重力多体系シミュレーション用のアルゴリズムとハードウェア

国立天文台
理論研究部/天文シミュレーションプロジェクト (CfCA)
牧野淳一郎

今日の話の構成

1. 重力多体系のシミュレーションではどんなことをしているか？
2. 重力多体系のシミュレーションでは何が難しいか？
    - 時間方向
    - 空間方向
3. どんな方法を使うか？
4. 計算機の方向

多体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

そもそもどんな対象を考えるか？

大抵の天文学的対象：自己重力系
一つの星：重力とガスの圧力がつりあう
それよりも大きい構造：重力を構成要素の運動エネルギーで支える

- **太陽系**
- **星団**
- **銀河**
- **銀河団**
- **宇宙の大規模構造**

例: 星団

- 球状星団
- その他、丸い星団

丸い: ある程度熱力学に緩和していることを意味する。(円盤銀河なんかとは違う)
しかし、熱平衡状態ではない (自己重力系には熱平衡状態は存在しない)

球状星団 M15

地球からの距離 10kpc (1pc: 3 **光年くらい**)
星の数: 200 **万くらい**
質量: **太陽の** 50 **万倍**
半径 (質量の半分がある): 4 pc

球状星団 G1

アンドロメダ (M31) にある、局所銀河団で最大の球状星団

質量:

見積もり (a) 1.5×10^7 太陽質量 (Meylan et al. 2001)

見積もり (b) 8×10^6 太陽質量 (Baumgardt et al. 2003)

2 倍の違い: 力学モデルの違いから。

球状星団を研究する理由

沢山あるが、、、

- 銀河でもっとも古い星の集団 — 球状星団がどうやってできたかは銀河自体がどうやってできたかと関係
- 出来てから星形成やガスとの相互作用が基本的にはない — 理想的な「自己重力質点系」に近い
- より複雑な銀河中心等を理解するための鍵
- 激しい星形成領域 — 比較的大きな星団になっているものが多いらしい。何が見えるか、何が起こるかを理解することは星形成過程の理解にも重要。
- 中間質量ブラックホール ???

Portegies Zwart 作ムービー

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・計算方法の限界 (今日の話の主題)**
- **素過程の理解 (今日は省略)**

解く方程式 — 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

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

数値計算は基本的にはこれを使う

何故多体問題を直接数値計算するのか？

もうちょっと違う方法

- 解析的ななんか
- 分布関数に対するフォッカープランク方程式

現実問題として、、、

- 解析的には解けない
- 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。

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

基本的な事実: 速く計算できればそれだけ良い計算が可能になる。(例は時間に余裕があればいくつかあげたい)
それしか方法がないわけではないが、「速く計算できるようにする」のは極めて重要な方法。

- **計算法を改良する**
- **速い計算機を買う**
- **速い計算機を作る**

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

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

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、どんな方法を使っているかという話を簡単に。

計算法 — 時間領域

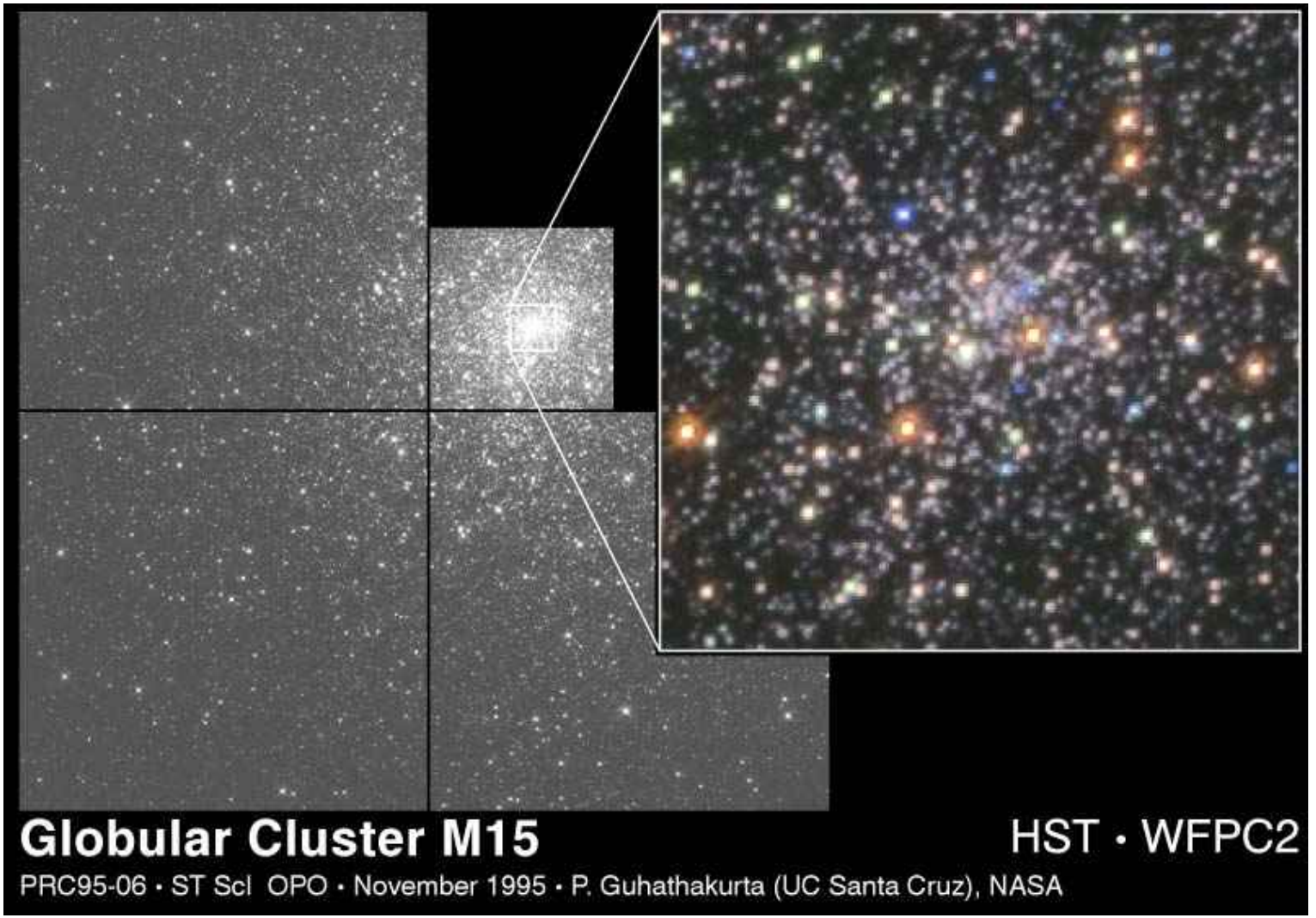
算数としては、単に常微分方程式の初期値問題の数値解。
ナイーブに考えると、いろんな公式がライブラリであるので、それを使えば済みそうな気がする。
それだけでは済まないのが問題。
済まない理由:

- 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: 球状星団の例



構造形成による効果 (続き)

- 「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？

星の運動のタイムスケール: 中心に近いほど短くなる。
等温カスプ (密度が半径の -2 乗に比例): タイムスケールの分布がべき乗になる。

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りすぎる。

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。

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 $O(N^{1.3})$
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。

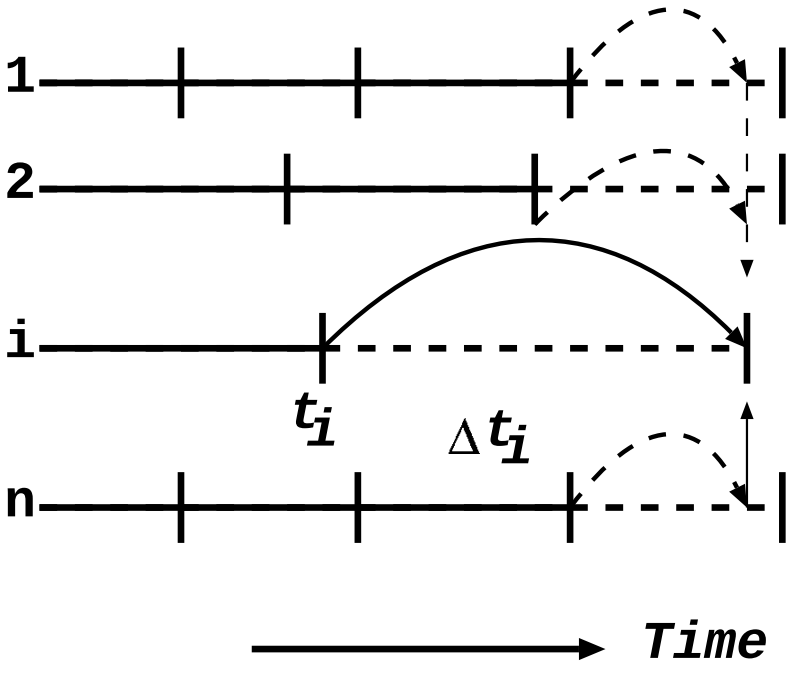
独立時間刻みの原理

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。

ある粒子の時刻 $t_j + \Delta t_i$ で他の粒子の位置が高精度で必要:
予測子・修正子型の公式を使う。

独立時間刻み

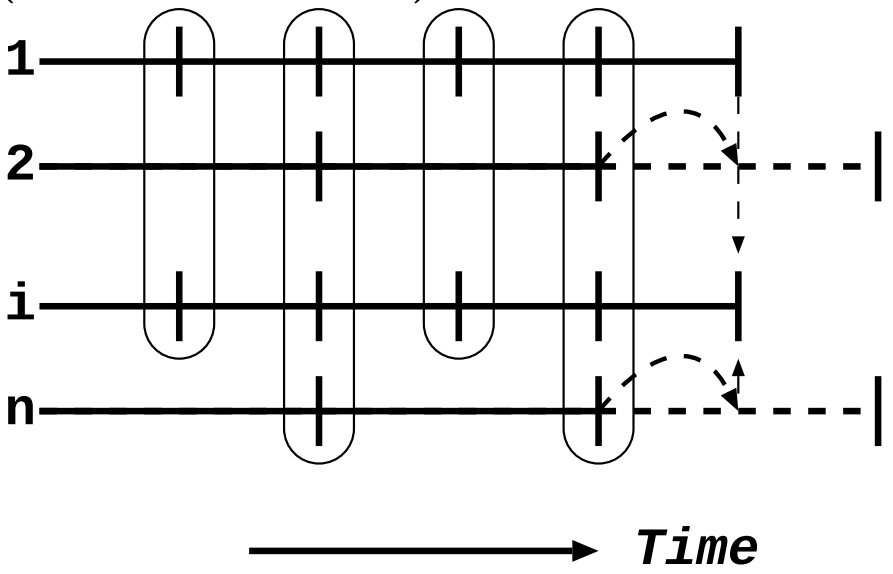


(Aarseth 1963)

- 各粒子にそれぞれ時刻と時間刻みを与える
- 「イベント駆動」時間積分 — $t_i + \Delta t_i$ がもっとも小さい粒子が積分される

ブロック時間刻み法

(McMillan 1986) ベクター／パラレル計算機のための改良



- 時間刻みを 2^{-k} に制限する。
- $t_i + \Delta t_i$ が同じ（Δt_i は違ってもよい）粒子は同時に積分される。

$O(N_c^{2/3})$ 個の粒子（N_c は高密度コアのなかの粒子数）を並列計算

— そんなに大きな数ではない。

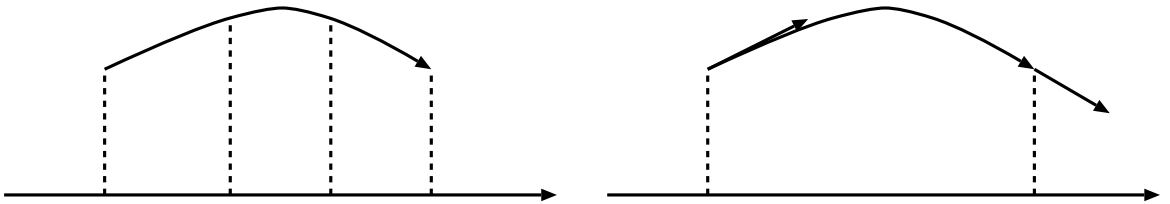
時間積分公式に対する要請

- 高次の予測子が必要 (他の粒子の位置が必要)
- 可変時間刻みが必要
- 積分区間の途中で加速度を計算するような方法は使えない。
  - 線形多段階法は使える
  - ルンゲ・クッタは使えない
  - シンプレクティック法は単純には使えない

時間積分公式

次数：「4 次くらいが適当」(JM 1990)。
もっと高いほうが良い: (Nitadori and JM 2007)

- Aarseth scheme (Aarseth 1963): **可変刻みのアダムス法、** PEC **モード、4 次、2 階の方程式用。**
- Hermite scheme (JM 1990): **ラグランジュ補間（ニュートン補間）の代わりに、加速度の一階時間導関数も使ってエルミート補間を構成。**



高次エルミート

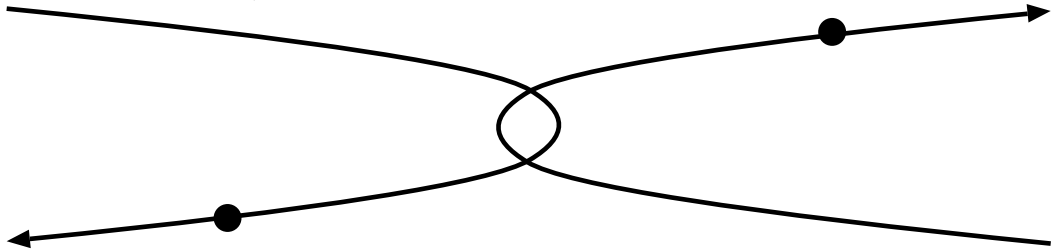
Nitadori and JM 2007

- 2 階導関数まで直接計算してエルミート補間: 6 次公式
- 3 階導関数まで直接計算してエルミート補間: 8 次公式
- 予測子は前のステップの値を使って構成

近接遭遇

連星に限らず、2 つの星の距離が極端に近づくと計算が破綻する (相互作用ポテンシャルが発散するので数値誤差も発散する)。なんらかの対処が必要。

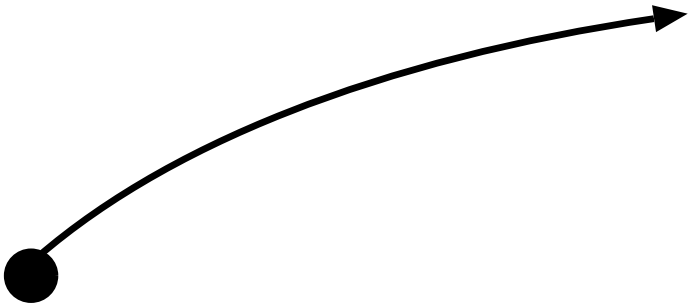
原理的には “straightforward”: 2 体が十分近づいたら、解析解 (またはそれ+摂動) でおきかえればいい。



具体的には

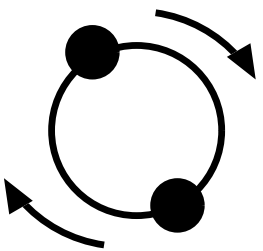
- 2 つの粒子が近付いたかどうかをチェックする
- 近付いていたら、その 2 粒子の重心および相対位置（内部運動）を新しい変数として時間積分する
- 相対運動に対して、回りの粒子からの摂動が十分に小さければ数値積分しないで解析的に解く
- 2 つの粒子が十分離れたらその重心と相対位置から元の粒子を復元する

連星



普通の星:他の星全体が作るポテンシャルの中を運動

連星: 2 つが重力的に結合
できかた:



- 生まれる時に連星 (結構多い)
- 3 体相互作用

連星と計算量

連星は軌道周期が短い
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 $=$ 軌道長半径 $\sim$ 系の大きさ$/N$
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

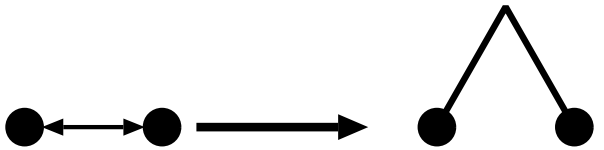
2体以上の系の扱い

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？

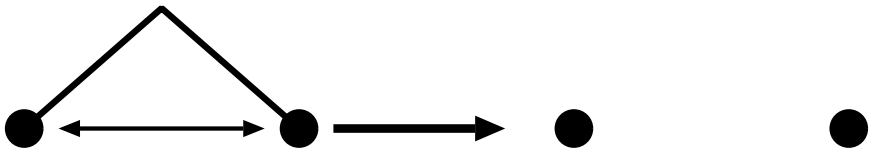
- 「3体」として認識、それ用の正則化
- 2体+1体としては認識、再帰的に扱う

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

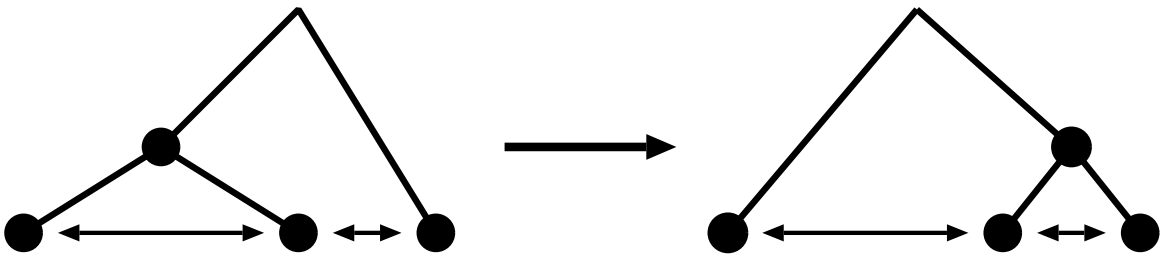
構造の組み換え



$r < r_{close}$



$r > \gamma r_{close}$



アニメーション

ちょっと息抜き

大規模構造
アニメーション 1　アニメーション 2

銀河円盤
アニメーション a1
アニメーション a2
アニメーション b1
Star formation with SPH
銀河円盤 (ガスあり 1)
銀河円盤 (ガスあり 2)

