# GRAPE-DR プロジェクト
# — 次世代 GRAPE の概要

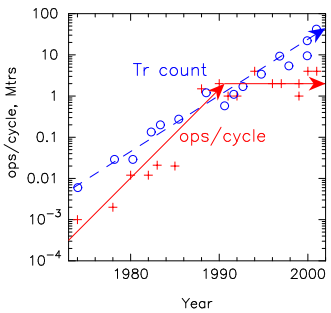
牧野淳一郎
(東大天文)

- 動機
- アーキテクチャ
- ターゲットアプリケーション
- 予算
- 開発課題

## 科学技術用高速計算機の現状

- 「汎用性」の低下
  アーキテクチャの複雑化
  - メモリ階層
  - スカラー → ベクトル → ベクトル並列 (分散メモリ)
    「あらゆる応用プログラムで万能に速い」計算機はなくなった
- 半導体利用効率の低下
  - チップあたりトランジスタ数は 10 年で 100 倍
  - チップあたりの演算器数は過去 15 年間で 2 倍程度

## 半導体利用効率の低下




マイクロプロセッサのトランジスタ数と 1 クロックあたりの浮動小数点演算数

演算数は 1 に達したところで進歩がほとんど止まっている

## 進歩が止まった理由

- ハード的な問題
  - memory wall: プロセッサ — メモリ間のデータ転送速度が足りないので、演算器を増やしても性能が上がらない
  - スーパースカラーでは実行ユニット数を増やすとハードウェアが急速に大きくなる。
- ソフト的な問題 — 命令レベルの並列性には限界

科学技術計算の多くは高い並列性を持つ — 「本来は」並列性の限界はあまり問題ではない

## 汎用プロセッサの進化の方向

- Simultaneous Multithreading
  - 基本的には、1 プロセッサを時分割して複数のプロセッサに見せる
  - キャッシュミスのペナルティをある程度隠蔽できる
  - メモリのバンド幅不足は解決しない
- On-chip multiprocessor
  - 2-4 個のプロセッサを 1 チップに載せる
  - メモリのバンド幅、レイテンシどちらもより大変になる

## 専用計算機

アプリケーションに専用化したプロセッサ:多数の演算器を使える

商用で成功しているもの

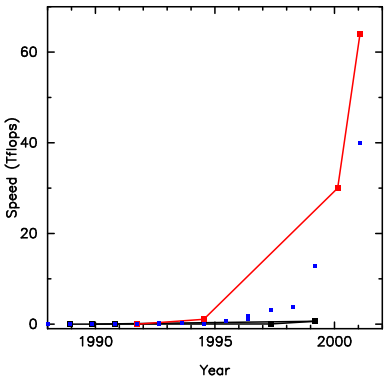
- PC 用グラフィックカード
- Sony PS2 などのゲーム機

演算器は 10 個程度

科学技術計算用専用計算機

- 多体問題専用計算機
  - 粒子間相互作用計算に特化した演算パイプライン
  - 400 個程度の演算器を 1 チップに集積 (GRAPE-6, MDM)

## GRAPE の進化




赤: GRAPE

青: 汎用ベクトル/並列

速度はほぼ同等。
開発費は 2 桁違う。

## 現状の整理

- ベクトル並列 (地球シミュレータ)
  - 計算/メモリ転送速度: 2 程度
  - 高い実効性能・ 高い価格
- スカラー並列 (ASCI、 PC クラスタ)
  - 計算/メモリ転送速度: 20 程度
  - 低い実効性能・そこそこの価格
- GRAPE 型専用計算機
  - 計算/メモリ転送速度: 200 程度
  - 高い実効性能・低い価格

## 専用計算機の「限界」

開発費の高騰

| | | |
|---|---|---|
| 1990 | $1\mu$m | 1500万円 |
| 1997 | $0.25\mu$m | 1億円 |
| 2004 | 90nm | 3億円以上？ |

