V-GRAPE とその気候モデルへの応用可能性について

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
牧野淳一郎

CCSR セミナー 2007/1/24

今日の話の構成

- 天体の粒子シミュレーション
- これまでの GRAPE
- GRAPE-DR の考え方
- 開発状況
- ソフトウェアの実際
- 次世代スーパーコンピューターと V-GRAPE
- 気候モデルへの応用可能性
- スーパーコンピューターの発展の方向
- まとめ

天体の粒子シミュレーション

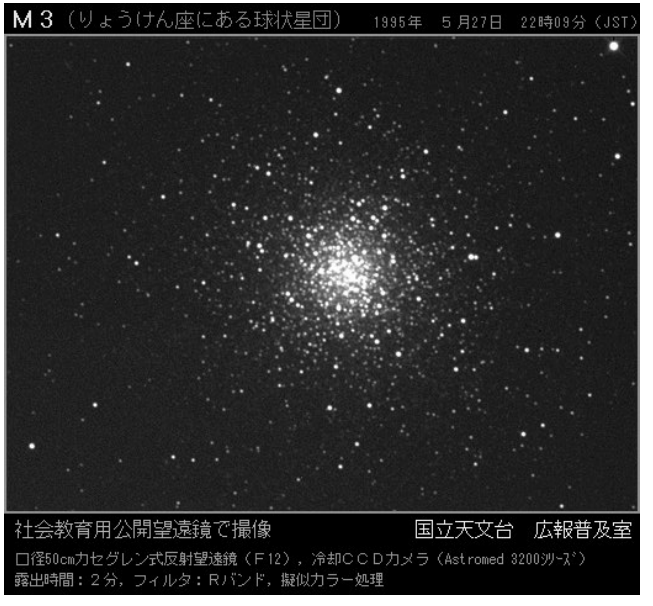
- 対象の例
- 基本的な考え方

対象の例

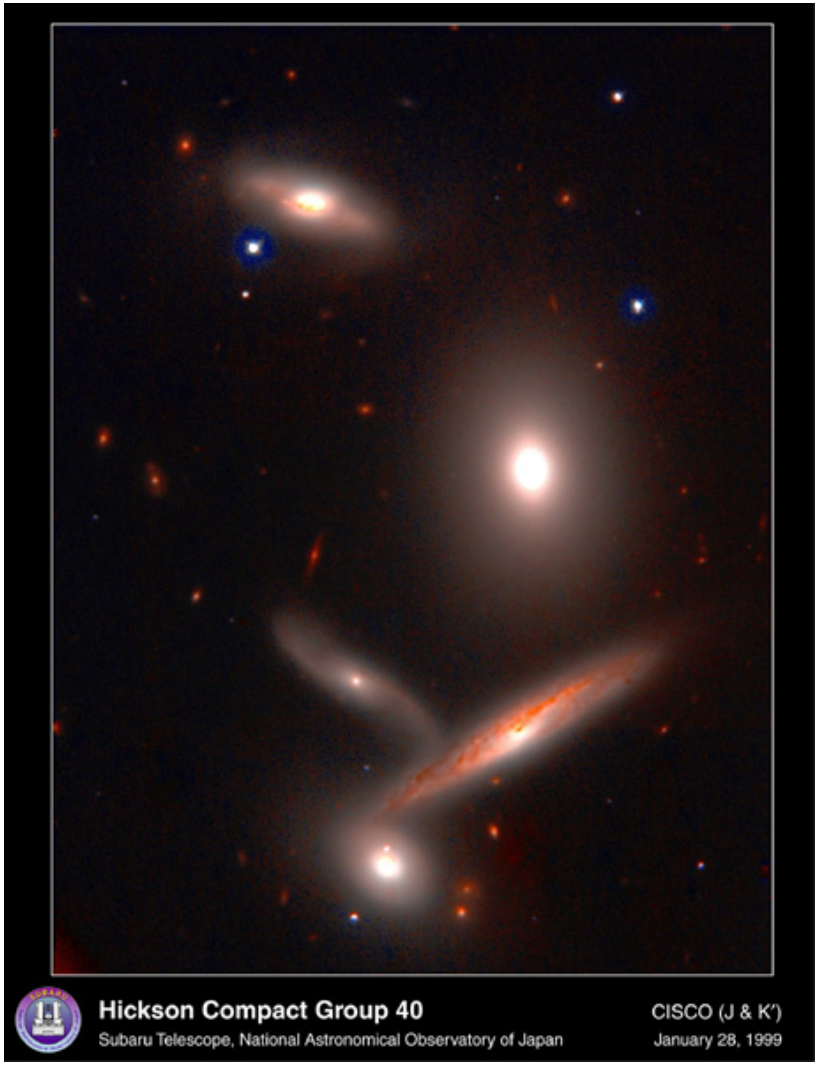
銀河


M51（子持ち銀河）
1995年　5月19日　23時10分（JST）
社会教育用公開望遠鏡で撮像
国立天文台　広報普及室
口径50cm力セグレン式反射望遠鏡（F12），冷却CCDカメラ（Astromed 3200ｼﾘｰｽﾞ）
露出時間：2分，フィルタ：Rバンド，擬似カラー処理

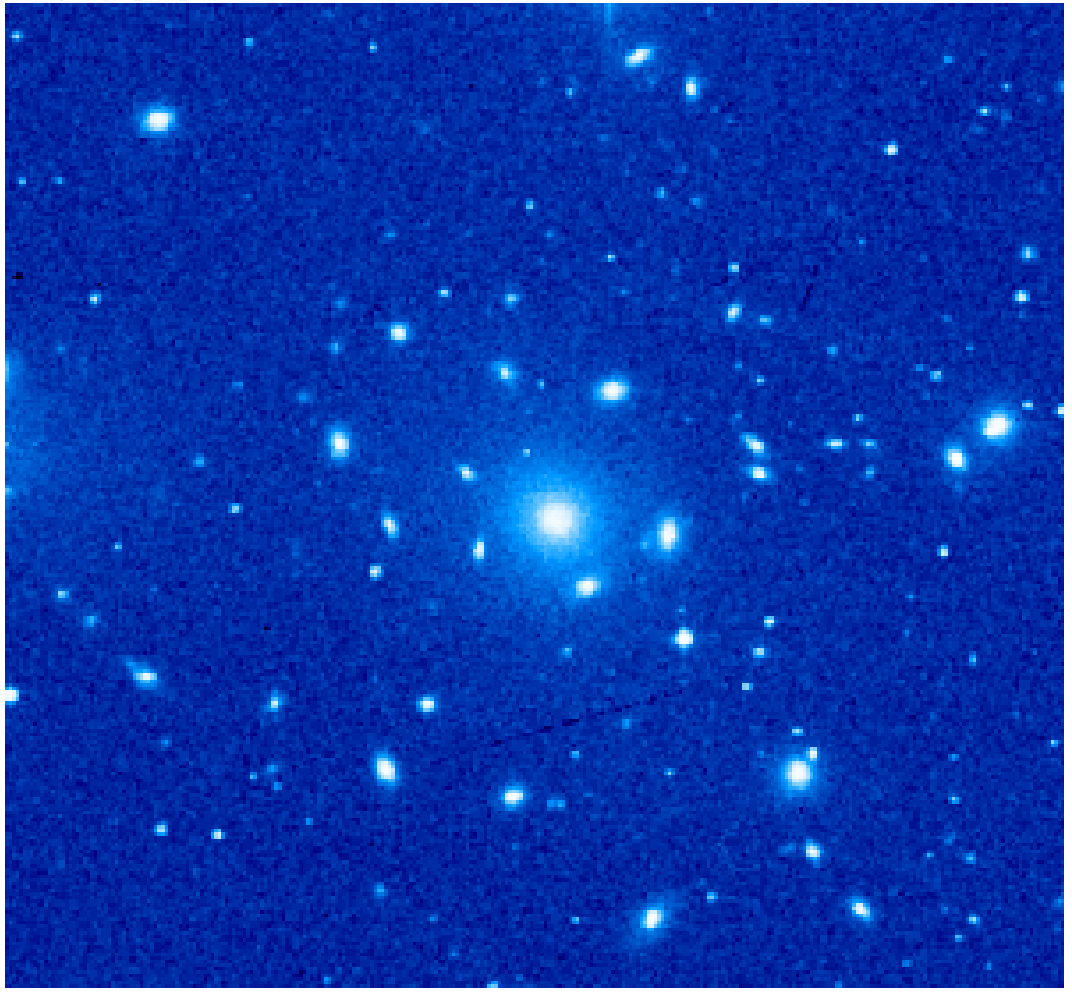
球状星団


M3（りょうけん座にある球状星団）
1995年　5月27日　22時09分（JST）
社会教育用公開望遠鏡で撮像
国立天文台　広報普及室
口径50cm力セグレン式反射望遠鏡（F12），冷却CCDカメラ（Astromed 3200ｼﾘｰｽﾞ）
露出時間：2分，フィルタ：Rバンド，擬似カラー処理

銀河群

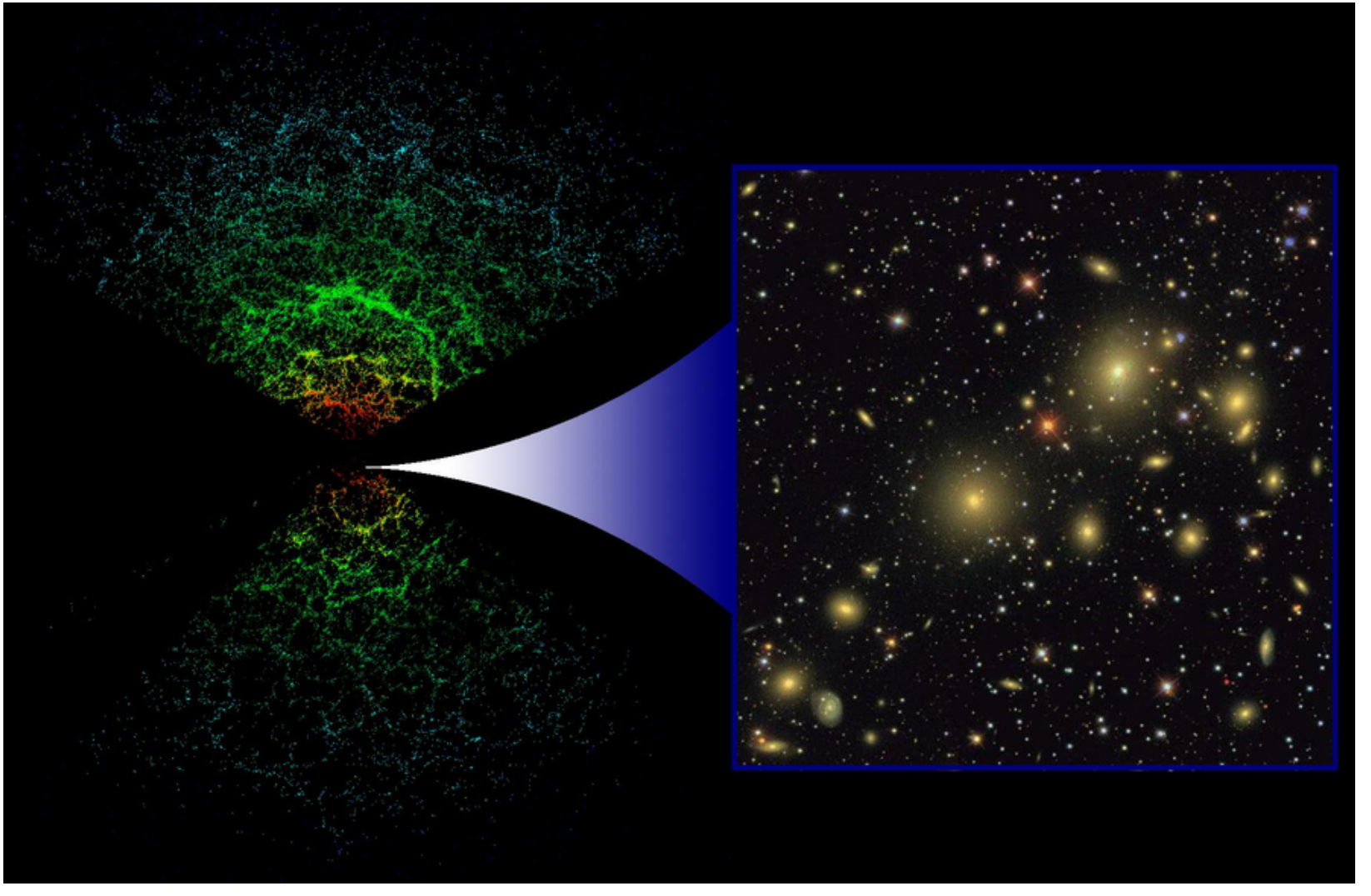

Hickson Compact Group 40
Subaru Telescope, National Astronomical Observatory of Japan
CISCO (J & K')
January 28, 1999