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。

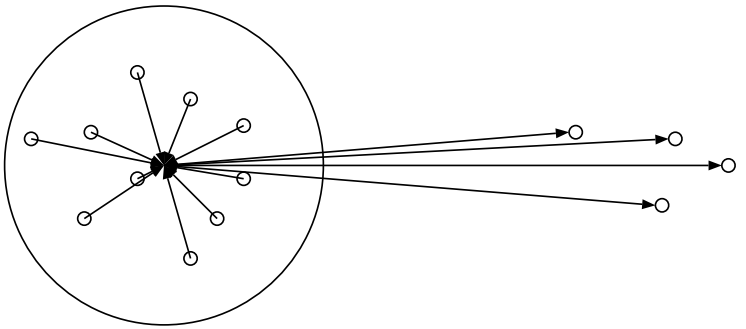
広く使われている方法： Barnes-Hut treecode
有名な方法： 高速多重極法

ツリー法、FMMの基本的発想

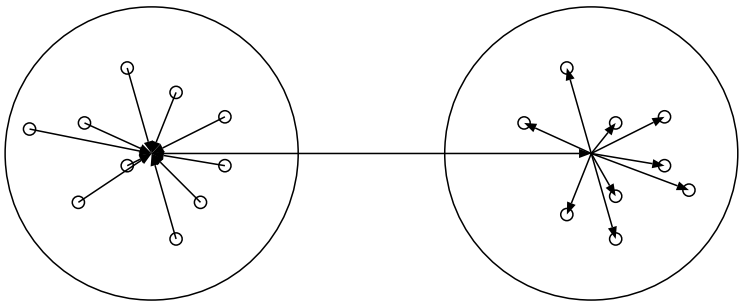
遠くの粒子からの力は弱い

↓

まとめて計算できないか？



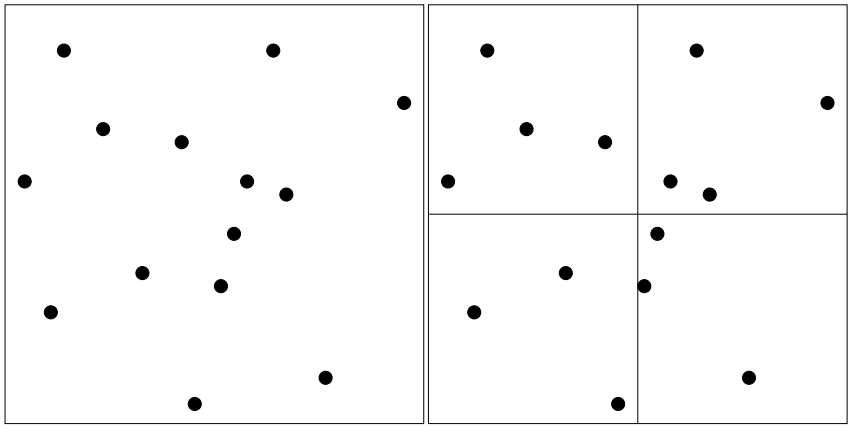
Tree



FMM

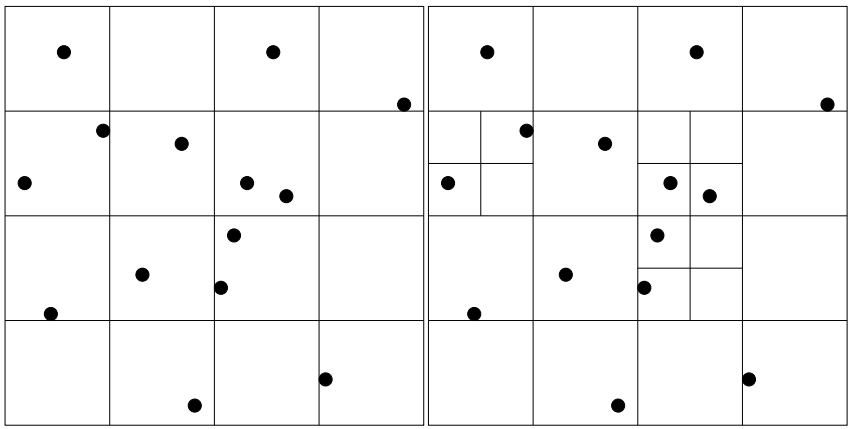
- ツリー：力を及ぼすほうだけをまとめて評価
- FMM：力を受けるほうもまとめて評価

どうやってまとめるか？ — ツリー法の場合



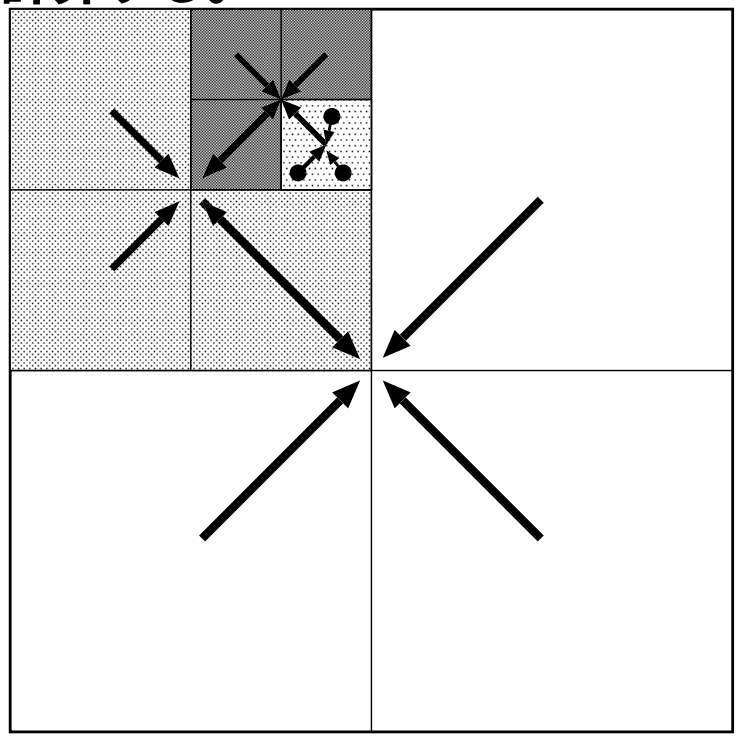
階層的なツリー構造を使う。

- 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。

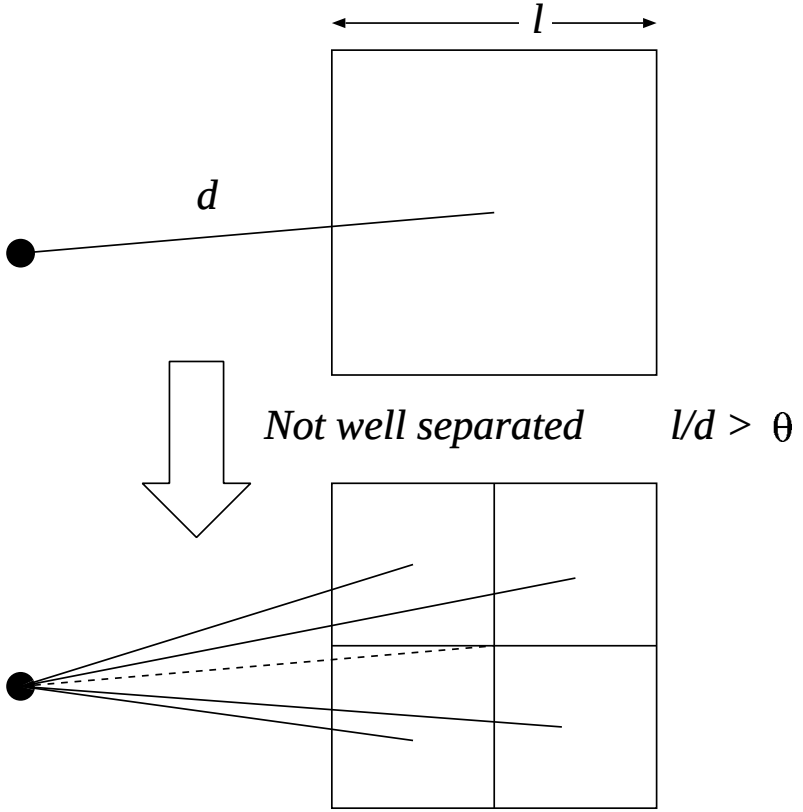


- 最下層のセル:そのなかの粒子が作るポテンシャルを多重極展開
- それ以外：子セルの多重極展開の展開中心をシフトして加算

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

ツリー法の効果

- 計算量のオーダーが $O(N^2)$ から $O(N \log N)$ に減る。
- 現状では、例えば天文台の $\mathrm{Cray}\ 1024$ コアを使って 2048^3 粒子が 1 ステップ数分とか。
- もしも直接計算したら 1 ステップ数十年かかる。

独立時間刻みとツリー法の組合せ

- 原理的にはこれが望ましいに決まっている
- 研究も昔からある。McMillan and Aarseth 1993 とか
- (牧野は 1987 年あたりに色々やったけど論文書いてない)
- あまり上手くいっていない

問題点:

- 現在の殆どの実装は、一番短いステップ毎にツリーを作り直す。そうすると、ツリーを作る時間が全部になって速度があがらない。
- 部分的にツリーを作り直すとかも試みられているが、特に並列化と組み合せるとコードが複雑になりすぎて手に負えない。

並列化向けのアプローチ

例えばこんなのを扱いたい
銀河中心近くでの高密度星団の進化

- 星団を銀河に埋め込む
- 星団内は精度が欲しい
- 銀河はまあ適当でもよい

ここ数年で色々ごまかす方法を考えた。

- BRIDGE scheme
- もうひとつ別 (押野、in preparation)

BRIDGE

MVS (混合変数シンプレクティック) に類似

単純な陽的シンプレクティック: ハミルトニアンを運動エネルギーとポテンシャルに分解、交互に積分
MVS: 惑星の運動を、太陽の回りのケプラー運動とそれ以外の相互作用に分解。

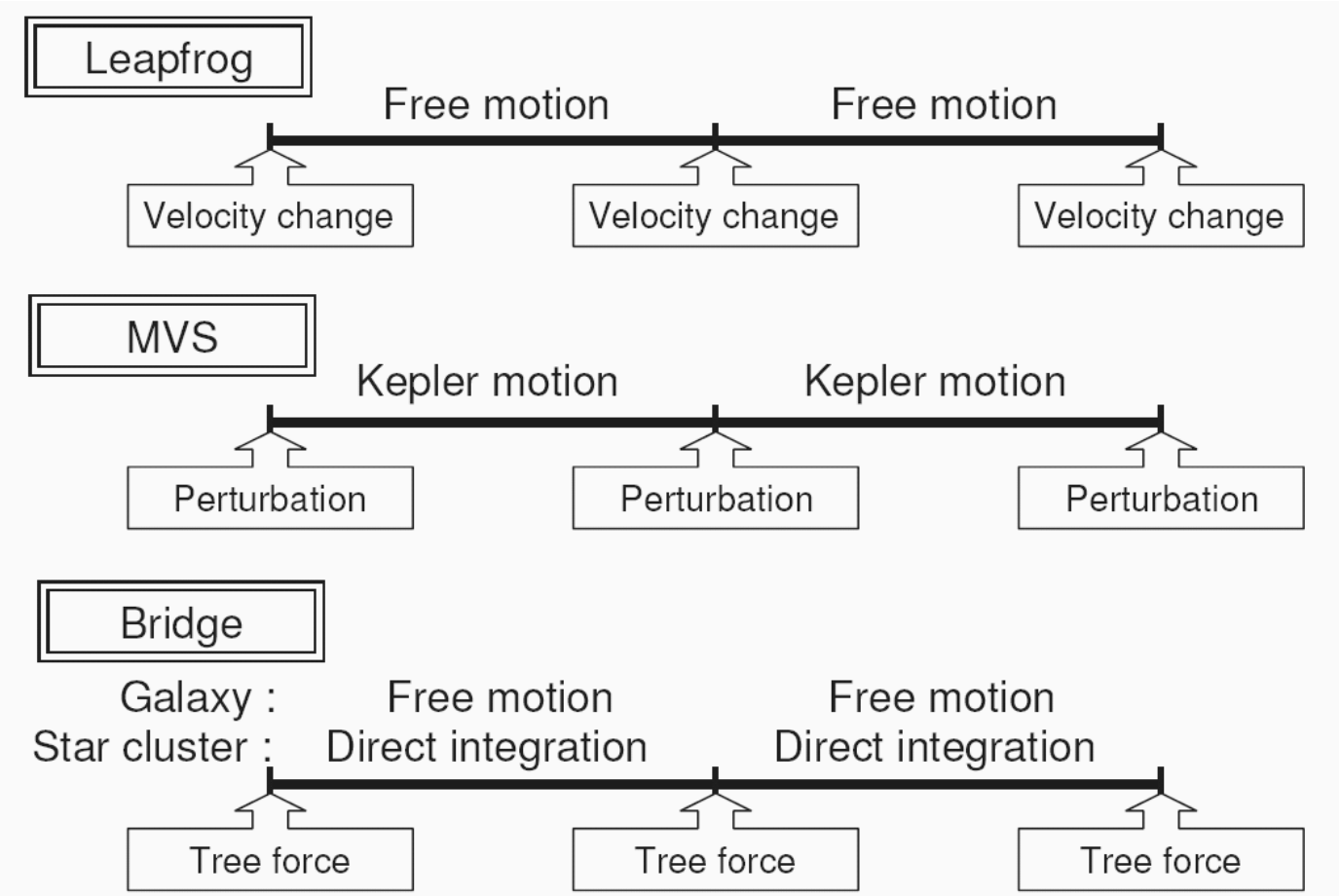
我々の方法:ハミルトニアンを

- 運動エネルギーと星団内ポテンシャル
- それ以外のポテンシャル

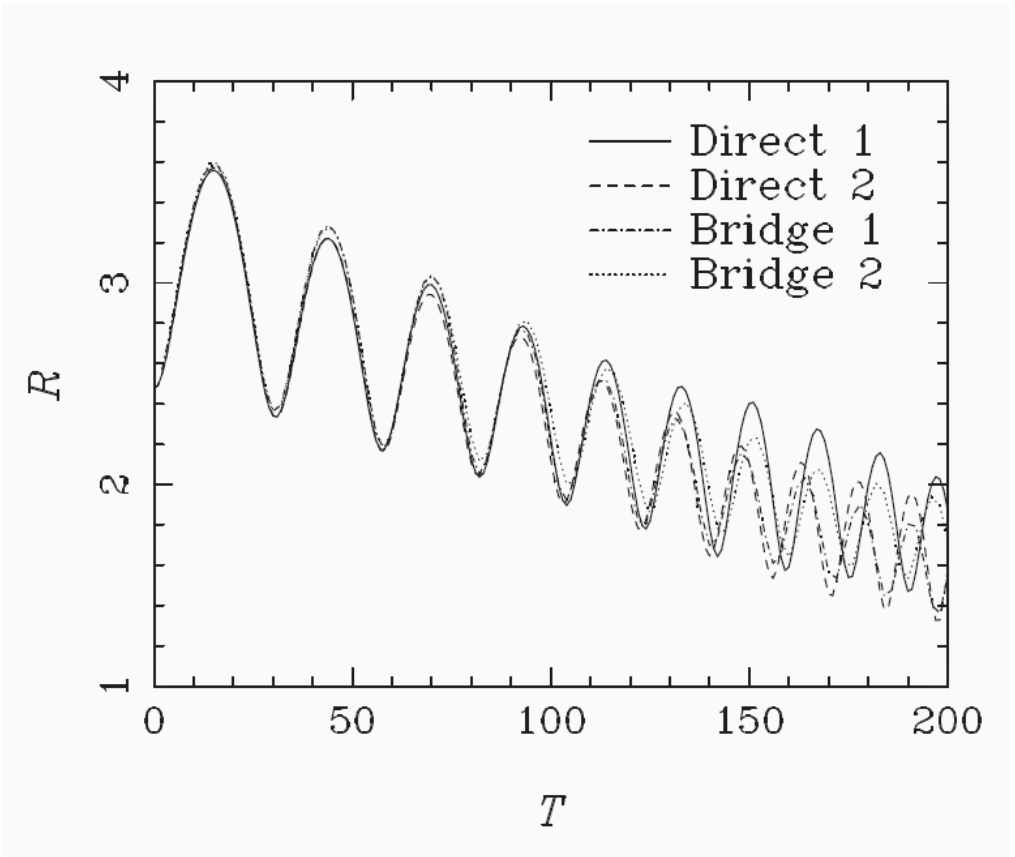
に分解、前者を (シンプレクティックでない) 独立時間刻みで高精度に積分

BRIDGE (Bridge is for Realistic Interactions in Dense Galactic Environment)

How does it work?



Test result



- $N = 100\text{k} + 2\text{k}$
- Similar model as in Fujii et al. 2996
- Two runs: different random seeds
- Results agree well.
- Energy error: dominated by the parent galaxy.

