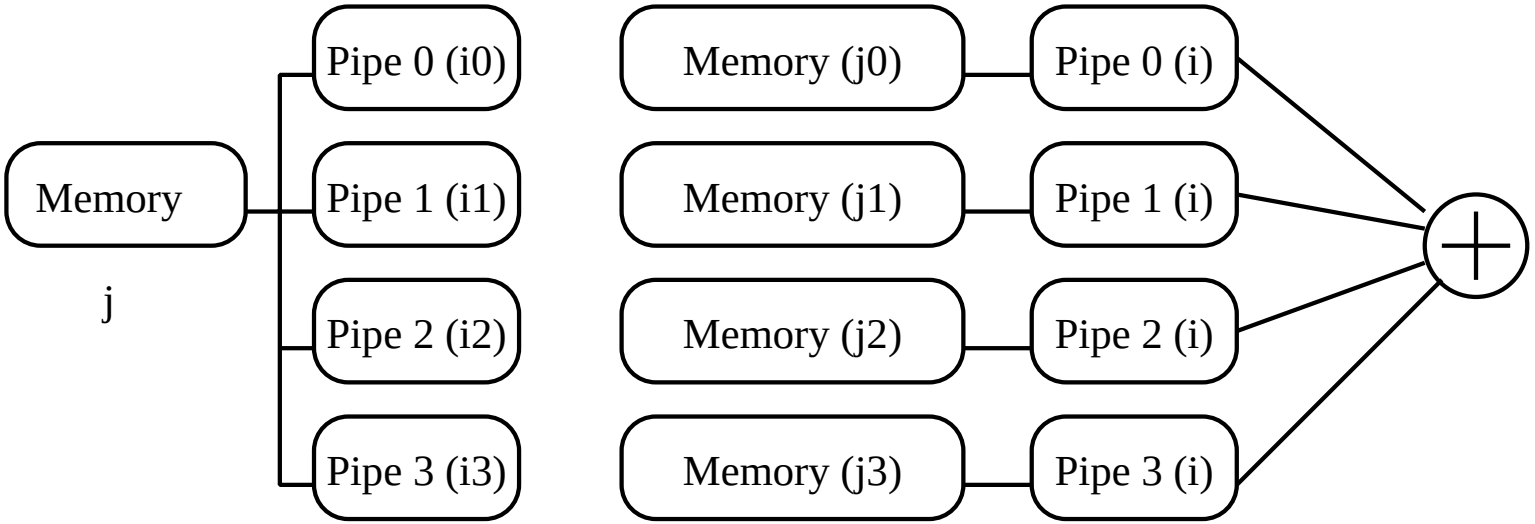
方針は 2 つ

- 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。



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

Memory wall の回避

あるいは通信量の削減

ホスト — GRAPE 間: N 個の粒子、 N^2 の計算

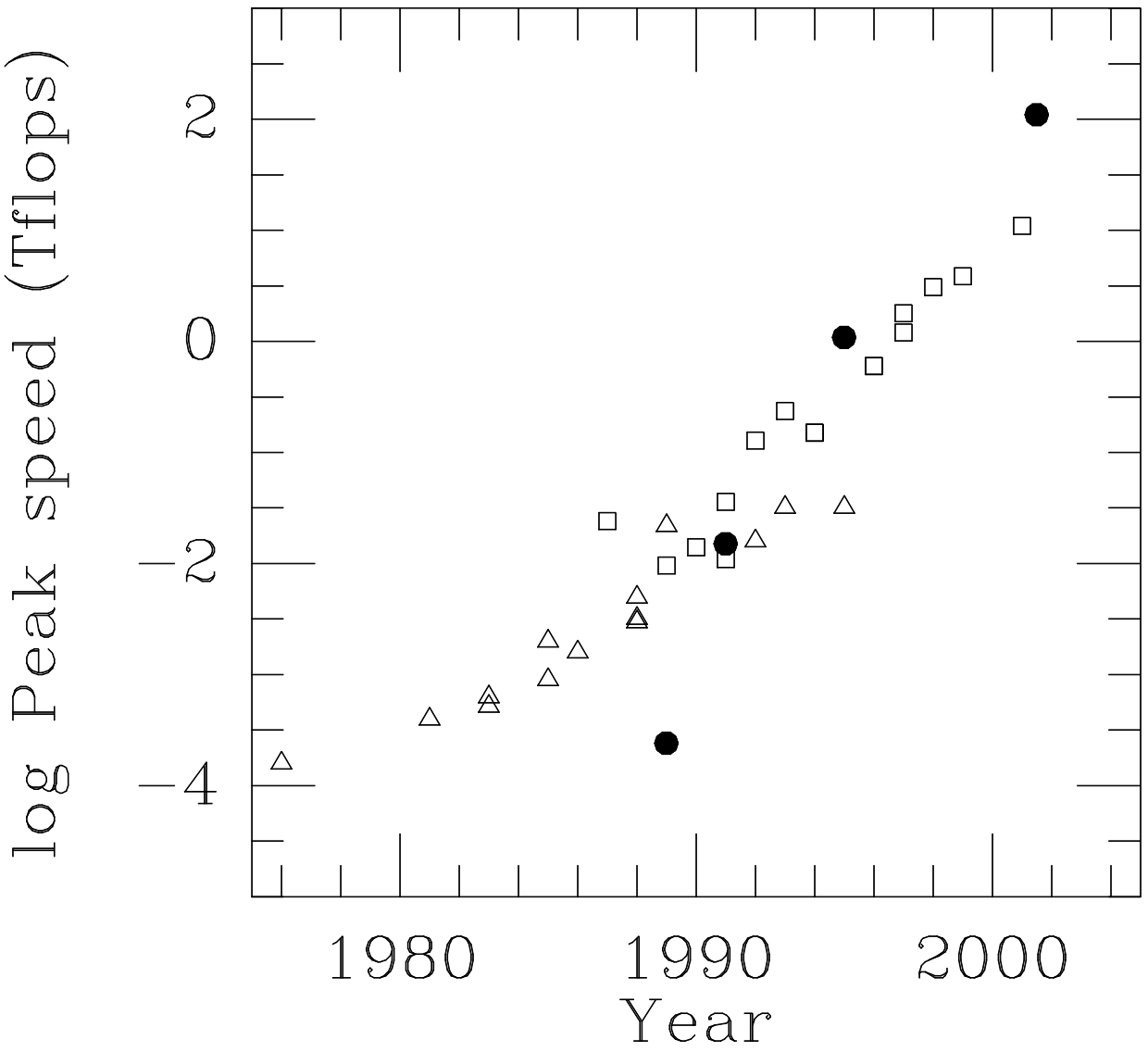
ボード/チップレベル:

- i 並列 (メモリ 1 つから同じデータを受け取って、別の粒子への力を計算)
- 仮想マルチパイプライン
- 演算パイプライン自体:中間結果をレジスタに戻さない