銀河団



http://antwrp.gsfc.nasa.gov/apod/ap950917.html

大規模構造 (距離情報あり) — SDSS スライス

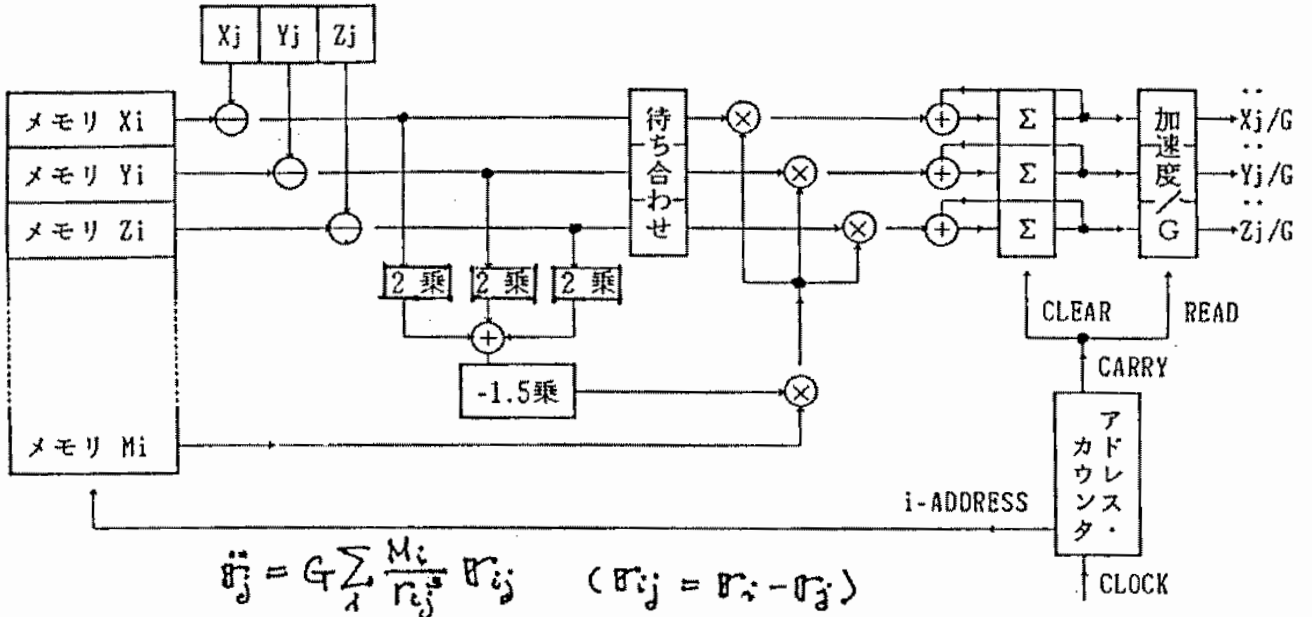


GRAPE の考え方

- **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

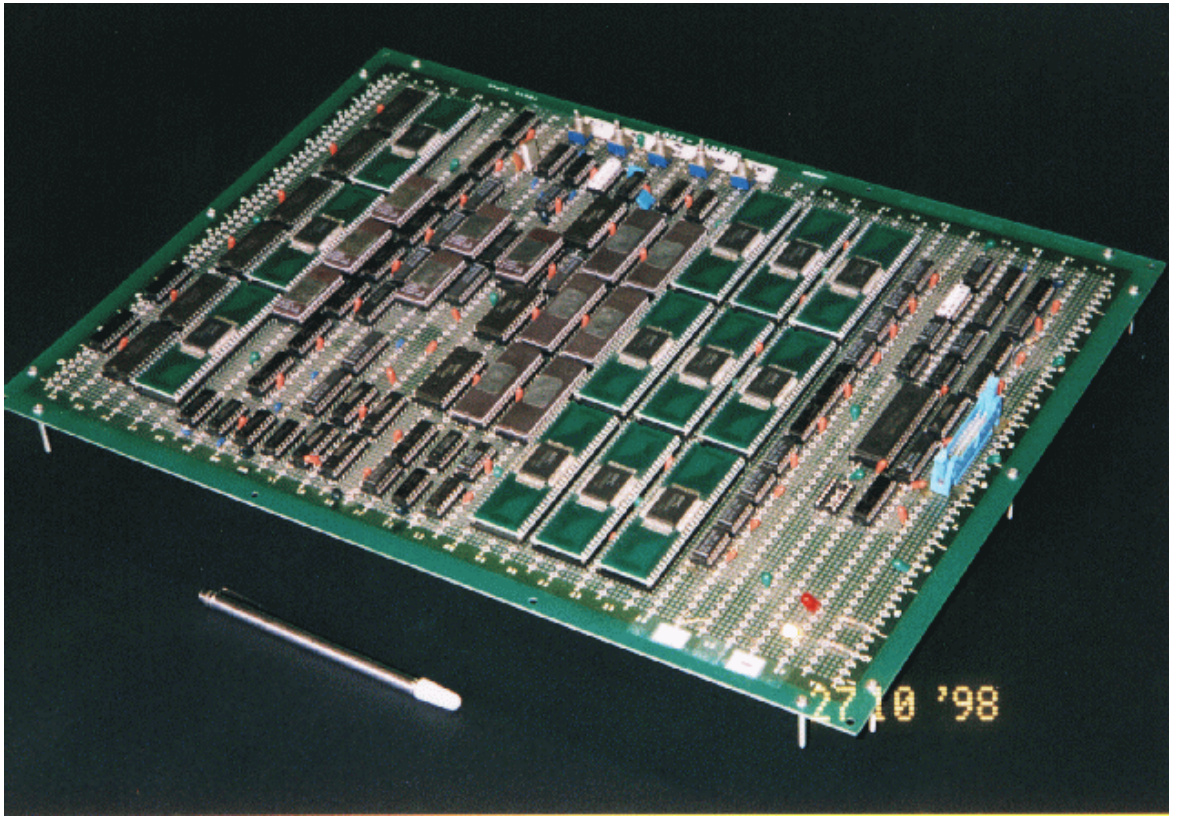


$$\ddot{\vec{r}}_j = G\sum_i \frac{M_i}{r_{ij}^3}\vec{r}_{ij} \qquad (\vec{r}_{ij} = \vec{r}_i - \vec{r}_j)$$