もうひとつの方法

BRIDGE は銀河+星団 1 つだと素晴らしく上手くいくが、限界もある

- 星団複数だと計算大変
- 実際に銀河の中で星団が生まれてくるような現象には使いにくい

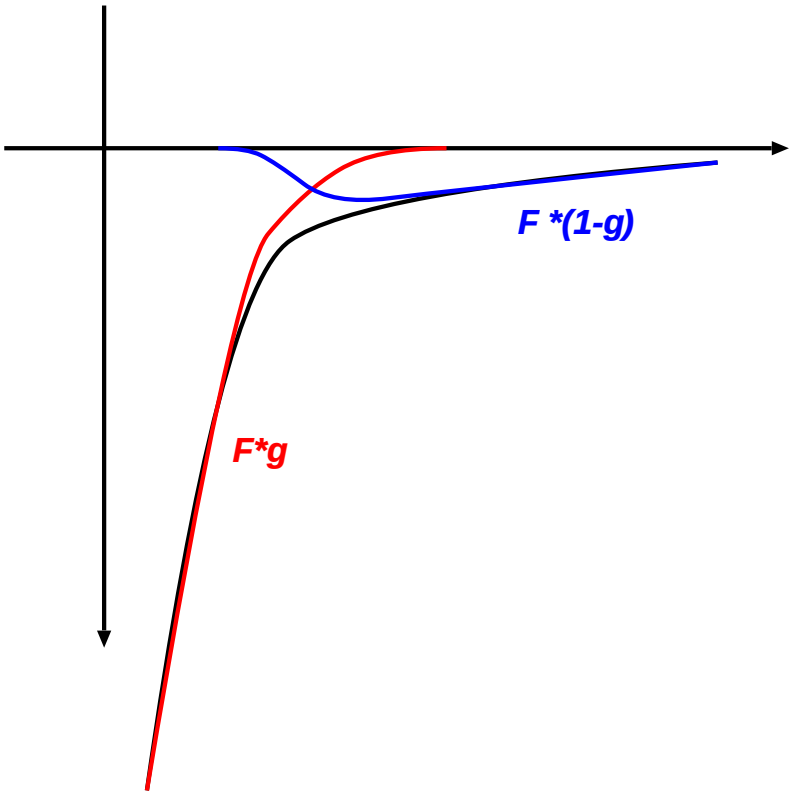
というわけで、

- 粒子の種類によってではなく、距離でわけたらどうか？

具体的には

2 粒子間の重力を形式的に 2 つのタームに分ける。

$$F_{ij} = -Gm_i m_j \frac{r_{ij}}{|r_{ij}|^3} = F_{ij}(1 - g(|r_{ij}|) + F_{ij}g(|r_{ij}|)$$



こんな感じに分解

- $F * g +$ 運動エネルギー を独立時間刻みで高精度に積分
- $F * (1 - g)$ はツリー+リープフロッグで積分 (こっちはシンプレクティック)
- g はコンパクトサポートで、積分公式に必要な回数だけ微分できる必要あり (スプラインを使う)

開発状況

- とりあえず、惑星形成の計算向けに実装、テスト中
- 上手く動いているような気がする
- MVS と同じく、太陽重力は高精度側に入れる
- 並列化と相性がよい。

GRAPE と GRAPE-DR

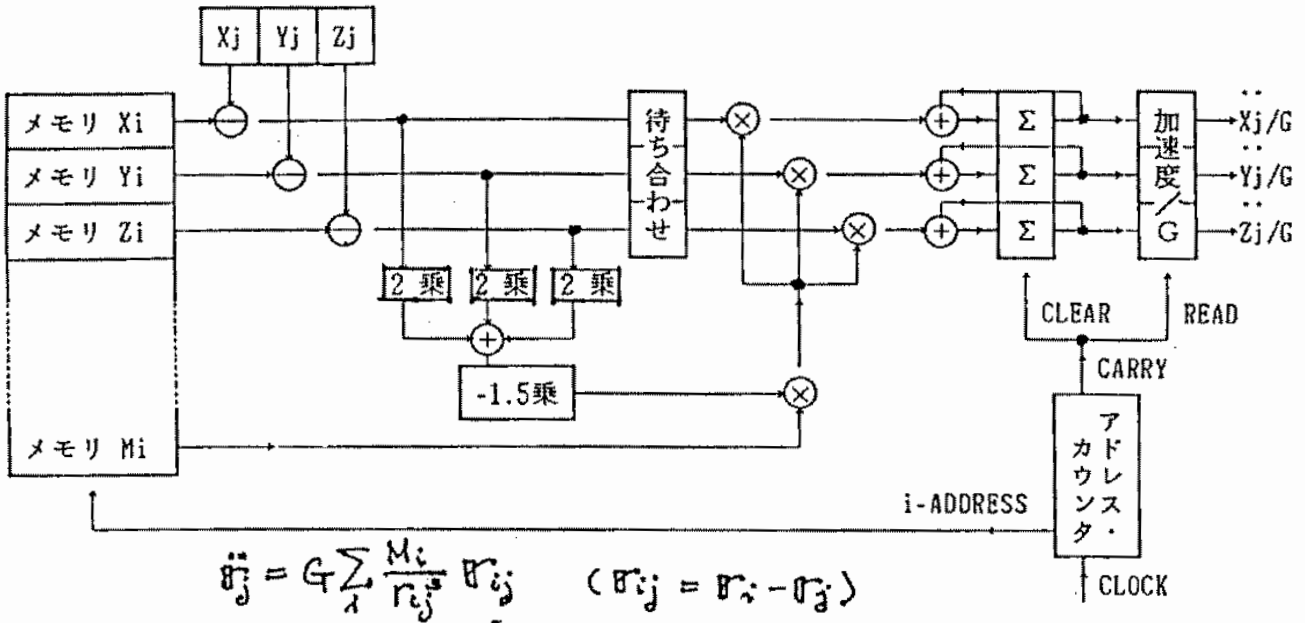
- GRAPE の考え方
- GRAPE-DR
- その次

GRAPE の考え方

- 重力多体問題: 粒子間相互作用の計算が計算量のほとんど全部
- 効率のよい計算法 (Barnes-Hut tree, FMM, Particle-Mesh Ewald(PPPM), ...): 粒子間相互作用の計算を速くするだけでかなり加速できる
- そこだけ速くする電子回路を作る (「計算機」というようなものではない)

近田提案

1988 年、天文・天体物理夏の学校

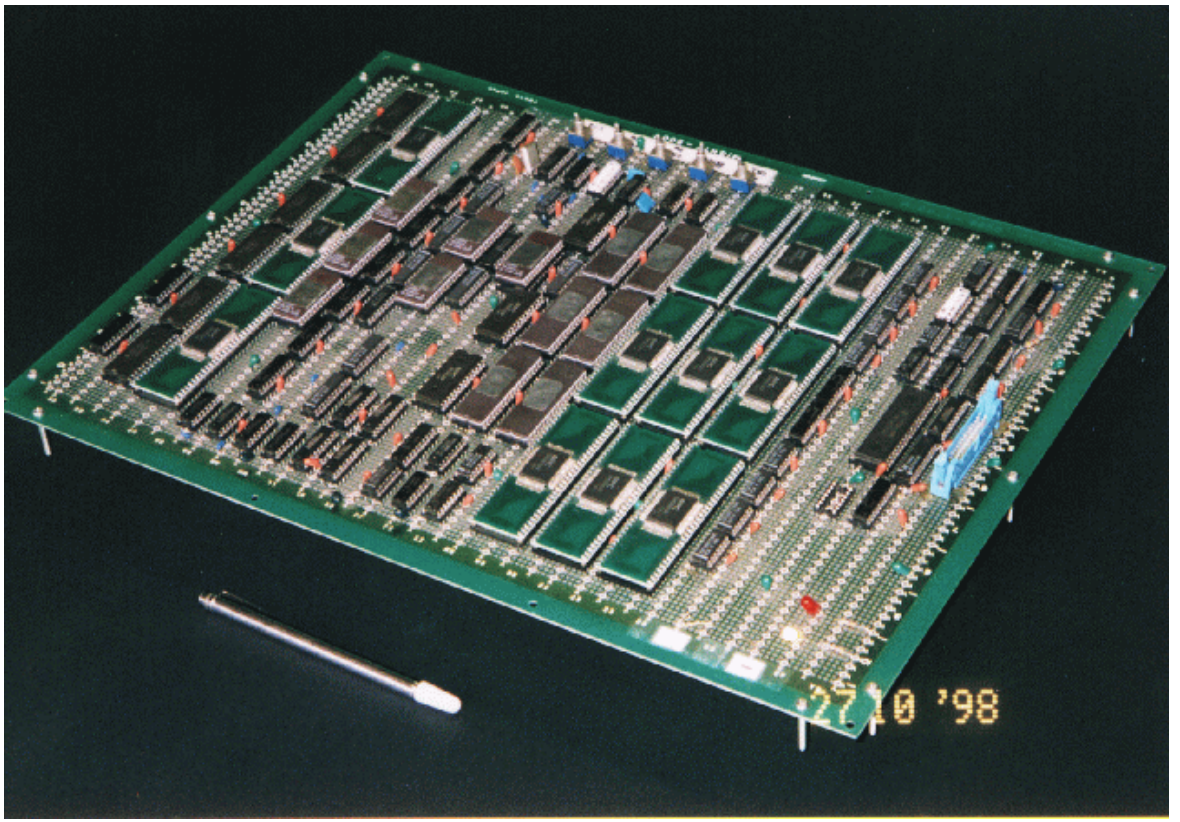


$$\ddot{\vec{r}}_j = G\sum_i \frac{M_i}{r_{ij}^3}\vec{r}_{ij} \qquad (\vec{r}_{ij} = \vec{r}_i - \vec{r}_j)$$

+，－，×，２乗は１operation，　-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．

GRAPE-1(1989)



演算毎に語長指定。固定 16–対数 8–固定 32–固定 48
240Mflops 相当

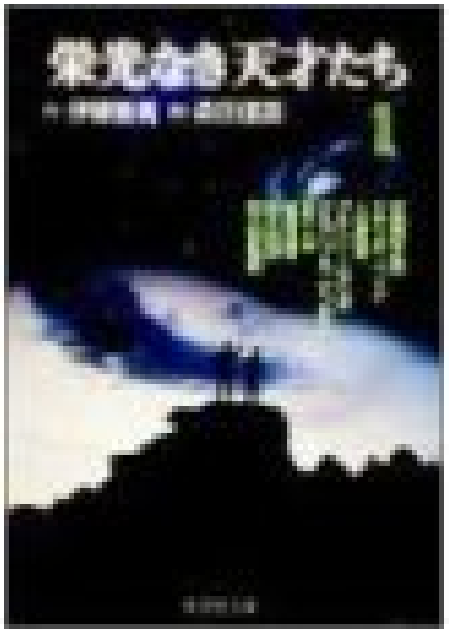
開発はどんなふうだったか

ハードウェアは私が知らないうちにできていた (伊藤君が作った) ので良く知らない。

ホストの GPIB インターフェースの性能がでない (特に Sony NEWS を使った時に) のでなかなか大変だった。

最終的にはユーザープロセスから GPIB 制御チップをいじり回すプログラムを書いた。

伊藤君

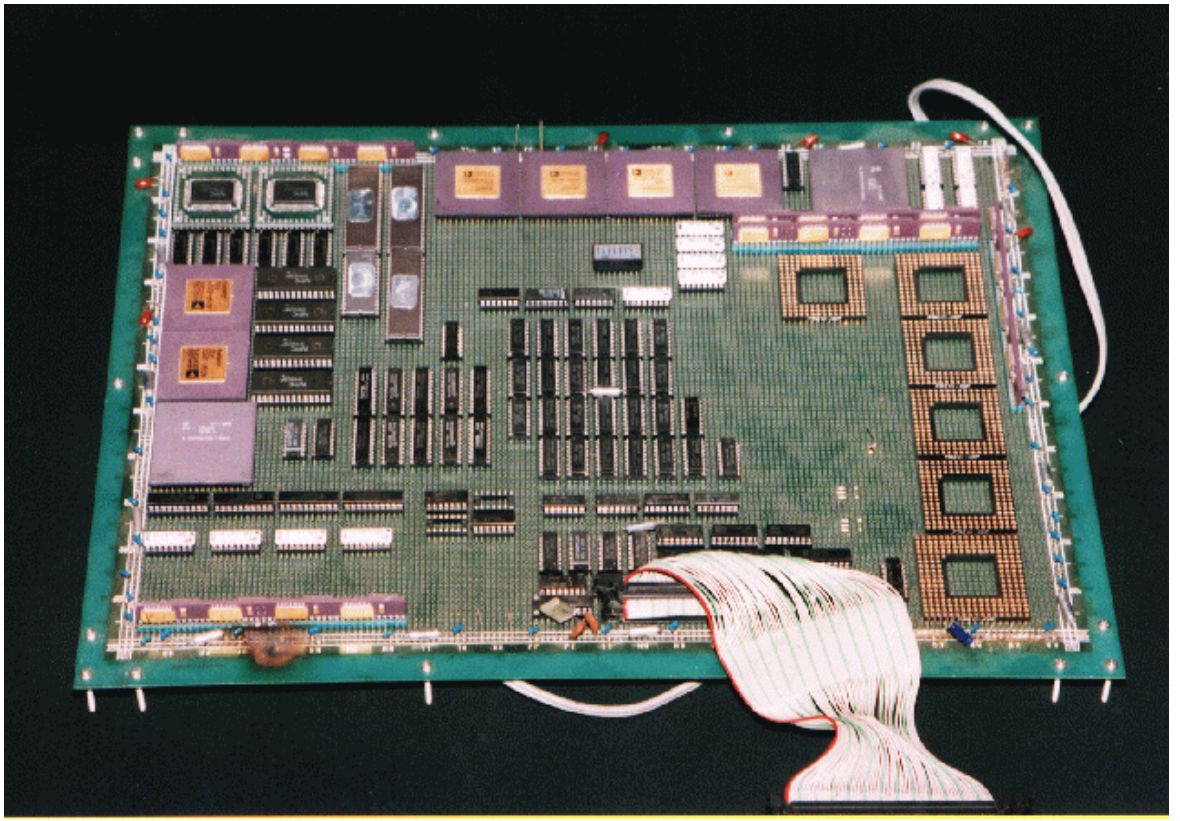


「栄光なき天才たち」の原作者としてのほうが有名という話も。

学部の頃から GRAPE-2 の開発の後くらいまで原作者をしていた。

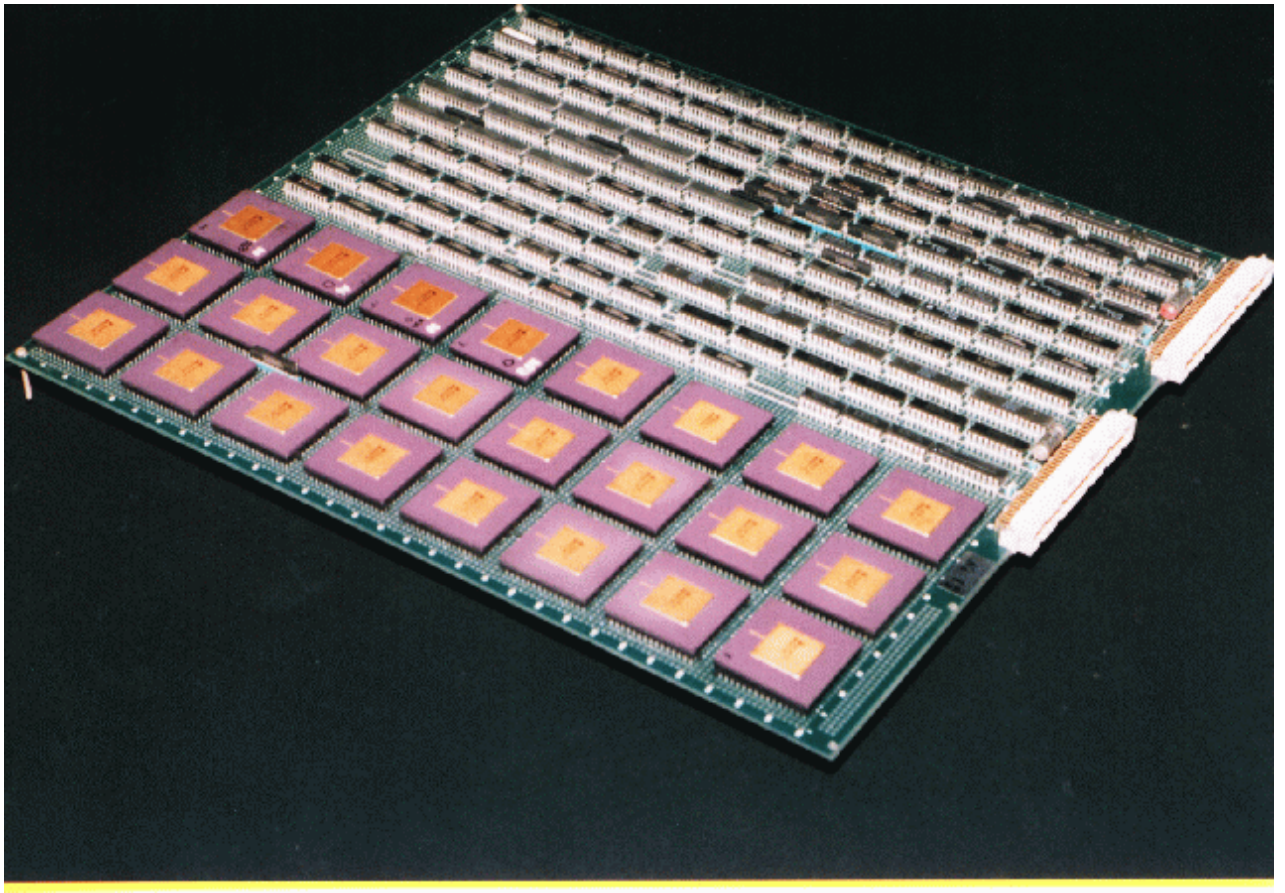
現在千葉大教授

GRAPE-2(1990)