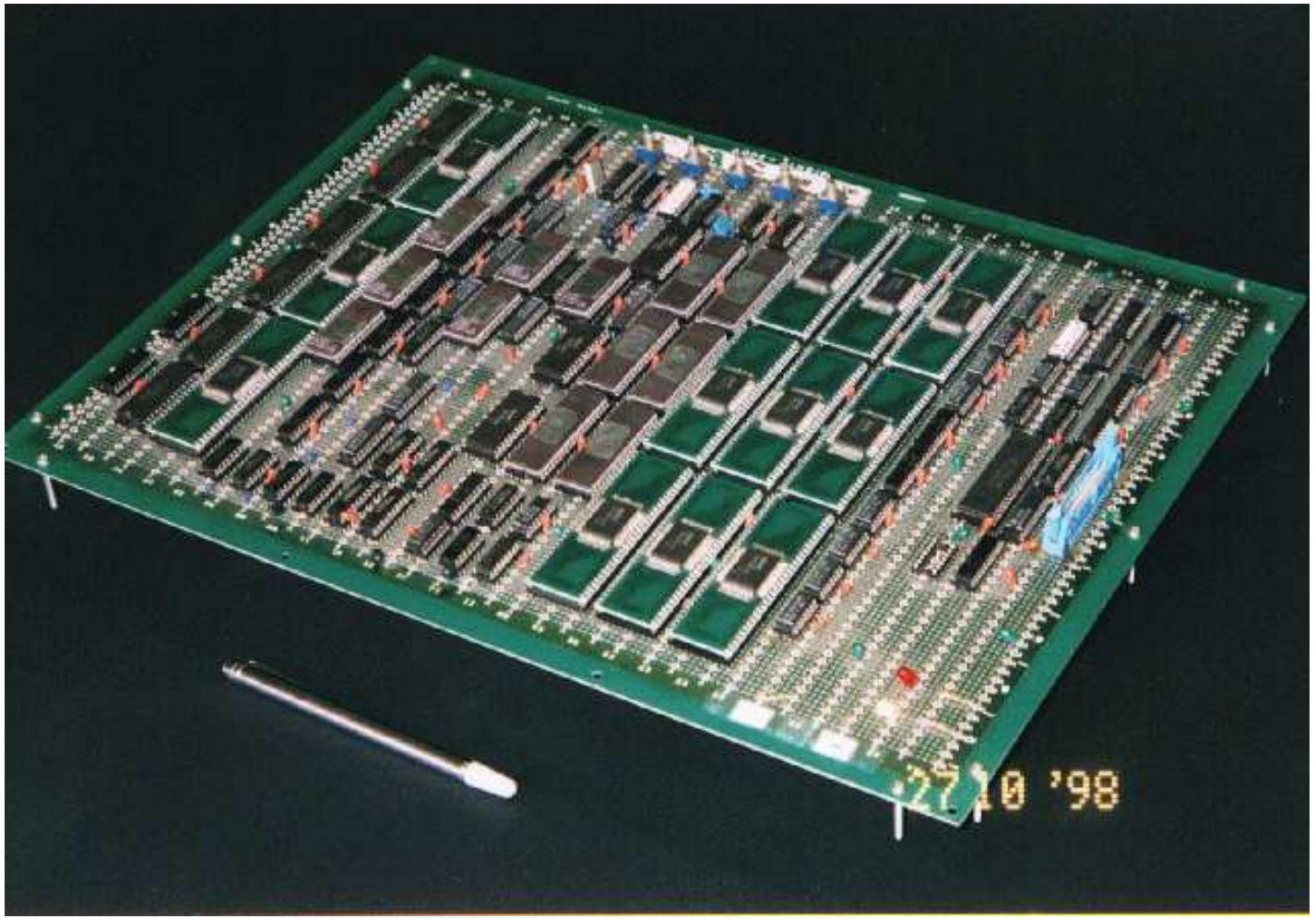
GRAPE アーキテクチャの発展

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-1 の中身

「初めてのデジタル回路」

当初の目標

- とりあえず演算パイプラインのようなものを作る
- ホスト計算機につないで動かす
- 意味がある計算ができるかどうかはあまり気にしない

最初の構成

- 演算は ROM テーブルルックアップ。データは 8 ビット
- 通信は GPIB

計算/通信比: 粒子数 1 万くらいで性能がでればいいことにすると
動作クロック 5 MHz くらい = 1 粒子のデータのやりとりに 1ms くらいは使っていい
データ量が数十バイトくらい → データ転送速度は 100KB/s くらい欲しい

シリアルでは不足
SCSI は難しくて良く分からなかった

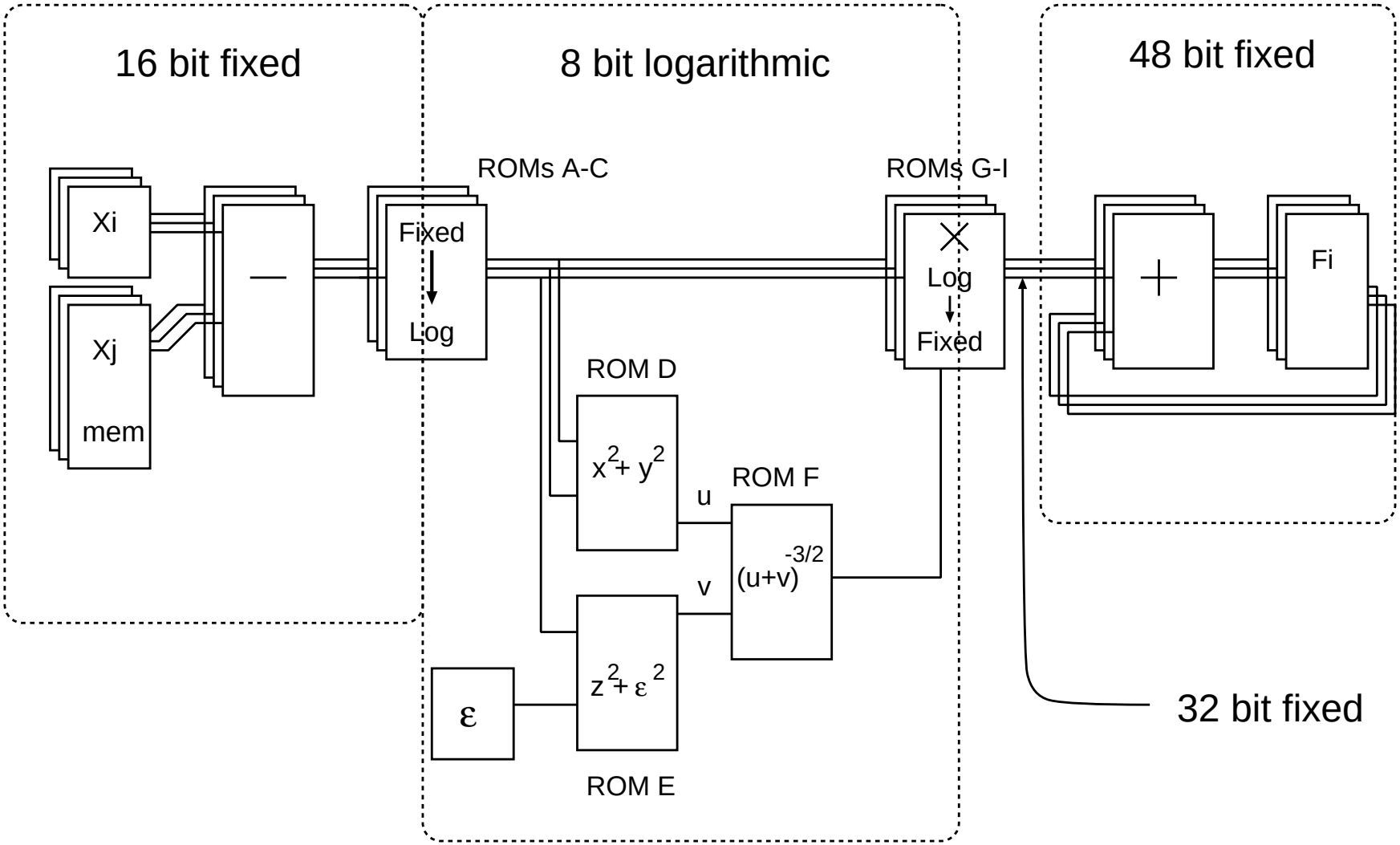
実用になるものへ

途中の計算 (2 粒子間の力の相対精度) は低くても、

- 最初の座標の引き算
- 最後の足し上げ

だけを (そこそこ) 高い精度でやれば、問題によっては十分使
える (ツリーコードに比べれば高い精度になる)
ということに気が付いたので、そっちに変更
最初の座標の引き算: 16bit 固定小数点、最後の足し上げ:48bit
固定小数点
とする。

GRAPE-1 パイプライン



開発中のトラブル

ハードは割合順調に出来たらしい (実は伊藤君が担当だったので良く知らない)

できてから、性能がでない、、、

当初: PC-98 ホスト、どこかの GP-IB ボード (TI の GP-IB インターフェースチップがのってるだけのもの)。
通信速度は問題なし

ホストが遅い、メモリものらない、、、、というので何故か GP-IB インターフェースカードがあった Sony NEWS-830 につなぐと、、、、

通信オーバーヘッドというもの

NEWS につなぐと全くまともな性能が出ない。

原因: GPIB を使った通信は read/write システムコールを使う。で、これが 1 回呼ぶと 1ms くらいかかる。

GRAPE-1: 1 粒子もらって、それへの力を計算して、結局返す。

NEWS 830 は I/O プロセッサ付き: オーバーヘッドは一層大きい

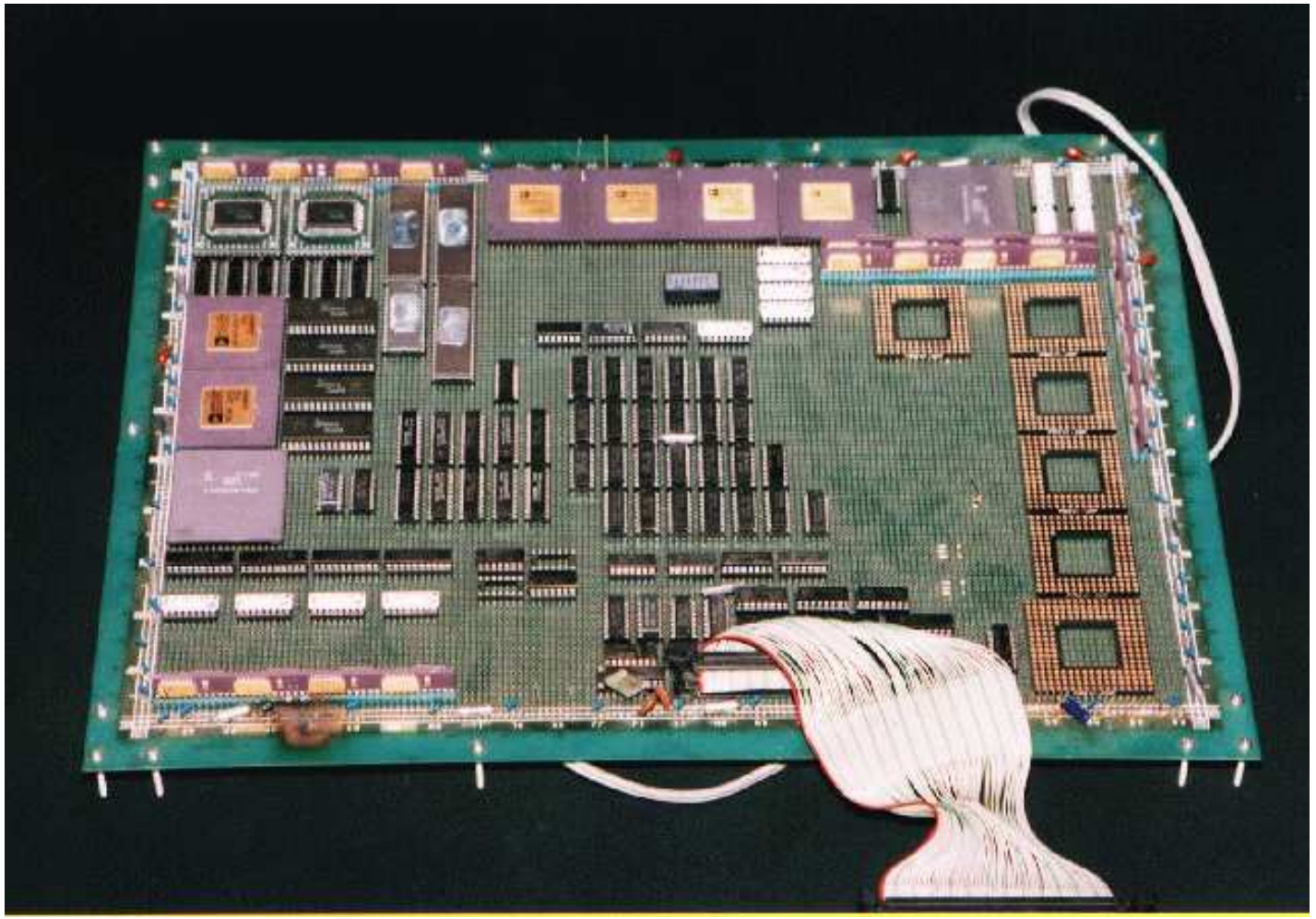
最初の対応: PC-98 をバッファリングに使う。

次の対応: I/O プロセッサのない Pop NEWS にして、GP-IB コントローラのレジスタをユーザープロセスがいじりまわす。これで 100KB/s くらいでるようになった。