+，－，×，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．

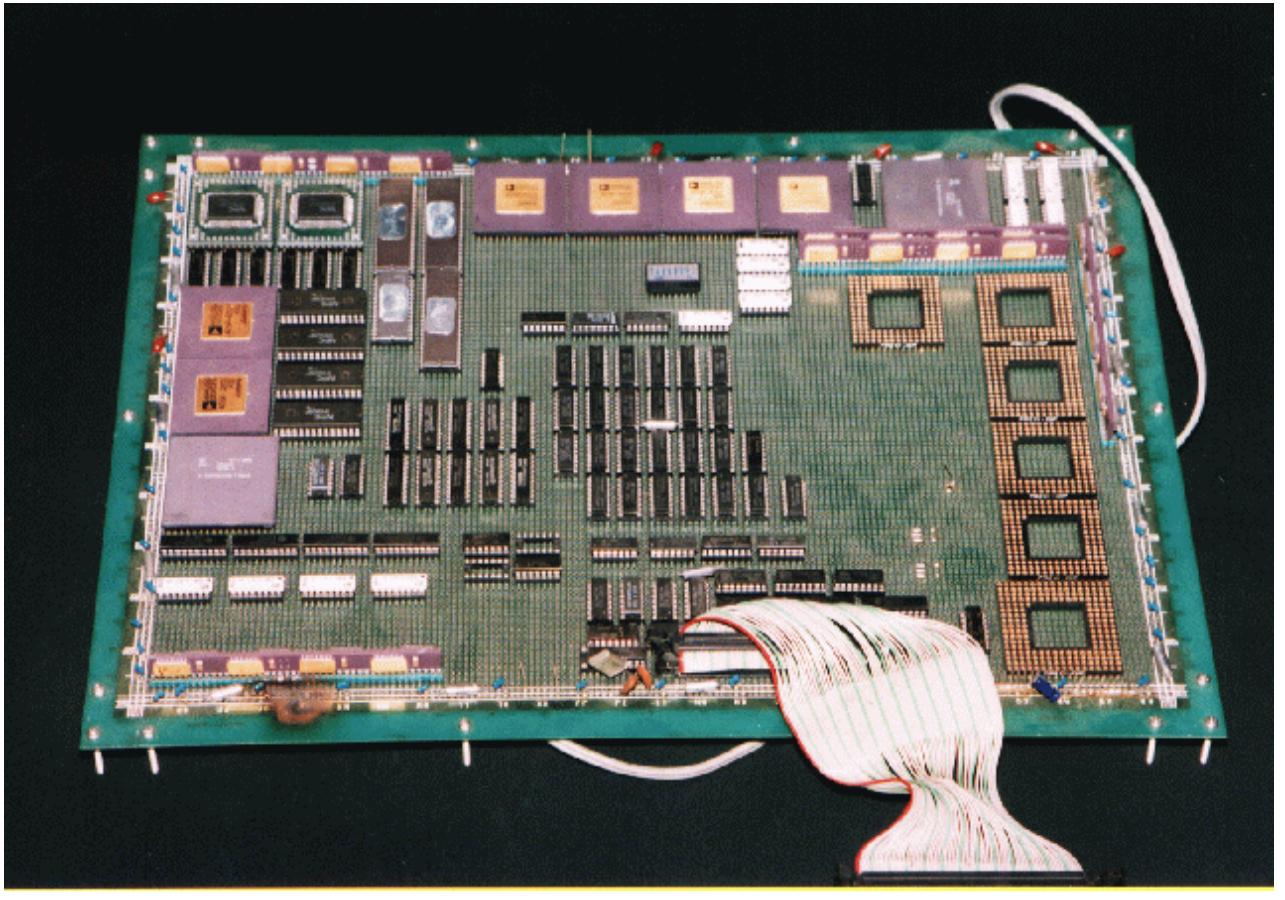
図２．N体問題のj-体に働く重力加速度を計算する回路の概念図．

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**

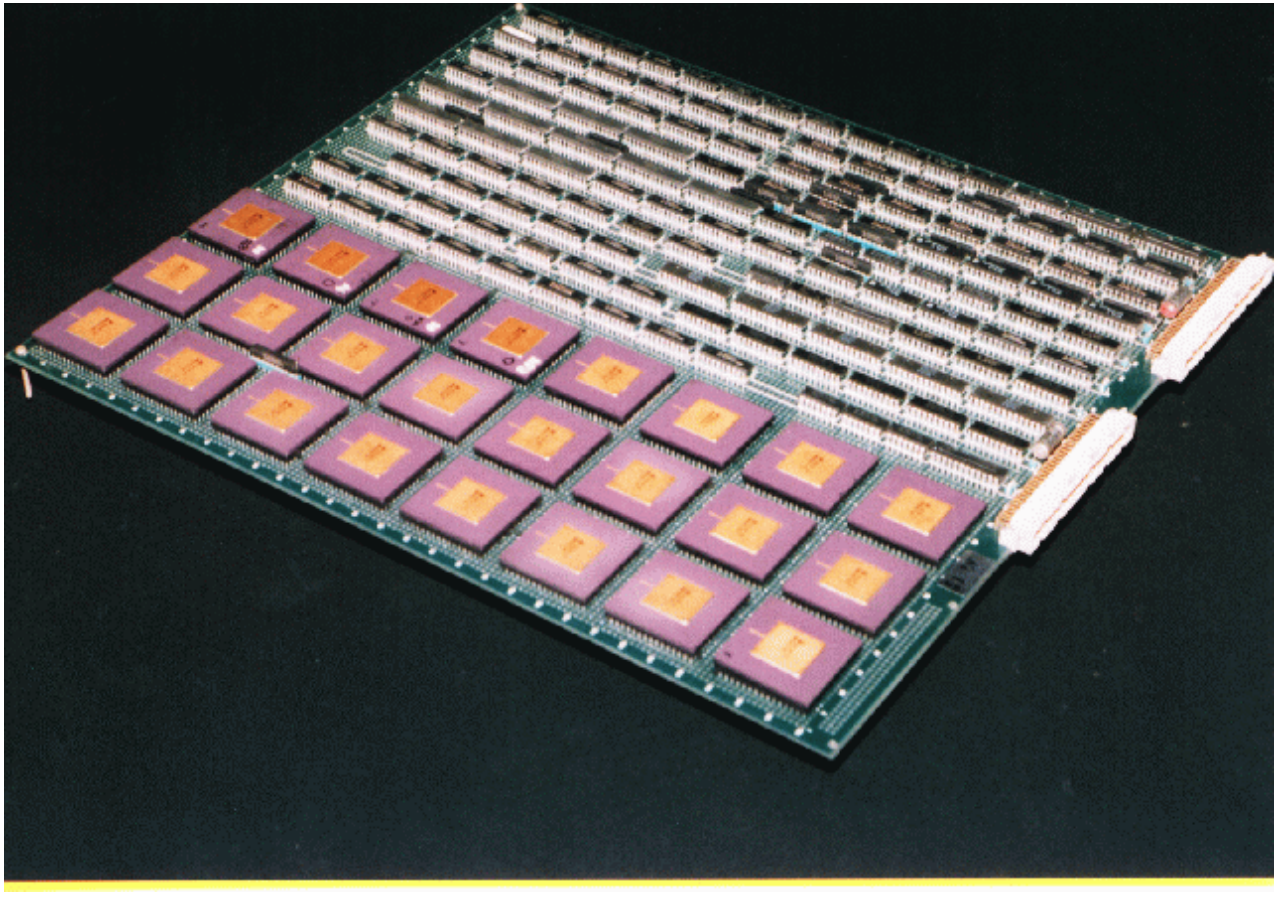
GRAPE-2(1990)



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)

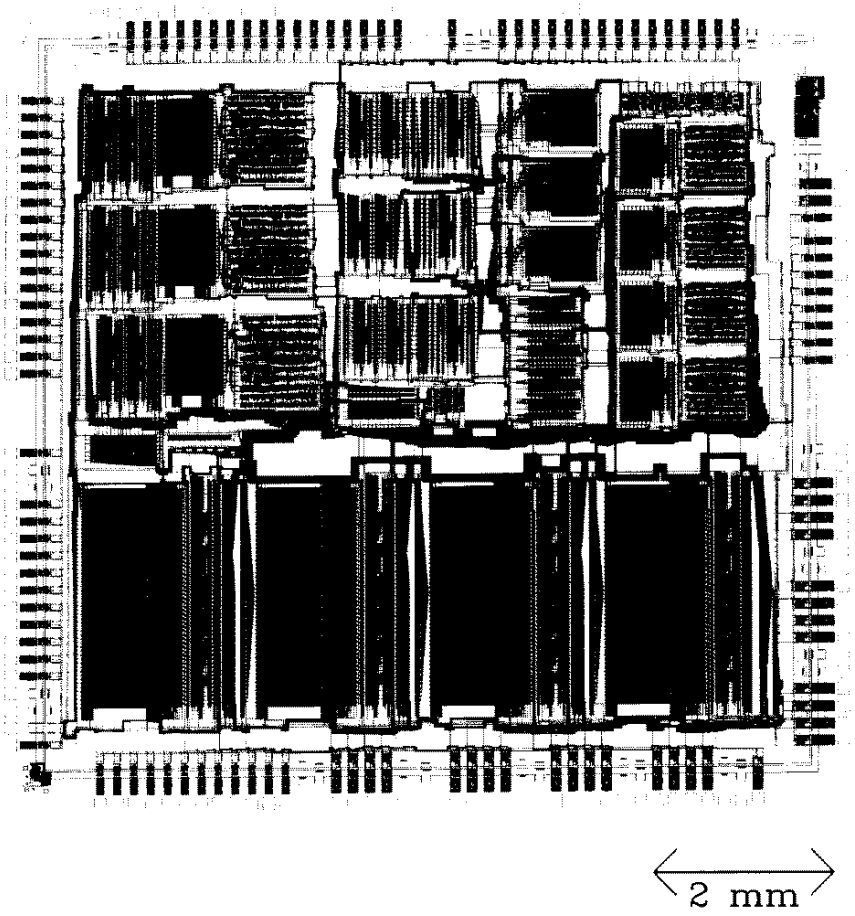
40Mflops

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

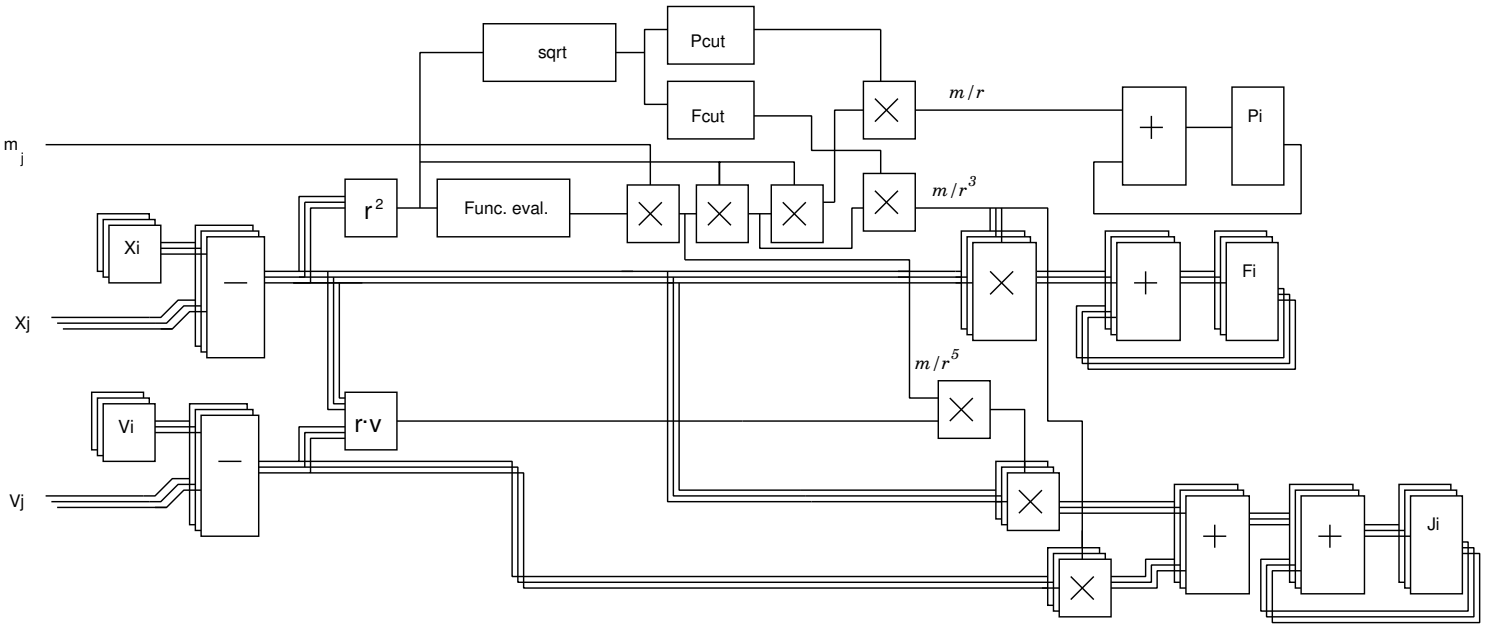


- 1μm プロセス
- 11万トランジスタ
- 20 MHz クロック動作
- 600 Mflops 相当

GRAPE-4(1995)

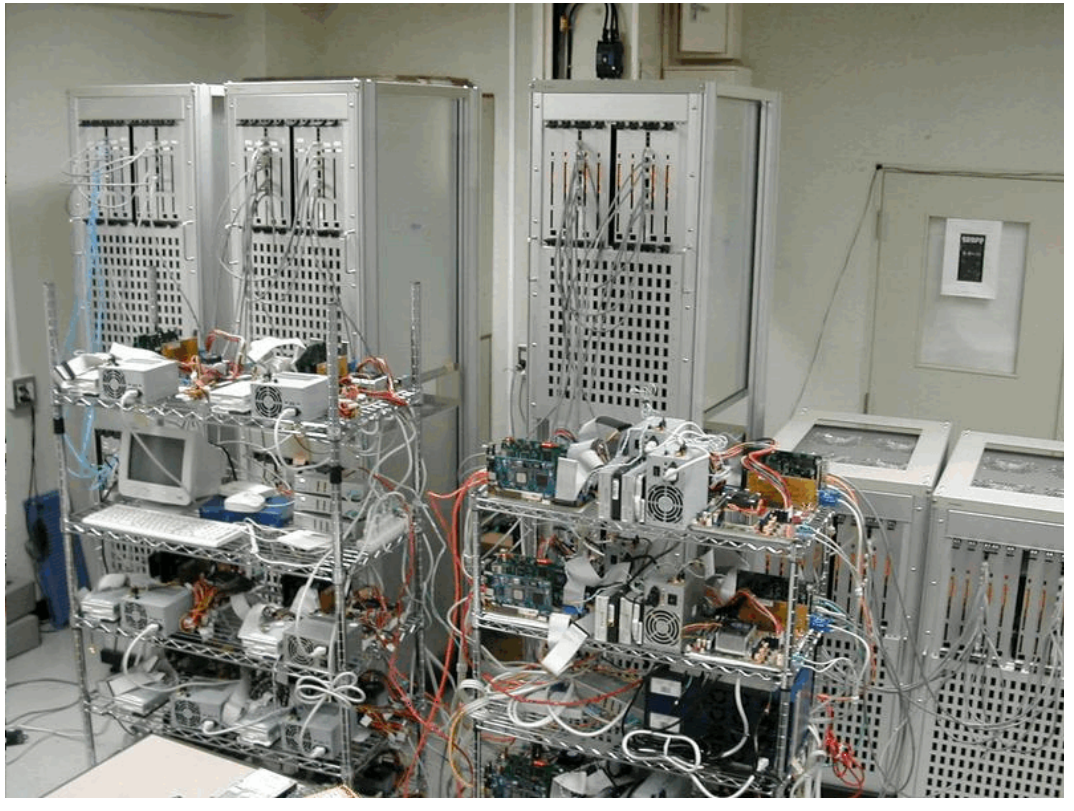
トータル 1792 チップ、 1.1 Tflops

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

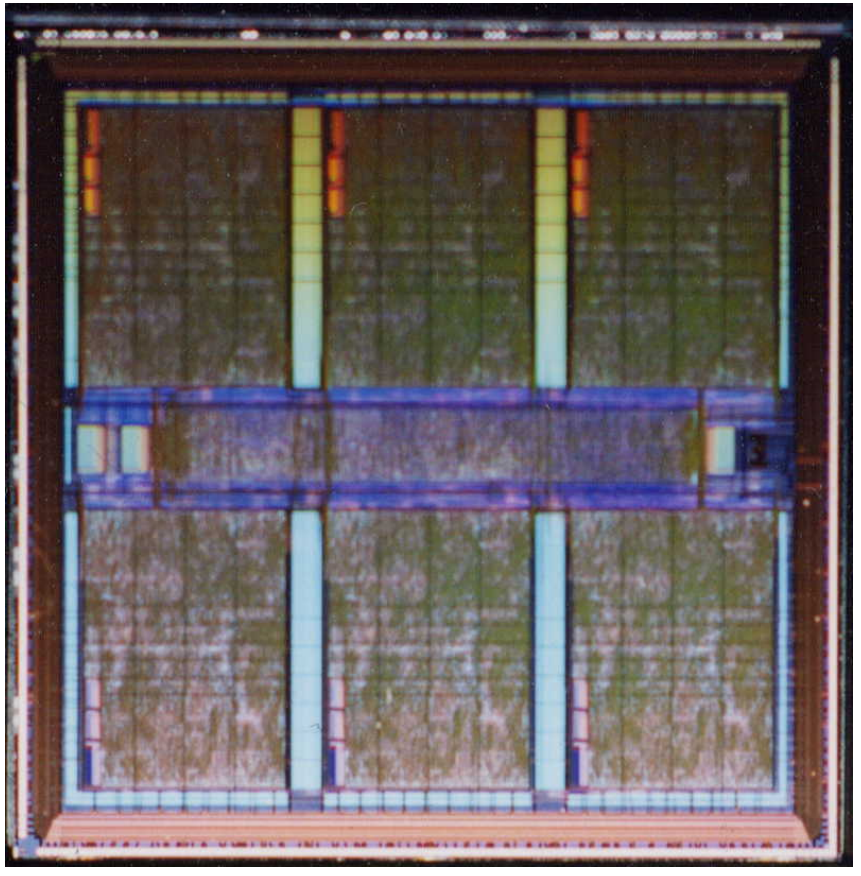
GRAPE-6(2002)