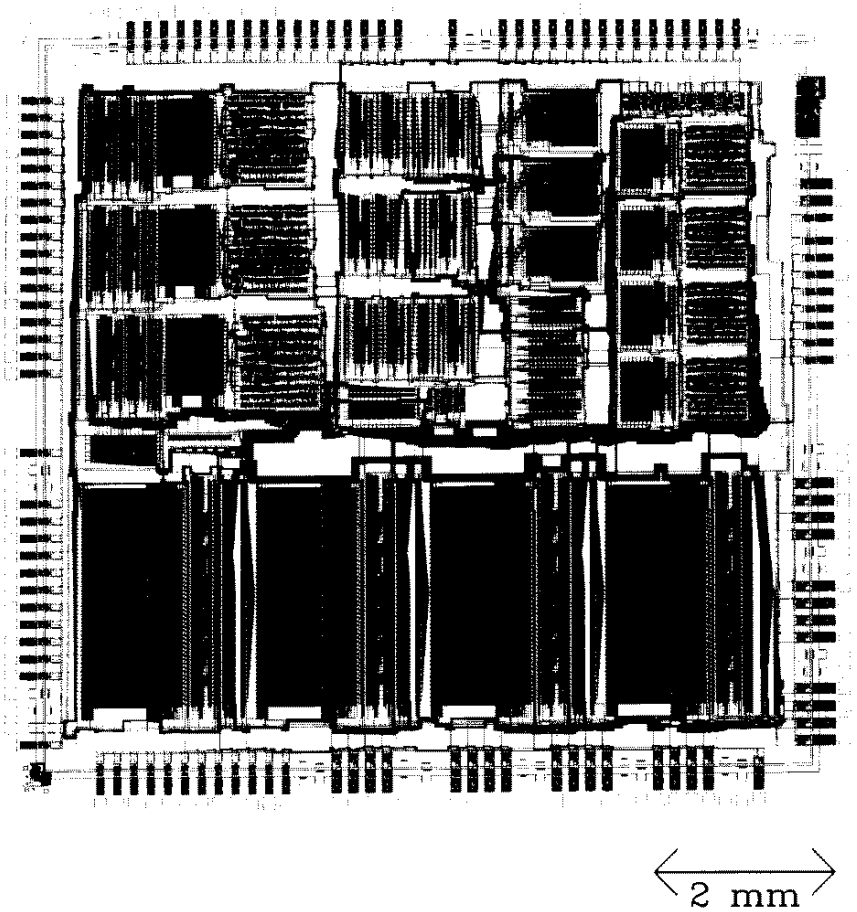
8ビット演算とかは止めて普通に浮動小数点演算 (倍精度は最初と最後だけ)
40Mflops

GRAPE-3(1991)



カスタムチップ 24 個 1 ボード、 10MHz 動作、 7.2Gflops

GRAPE-3 チップ



1μm プロセス

11万トランジスタ

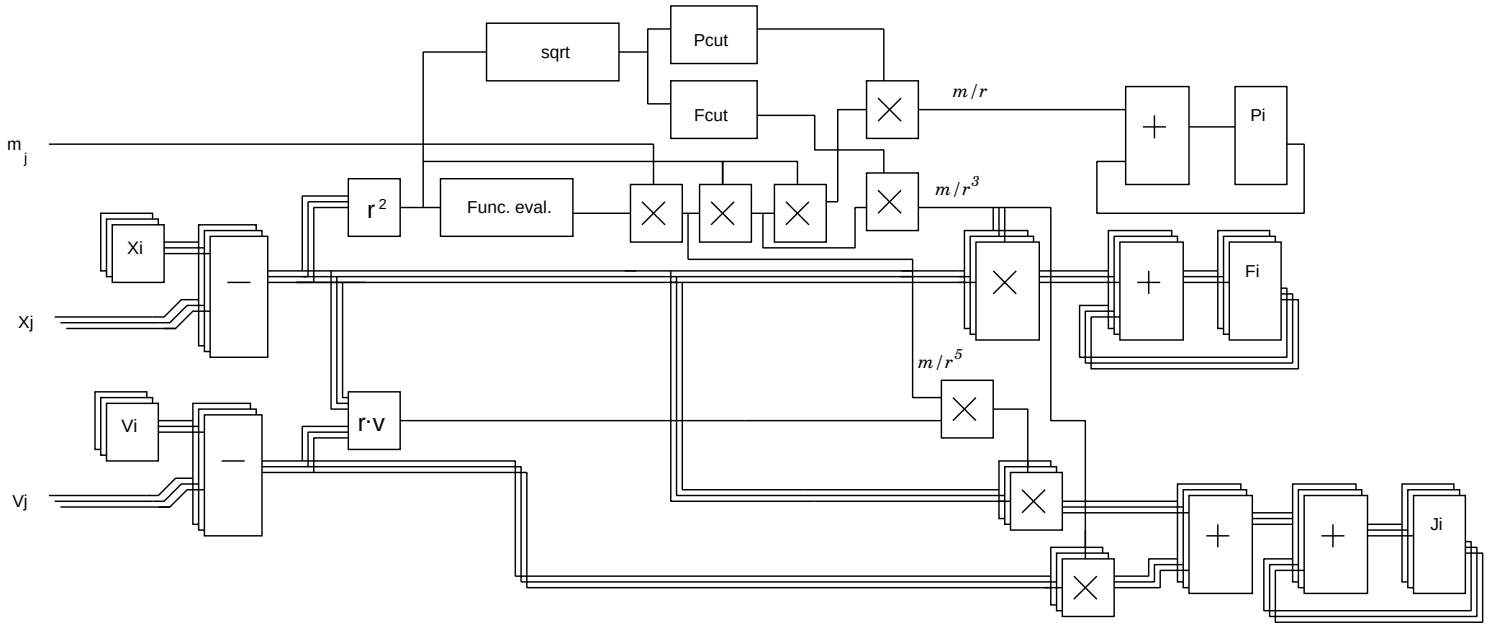
20 MHz クロック動作

600 Mflops 相当

GRAPE-4(1995)

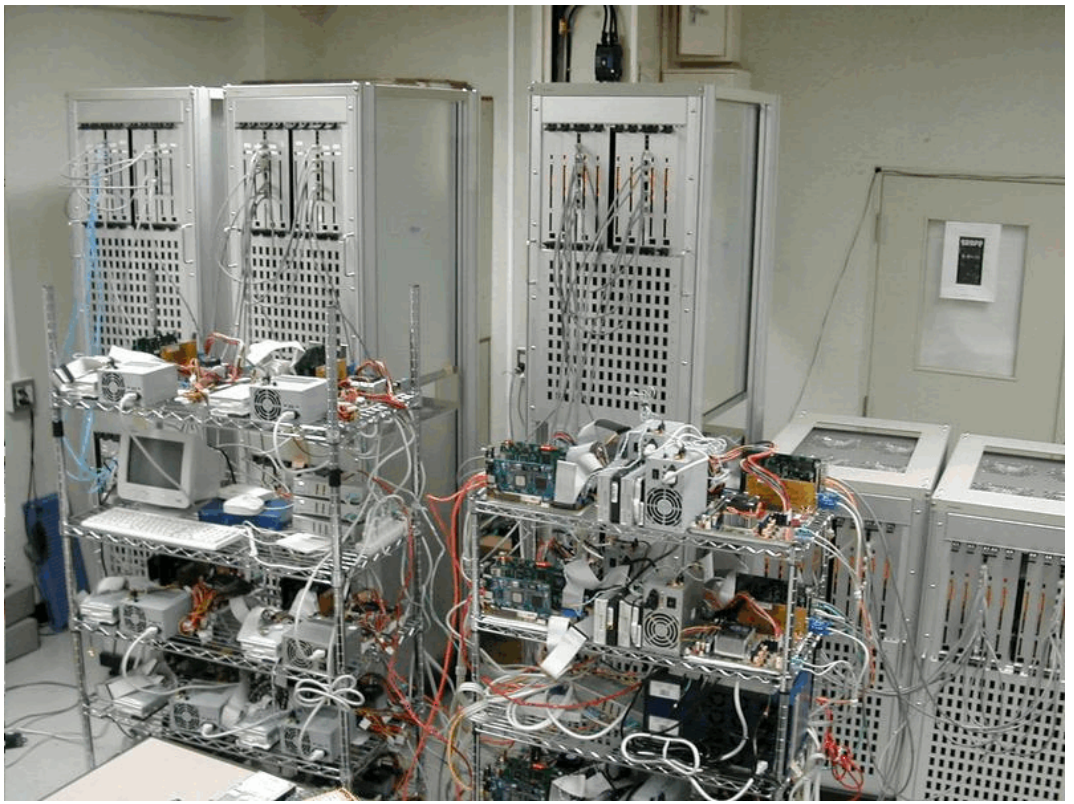
トータル 1792 チップ、 1.1 Tflops

GRAPE-4 パイプライン



1μm プロセス、 10万ゲート (40万トランジスタ)、640Mflops
泰地君設計

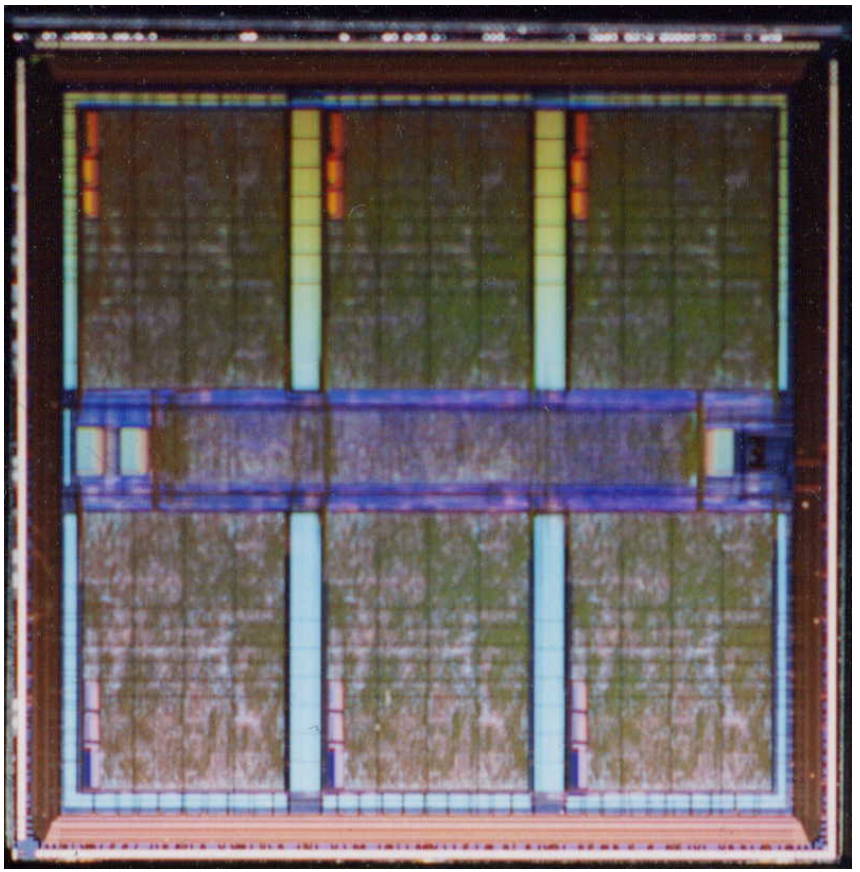
GRAPE-6(2002)



2002 年現在の 64 Tflops システム

4 ブロック
16 ホスト
64 プロセッサボード

パイプライン LSI

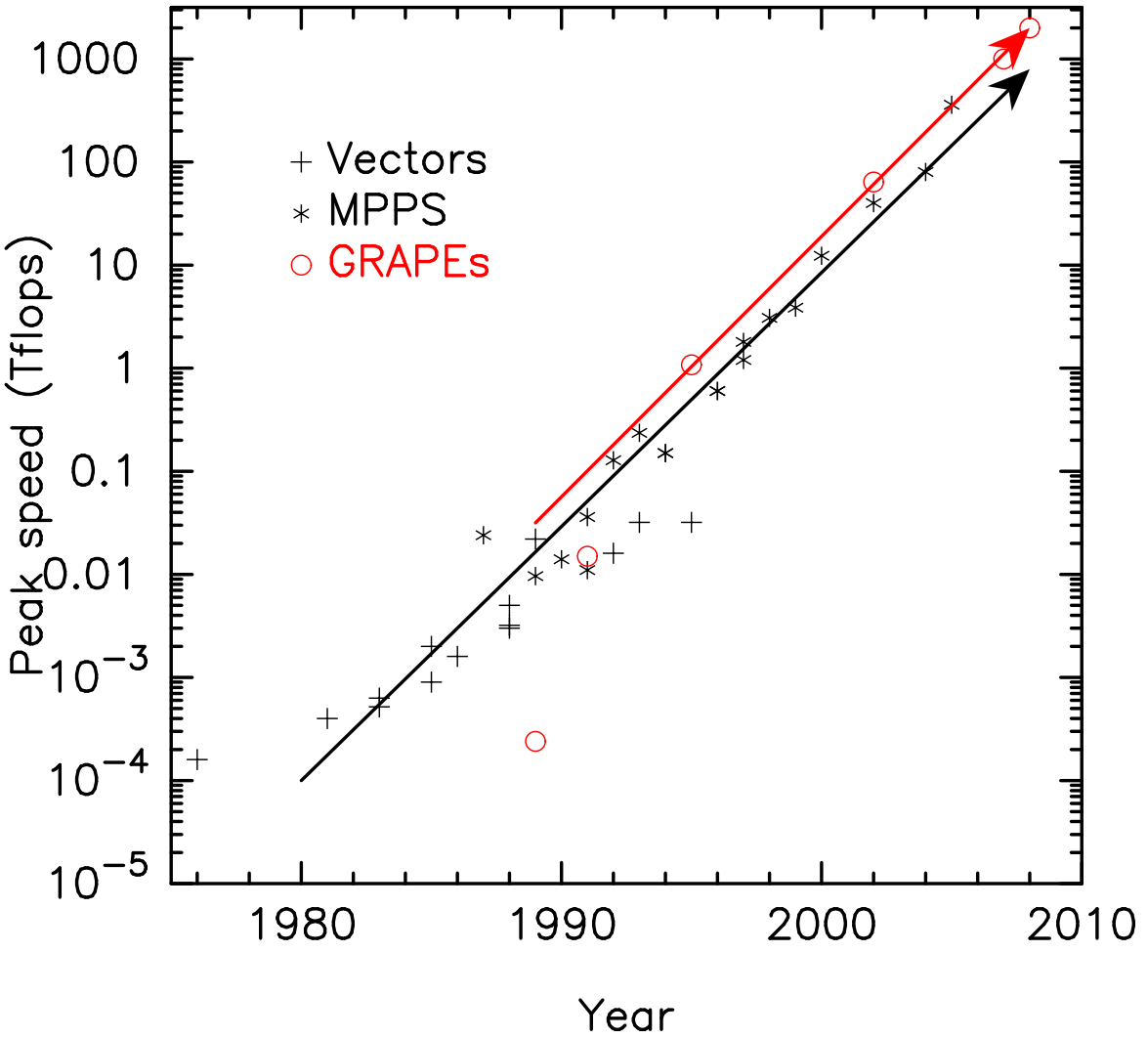


- 0.25 μm ルール
  (東芝 TC-240, 1.8M ゲート)
- 90 MHz 動作
- 6 パイプラインを集積
- チップあたり 31 Gflops

現在のマイクロプロセッサと比べてみる

	GRAPE-6	Intel Xeon 5590	IBM BG/P
Year	1999	2009	2007
Design rule	250nm	45nm	90nm
Clock	90MHz	3.3GHz	850MHz
Peak speed	32.4GF	53GF	13.6GF
Power	10W	130 W	15W?
Perf/W	**3.24GF/W**	0.4 GF/W	0.91GF/W

ピーク性能の進歩



GRAPE-4 以降、
完成した時点で世界
最高速を実現

商業版 GRAPE

- GRAPE-3 から商業版
- GRAPE-6 を購入した機関

American Museum of Natural History
Drexel University
Indiana University
Rochester Institute of Technology
Rutgers University
University of Michigan
University of California
McMaster University
The University of Cambridge
University of Edinburgh
Observatoire Astronomique Marseille-Provence(OAMP)
Astronomisches Rechen-Institute (ARI)
Ludwing-Maximillans University
Max-Planck-Institute fur Astronomic

GRAPE-6 を購入した機関(つづき)

University of Bonn
University of Mannheim
Holland University of Amsterdam
Nanjing Univesity
Citec Co., Ltd
Gunma Astronomical Observatory
Hokkai-Gakuen University
Kansai University
Kyoto University
National Institute for Fusion Science (NIFS)
National Astronomical Observatory of Japan
Osaka University
The University of Tokyo
Tokyo Institute of Technology
University of Tsukuba

約 30 機関、 60Tflops。MDGRAPE-2 も同様

GRAPE-6 の次は？

MDGRAPE-3 の次: MDGRAPE-4 (次世代の迷走のせいでどっかにいった？)