教訓

- 通信ソフトウェアは厄介である
- 教科書的に「良い」方法が良い結果になるとは限らない
- 結果が良ければ「邪道な」ことでもなんでもかまわない

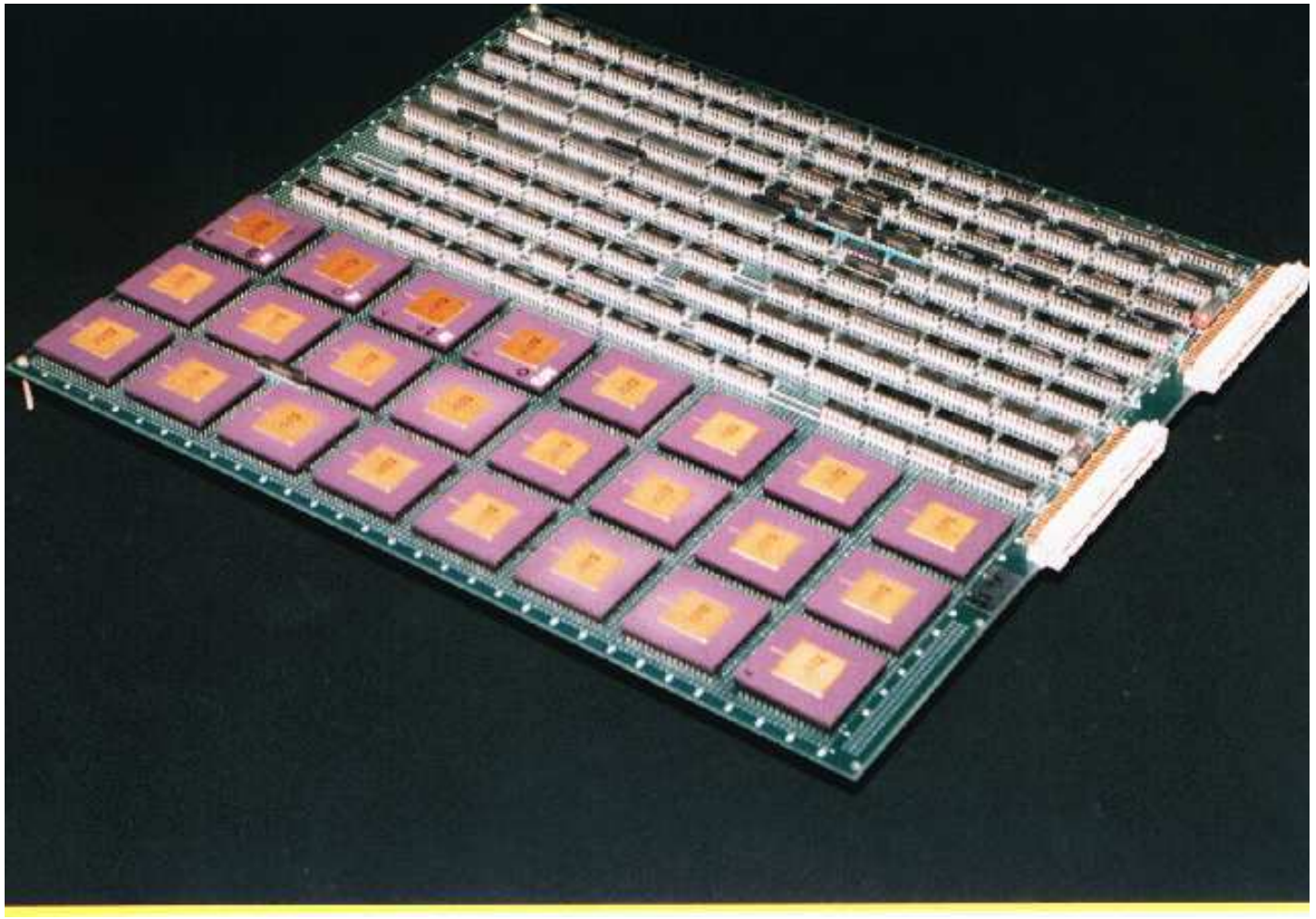
GRAPE-2



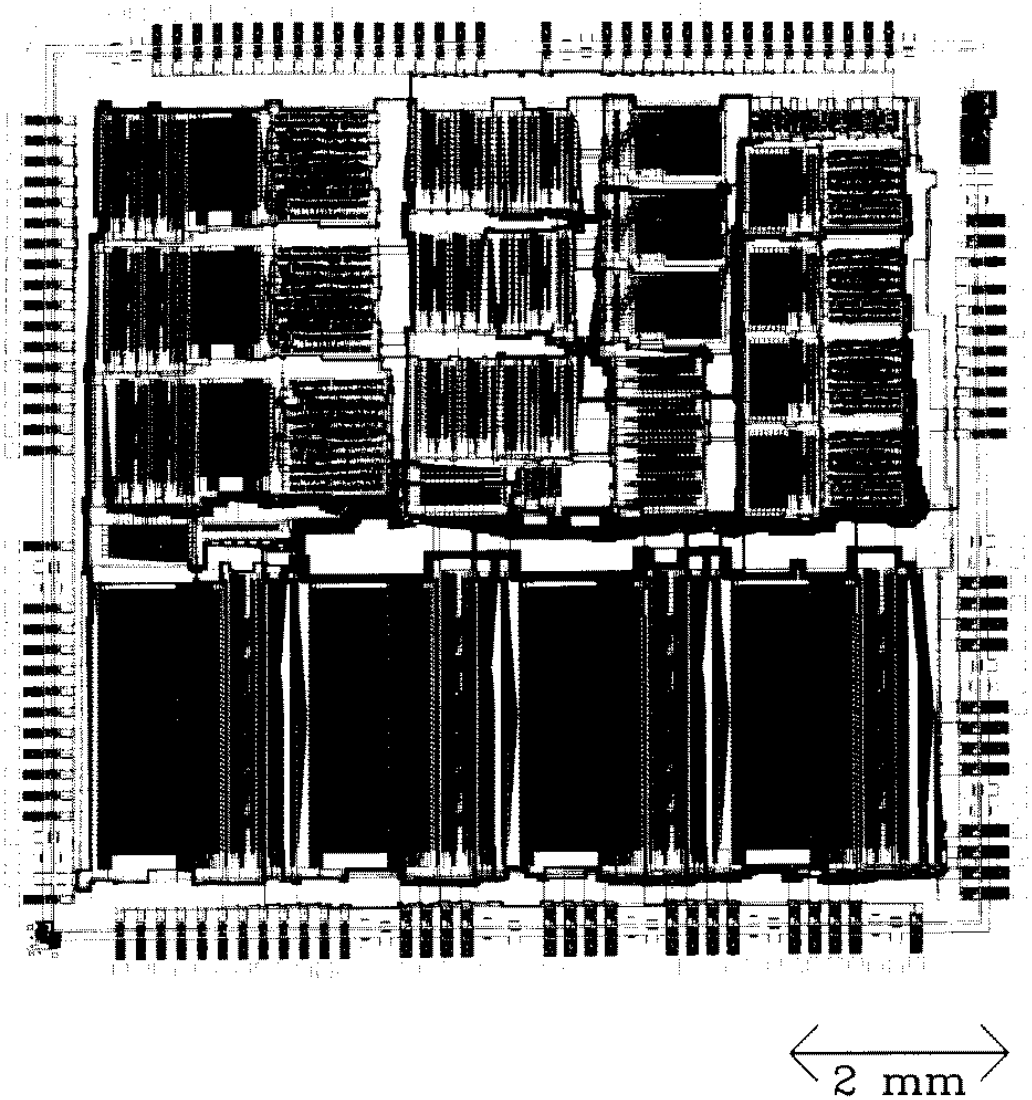
GRAPE-2 のまとめ

- 8 ビット演算とかは止めて普通に浮動小数点演算 (倍精度は最初と最後だけ)
- 通信は VME バス
- ホストは VME バスに PIO アクセス。 DMA は使わない (GRAPE はマスターにならない)

GRAPE-3



GRAPE-3 チップ



GRAPE-3 チップ

- 仕様決定、シミュレータ (C で記述) は牧野がやった
- 論理設計以降は富士ゼロックス (橋本、富田)
- SCS Genesil で設計
- ファブは NS. 1μm