ある程度広い応用を持つものでないと難しい

FPGA、「リコンフィギャラブルデバイス」は？

トランジスタ利用効率で大きく劣る

↓

データ語長が短い応用でないと高性能は難しい

## 別のアプローチはないか？

目標:

- 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

明らかな矛盾？

GRAPE の高性能: 多数の演算器を 1 チップに集積し、小さなメモリバンド幅で全演算器を有効に働かせたことによる

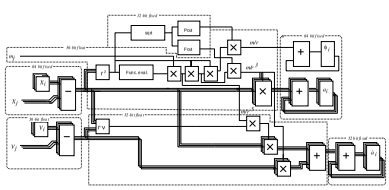
相互作用の計算式に専用化したパイプラインが必須というわけではない。

## 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

## GRAPE的専用プロセッサ



利点:

- シリコンの利用効率は極めて高い
- 動作クロックも上げやすい

欠点

- アプリケーション限られる。多種類作るのはリソースがかかり過ぎる

## 再構成可能プロセッサ

FPGA ベース

- 任意のロジックを実現可能
- 集積度、速度は大きなペナルティがある
- 精度が低くてもいい応用には向く

いわゆる「動的再構成可能プロセッサ」

- 8-32 ビットの単純な ALU を多数集積
- その間をプログラマブルな配線でつなぐ
- 集積度、速度はやはり大きなペナルティ

## パイプラインは必要？

パイプラインを使う (使える) 理由:

- 規則的で並列性の高い計算をする
- 入力データ一つに対して多数の計算をする
- 専用パイプラインの場合には配線リソース等が不要になる

## パイプライン (ベクトル) 処理と並列処理

ベクトル化できる処理 (プログラム) と並列化できる処理:
基本的は同じ。

パイプラインの代わりに単純な SIMD 並列処理ではいけないのか？

「教訓」: SIMD 超並列プロセッサは歴史上の存在

Goodyear MPP
ICL DAP
CM-1/2
MasPar MP-1

## 何故 SIMD 超並列は消えたか？

基本的な理由: memory wall
多数のプロセッサを LSI に集積 → 並列動作するには (普通のプログラミングモデルでは) プロセッサ数にみあうだけのメモリバンド幅が必要。

現在ではこれは全く不可能。

例: 2 億トランジスタ → 64 ビット乗算器が 1000 個。
データを供給するには、コアの動作クロックと I/O の動作クロックが同じでも 19 万本の配線 が必要

しかし、 GRAPE ではそんな問題は起きていない

GRAPE と同じようなことをやらせるなら I/O 速度への要求は何桁も下がる。

## GRAPE における並列性

する計算:

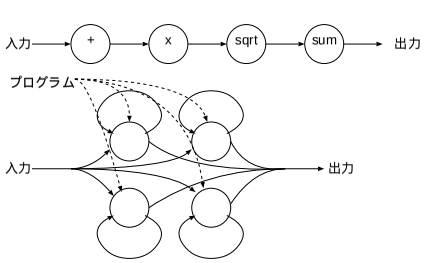
$$a_i = \sum_j f(r_i, r_j, m_j)$$

$i$ についても $j$ についても並列 ($j$ は総和が必要)

1. パイプライン演算 ($j$ 並列)
2. 物理/仮想マルチパイプライン ($i$ 並列)
3. 複数チップ ($i$ 並列/$j$ 並列)

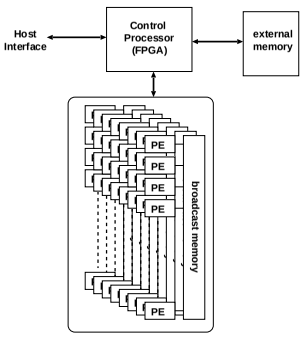
「パイプラインであること」はあまり本質的ではない？

## パイプライン処理と SIMD 並列処理




パイプラインでできることは SIMD 並列でもできる (縮約演算は追加ハードウェア必要)
専用パイプラインはいろいろメリットある
再構成可能だと、、、

## GRAPE-DR の アーキテクチャ




- 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ユニット (BU) 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