そもそも MDGRAPE-3 にあたるものは？ → GRAPE-DR

GRAPE-DR 計画とは何か？

「基本的には」次期 GRAPE 計画

- 2004 年度から 5 年計画
- 目標ピーク性能: 2 Petaflops
- チップ数 4096
- 単体チップ性能 0.5Tflops

と、これだけなら今までの GRAPE が速くなっただけ。

実際のアーキテクチャ: 今までの GRAPE とは全然違う

- なぜ違うか
- それで何ができるか

「次期 GRAPE」の実際的な問題

天文だけ (しかも理論だけ (しかも N 体だけ)) の機械としてはチップ開発コストが大き過ぎる

チップ開発費

1990	1μm	1500 万円
1997	0.25μm	1 億円
2004	90nm	3 億円以上
2009	45nm	10 億円以上

ある程度広い応用を持つものでないと予算獲得が難しい

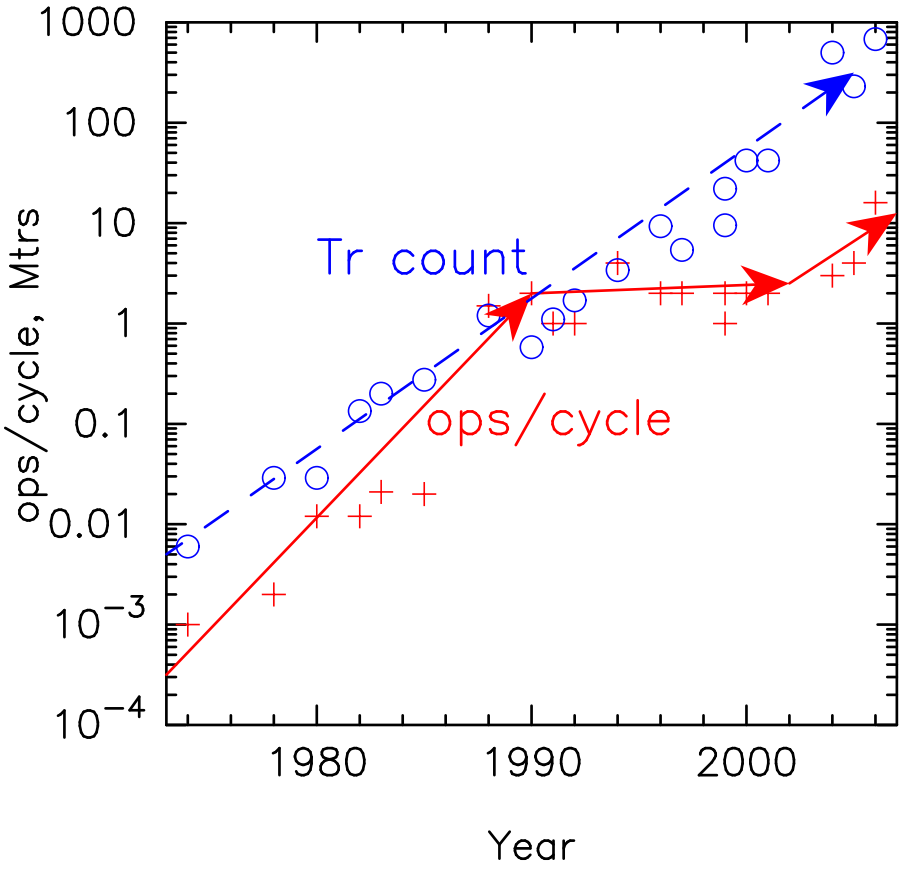
GRAPE-DR の基本的な考え

- 応用に特化し、多数の演算器を 1 チップに集積、並列動作させて高い性能を得た専用計算機の特徴を生かす
- しかし広い応用範囲を実現する

そんなことができるか？が問題

トランジスタは沢山ある

マイクロプロセッサの「進歩」



- **トランジスタ: 15 年で 1000 倍**
- **演算器の数: 同じ期間に 8 倍**
- **100 倍分がどこかに消えた**
- **最も良かった時でもチップ上の演算器の割合は 5% くらい**

多数の演算器を詰め込む方法

境界条件: メモリバンド幅は増やしたくない (システムコストはほぼメモリバンド幅で決まる)
可能な方策

1. GRAPE 的専用パイプラインプロセッサ
2. 再構成可能プロセッサ
3. SIMD 並列プロセッサ

SIMD 並列処理

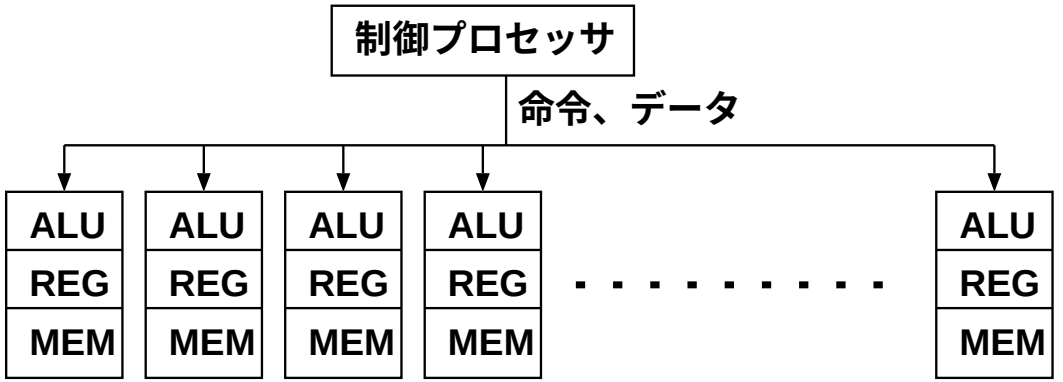
- パイプラインプロセッサをやめにして、「プログラム可能なプロセッサ」を沢山載せる。
- SIMD (Single Instruction Multiple Data): 全プロセッサが同じ命令を実行
- 基本的には、全プロセッサがソフトウェアで GRAPE をエミュレーションする。

SIMD 並列処理って？

- 古典的 SIMD 並列計算機
- SSE、MMX とかの SIMD 拡張命令
- GRAPE-DR における SIMD

古典的SIMD並列計算機

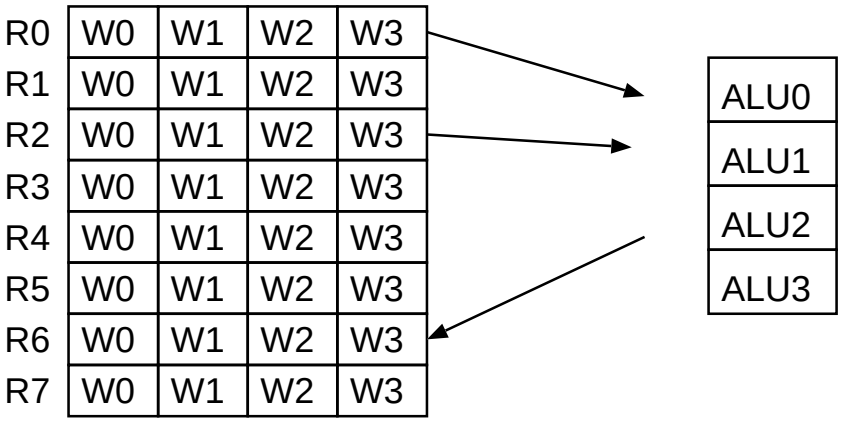
Illiac IV, Goodyear MPP, ICL DAP, TMC CM-2, MASPAR MP-1



- 1960年代に発生、80年代に絶滅
- 半導体技術の向上に対応できないアーキテクチャ: 計算速度とメモリアクセス速度が比例する必要あり。
- メモリ階層をつける: プロセッサが複雑になりすぎて SIMD の意味が無くなる。

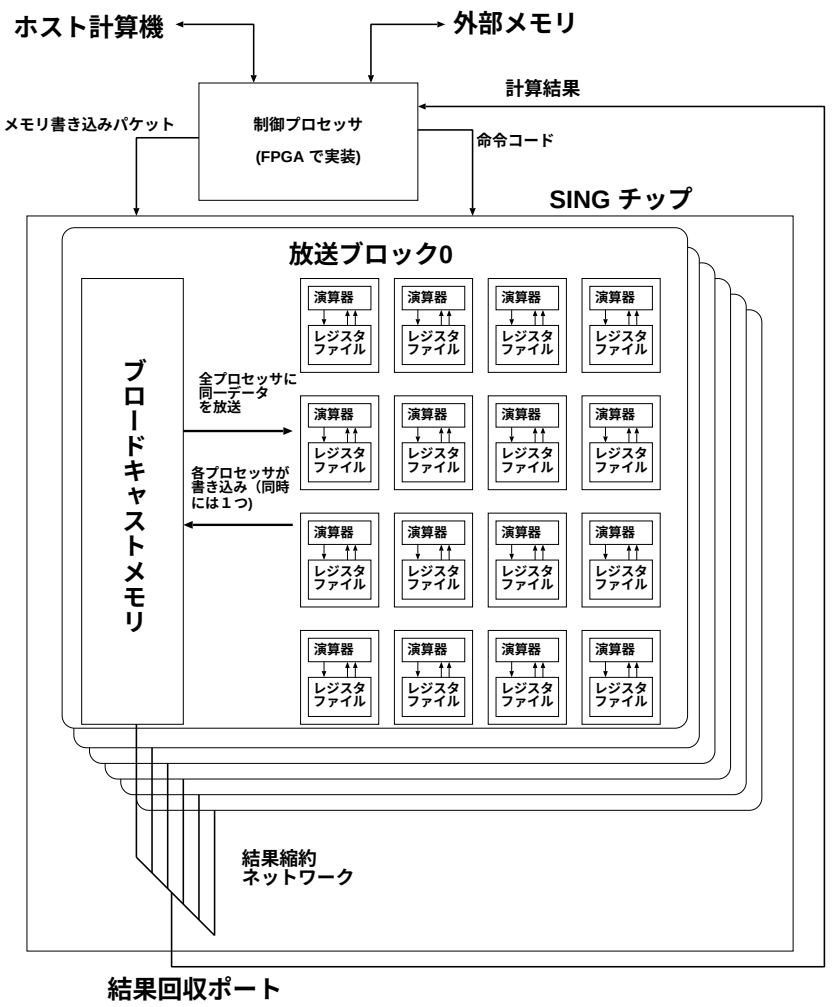
SIMD 拡張命令

SSEx が有名
128 ビットなり 64 ビットのデータを 4 語に区切って、それぞれの要素に対する演算を 4 個の演算器で同時に処理



1 つのプロセッサの中の話: キャッシュとデータをやりとり
並列度 4 程度が限界？それ以上増やすとキャッシュの速度が追いつかなくなる。

GRAPE-DR における SIMD



- 非常に多数のプロセッサエレメント (PE) を 1 チップに集積
- PE = 演算器 + レジスタファイル (メモリをもたない)
- チップ内に小規模な共有メモリ (PE にデータをブロードキャスト)。これを共有する PE をブロードキャストブロック (BB) と呼ぶ。
- 制御プロセッサ、外部メモリへのインターフェースを持つ

基本的な計算モデル

これはハードウェアの制限ではなくて、あくまでも現在のソフトウェアが表現したいものがこう、という話。

計算するもの:

$$g_i = \sum_j f(x_i, x_j)$$

つまり、

- 粒子 i に働く力は他の粒子 j からの力の合計
- j から i に働く力は 粒子 j, i のデータから計算できる
- j は別に全粒子である必要はない (近傍を選択とかはソフトウェアで可能)

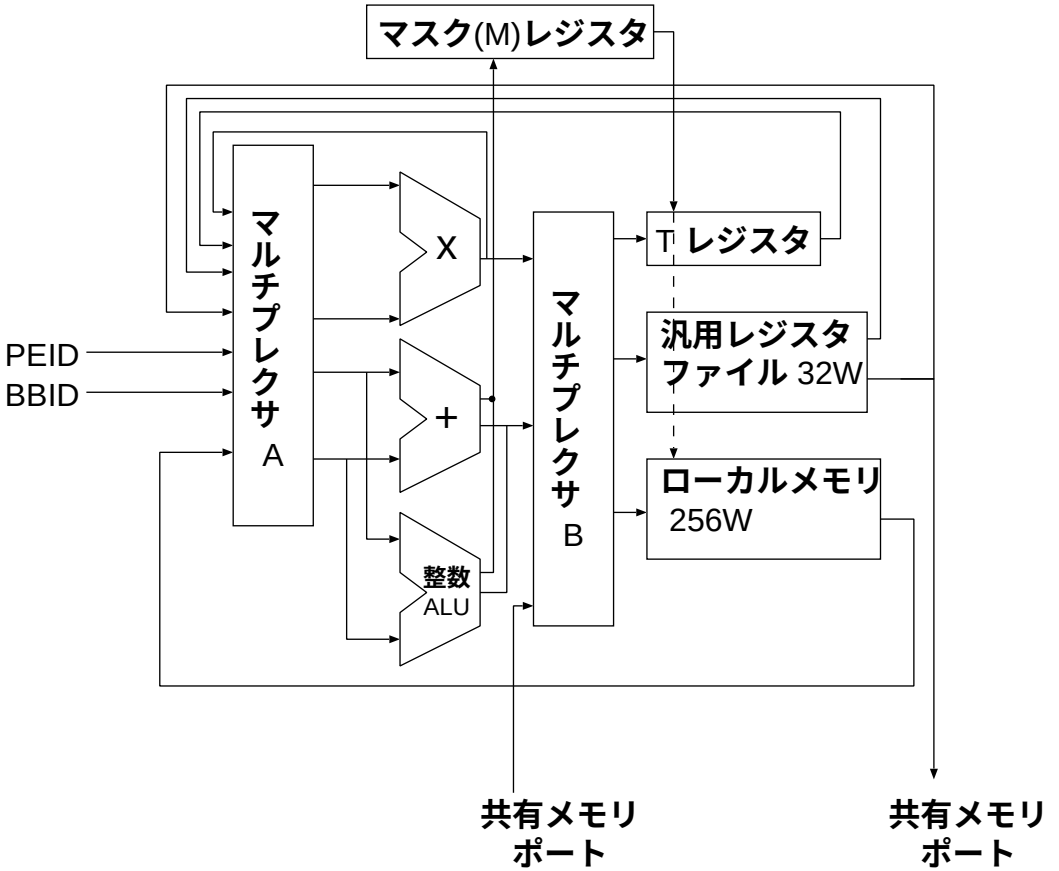
どうやってプログラムを書くか？

GRAPE でやっていたことなら簡単にできるようなシステムを作った。

- 粒子間相互作用を計算するコードをアセンブラまたはコンパイラで記述
- アプリケーションプログラムが呼ぶ関数、構造体定義が自動生成される
- 複数チップ、チップ内複数プロセッサでの並列化は自動的に行われる
- OpenMP みたいな表記からの自動変換もできる

ユーザー・アプリケーションプログラマは、演算カーネルの効率以外は気にすることはない。複数 PE への割り当て、ループブロッキング等はハードウェアが勝手にやる。

PE の構造

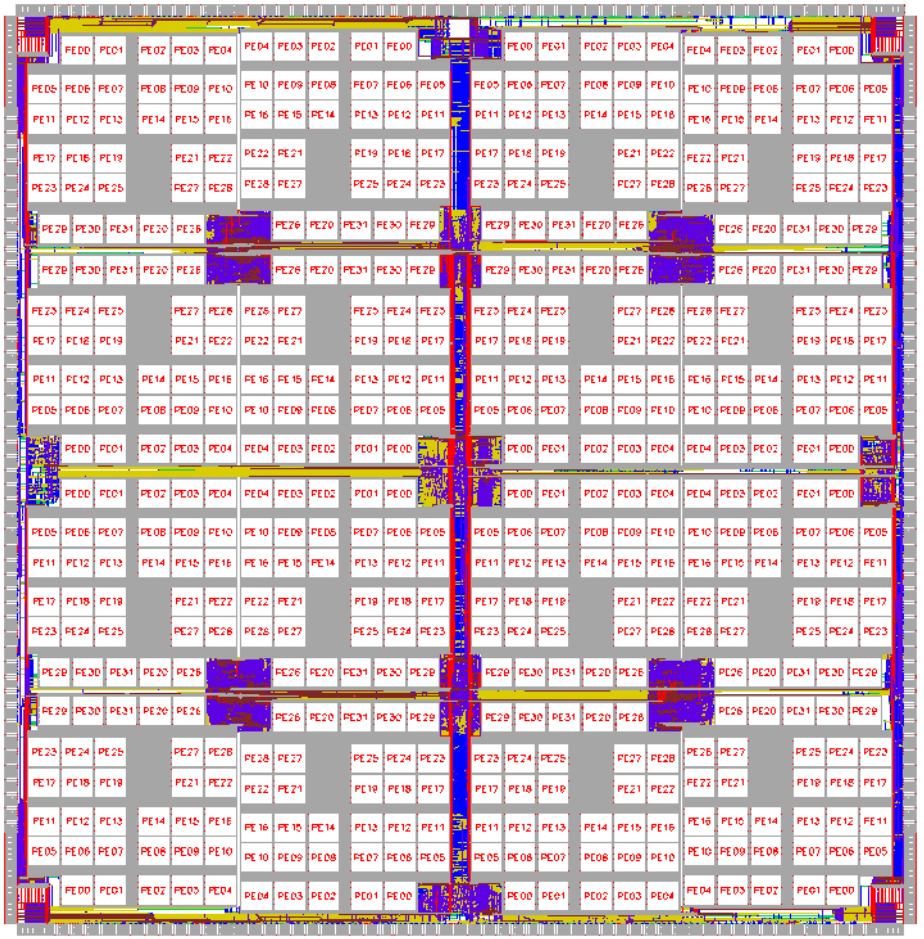


- 浮動小数点演算器
- 整数演算器
- レジスタ
- メモリ 256 語
  (K とか M ではない。)