チップを作るのはどれくらい心臓に悪いか?ということ

2回チップを作る予算があったことはない=一発勝負

普通の責任分担

- テストベクタが通らなければ向こうのせい
- テストベクタが通ればこっちのせい

「理論上は」「完璧な」テストベクタを作ればこちらのせいである問題は起こりえない。

実際はそうはいかないけど、、、

普通のチップとの違い

GRAPE の場合、マイクロプロセッサとかに比べるとはるかに楽

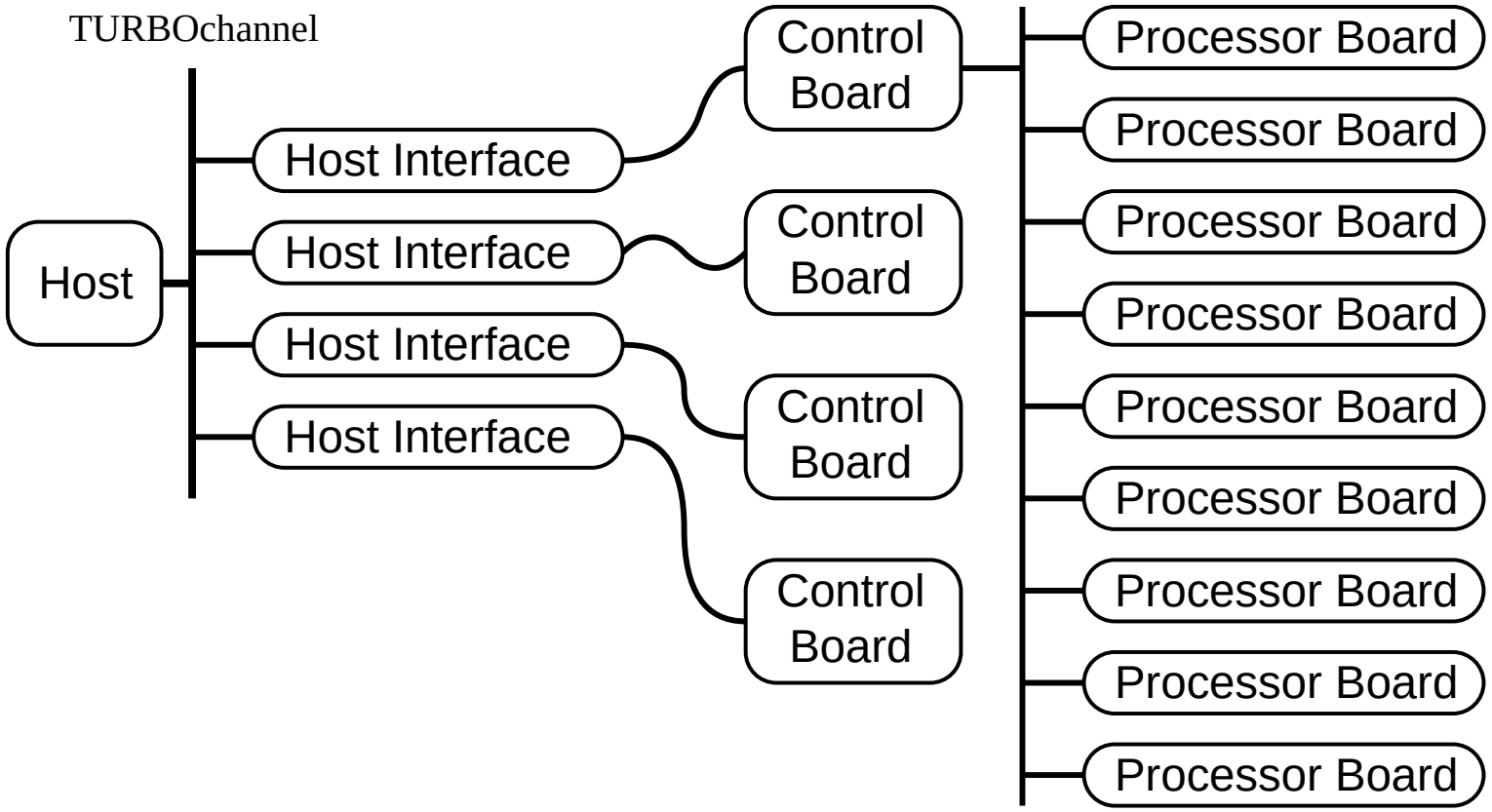
「内部状態」というものがほとんどない。計算中/停止の違いくらい。これはアキュムレータの制御だけ。

