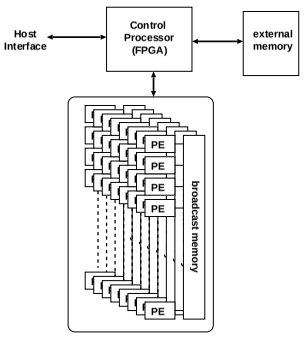
GRAPE-DR 開発状況

牧野淳一郎

概要

- アーキテクチャのおさらい
- チップ開発状況
- ソフトウェア開発状況
- まとめ

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)
- PE はプログラムによって並列動作する
- 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

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:

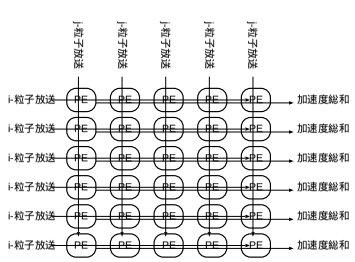
- 単純なネットワークでいい (放送とランダムアクセス)
- メモリもプロセッサあたり少しだけあればいい

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

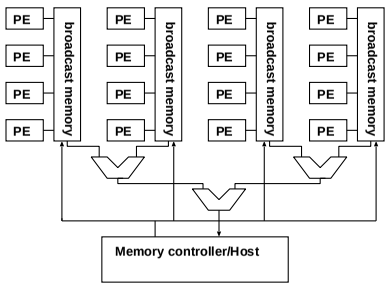
PE は放送メモリとだけ接続

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

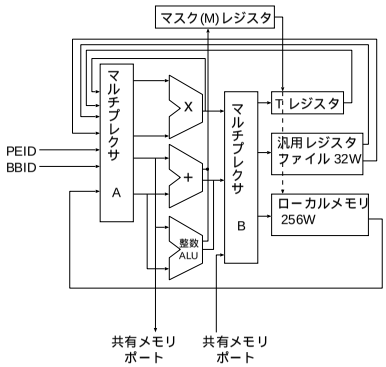
チップ外ポートから放送メモリには

- 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。

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。

PEの詳細

データ形式

単精度浮動小数点: 36 ビット (符号1、指数11、仮数24)
倍精度浮動小数点: 72 ビット (符号1、指数11、仮数60)
36/72 ビット固定小数点数

演算命令

乗算は単精度のみ (倍精度のための部分積をサポート) 倍精度乗算を 2 サイクルでするために 25×50 ビットの乗算器
整数演算、加減算は倍精度のみ (メモリ/レジスタからの読出し/格納時に単・倍変換ができる)
特殊な浮動小数点命令: 仮数を正規化しないまま演算を続ける。これにより、演算順序によらないで結果が同じになることを保証する (GRAPE-6 の積算と同様)

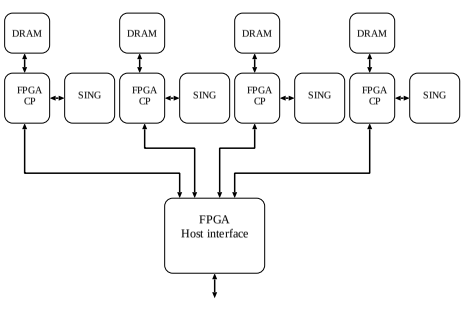
PEの詳細 (続き)

- パイプラインは8ステージ。
- 基本命令は4データに対するベクトル命令。4サイクルに1回しか命令ははいらない。
- T レジスタについてのみ直前の命令の実行結果を利用可能。
- T レジスタはアドレスレジスタになる (間接アクセスが可能)

サポートする命令等は基本的にはSIMD 計算機、例えば CM-2, MasPar MP-1 なんかとあまり変わらない。但し、PE がはるかに強力になっている。

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、、、

開発状況

- チップ命令セット一応決定
- チップシミュレーター一応動作
- VHDL 記述完成
- アプリケーションでの検証
  - 単純 GRAPE 完了
  - GRAPE-6 相当 完了 (ネイバーリストなし)
  - SPH これから
  - 行列乗算 これから

ファブ、製造プロセス内定、レイアウト作業スタート (4月)

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

GRAPE 的モデル

1. 初期設定をする。定数を書くとか。
2. ホストが j 粒子をボードに送る。これは FPGA につながった外部メモリに格納される。
3. ホストが i 粒子を送る。これは PE メモリに格納される。
4. 力の計算を始める前の準備をする。積算レジスタのクリア等
5. 最初の j 粒子を外部メモリから 放送メモリに転送
6. 放送メモリから PE メモリに j 粒子を転送し、実際の相互作用計算をする。 これと並行して次の j 粒子を放送メモリに転送する。
7. 上を指定された数繰り返す。
8. 求まった結果を縮約しながらホストに送る。
9. 力を計算するべき i 粒子が残っていれば iの転送に戻る

GRAPE 的モデルの実現

安直な方法:

- 制御プロセッサ FPGA の中に各オペレーションに対応するシーケンサを入れる
- それぞれがホストからのコマンドで起動される

GRAPE の制御部分とほとんど同じ。とりあえずこの方向で考える。
で、使う人が書くものは何か？

使う人が書くもの

最低限必要なもの: 相互作用の計算法自体。
とりあえず提供するもの: アセンブラ。
通信制御は全て変数に対して「どう転送されるか」を指定する形。

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  6,10,14
```

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;
};
```

アセンブラの現状

- このサンプルはレジスタは番号でだけアクセス
- 現在はレジスタに名前はつけられる
- まあ、なれるとプログラム書ける
- とはいえ、かなり大変。最適化コンパイラ作る人募集中。

ニュートン反復における丸めの問題

普通にニュートン反復で r^{-3} を計算すると丸めにバイアスがでる、という話。
ニュートン法の近似式は、 $a = r^2$ として、

$$x_1 = \frac{x(3 - a^3x^2)}{2}$$

ほぼ収束している場合: $a^3x^2 \sim 1$, $3 - a^3x^2 \sim 2$
x が左側 $\rightarrow$ $a^3x^2 < 1$ $\rightarrow$ 括弧内は 2 より小さい
x が右側 $\rightarrow$ $a^3x^2 > 1$ $\rightarrow$ 括弧内は 2 より大きい
この2つで丸めが違うので、真の値からのずれが右側の時に大きく なる。

対応

以下の変形をする

$$\Delta x = \frac{x(1 - a^3x^2)}{2}$$
$$x_{new} = x + \Delta x$$

つまり、 1 とか 2 という数字が発生しないようにする。
(a^3x^2 は相変わらず 1 だけど、、、)
とりあえずバイアスは測定可能な値ではなくなった。

まとめ

- GRAPE-DR チップ設計は大体予定通り進行中。
- ファブは内定。予算には収まりそう。
- アセンブラが GRAPE 用 (粒子計算用) は動く。チップシミュレーションもできた。
- プロトタイプボード設計 ... 藤野様/福重様よろしくお願いします。