プロセッサチップとプロトタイプボード

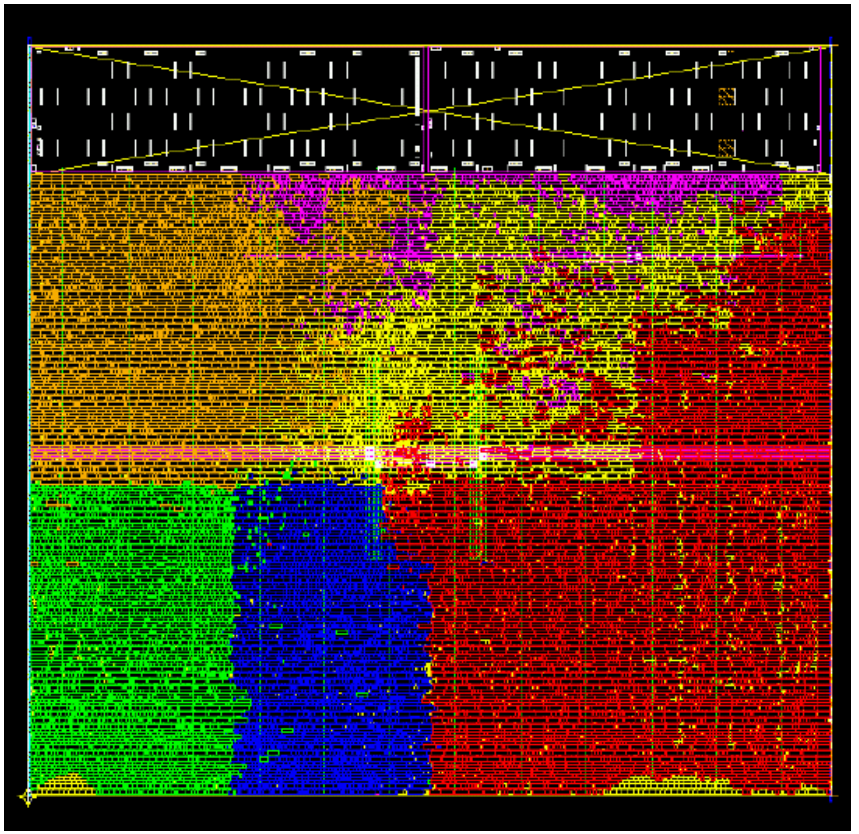
チップ写真

レイアウト



- 32PE(要素プロセッサ)のブロックが 16 個
- 空いているところは共有メモリ
- チップサイズは 18mm 角

PE レイアウト



0.7mm by 0.7mm

Black: Local Memory

Red: Reg. File

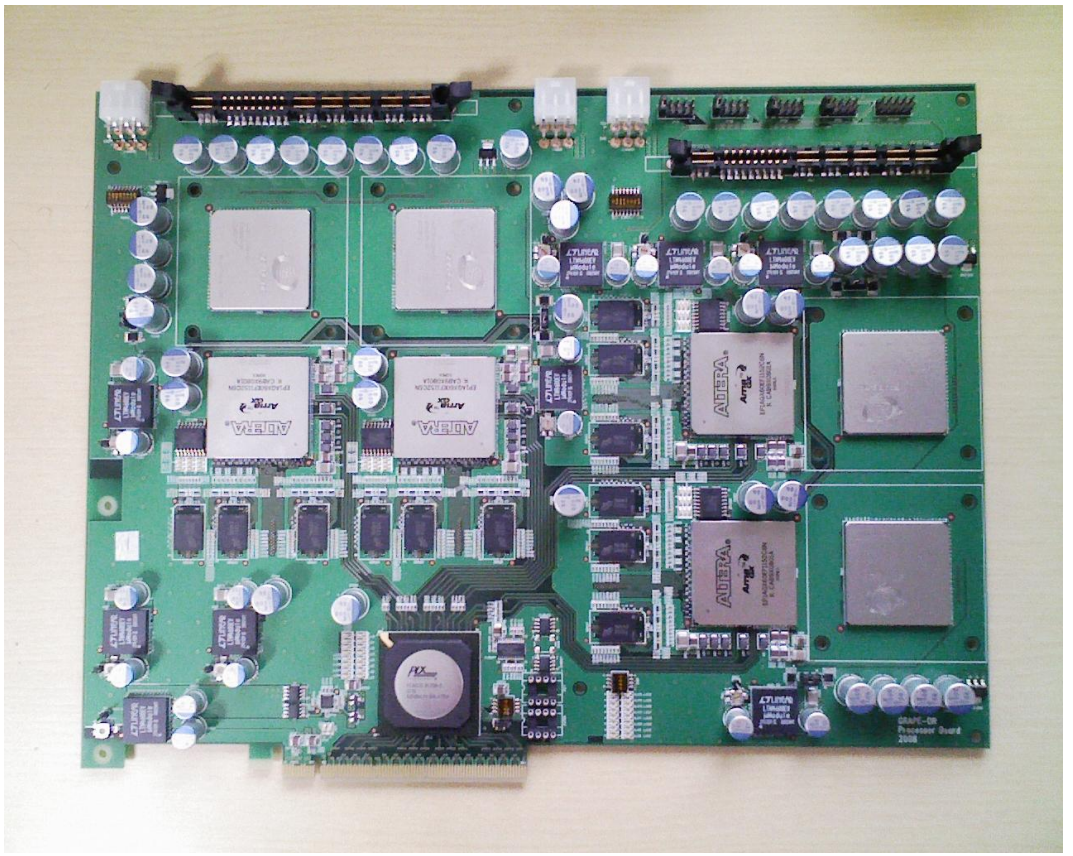
Orange: FMUL

Green: FADD

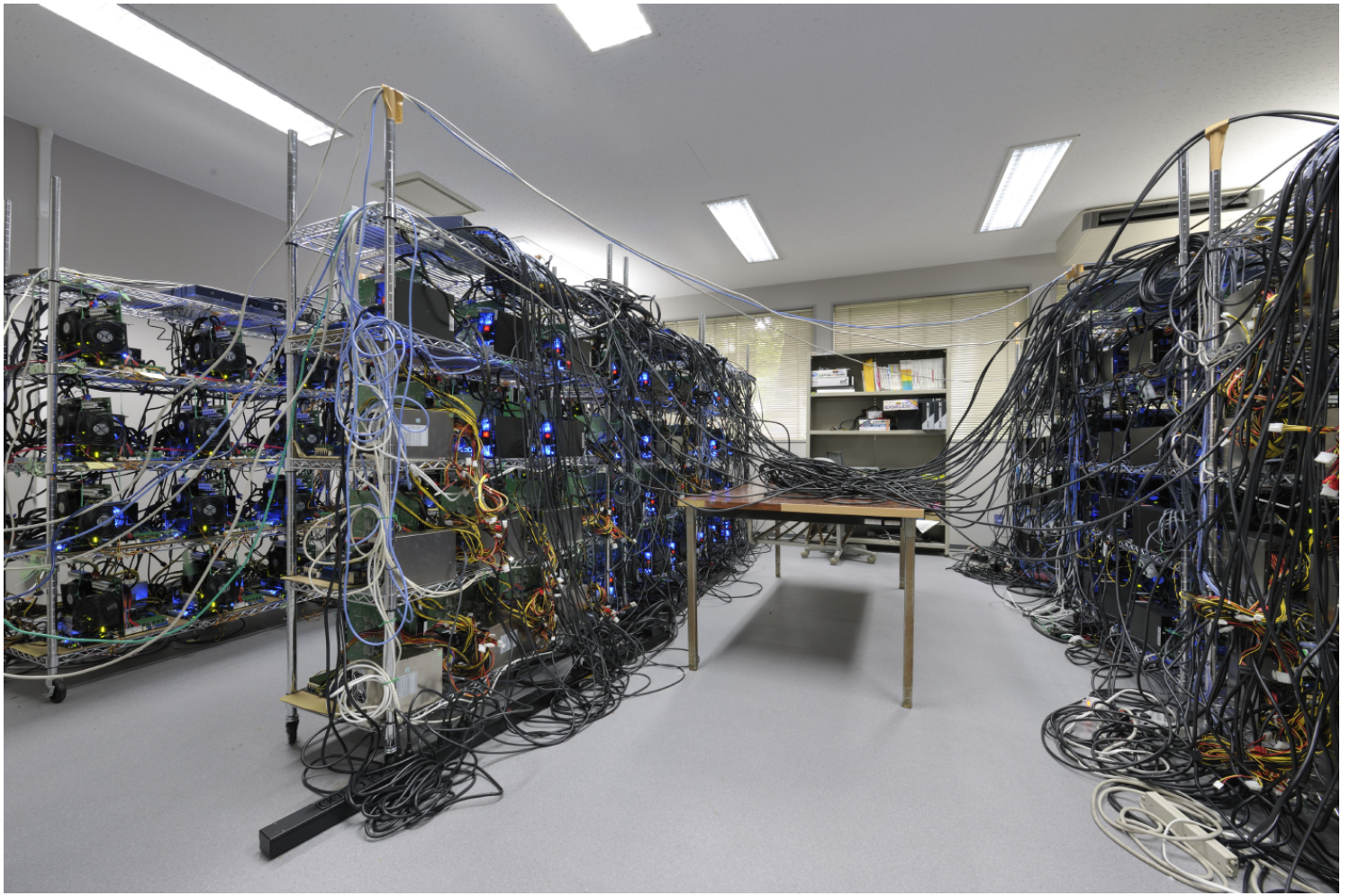
Blue: IALU

量産型ボード

- PCI-Express カード (16 レーン、通信速度実力 2GB/s)
- 4 GRAPE-DR チップ
- 実力クロック 400MHz、、、 1.6TF(SP)/0.8TF(DP)



GRAPE-DR cluster system



GRAPE-DR cluster system

- 128-node, 128-card system (105TF theoretical peak @ 400MHz)
- Linpack measured: 24 Tflops@400MHz (still lots of tunings necessary....)
- Gravity code: 340Gflops/chip, working
- Host computer: Intel Core i7+X58 chipset, 12GB memory
- network: x4 DDR Infiniband
- plan to expand to 384-node system RSN. (Cables and switches...)

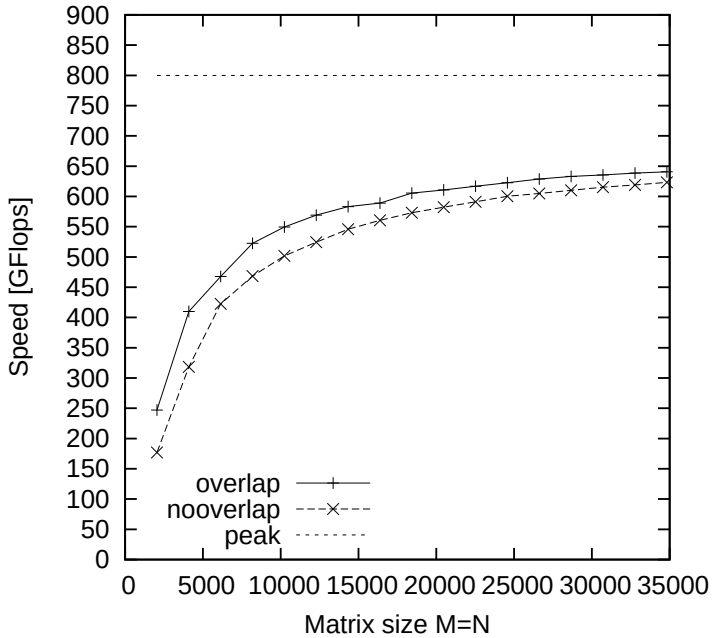
Performance and Tuning example

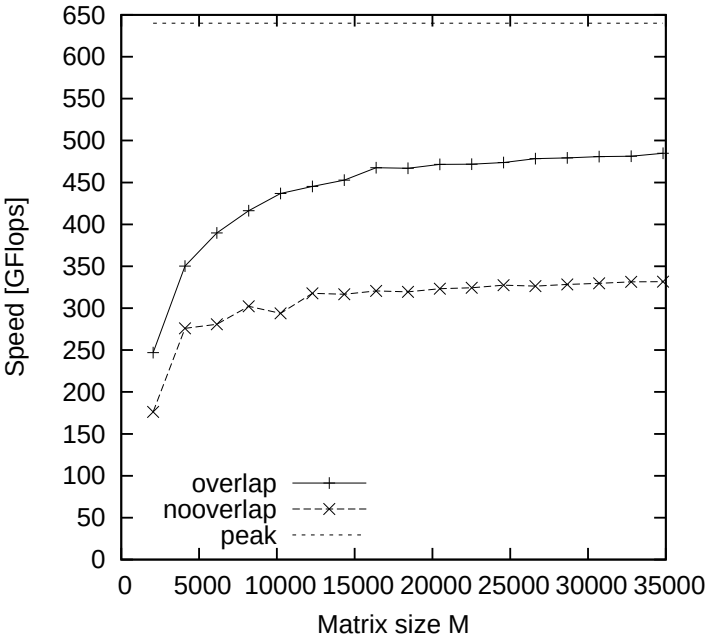
- **HPL (LU-decomposition)**
- **Gravity**

Based on the work by H. Koike (Thesis work)

LU-decomposition

DGEMM performance





M=N, K=2048, 640 Gflops **N=K=2048, 450 Gflops**

FASTEST single-chip and single-card performance on the planet!

(HD5870/5970 will be faster...)

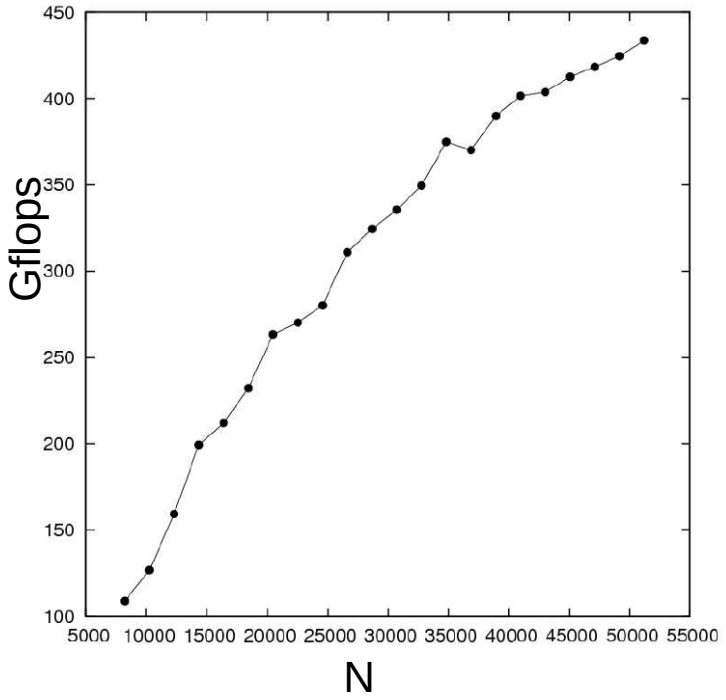
DGEMM tuning

Key to high performance: Overlapping communication and Calculation

- **PE kernel calculates C(8,2)= A(32,8) * B(8,2)**
- **512 PEs calculate C(256,2)= A(512,256)* B(512,2)**
- **Next B sent to chip while calculation**
- **Previous C sent to host while calculation**
- **Next A sent from host to GDR card while calculation**

Everything other than the transfer of B from host to GDR card is hidden.

LU-decomposition performance



Speed in Gflops as function of Matrix size

430 Gflops (54% of theoretical peak) for N=50K

LU-decomposition tuning

- Almost every known techniques
  - except for the concurrent use of CPU and GDR (we use GDR for column factorization as well...)
  - right-looking form
  - TRSM converted to GEMM
  - use row-major order for fast $O(N^2)$ operations
- Several other "new" techniques
  - Transpose matrix during recursive column decomposition
  - Use recursive scheme for TRSM (calculation of L^{-1})

3 weeks to develop the code from scratch

HPL (parallel LU) tuning

- Everything done for single-node LU-decomposition
- Both column- and row-wise communication hidden
- TRSM further modified: calculate UT^{-1} instead of $T^{-1}U$
- More or less working, tuning still necessary

Two months for coding and debugging so far.

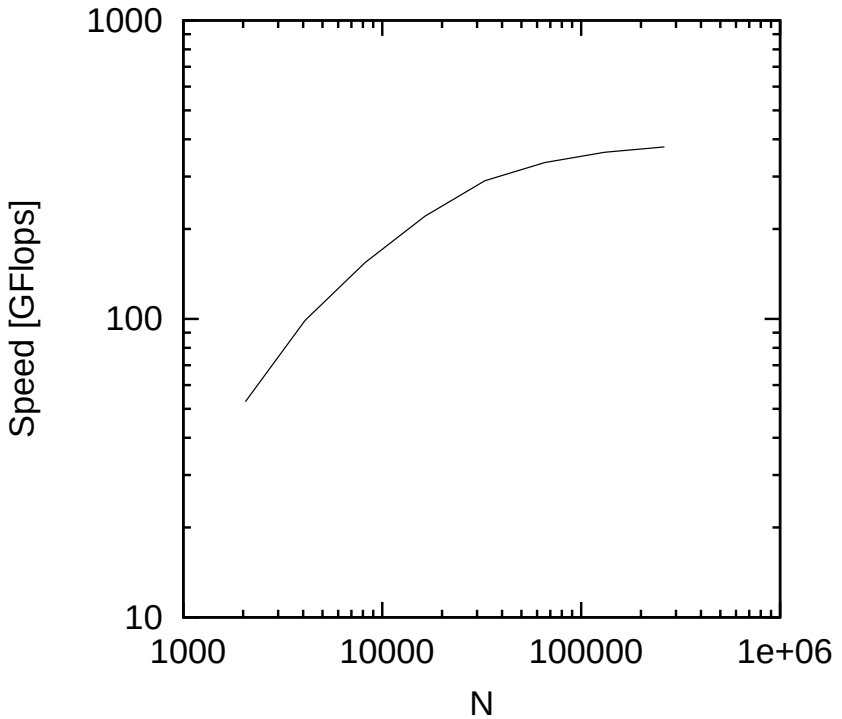
N=30K, single node: 290Gflops

N=60K, 4 nodes: 1170 Gflops

(Super-linear because of partial hiding of panel factorization)

Gravity kernel performance

(Performance of individual timestep code not much different)



Assembly code (which I wrote) is not very optimized yet... Should reach at least 600 Gflops after rewrite.