演算回路の仕様自体が問題。小規模なシミュレーションでは動いても、実際にハードウェアができて初めて可能になる大規模な計算で問題なく動くかどうかはわからない。

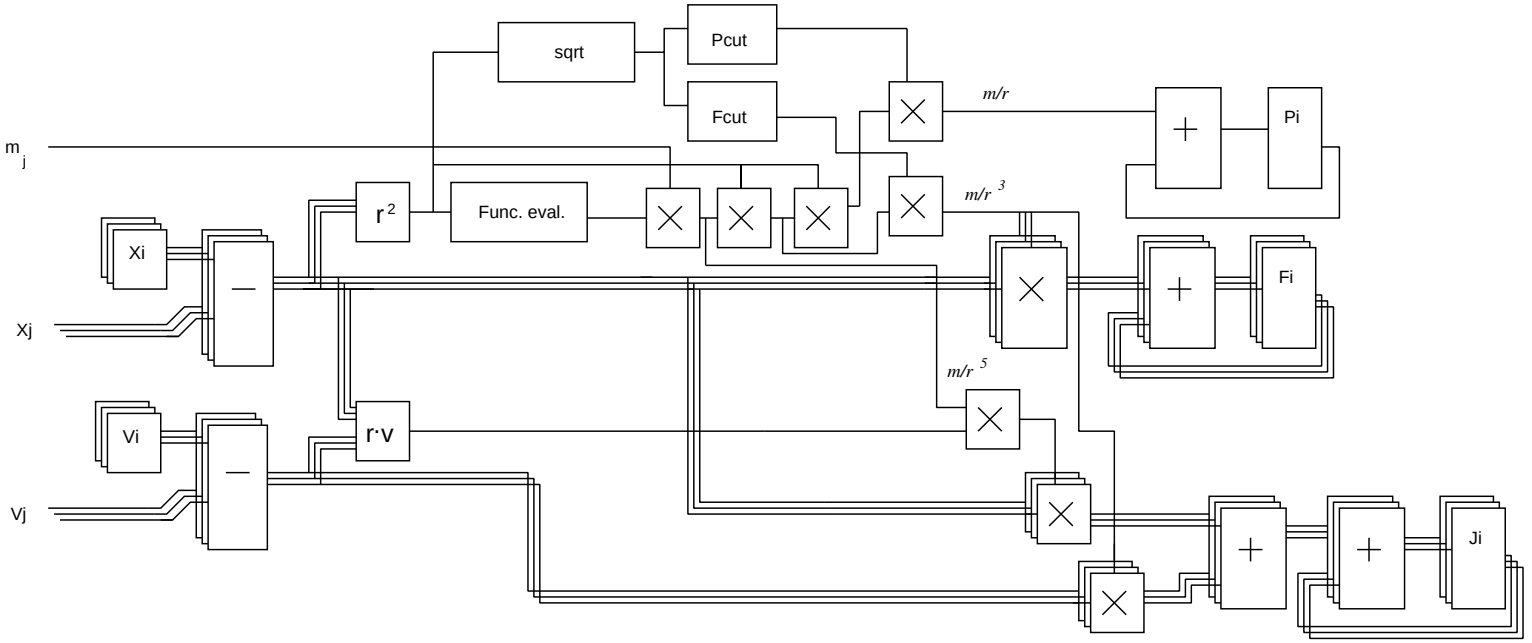
その他、予想もしない問題は起きる。

GRAPE-4

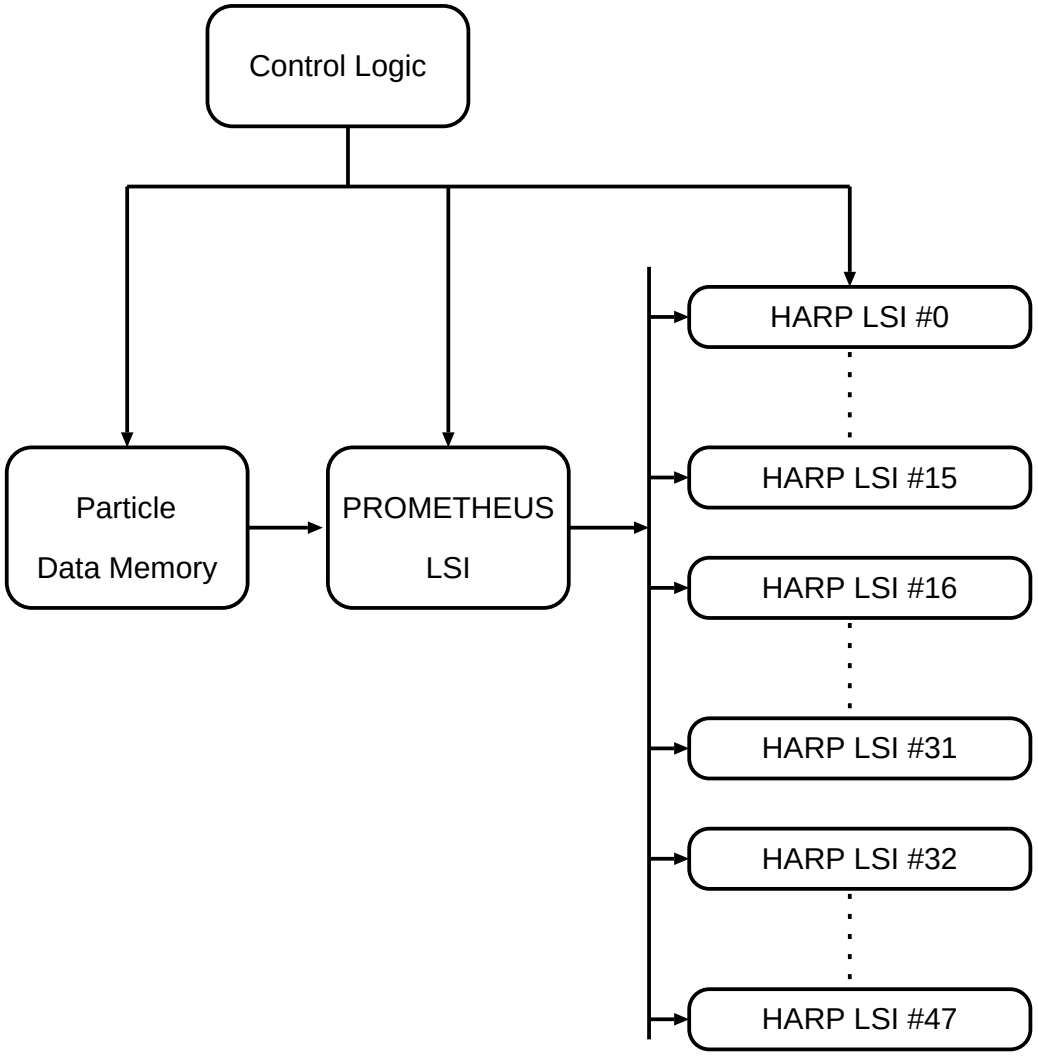
GRAPE-4 の構成


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



GRAPE-4 演算ボード



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-4 における通信

演算性能:GRAPE-3 の 100 倍

通信:単一ホストでできるベストに近いところで済ませられればそうしたい。

とすると 100 MB/s くらい。

1 粒子データ 200 バイトくらい

とすれば、 10^5 粒子くらいでまあまあの性能がでる計算になる。

まあ、賞でも取ろうという時以外は機械を分けて使うので、これくらいの見積りで十分。

100MB/s をどうやって出すか? (1993年に)

- DMA は必須
- ホストは、、、、 DEC Alpha、バスは TurboChannel

オーバーヘッドを避けるためにはDMA のたびにカーネルにいったりしては困る。
というわけで
ユーザープロセスのバッファに使うページの物理アドレスをカーネルの内部関数をなんかして調べて、それをユーザープロセスがインターフェースカードの DMA アドレスレジスタに直接書く。

最近なら？

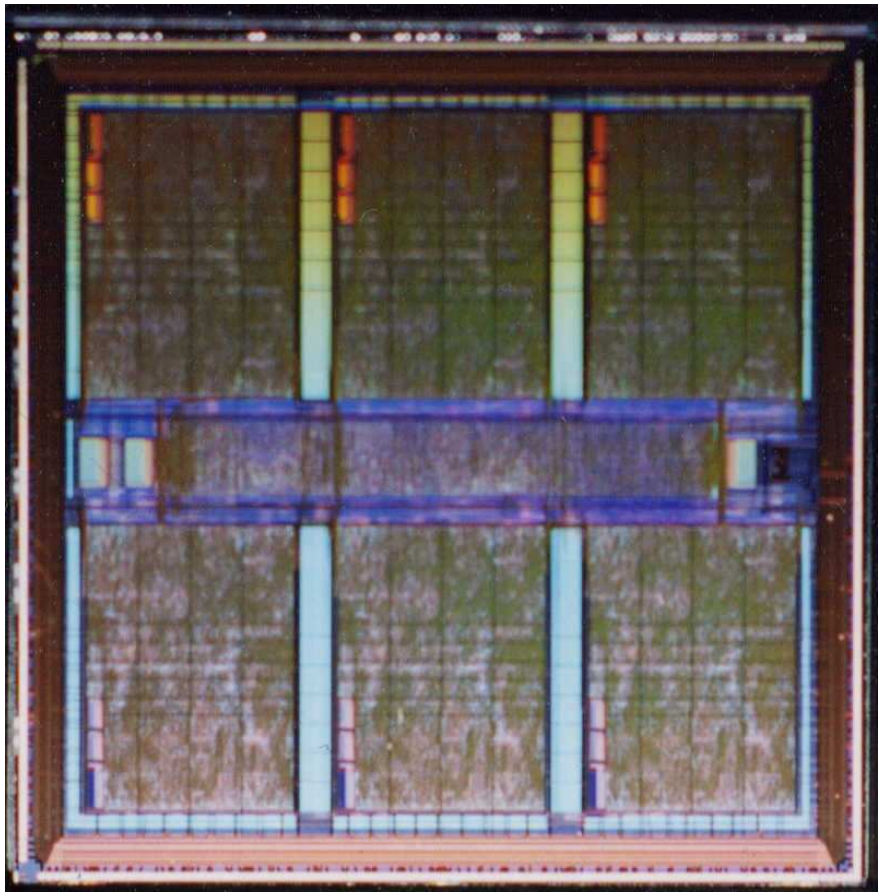
最近のまともな OS では、カーネルが確保したバッファをユーザー空間にマップしてくれるみたいなので同様のことがもうちょっと安全に実現できる。

某所で GRAPE-4 と同時期に開発していた某並列計算機では「ユーザーモードDMAによる通信」というのが売り物だったような、、、

GRAPE-6 について

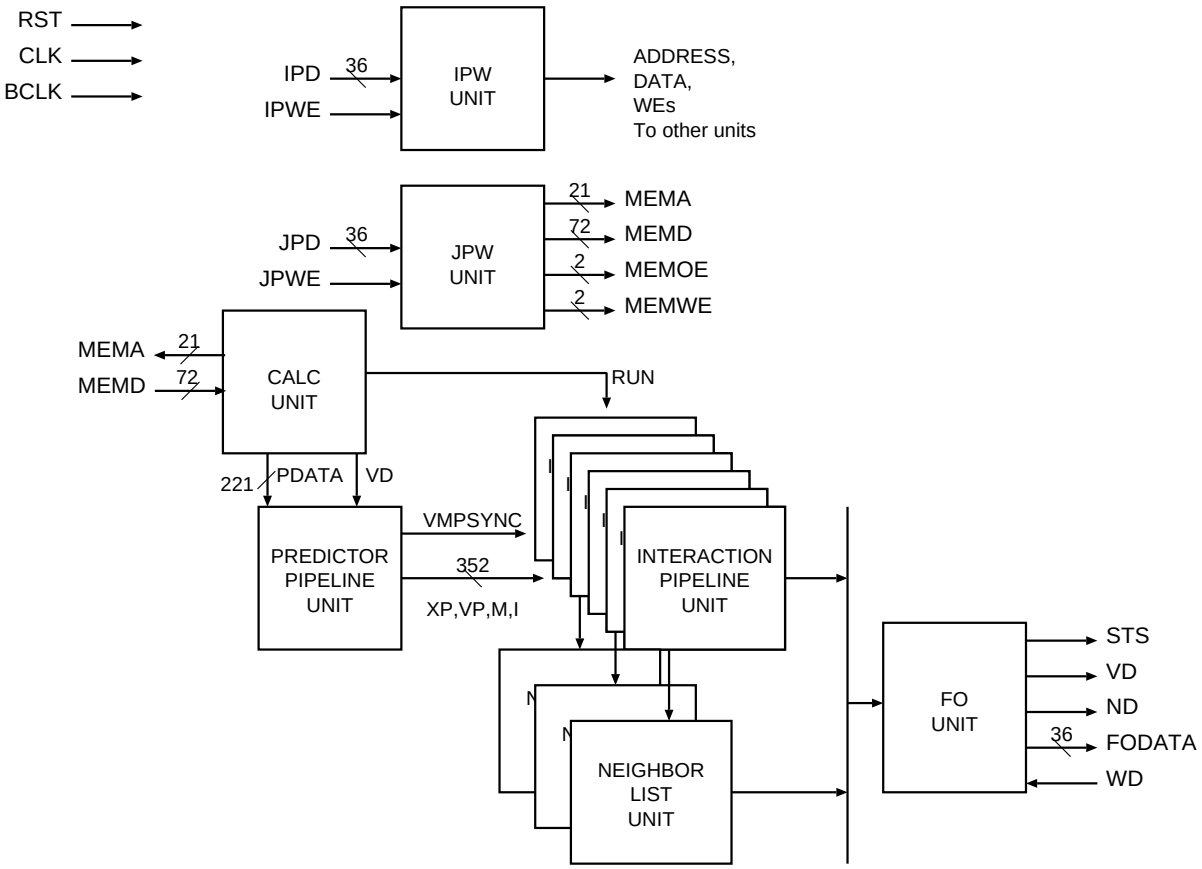
- プロセッサチップ
- プロセッサモジュール
- プロセッサボード
- ネットワークボード
- 全体