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

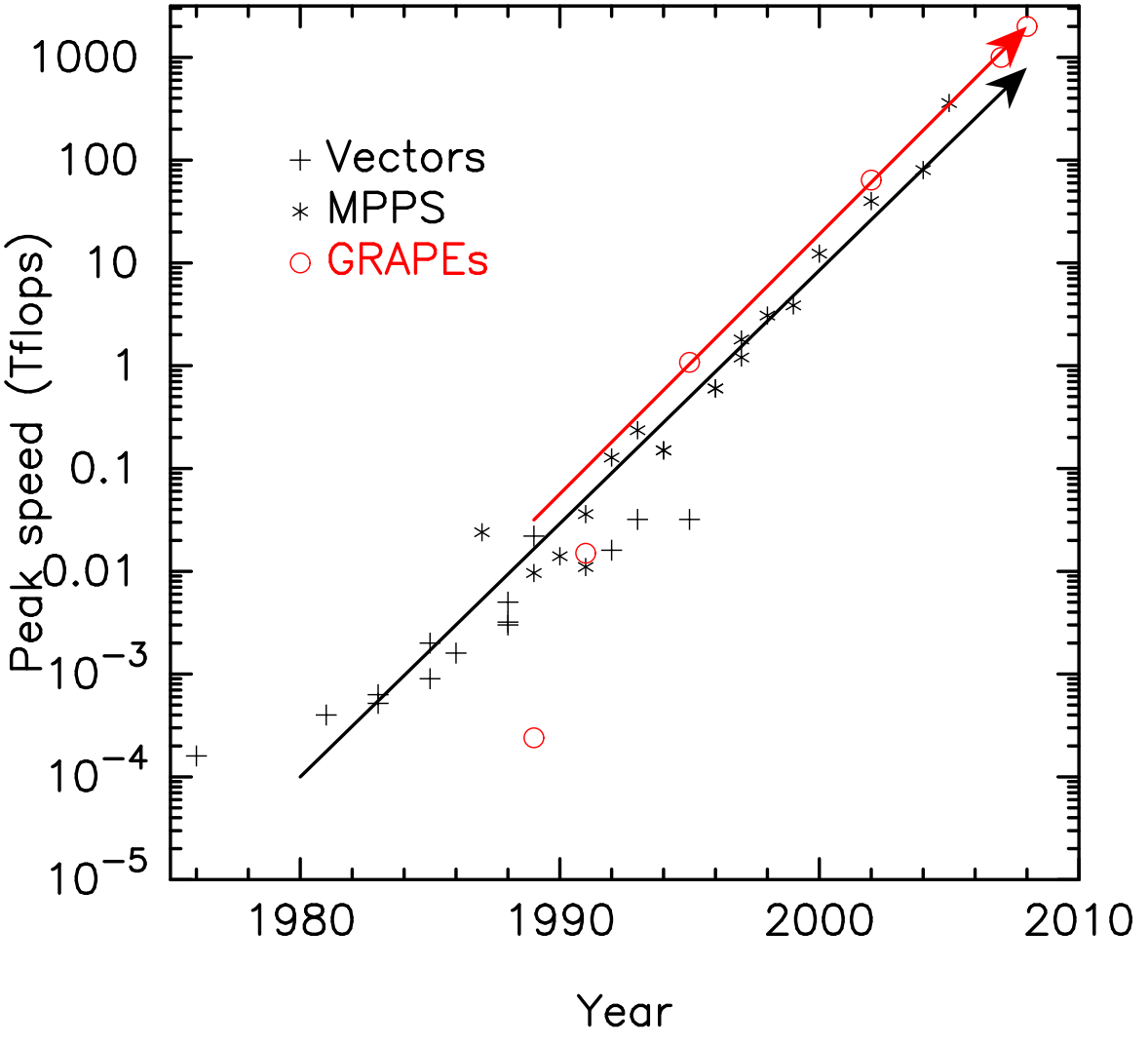


- 0.25 μm ルール
  (東芝　TC-240，　1.8M ゲート)
- 90 MHz 動作
- 6 パイプラインを集積
- チップあたり 31 Gflops

2006 年のマイクロプロセッサと比べてみる

	GRAPE-6	Core 2 Extreme
デザインルール	250nm	65nm
動作クロック	90MHz	2.93GHz
ピーク性能	32.4Gflops	23.44Gflops
消費電力	10W	75W
1W あたり性能	3.24Gflops	0.313 Gflops

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

GRAPE-6 の次は？

MDGRAPE-3 の次: MDGRAPE-4, 20Pflops@2010

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
2006	65nm	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

ではどうするか

1. **やめる**

ではどうするか

1. やめる
2. 安くあげる方法を考える

ではどうするか

1. やめる
2. 安くあげる方法を考える
3. なんかお金を取る方法を考える

ではどうするか

1. やめる
2. 安くあげる方法を考える
3. なんかお金を取る方法を考える

GRAPE-DR では (3) を選択

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

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

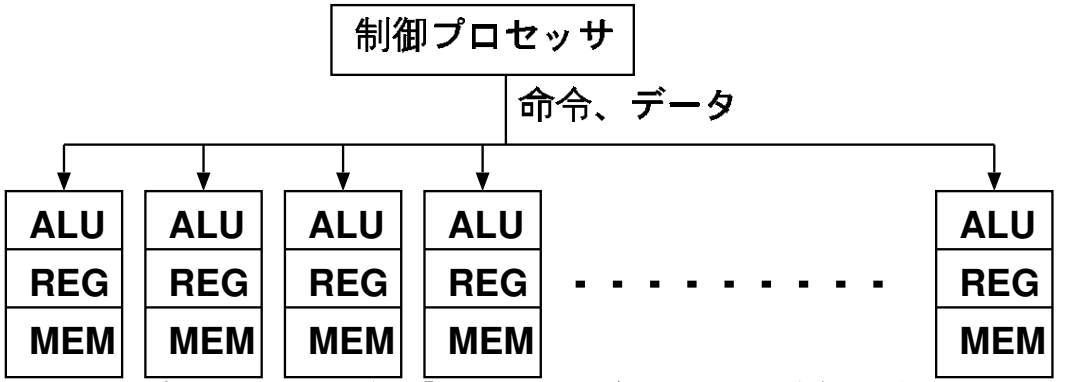
基本的には、全プロセッサがソフトウェアで GRAPE をエミュレーションする。

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

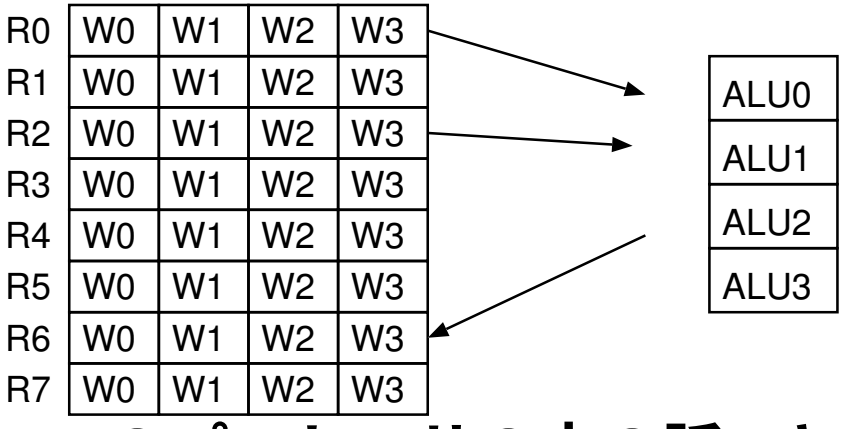


1960年代に発生、80年代に絶滅
半導体技術の向上に対応できないアーキテクチャ: 計算速度とメモリアクセス速度が比例する必要あり。
メモリ階層をつける: プロセッサが複雑になりすぎて SIMD の意味が無くなる。

SIMD 拡張命令

Pentium III, 4 が有名

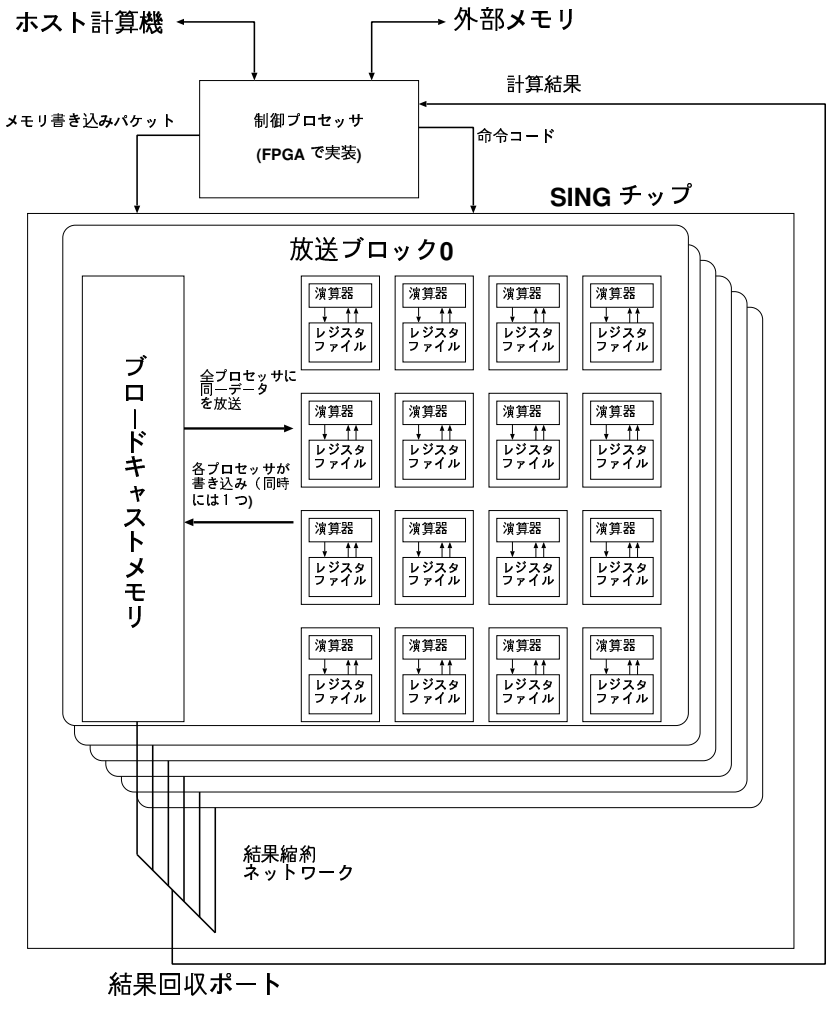
128 ビットなり 64 ビットのデータを 4 語に区切って、それぞれの要素に対する演算を 4 個の演算器で同時に処理