GRAPE, GRAPE-DR についてのまとめ

- 天文シミュレーション専用計算機 GRAPE はそれなりに成功したような気がする。
- GRAPE-DR で用途を広げる試みは上手くいきそうな気がするけど本当にそうかどうかは今後明らかになる。

計算機の発展の方向

歴史

- **1960 年代 単に、 高くて速い計算機**
  - **代表例:** CDC 6(7)600 (Cray 設計)
- **1970 年代 ベクトル プロセッサ**
  - **代表例:** Cray-1, CDC-Star
- **1980 年代 いろいろあった**
  - **日本のベクトル**
  - **クレイの並列**
  - **いろんな並列計算機**

歴史のつづき

- 1990 年代
  - ベクトル 衰退
  - 並列計算機 会社倒産
  - PC クラスタ 生き残る

何故そうなったか？

何故そうなったか？

- ベクトルは高くなった
- 並列計算機 やっぱり高くなった

何に比べて？

- PC クラスタ

どれくらい高いか？

1975:	Cray-1	100MF	10M$	Cray 50倍お得
	PDP-11/70	10kF?	50K$?	
1985:	Cray XMP	1GF	10M$	XMP 20倍お得
	PC-AT	30kF?	5K$	
1995:	VPP-500	100GF	30M$?	VPP 3倍損
	Dec Alpha	300MF	30K$	
2005:	SX-8	10TF	50M$?	SX-8 60 倍損
	Intel PD	12 GF	1K$	

30年間で 3000倍変わった

普通のスパコンは割高になった

こうなった理由

- チップ間配線とチップ内配線の違い
- 量産効果

チップ間とチップ内

1970 年代: IC を並べて沢山配線して計算機を作った。プロセッサ内もメモリまでも同じ。
1980 年代終わり以降: パイプライン演算器をもった、 Cray-1 みたいなものが 1 チップにいくらでも入るようになった
2009 年現在:

- 1 チップに 10^8 ゲート以上 (Cray-1 なら 300 台) 入る
- チップ内動作クロックは GHz 以上
- チップの外への配線は多くて数百
- 外の配線の速度は高くて GHz

つまり:
1 チップ計算速度: Tflops は容易 (まあ、、メーカーさんが作れば、、、)
データ転送速度: 100GB/s がせいいっぱい

ベクトルプロセッサとマイクロプロセッサ

- Cray-1、その後の普通のベクトルプロセッサ: 主記憶のバンド幅と演算速度が大体つりあう。つまり、メモリが 100GB/s なら 数十 Gflops。
- マイクロプロセッサ: そういうことはあまり考えないで演算速度あげた。10GB/s で 50Gflops とか。
- それでも演算器は少ない
- システムの価格は演算速度ではなくチップ間転送バンド幅で決まっている。

マイクロプロセッサが HPC の将来か？

x86 マイクロプロセッサの進歩はスーパーコンピューターの進歩を 20 年遅れで追いかけている

フルデコード乗算器	CDC 7600 (1971)	i80860(1989)
密結合マルチプロセッサ	Cray XMP (1982)	Pen D (2005)

スーパーコンピューターのこの後の進化:
1992 頃まで: 16 プロセッサ程度までのゆっくりした進化
1993: 航技研 NWT (VPP500)。アーキテクチャの革新
1995 以降: それでも衰退

今後も同じなら、マイクロプロセッサも
2010-2015: なんらかの革新
2015 以降: それでも衰退

マイクロプロセッサに代わるものは？

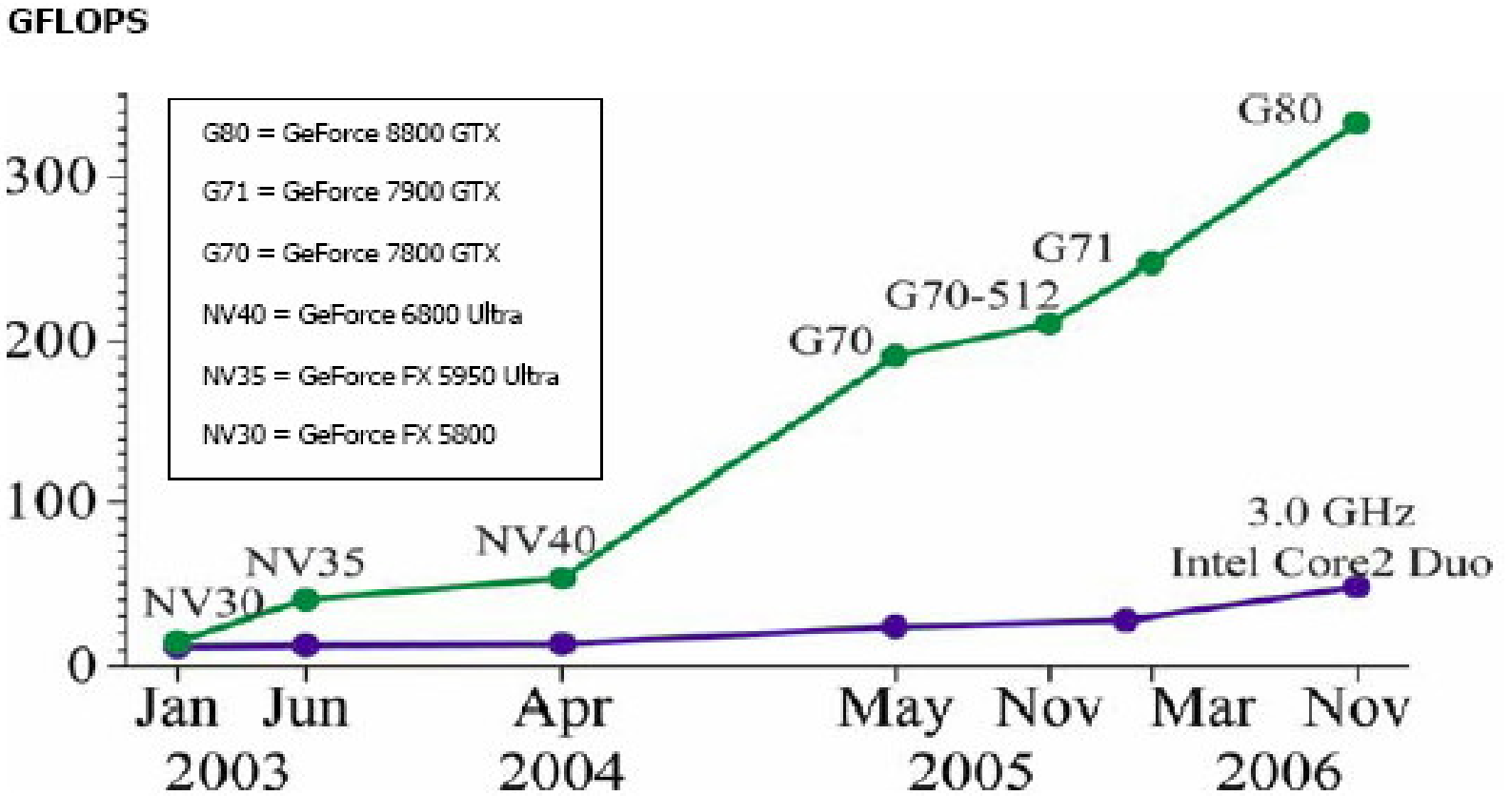
- 専用アーキテクチャ？
  - 10 億の予算とれれば。
- FPGA とか？
  - 20 年前から「未来のプロセッサ」。今でもそう。
- GPU とか？
- それ以外？

GPGPU

グラフィックカードを計算に使う。

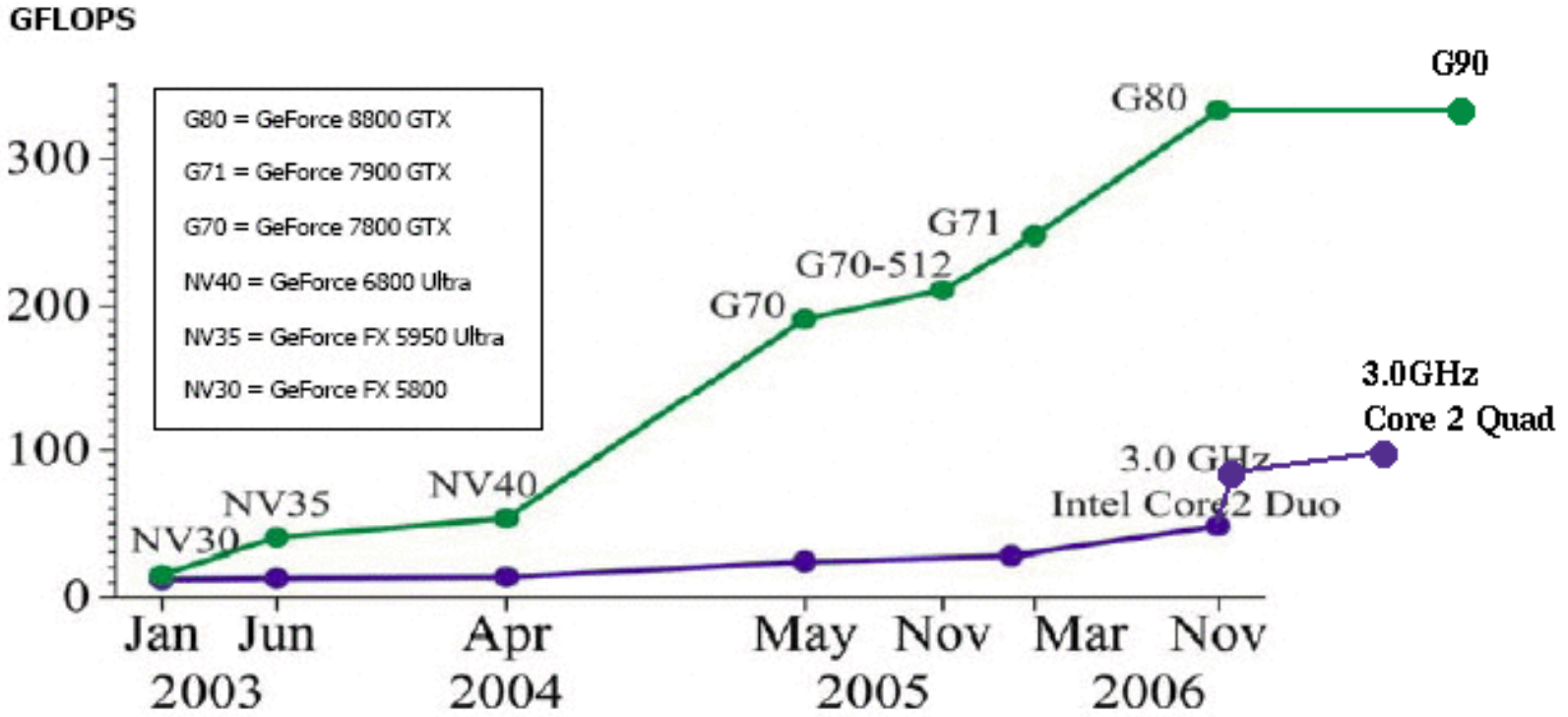
- nVidia 8800: C でプログラム書ける。ボード上のメモリ 768MB、メモリ速度 90GB/s(SX-9 の 1/3)
- データを全部ボード上に置いて GPU の C で全部プログラム書き直せば 100Gflops くらい出るかも。(単精度なら、、、倍精度は速度 1/8 らしい)
- カードは安い (10 万円以下)
- 将来性は怪しい

GPGPUs — メーカーさんの見せる図



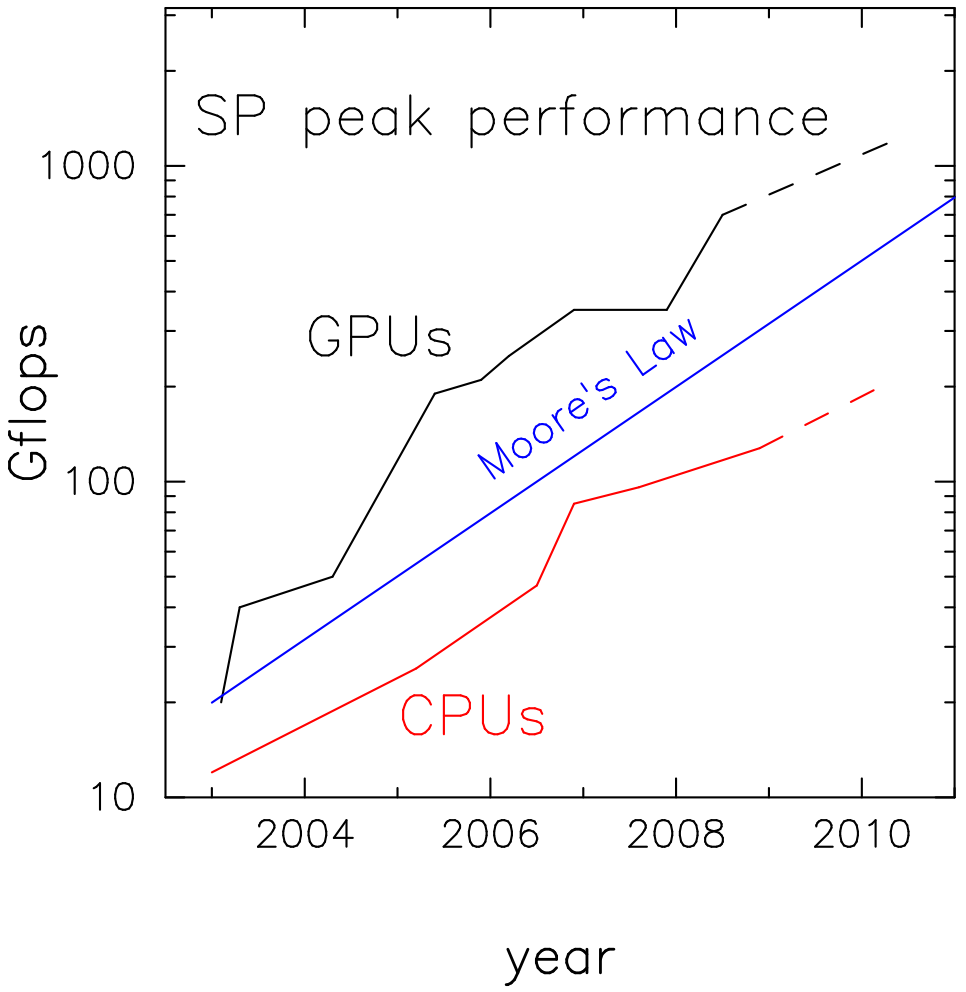
「ムーアの法則より速い!」

GPGPUs —2007年末まで



あれ？

GPGPUs — 対数グラフでは



- 2005 年以降減速
- 汎用マイクロプロセッサがキャッチアップ
- 倍精度 ??
- メモリバンド幅リミット
- ピーク性能で CPU の 10 倍前後
- 実アプリケーションでは xxx (100 倍とかいう話はデタラメ)

計算機に関するまとめ

- GRAPE では、重力相互作用計算に専用化したパイプラインプロセッサをフルカスタム LSI で実現することで、同時期のマイクロプロセッサの数十倍の性能を数分の一の消費電力で実現してきた。
- GRAPE-DR では、SIMD 方式でプログラム可能にすることで GRAPE より広い応用を実現する。コストは数倍あがるが我慢する。
- サンプルチップは完成し、正しく動作をすることが評価ボードで確認できた。
- x86, GPGPU の将来がどういうものかは？なので、他のことも考えましょう。

おまけ:次世代スーパーコンピューター

- 文部科学省の計画: 2012 年度に 10+ Pflops、予算 1100 億 (100 億増えたり減ったり)

開発に意味はあるか？という話。

- Cray XT6@2010: 1Tflops 500-1000 万円。
- Cray 2012 年後: 1 Tflops 2-300 万円、 10Pflops 2-300 億円
- GRAPE-DR: 1Tflops 100 万円@2008。5 年後？