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

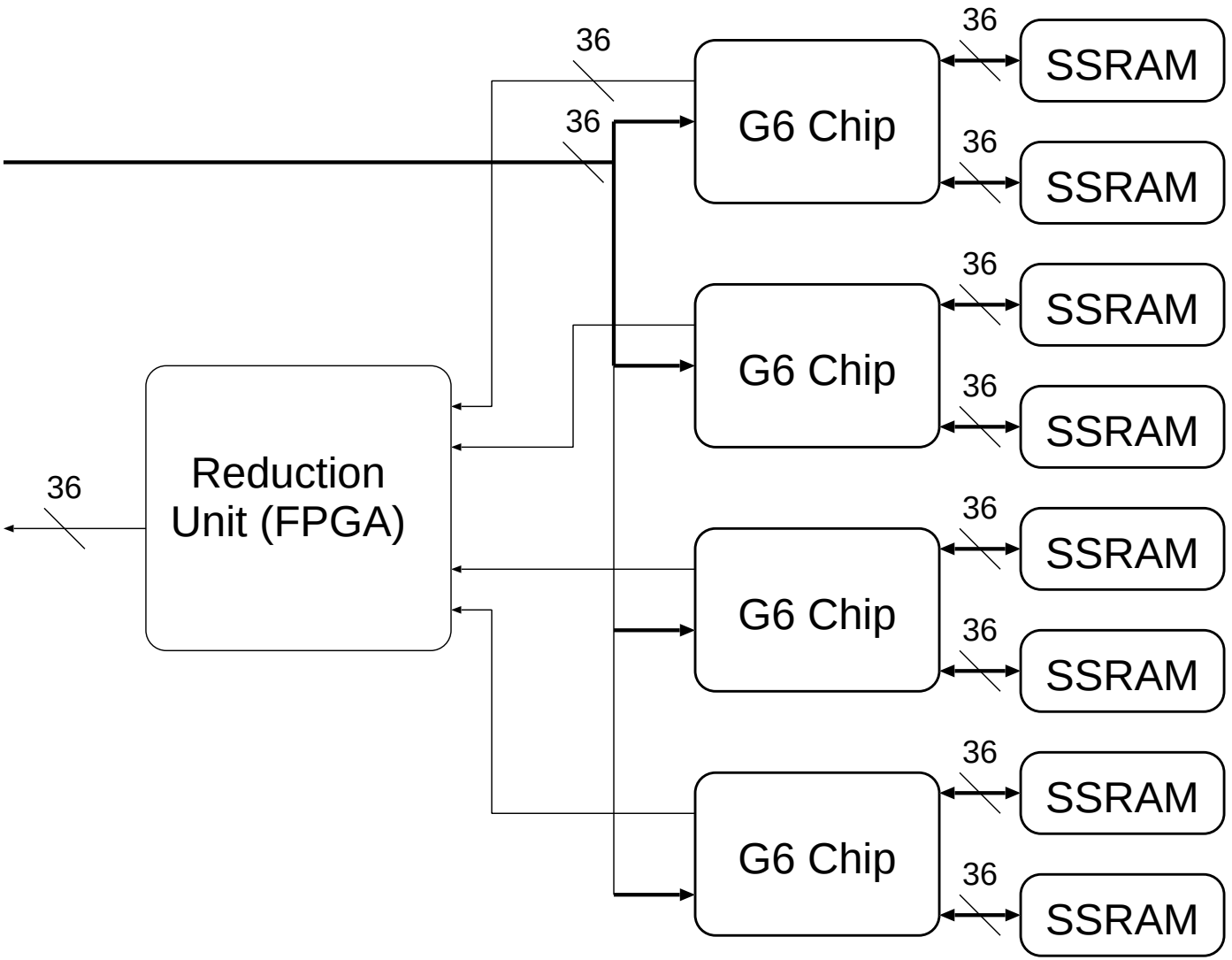


GRAPE-4 プロセッサボードの全機能を集積

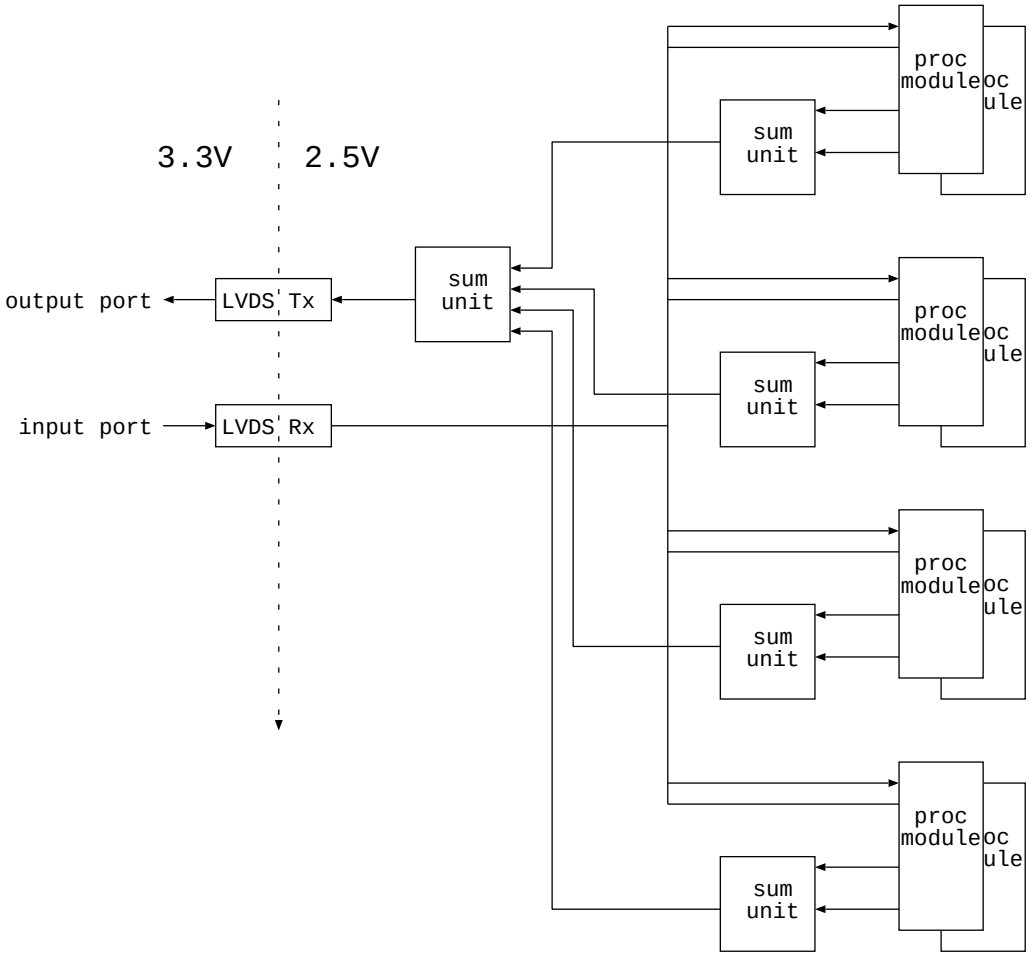
- ホストIF
- メモリIF
- パイプライン本体
- 制御回路

GRAPE-6 processor module

GRAPE-6 processor module



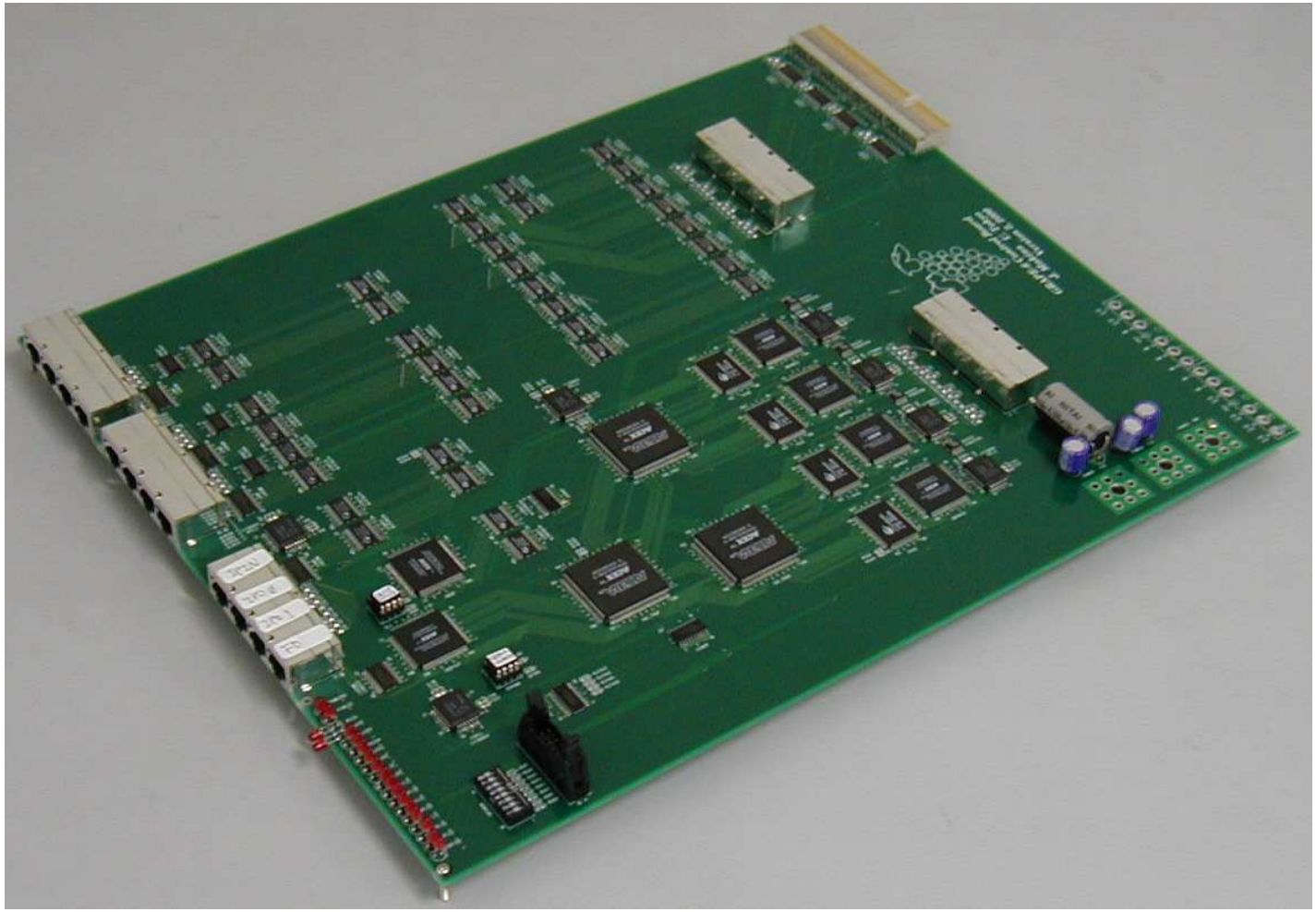
GRAPE-6 Processor board



- 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)