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

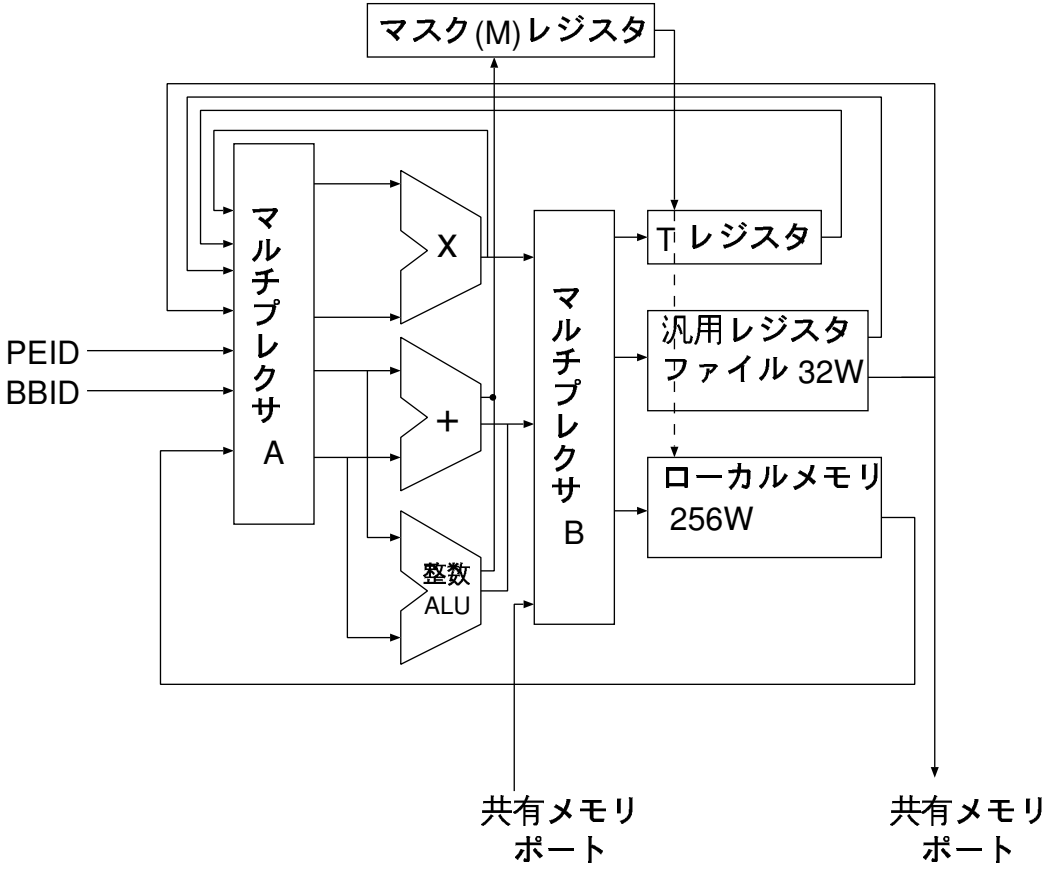
並列度 4 程度が限界？それ以上増やすとキャッシュの速度が追いつかなくなる。

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

PE の構造

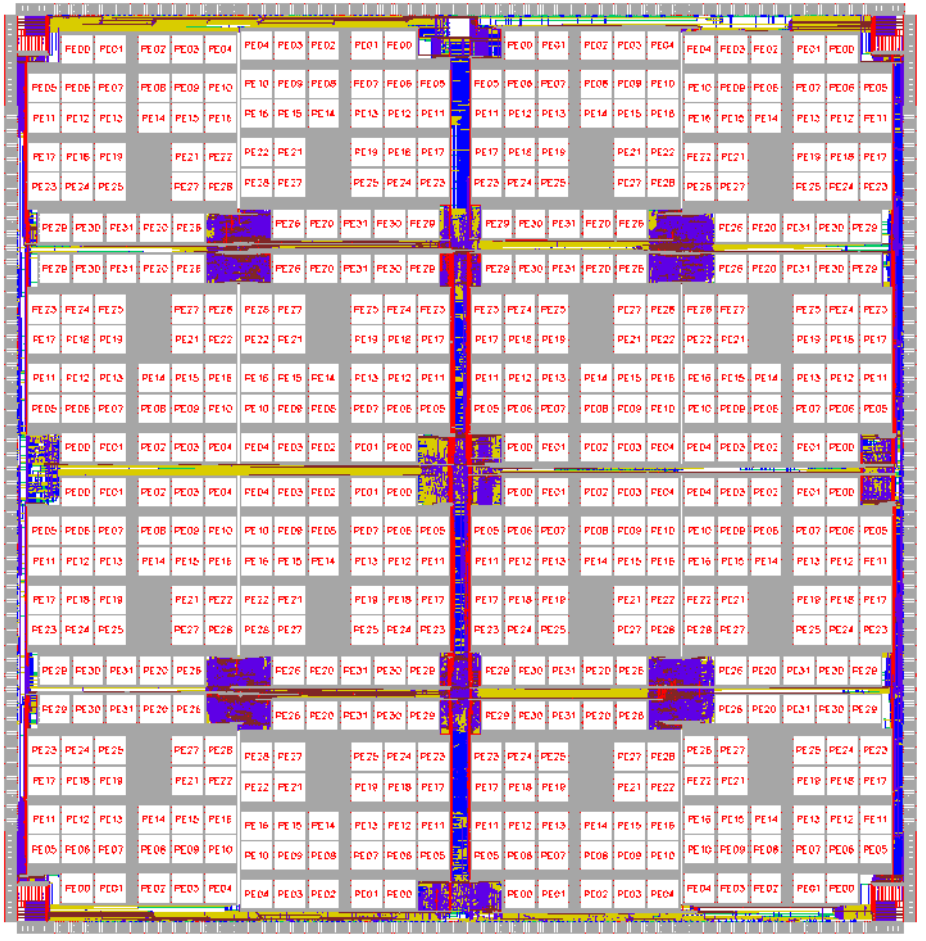


- 浮動小数点演算器
- 整数演算器
- レジスタ
- メモリ 256 語 (K とか M ではない。)

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

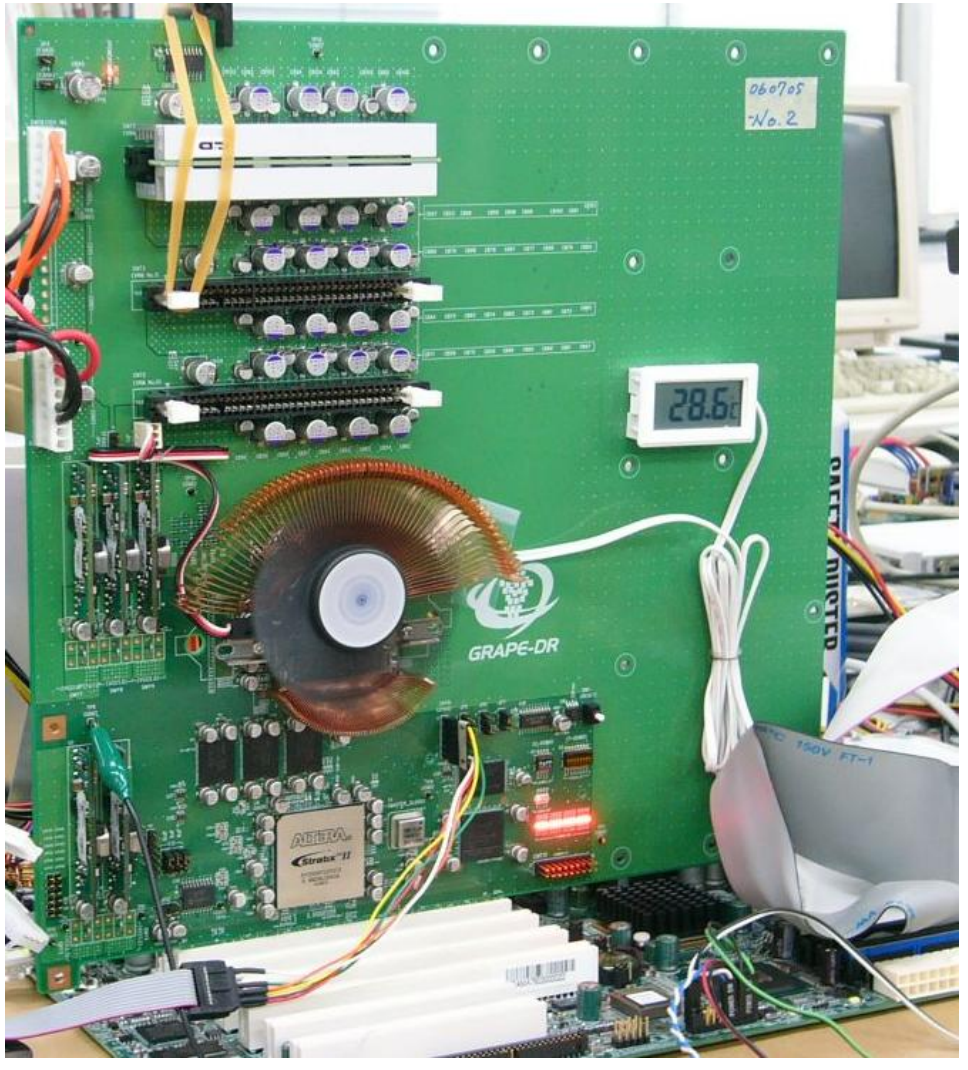


- 32PE(要素プロセッサ) のブロックが16個
- 空いているところは共有メモリ
- チップサイズは18mm 角

GRAPE-DR の開発状況

500MHz。いくつかのアプリケーションが動作。
重力多体問題、分子動力学
100 Gflops くらいはでている (藤野、修士論文)

GRAPE-DR 別ボード



- こっちが「プロジェクト公式」
- 中身は殆ど同じ
- 何故か大きい
- LINPACK が動作したらしい

量産型ボード

- PCI-Express カード (8 レーン、通信速度 2GB/s)
- 4 GRAPE-DR チップ (予定)
- 現在、 PCI-Express を使った 1 チップボードを設計中

全体システム

- 目標:理論ピーク $1\ \mathrm{Pflops} = 4096$ チップ
- ボード2枚のせた $\mathrm{PC}\ 512$ 台で構成する予定
- 製造は年次進行

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

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)

コンパイラ

(中里 2006)

```
/VARI  xi, yi, zi, e2;
/VARJ  xj, yj, zj, mj;
/VARF  fx, fy, fz;
dx = xi - xj;
dy = yi - yj;
dz = zi - zj;
r2 = dx*dx + dy*dy + dz*dz + e2;
r3i= powm32(r2);
ff = mj*r3i;
fx += ff*dx;
fy += ff*dy;
fz += ff*dz;
```

これから GRAPE 並のことをするアセンブラ、インターフェースライブラリを生成。
基本的なアイデアは PGR (FPGA を使ったPROGRAPE用コンパイラ、濱田D論2006) と同様

インターフェース関数

中身はコンパイラ/アセンブラ記述から自動生成される。

```
int SING_send_j_particle(struct grape_j_particle_struct *jp,
                         int index_in_EM);
int SING_send_i_particle(struct grape_i_particle_struct *ip,
                         int n);
int SING_get_result(struct grape_result_struct *rp);
void SING_grape_init();
int SING_grape_run(int n);
```

インターフェース関数の機能

- 粒子データを送る
- 計算させる
- 結果を回収する

もうちょっと汎用に使うには？

とりあえず、「粒子間相互作用」でなくても、無理矢理その形に書けば大抵のことは表現できる

- **格子・粒子相互作用**
- **格子・格子相互作用**
- **普通の、依存性のない並列計算**

量子化学計算、2 電子積分を実装中 (理研・名古屋大学)

V-GRAPE

- GRAPE-DR は大体出来た
- 設計でどこに失敗したかも大体わかってきた
- 次世代スーパーコンピューターでアクセラレータがどうこうという話がある

次を作るとすれば何をするか？
= V-GRAPE

GRAPE-DR における問題点

- あまり速くない
  — 倍精度256Gflops は名目速度では MDGRAPE-3 並
- ボード製造コストが高い
  — メモリ制御用 FPGA、メモリ自体等部品が多い
- FFT、CG 反復といった、計算量が結構多いがメモリアクセスも多い計算で性能がでない

速くないのは設計があんまりまだ頑張ってないせい。演算ユニットの作りが適当。最適化の余地は大きい。

下の 2 つは発想の転換が必要。

FFT、CG法で遅いのは何故か

理由: そもそもそのへんを速くすることを考えてない。

考えれば速くなる ???