	次世代	AMD	GRAPE-DR
B/F	0.5	0.4	0.016

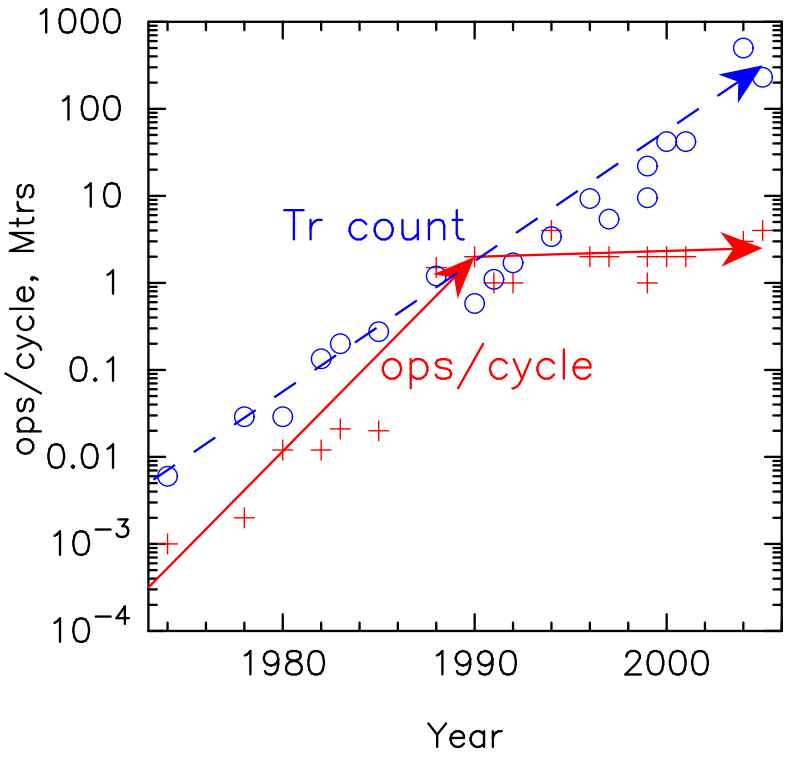
予備資料

Memory Wall への対応法

- ベクトルプロセッサ
  - 正攻法で高速、多数のメモリ配線を用意
  - 利点: 多くのアプリケーションプログラムで高い実効性能
  - 欠点: 高価
- スカラープロセッサ
  - 多階層のキャッシュ
  - 利点: 安価
  - 欠点:
    - 設計が複雑
    - アプリケーションを性能がでるように書くのは困難
      - キャッシュの振る舞いを間接的に制御する必要あり
      - プリフェッチでは不十分

これでいいのか？

- プロセッサチップでのトランジスタの利用効率
  - 1990年から一貫して低下を続けてきた
- スカラープロセッサ
  - ほとんどの面積がキャッシュとその制御回路
- ベクトルプロセッサ
  - ほとんどの面積がI/Oとベクトルレジスタ



トランジスタの有効利用法を考え直す時期

システム構成の考え方

- アプリケーション実行はホスト計算機との連携で行う
- できるだけ強力なホスト計算機・ネットワークを使いたい
- 以下に示すシステム構成は最低線のもの

プログラミング環境

- 加速ボードのメーカーが必ずいうこと:
  - アプリケーションプログラムから加速ボード側で実行できる部分を自動検出、変換できます
  - ソースコードに手を加えずに性能向上可能です
- 30 年前から同じことをいわれてきた
  - 今更騙される人はいない
- より現実的、効果的なアプローチ
  - 加速ボードでは単純なカーネルルーチンだけを実行
  - 他の部分はホスト上のプログラム。
    これは「基本的には」改変なし

V-GRAPE で提供する環境

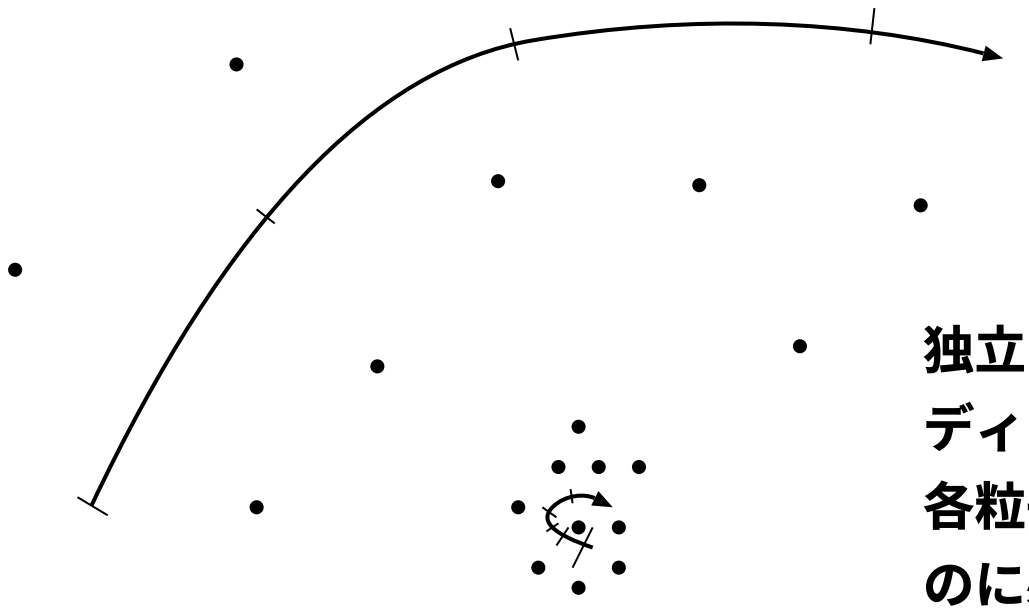
- カーネルルーチン
  - 行列乗算、対角化等の行列演算ルーチン
    - BLAS, LAPACK のサブルーチンの形に
  - 粒子間相互作用計算ルーチン
    - 単純な粒子間相互作用程度はアセンブラでも書ける
  - 二電子積分、交換積分などグランドチャレンジに必要なルーチン
- アセンブラ
  - 昔はみんなこれで書いていた
- PE 上でのコードに対して、「高級言語」を提供
  - PGDL (理研 濱田、中里、FPGA 再構成計算用コンパイラ)
  - SPH 法のための専用パイプライン (演算器 150) の合成に初めて成功

まとめ

- 重力多体系向けの計算方法を概観した
- 時間スケール・空間スケールの幅があるために、それぞれについて適応的な方法が使われている
  - 時間: 独立時間刻み
  - 空間: ツリー法
- 2 つの組合せは難しい。
- 限られた状況では上手くいくような方法はいくつかできた

おまけ:独立時間刻みというのは本当に大丈夫なのか？

何が問題か？

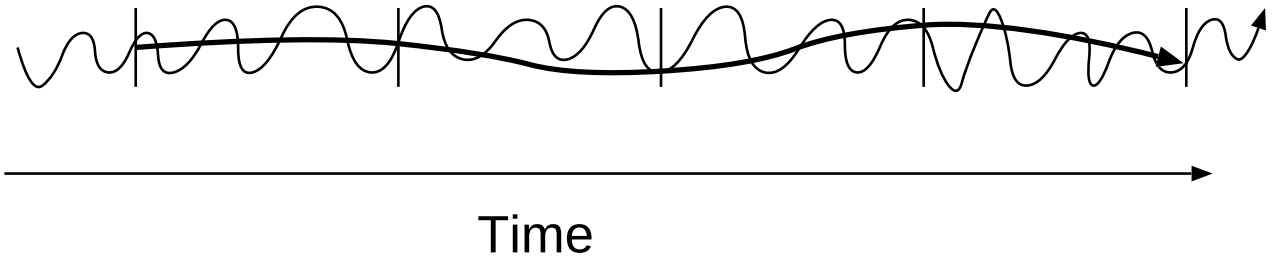


独立時間刻みの基本的なアイディア:
各粒子は自分の軌道を分解するのに必要十分な時間刻みを取る

タイムスケールが短い粒子から長い粒子への力はどうなっているのか？

どうなっていると思われるか？

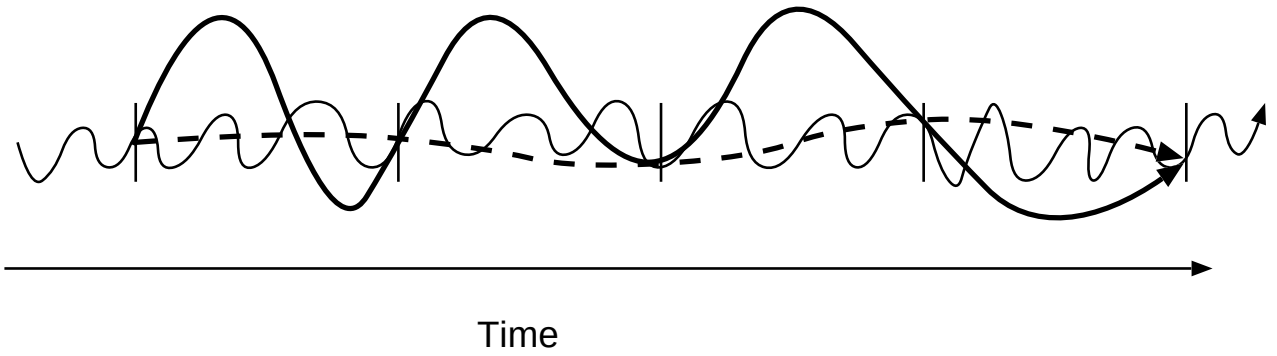
全然デタラメなはず。



- **力は時間スケールの小さい変動をするはず。**
- **タイムステップが長いと、この力がランダムにサンプルされる。(タイムスケールの短い粒子の軌道が完全に周期的でないなら)**

エルミート公式の場合

もっと悪い。

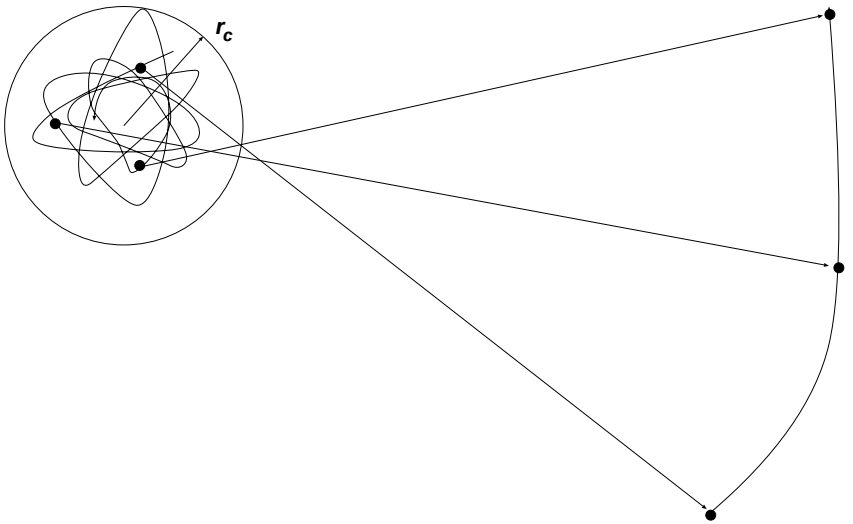


- **導関数の情報を使うべく頑張る**
- **でも無理**
- **無理に計算する**

どういう時に起こるか？

星団の場合:

- **コア領域の粒子の軌道タイムスケールが星団の典型的な星の時間刻みより短くなると起こる。**
- **大雑把にいって:** $r_c \ll r_h N^{-1/3}$
- **収縮から膨張に移る時:** $r_c \sim r_h/N \ll r_h N^{-1/3}$

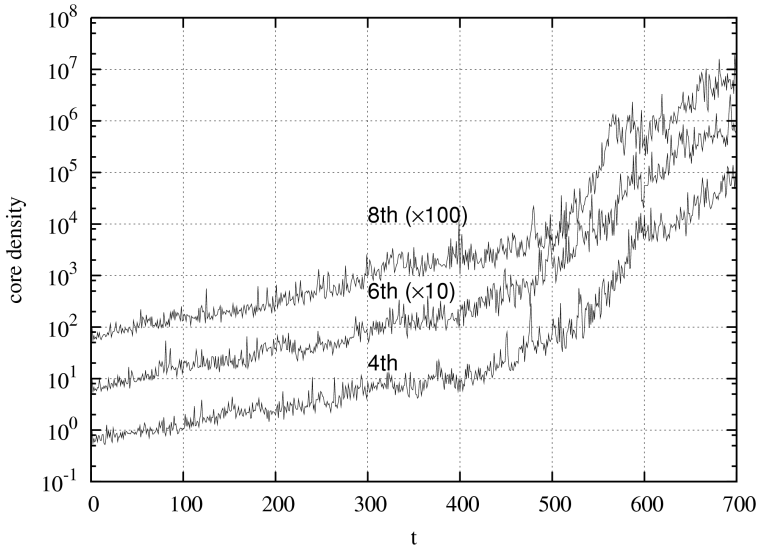


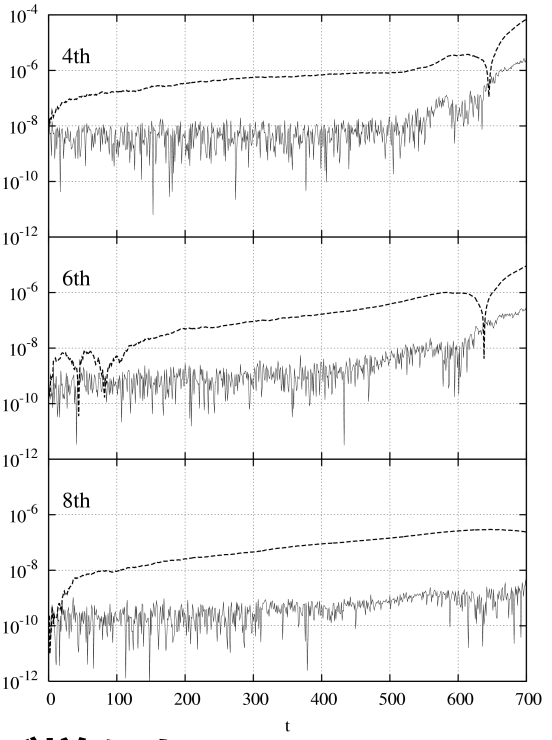
本当に問題は起こるか？

コアが小さくなるとエネルギー誤差が急激に増大する

ということは昔からなんとなく知られていたが、コア自身の積分誤差だとみんな思っていた。
そうではないらしいと判明: Nitadori and JM 2007 の数値実験
やったこと: 1024 体プラマーモデルをコラプスまで計算

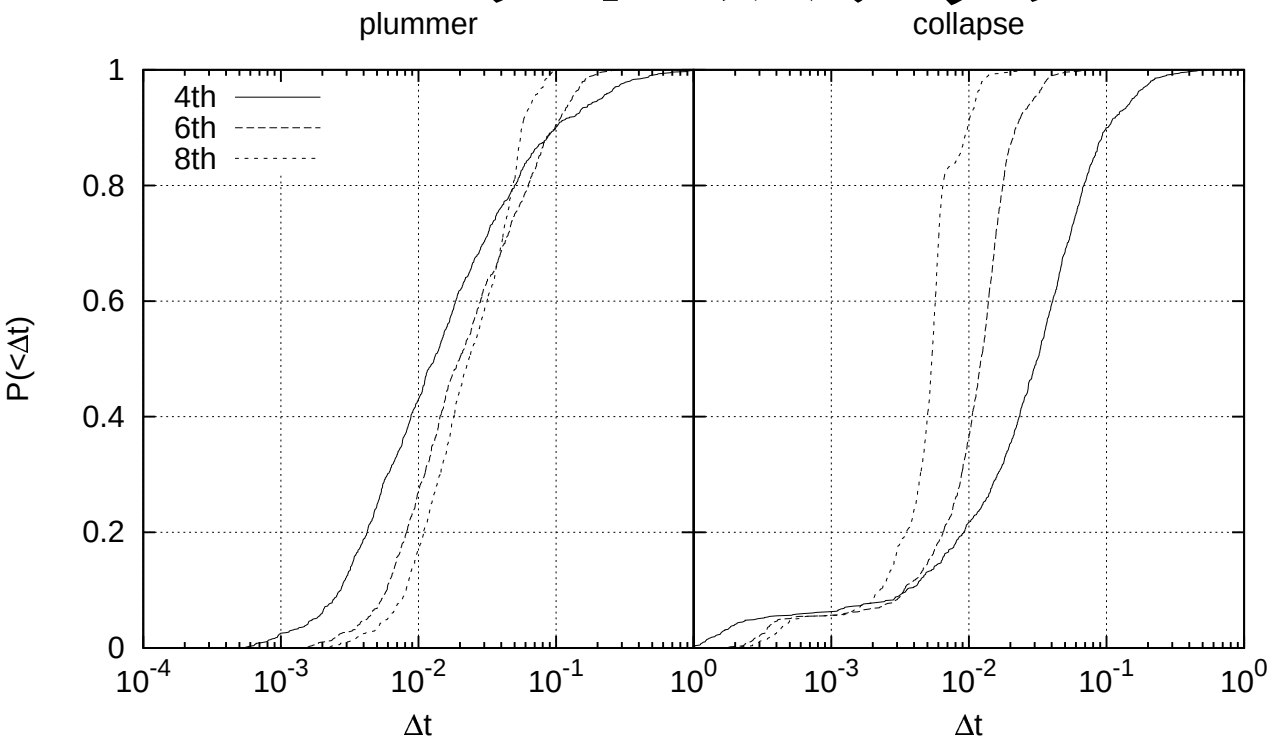
中心密度と誤差





8次スキームだとコラプスしても誤差が増えない。

タイムステップ



左:初期条件、右:コラプスした後

- 基本的に外側のほうがタイムステップ長い
- 8 次スキームは全体に時間ステップが短くなっている
- 4 次スキームでは外側のほうは長いまま

解釈

- 8 次スキームでは、外側の粒子がコアの粒子の軌道変化のためにタイムステップが短くなる
- そのため、コラプスしても誤差が増えない
- 4 次スキームではそうならない
- 6 次は中間的
- 8 次スキームは安全だが、「独立時間刻み」といえるか？
- 一番短いタイムステップの 30 倍程度が限界？