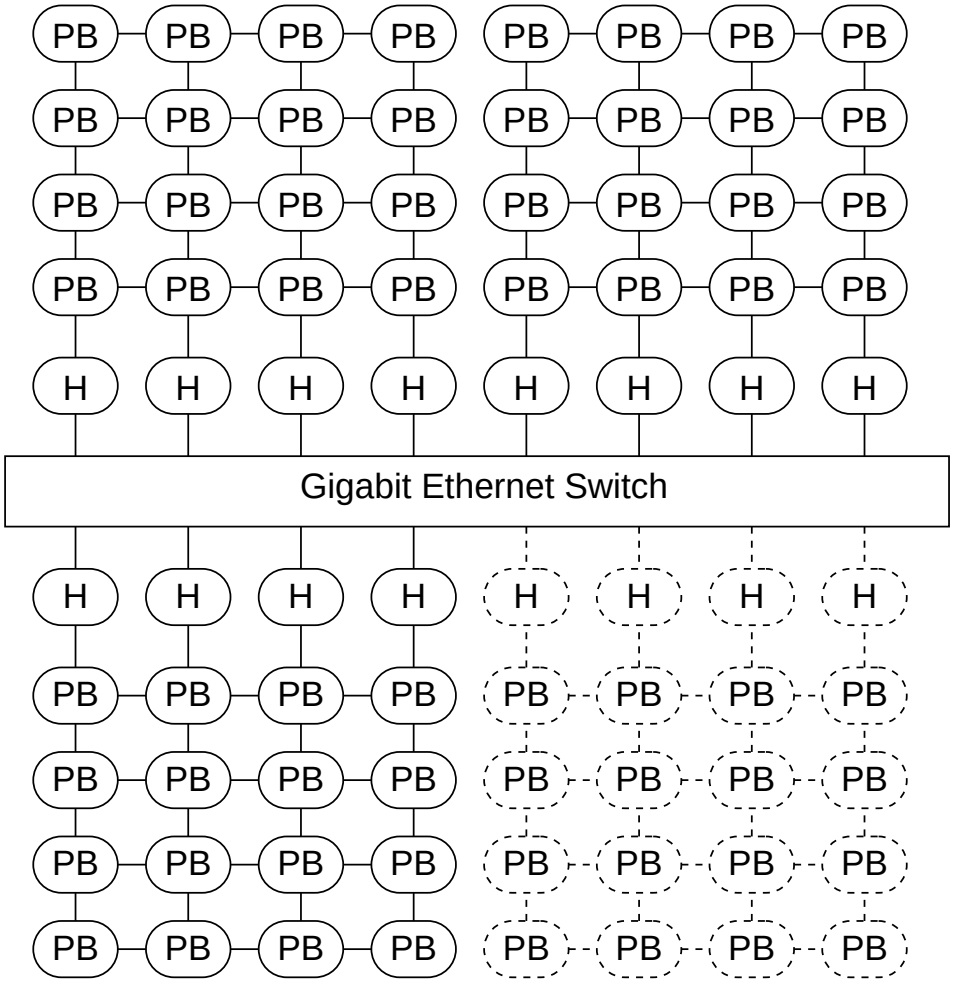
GRAPE-6 processor board

GRAPE-6 network board



GRAPE-6 host interface board

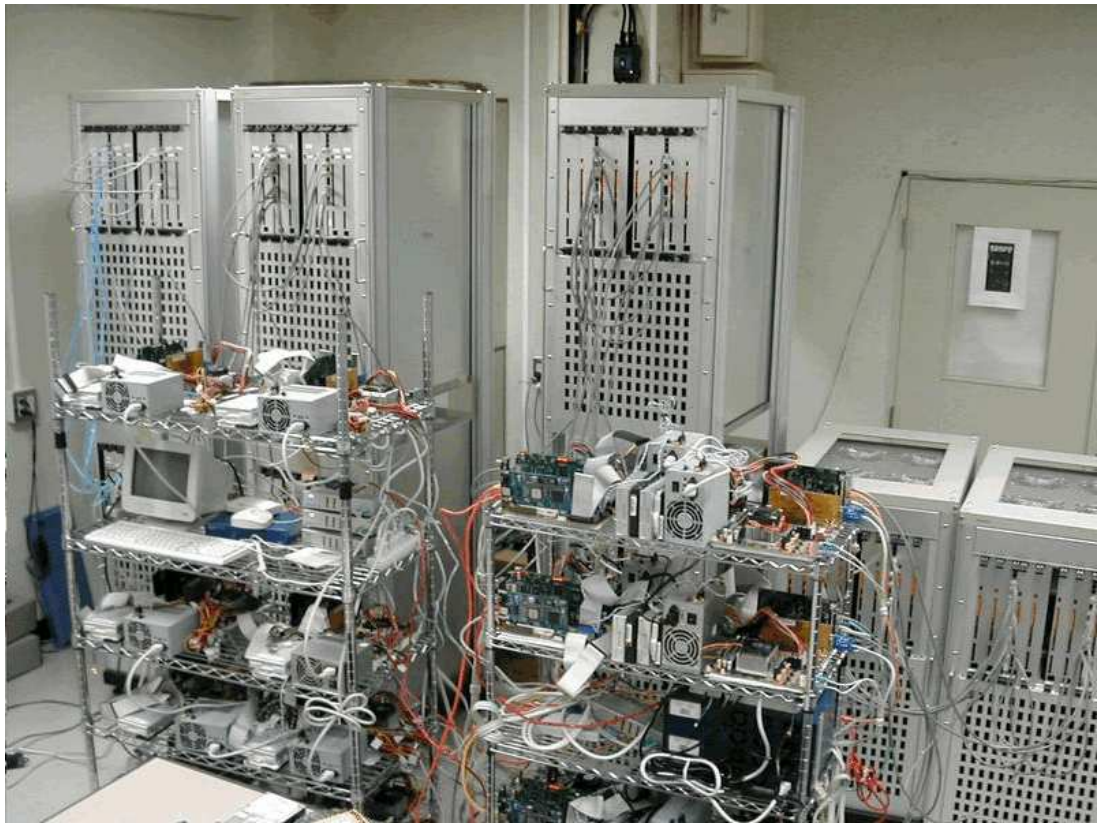
The full 64 Tflops GRAPE-6 system



- 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.