原理的な限界はどこか、が問題。

FFT の場合

FFT をする前のデータはホスト計算機にある

↓

それを アクセラレータに転送して FFT する

↓

結果を回収する

という手順だと:

- 原理的に計算量は通信量の $10 \log n$ 倍くらい。
- 転送速度 4GB/s として、 10 Gflops 程度。
- CPU 使ったほうが速い

CG 法

に限らず疎行列の反復解法:
格子点当りの計算量は $O(10)$ × 反復回数

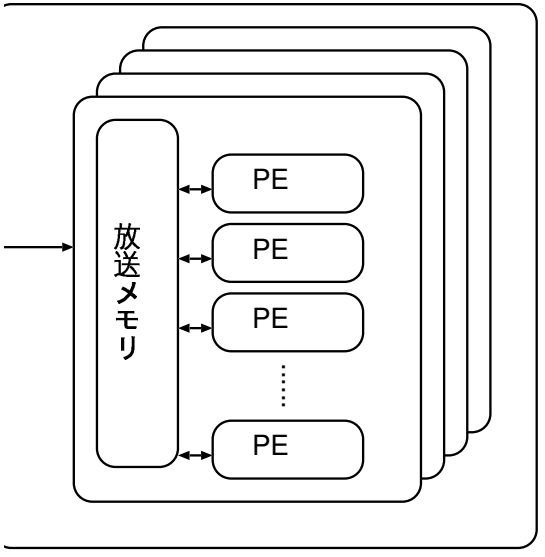
- **格子点データをチップ内にもつ**
- **一度データ送ったら収束するまでチップ内だけで反復する**

というふうにできれば性能は出る。

チップ内メモリの量

- 現在の GRAPE-DR: 1MB
- Intel Itanium とかのとても高いチップ: 24MB?
- オンチップDRAM や 1T-SRAM を使うと: 32 MB くらいならやってくれそう

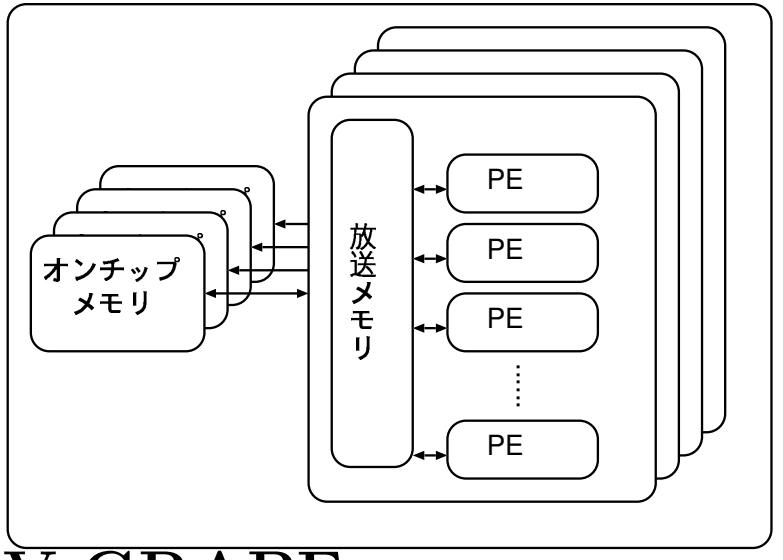
V-GRAPE



GRAPE-DR

大きなメモリはチップ外

バンド幅小さい



V-GRAPE

大きなメモリを中に

メモリと PE ブロックの接続

方式は検討中

V-GRAPE の利点/欠点

利点:

- アプリケーションが増える
- メモリインターフェースがないので開発が楽
- 製造も (チップ製造以外は) 楽、低コスト
- ボードにプロセッサチップを沢山のせられる

欠点:

- メモリにチップ面積を喰われるのでピーク性能低下
- チップ開発コストはあがる

気候モデルへの応用

といっても私は専門家でないし、むしろ皆さんの考えを聞きたい。

とはいえ、それでもで終わるのでは申し訳ないので、少し。

全然専門ではないので、勘違いしていたら教えて下さい

どこの計算量が多いか？

(GRAPE に限らず) 高速化の基本方針

- 一番時間がかかっているところを見つける
- そこにかかる時間を減らす方法を考える

GRAPE の場合:

- 基本的には相互作用計算を速くした
- 相互作用計算がすごく速くなったら、他のところを (相互作用計算が増えても) 速くする方法も考えた

どこの計算量が多いか？

という以前にどんな計算法を使ってるのか？気候モデル (全球モデル) の場合？

- **SC2002 でゴードンベル賞のコード**
- **NICAM**

SC2002 のコード

Shingu et al, "A 26.58 Tflops Global Atmospheric Simulation with the Spectral Transform Method on the Earth Simulator "

- ルジャンドル変換によるスペクトル法
- $3840 \times 1920 \times 96$ グリッド
- 概ね、計算量の 90% 程度がルジャンドル変換

これは GRAPE-DR で高い効率で実行できるはず。
緯度方向に n データがあると n^2 の計算量

NICAM

http://www.gfd-dennou.org/seminars/wtk/2006-05-20/iga/pub/ を参考に。

- 正 20 面体メッシュ
- 基本的には圧縮性、2 次精度の有限体積法

次数が低い差分法なので、必然的に演算に対してメモリアクセスが多い。
1 タイムステップ毎にホスト計算機主記憶からデータを GRAPE-DR に送るような方法では性能がでない。

どれくらい性能がでないか

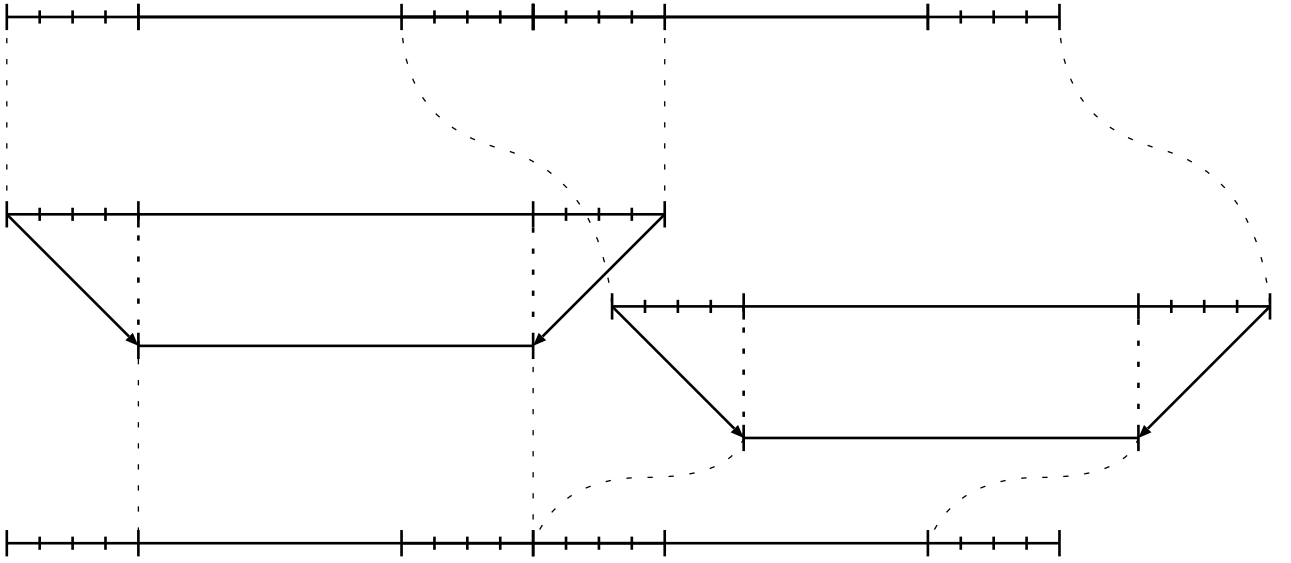
「2010年の計算」という名目でやった見積り:

- Glevel=14
- データ量 650TB
- 1ステップ当り 2P 演算くらい

(メモリ量、演算量ともかなり適当な見積り)
→ 1バイト当り 3 演算くらい。
データ転送速度で性能が決まる。
2GB/s のGRAPE-DR なら 6 Gflops しかでない (ピークの 1%以下)
V-GRAPE にしても良くならない

改善する方法はあるか？

陽解法であることを利用: 1度転送したら複数ステップ時間積分することを考える



- **オーバーラップがある領域に分割**
- **順番に、領域を V-GRAPE で計算。最終的にアップデートされるのはオーバーラップがないもの。**
- **全部更新したら次のステップに。**

こんなスキームを本当に書くのか？

- 原理的には可能
- LINPACK なんかではこの手のことを一杯やっている
- アプリケーション書く人がこんなことまでするのか？という気もする

とはいえ、

- おそらく陽解法なら V-GRAPE でなくてもこういうことを考えないと近い将来に性能でなくなるのではないか？

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

高い計算機と安い計算機を比べると

1975:

Cray-1	100Mflops	10M$
PDP-11/70	10kflops?	50K$?

Cray-1 50倍お得

1985:

Cray XMP	1Gflops	10M$
PC-AT	30kflops?	5K$

XMP 20倍お得

1995:

VPP-500	100Gflops	30M$?
Dec Alpha	300Mflops	30K$

VPP 3倍損

2005:

SX-8	10TF	50M$?
Intel PD	12 Gflops	1K$

SX-8 60 倍損

30年間で 3000倍変わった

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 — これはまあ説明するまでもない？

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
2007 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/ 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。
- システムの価格は演算速度ではなくメモリバンド幅で決まっている。

同じお金でより大規模な計算をしたいと思うなら、演算性能あたりの価格が安いシステムである程度の効率がでるようなアルゴリズム、実装を考えるべき

別のアプローチは？

- GRAPE-DR を止めにして GPGPU を使うとどうか？
- 計算量が増えるけれどそれに見合ったいいことがある方法はないか？

GPGPU

グラフィックカードを計算に使う。

- **最新の 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-7 の3倍)**
- **粒子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 400Gflops くらい出るかも。**
- **カードは安い (8万円くらい)**

計算量を増やす

高精度のスキームならいいことがあるか？

物事をポジティブに解釈すると:
通信リミットで性能が決まっている
＝ 複雑な計算をしても遅くならない

そういう需要があるなら、、、

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商品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。
- V-GRAPE ではコストを下げ、アプリケーションを増やすためオンチップメモリを増やす。
- 気候モデルには
  - ルジャンドル変換するなら GRAPE-DR でも
  - 低次陽解法なら V-GRAPE で頑張れば