## 「GRAPE として」使う

最も単純には

- 全ての PE が、自分の粒子への、同じ粒子からの力を計算
- 力を及ぼすほうの粒子データは外部メモリから供給

現実問題としては

- 違うブロードキャストユニットの同じ位置の PE には同じ粒子データを書く
- 各ブロードキャスト
- 力を及ぼすほうの粒子データはブロードキャストユニット毎に違うものにする。

つまり、 ブロードキャストユニット内で $i$ 並列、ブロードキャストユニット間で $j$ 並列とする。

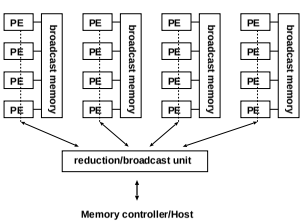
## 何故こういうややこしいことをするか？

消極的理由: GRAPE では予算がとれない。

積極的 (?) 理由:

- 応用を広げたい
- 新しいことをやってみたい

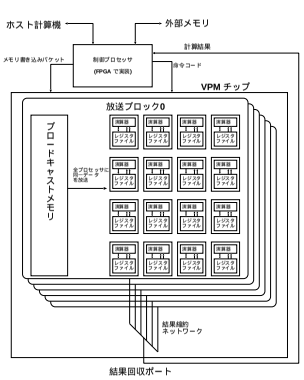
## チップ内のネットワーク




- ブロードキャストユニットにメモリポートからデータを放送とランダム書き込みの両方
- ブロードキャストユニットからのデータをリダクション (合計とか) しながらホストに返すネットワーク。

の 2 つが必要。

## ブロードキャストユニットの構成




- 命令コード、ブロードキャストメモリのアドレス等は外から供給される。
- 各 PE は同じデータを受け取る。

## PE の詳細

演算:(最低限) 以下をサポート

- 倍精度浮動小数点加減算
- 倍精度固定 (ブロック) 小数点加減算
- 単精度浮動小数点乗算。倍精度はマルチサイクルで
- 区分多項式近似・スケーリングに必要なビット操作
- 条件付き実行

メモリオペレーション
ローカル共有メモリ: ブロードキャスト、シリアルな読出し、書き込み
汎用ポート: リダクション演算、放送

## メモリコントローラ、シーケンサ

なるべくチップは設計を簡単にしたい

シーケンサは設計ミスがもっとも起きやすい場所
メモリコントローラはチップに作り込みたくない

面倒なものは全部 FPGA に任せる。
問題:命令ストリームを十分高速に渡せるか？
手抜きな対応: ベクトル命令なら問題なし。

## ターゲットアプリケーション

今までの GRAPE 的なものの置き換え

- 粒子法: 重力、分子間力、流体的相互作用、、、
- 境界要素法: ポアソン方程式、ヘルムホルツ問題、、、
- ルジャンドル展開による極座標流体計算

プロジェクトとしての政治的な目標

- 密行列ソルバ (内積型を利用) — Linpack で 500 Tflops 以上

もうちょっとまともな目標

- 分子軌道法での 2 電子積分
- 密行列の対角化、直交化

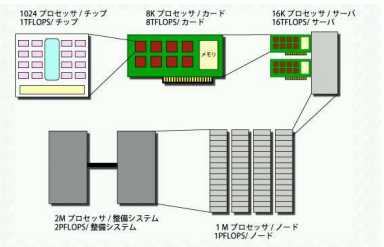
## 予算の名目

平成１６年度科学技術振興調整費新規課題
「分散共有型研究データ利用基盤の整備」

総額 3 億 $(-\alpha)\times$ 5 年

## 最終システムのイメージ

2007 年度に基本的には完成 (2008 年度に増強)
ピーク性能 2Tflops (単精度)
プロセッサチップ 2048 個。
プロセッサボード当り 4 チップ (PCI-X/Express)
ホストは 512 ノードの PC クラスタ