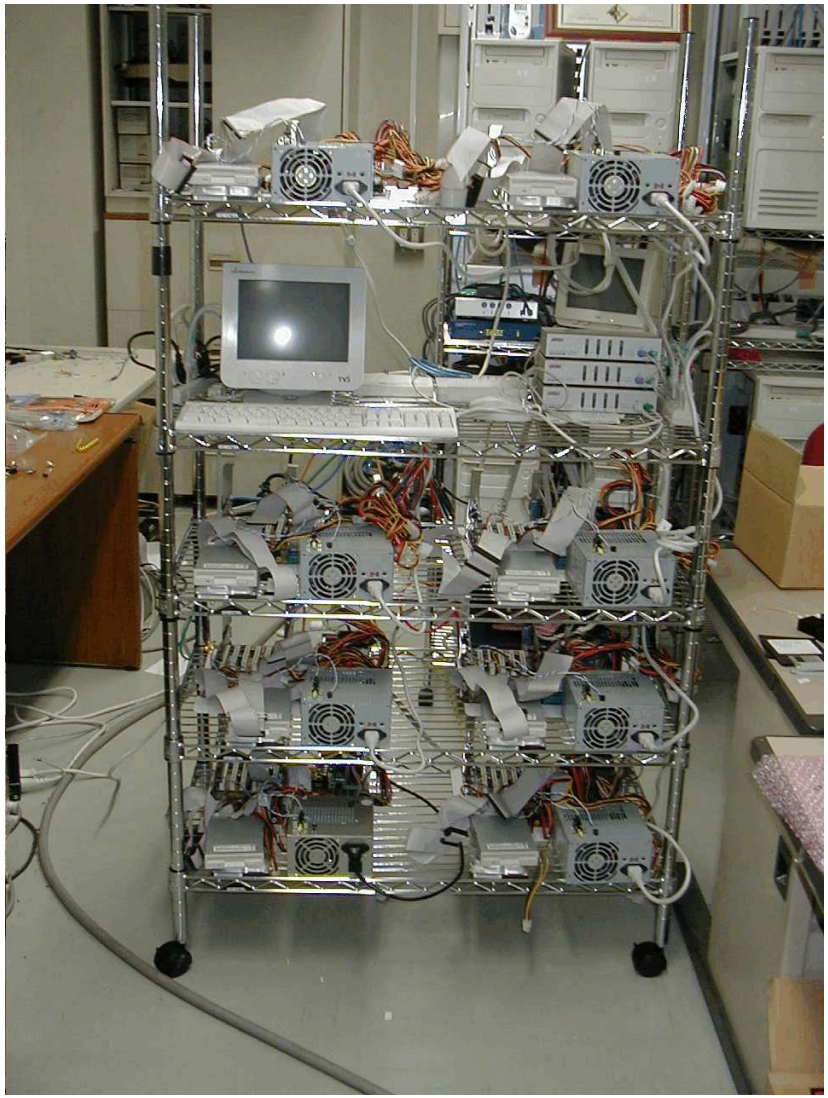
The 64-Tflops GRAPE-6 system

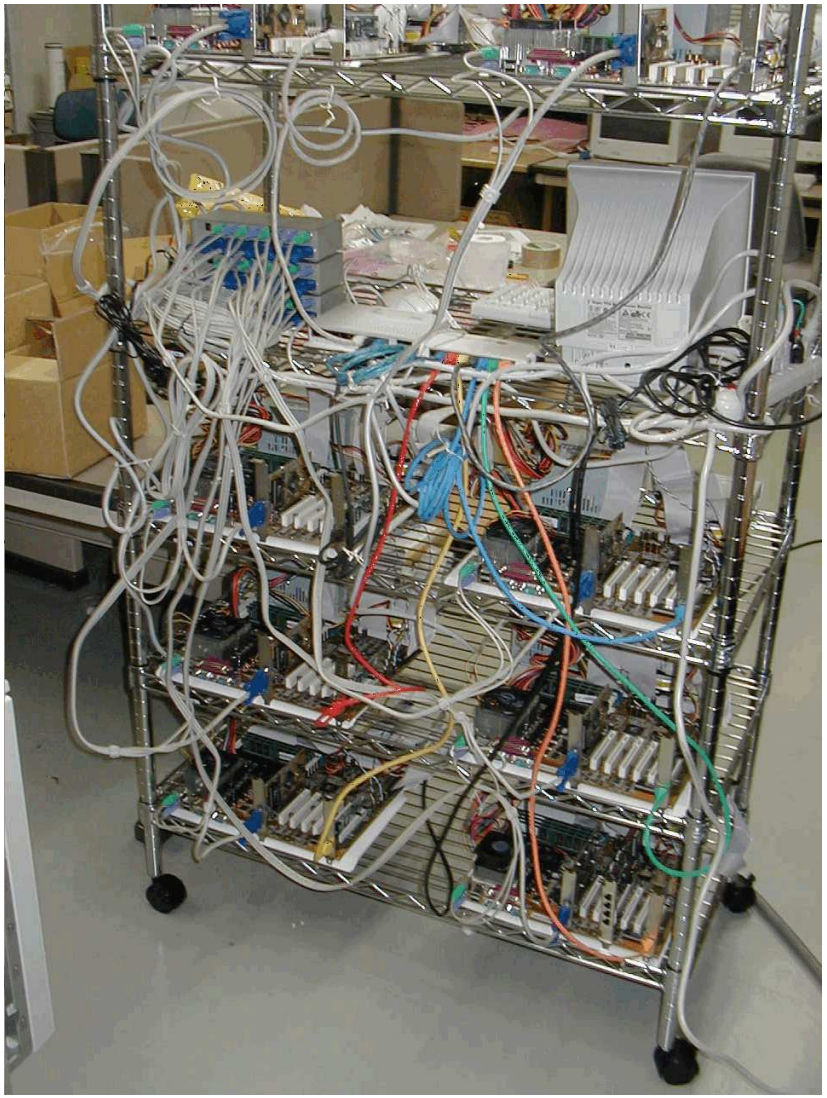


Present 64-Tflops system.

4 blocks with 16 host computers.

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。

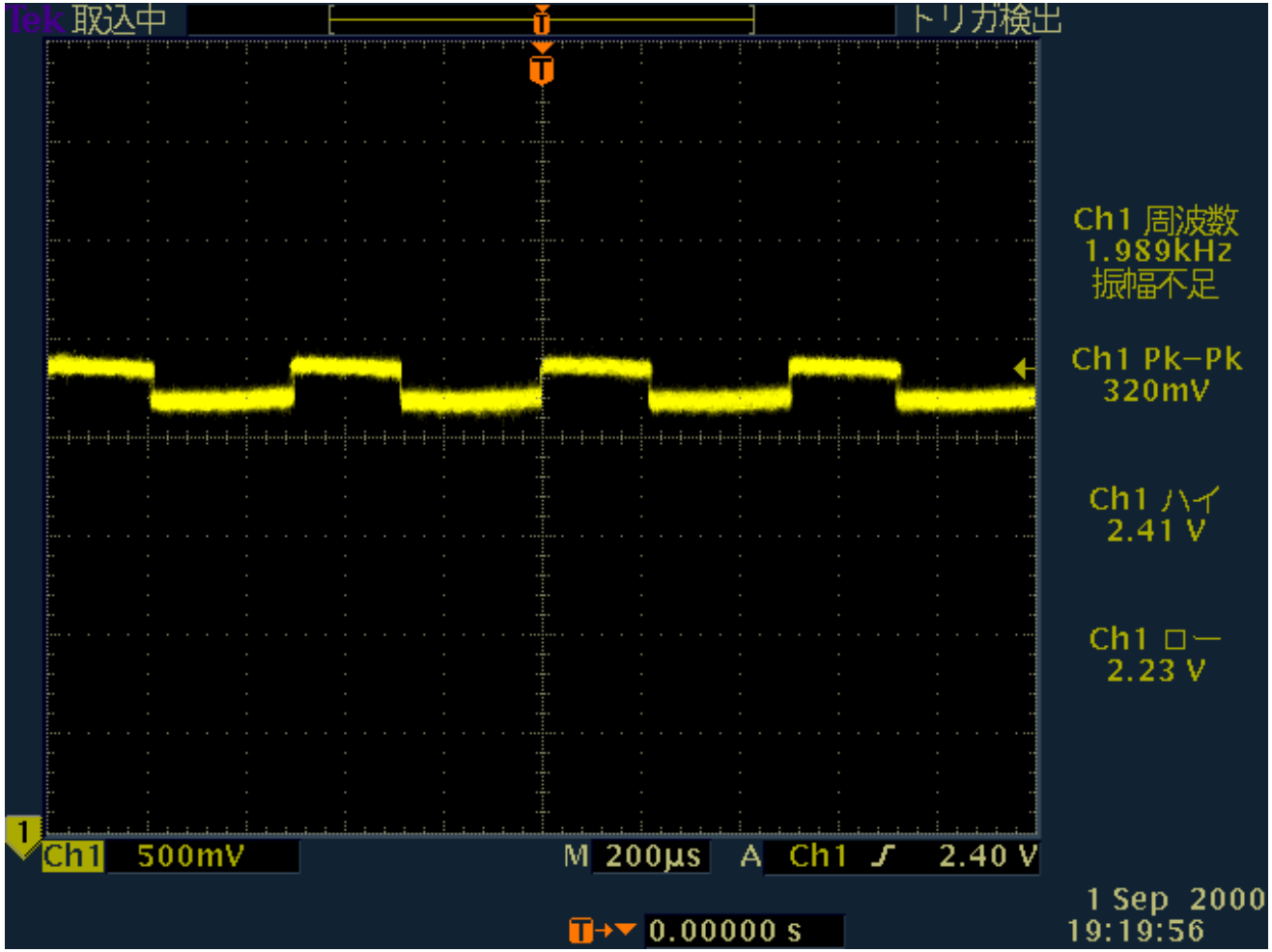
形式的な原因 : チップ製造を担当した会社の設計ガイドラインに、コアロジックの消費電力から電源ピン数等への制約がなかった (らしい)

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


Tek 取込中
トリガ検出
Ch1 周波数
1.989kHz
振幅不足
Ch1 Pk-Pk
320mV
Ch1 ハイ
2.41 V
Ch1 ロー
2.23 V
Ch1 500mV
M 200µs A Ch1 2.40 V
0.00000 s
1 Sep 2000
19:19:56

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。

もうちょっと細かい問題

GRAPE-6 における細かい問題の例

- 内部クロック 90MHz、外部クロックは 1/4 なのに、何故か 入力データのパリティチェック回路は内部クロックで動作。
- メモリにデータを転送する時、2 粒子以上のデータを連続して送れない (アイドルサイクルが必要)

後者はテストパターンの不備でもある。状態があるものの完璧なテストは難しい。

どちらも、性能に多少影響があるくらいですんだのは不幸中の幸い。

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

$\longrightarrow$ 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、、、

FPGA の大雑把な評価

同じテクノロジーのカスタムチップに比べて

- 集積度で 10-30 倍
- 速度で数倍 (この差は最近小さくなった)
- (量産) 単価で数倍 (これも縮小傾向)

劣る。

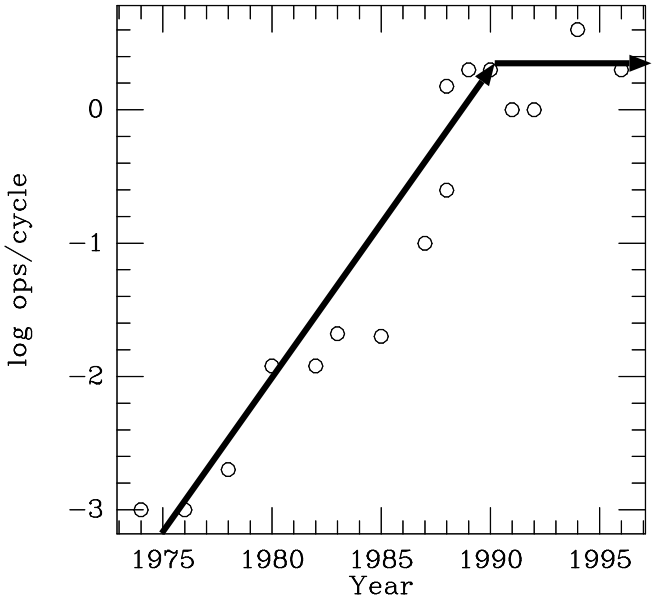
初期費用:実際上 0。

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。