  使えるかもしれない。

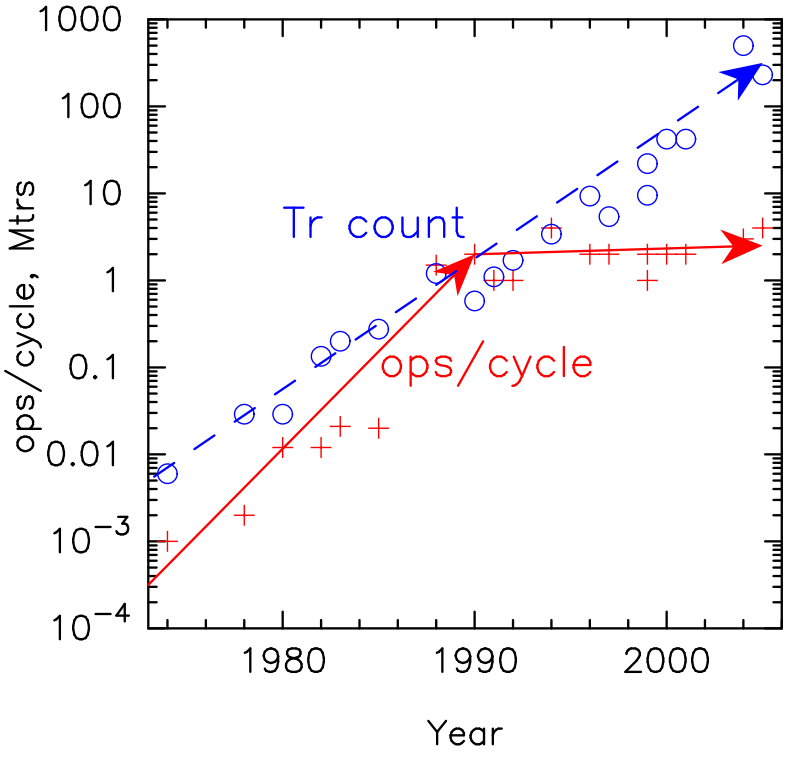
予備資料

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

ソフトウェアの継承性

カーネルルーチンだけに限る:

- それ以外の部分は計算機に依存しない
- その部分だけ移植すればよい
- 実はもっと汎用的な計算機でもその部分だけチューニングすればよい

アプリケーションのうち、マシン依存で変更するべき部分を特定することと同等

普通の並列計算機と何が違うか？

並列計算機の古典的な分類 (M. Flynn)
SISD/SIMD/MISD/MIMD

同時に処理される

- **命令列が一つ** (SI) **か複数** (MI) **か**
- **データ列が一つ** (SD) **か複数** (MD) **か**

この分類に入れるなら SIMD**。過去の** SIMD **機とは色々違う。**

- **上のレベル** (**並列ホスト**) **では** MIMD
- **接続ネットワークを相互作用計算に最適化**

接続ネットワーク

何故接続ネットワークが問題か？

SIMD 並列計算機を GRAPE として使う (粒子間相互作用の計算に使う) ことを考える。
単純な使いかた: 1000 個プロセッサがあれば、1000 個の粒子への力を計算。
利点:

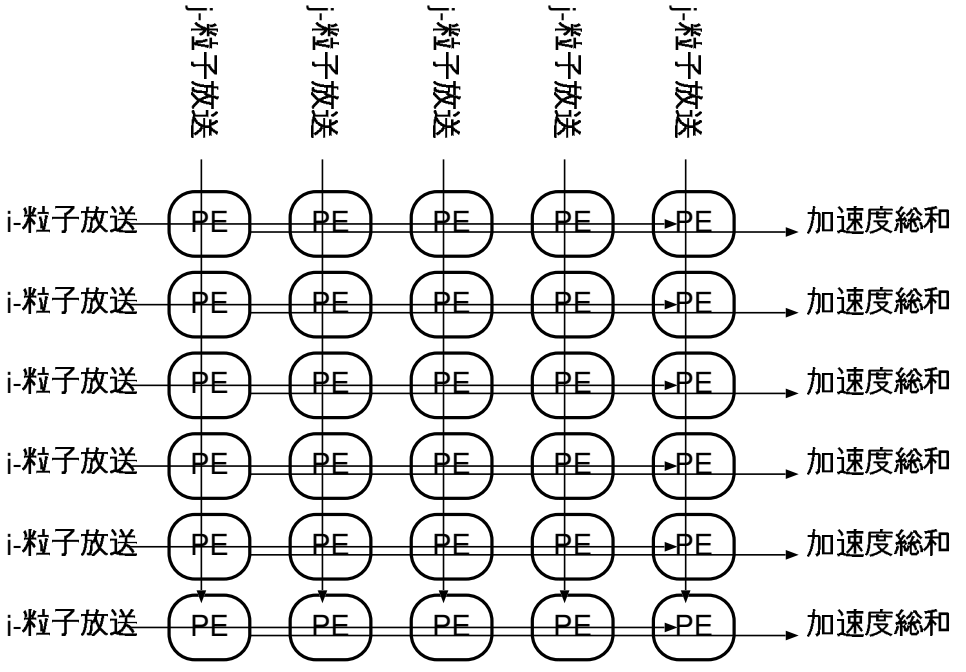
- **単純なネットワークでいい (放送とランダムアクセス)**
- **メモリもプロセッサあたり少しだけあればいい**

問題点:

- **独立時間刻みでは 1000 は多すぎる (複数チップ/ホストでもっと悪くなる)**
- **チップの高速動作が困難になる。**

みかけ上の並列度を下げる

複数のプロセッサ (PE) が同じ粒子への別の粒子からの力を計算、後で合計 (j-並列)



- **合計はチップ内でする必要あり (GRAPE-6 でボード上でやっているのと同じ)**
- **2 種類の「放送」が必要:論理的にプロセッサは 2 次元配列、縦、横両方の放送**

実際のチップ内ネットワーク

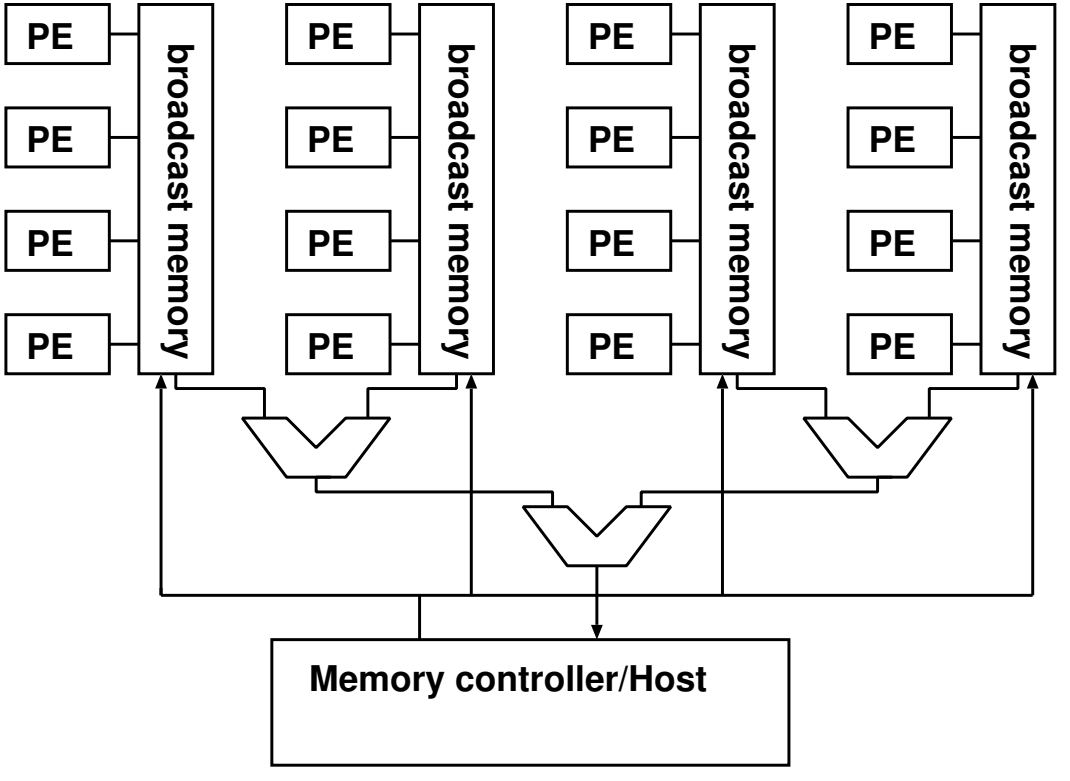
PE は放送メモリとだけ接続

放送メモリからそれにつながった PE には

- 放送
- ランダム読み書き

チップ外ポートから放送メモリには

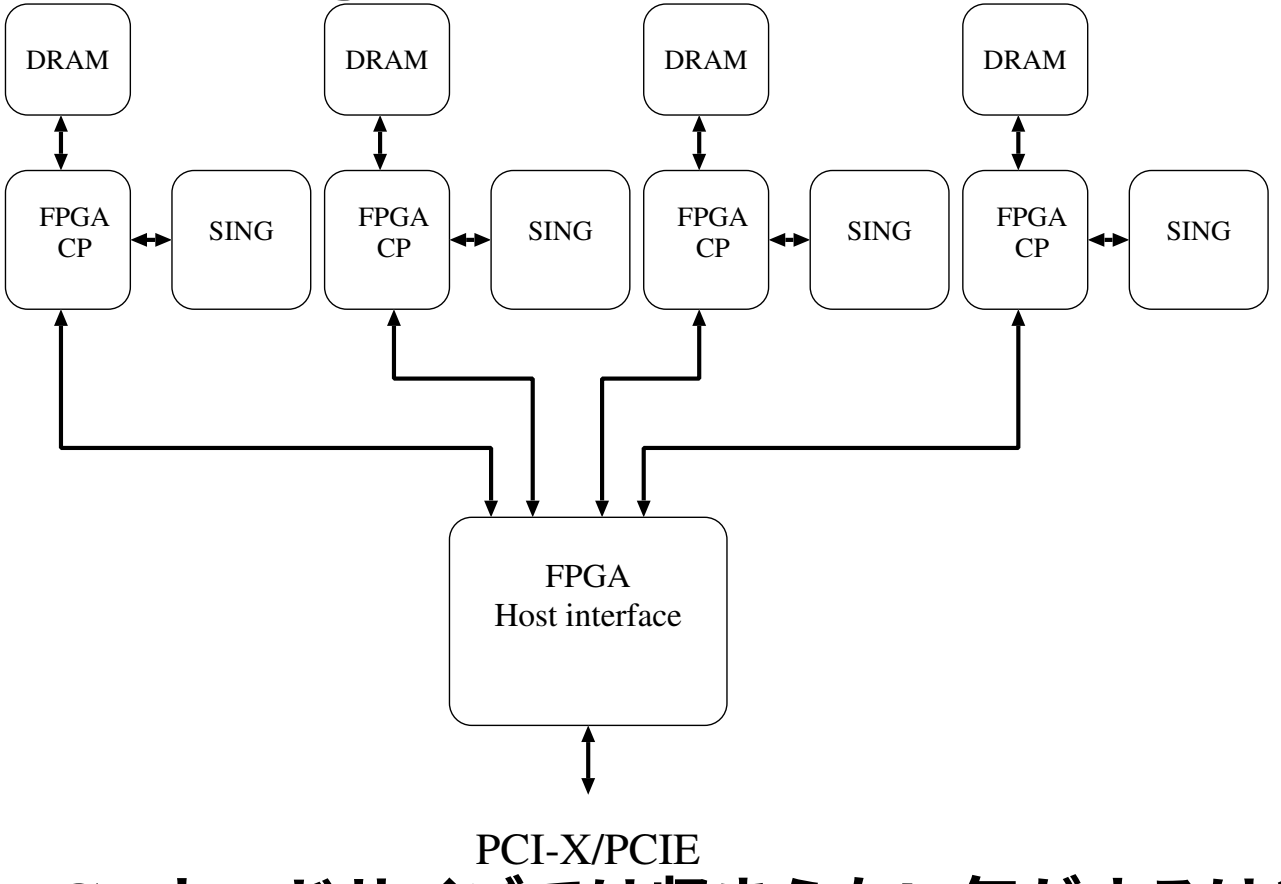
- 放送
- 縮約 (総和など) しながら読み出し
- ランダム読み書き



これで必要なことは全てできる。

ハードウェアのイメージ

SING: Sing Is Not GRAPE、チップの開発コードネーム



PCI カードサイズでは収まらない気がするけど、、、

どうやってプログラムするか？という話

- 今までの GRAPE とは全然違う: プログラムを書く必要がある
- 古典的な SIMD マシンともプログラミングモデルが違う
  - GDR の部分は「付加プロセッサ」:そこでプログラム全体が走るわけではない
  - 通信モデルが違う
  - 制御プロセッサはハードウェアレベルで変更可能 (FPGA で実装)