## プロセッサチップのイメージ

- 1024 PE
- 32 放送ブロック (各 32PE)
- 500 MHz 動作
- 1 Tflops ピーク (内積演算の時に)

## 開発スケジュール

(現実的には)

- 夏までに PE の詳細決定、シミュレータでの評価。
- それから 1 年くらいでテープアウトしたい。
- それ以上は先のことは今考えても、、、

## GRAPE に比べるとどれくらい損か？

GRAPE-6: 200万ゲート、 400 演算 = 5K ゲート/演算
PE の大きさ (推定):

| ユニット | サイズ |
|---|---|
| fadd | 7K |
| fmul | 8K |
| register file | 10K |
| 合計 | 25K |

場合によっては乗算・加算同時実行可能だとすると、合計では 12.5K ゲート/演算。純粋な GRAPE に比べると 2.5倍損
但し、レジスタファイルが 10K ゲート程度で十分かどうかは微妙。
共有メモリやインターフェース回路はサイズとしては無視できる。

## 類似のアプローチとの比較 (1)

- SIMD 超並列計算機 (Goodyear MPP, CM-1/2)
  - 外部メモリへのバンド幅、通信ネットワークでプロセッサ数が制限
  - プロセッサがメモリを共有することでプロセッサ数の制限を回避。応用範囲には制限。
- FPGA
  - トランジスタ利用率が低い
  - ソフトウェア開発が困難
  - どちらも回避

## 類似のアプローチとの比較 (2)

再構成可能計算機

- DSP ブロック入り FPGA
  - ゲート効率 FPGA よりはちょっといい？
- DAP/DNA, DRP
  - ゲート効率悪い
  - 動作速度遅い

他のアプローチとの基本的な違い:

チップ内にスイッチングネットワークをもたない

- (多少) できないことも発生
- デザインの単純化 → 高集積、高速動作

## ソフトウェア

PE のプログラムは、基本的にシーケンシャル
(SIMD アプローチの利点)

- アセンブリ言語
- 単純なコンパイラ

上位のソフトウェア

- 複数ホスト+超並列ボード上の並列化
- ホスト計算機/超並列ボードのタスク割り当て

## まとめ

- 次世代 GRAPE は予算がついた。
- 作るものは基本的には SIMD 動作する1チップ超並列プロセッサ。これで従来の GRAPE、MD-GRAPE の他、密度行列計算も可能にする。
- GRAPE 的なメモリ階層をもたせることで、 SIMD 計算機の難点を解消して高い価格性能比を実現する。

## 将来展望

- 多対多の相互作用向け SIMD 超並列計算機という新しい概念
  - 専用計算機並の高いコストパフォーマンス
  - 特定の物理系に縛られない広い適用範囲
  - 将来の半導体技術の発展を有効に利用可能
- ベクトル並列、スカラー並列と相補的
- compute-intensive な問題にブレークスルーをもたらす

## COTS アプローチ

Intel Pentium 4 等の “off the shelf” プロセッサ

- 大量生産 → 低コスト
- 大量生産 → 莫大な開発費 → 高い性能

単一スカラープロセッサで COTS の性能を超えるのは非常に難しい
性能が memory limited — プロセッサを別のものにしても速くならない

COTS の限界: backward compatibility.
VLIW すら難しい (Itanium の失敗)。

## チップ内部構成

- 共有メモリユニットの数 (共有メモリユニットあたりの PE 数)
- PE の構成
  - 演算方式 (語長、データ形式、関数計算のサポート)
  - 制御方式 (ベクトル命令/スカラー命令)
  - レジスタ構成
- メモリインターフェース方式
- 汎用プロセッサとのインターフェース