制御プロセッサの中身を決める = プログラミングモデルを決める

アセンブラ定義の概要

- 放送メモリ、 PE ローカルメモリに変数を置くことができる
- PE レジスタはアドレスで直接指定。(変数を置けるように変えるかも)
- ベクトル変数とスカラー変数がある
- 並列に実行する命令は明示的に指定する。

アセンブラの例

単純な重力計算の例

```
var vector long xi      hlt  flt64to72
var vector long yi      hlt  flt64to72
var vector long zi      hlt  flt64to72
var vector short idxi   hlt  fix32to36ru
bvar long xj            elt  flt64to72
bvar long yj            elt  flt64to72
bvar long zj            elt  flt64to72
bvar long vxj xj
bvar short mj         elt  flt64to36
bvar short eps2       elt  flt64to36
bvar short idxj       elt  fix32to36ru
var short  lmj
var short  leps2
var short  lidxj
var vector long accx rrn  flt72to64 fadd
var vector long accy rrn  flt72to64 fadd
var vector long accz rrn  flt72to64 fadd
var vector long pot  rrn  flt72to64 fadd
```

ここまでで変数宣言。 hlt, elt, rrn は通信タイプ。そのあとのは変換タイプ

アセンブラの例(続き)

```
loop initialization
vlen 4
uxor $t $t $t
upassa $ti $ti $lr40v
upassa $t $t   $lr48v
upassa $t $t   $lr56v
upassa $t $t   pot
loop body
vlen 3
bm vxj $lr0v
vlen 1
bm mj lmj
bm eps2 leps2
bm idxj lidxj
```

初期化(ループ前に実行)とループ本体の一部(放送メモリからの転送)

アセンブラの例(続き)

```
vlen 4
nop
upassa idxi  idxi  $t
uxor  $ti lidxj $t
moi 2
ulnot $ti $ti $t  # mreg 1 indicates i != j
moi 0
nop
fsub $lr0 xi $r6v  $t
fsub $lr2 yi $r10v ;  fmul $ti $ti $t
fsub $lr4 zi $r14v
fmul $r10v $r10v $r18v ; fadd $t leps2 $t
fmul $r14v $r14v ;  fadd $fb $ti  $t
fadd $fb $ti  $r18v $t # rsq is now in r18 t, dx, dy,dz are in
```

自己相互作用のチェック、座標の引き算と距離の2乗の計算

アセンブラの例(続き)

```
ulsr $ti      il"60"  $t $lr22v
ulsr $ti      il"1"   $t
uadd $ti      $lr22v  $t
usub hl"9fd" $ti      $t        # $lr8v は指数の 1.5 倍
ulsl $ti      il"60"  $lr30v
moi 1
uand il"1" $lr22v
moi 0
uand $r18v h"000ffffff" $t
uor  $ti h"3ff000000" $t
fmul $ti f"0.57" $t
fsub f"1.57" $ti $t
mi 1
fmul f"1.414" $ti $t
mi 0
nop
fmul $t $lr30v $t $r22v # Here the result is the initial guess
```

r^{-3} の初期推定値。これは手抜きな 1 次式

アセンブラの例(続き)

```
 fmul $r18v $r18v $r26v $t
 fmul $r18v $ti   $r26v $t
 fmul $ti f"0.5" $r26v # r26v is a**3/2
 fmul $r22v $r22v $t
 fmul $ti $r26v $t
 fsub f"1.5" $ti $t
 fmul $r22v $ti $t $r22v
 fmul $ti $ti $t
 fmul $ti $r26v $t
(反復ちょっと省略)
 fsub f"1.5" $ti $t
 fmul $r22v $ti $t $r22v
 fmul $ti $ti $t
 fmul $ti $r26v $t
 fsub f"0.5" $ti $t
 fmul $r22v $ti $t
 fadd $r22v $ti $t
 fmul lmj $ti $t $r22v
```

ニュートン反復

アセンブラの例(続き)

```
mi 2
fmul $r6v $ti ; upassa pot pot $lr0v
fmul $r10v $t ;  fadd $fb $lr40v  $lr40v accx
fmul $r14v $t ; fadd $fb $lr48v  $lr48v accy
fmul $r18v $t ; fadd $fb $lr56v $lr56v accz
fadd $fb $lr0v  pot
```

ポテンシャルと加速度の積算
この記述から、インターフェース関数とシミュレータを動かすのに必要なデータを生成する。

インターフェース関数

中身はアセンブラ記述から自動生成される。

```
int SING_send_j_particle(struct grape_j_particle_struct *jp,
                         int index_in_EM);
int SING_send_i_particle(struct grape_i_particle_struct *ip,
                         int n);
int SING_get_result(struct grape_result_struct *rp);
void SING_grape_init();
int SING_grape_run(int n);
```

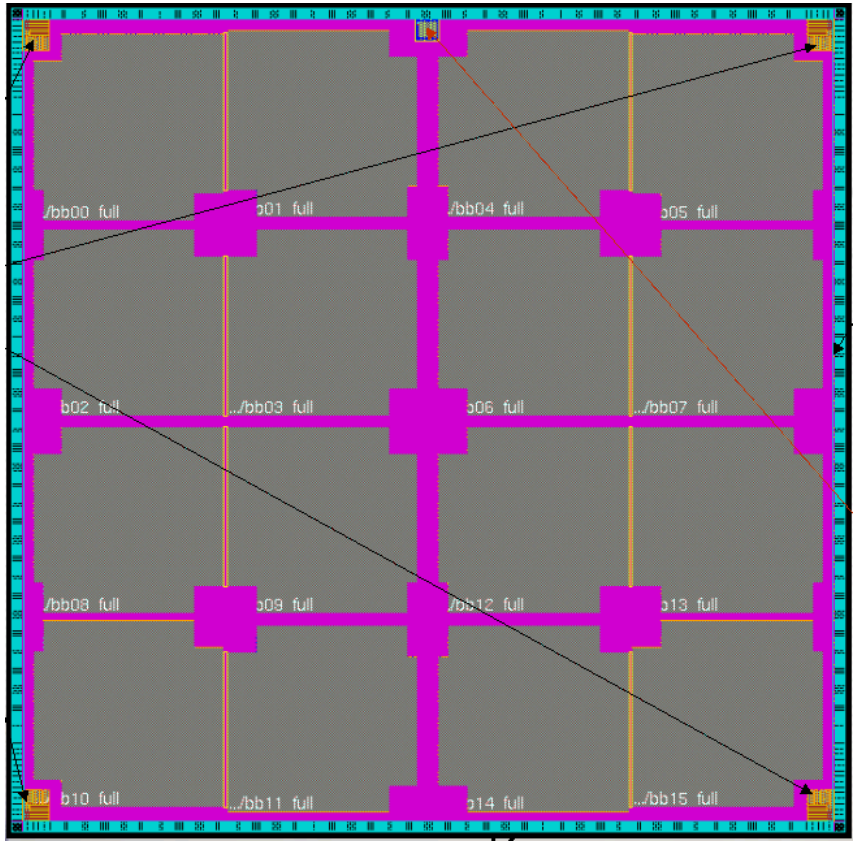
これにもう一層かぶせれば GRAPE-3/5 互換インターフェースはできる。

インターフェース構造体

これももちろん自動生成される。

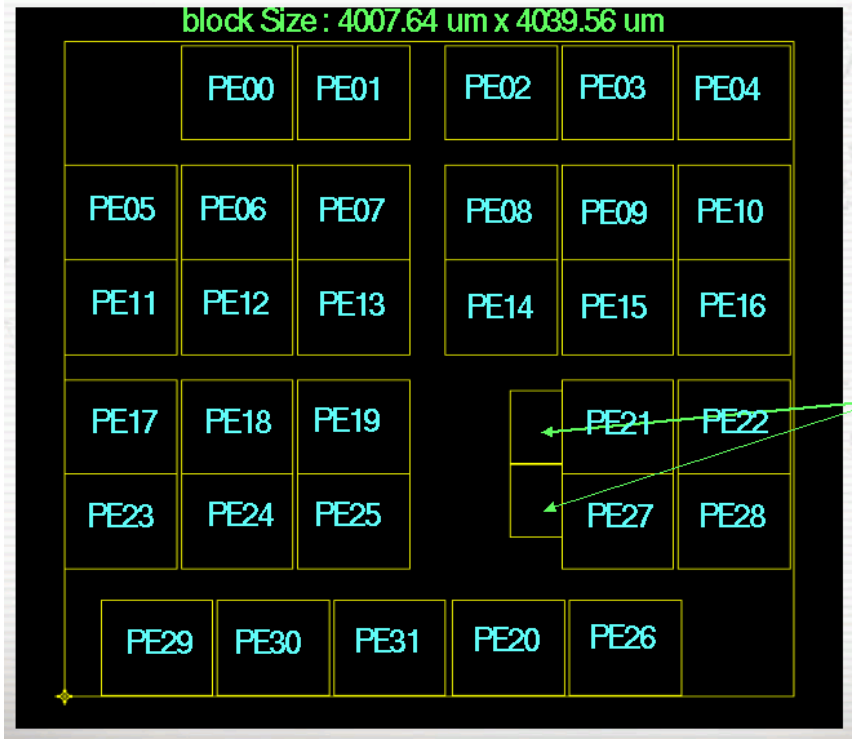
```
struct grape_j_particle_struct{
  double xj;
  double yj;
  double zj;
  double mj;
  double eps2;
  UINT32 idxj;
};
struct grape_i_particle_struct{
  double xi;
  double yi;
  double zi;
  UINT32 idxi;
};
struct grape_result_struct{
  double accx;
  double accy;
  double accz;
  double pot;
};
```

チップ全体フロアプラン



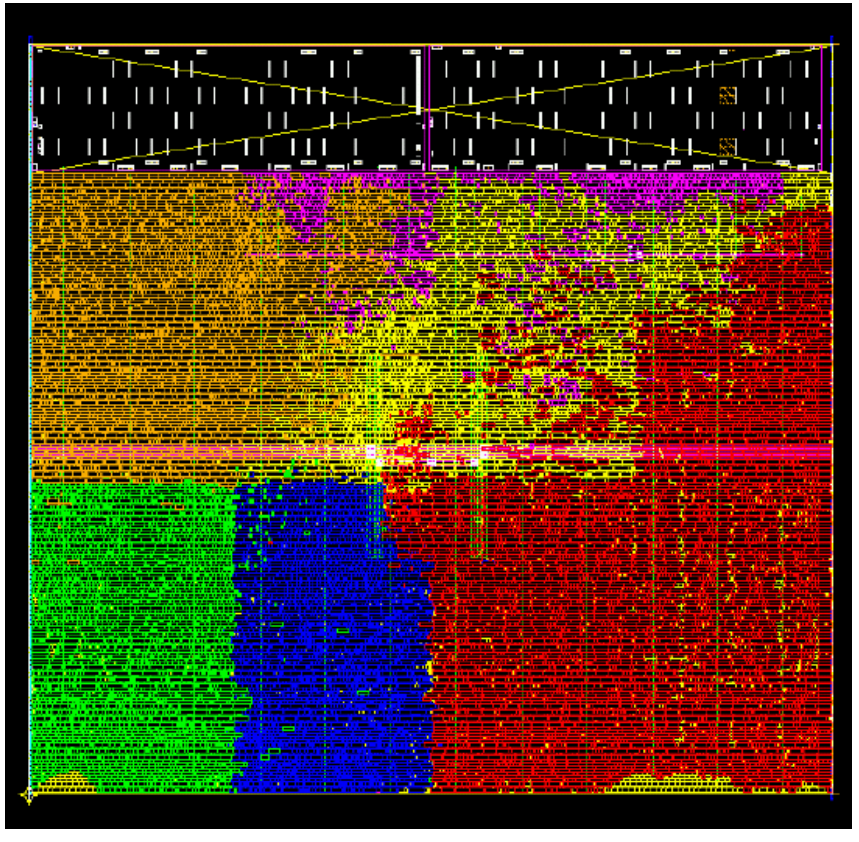
- 特におかしいとこなし
- サイズは大きい。
  17mm角

放送ブロックフロアプラン



- 特におかしいとこなし

PE フロアプラン



Module Name	Color
grf0	red
fmul	orange
fadd	green
alu	blue
lm	magenta
Others	yellow