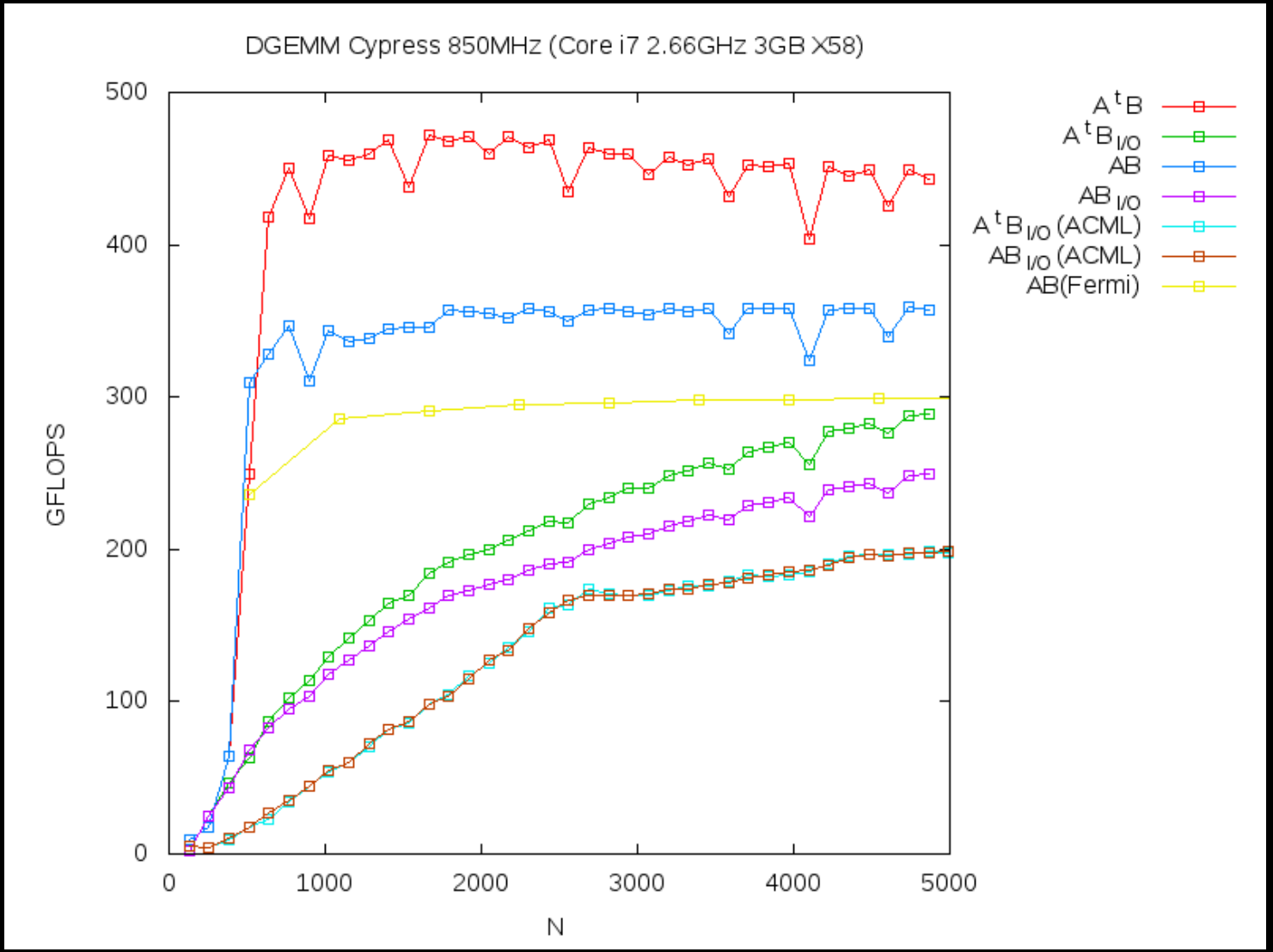
四倍精度演算専用プロセッサの開発と応用

中里直人(会津大学)
台坂博(一橋大学)
牧野淳一郎(国立天文台)
石川正, 湯浅富久子(KEK)

本題の前に別の仕事の宣伝

- 世界一高速なSGEMM/DGEMM/DDGEMM



概要

- 四倍精度演算とは?
- 四倍精度演算プロセッサのアーキテクチャ
- GRAPE-MPチップの開発
- GRAPE-MPシステムについて
- 今後の応用例

数値計算の現状

- 数値計算では倍精度が利用されている
  - 仮数部 53 bit, 指数部 11 bit, 符号部 1bit
  - 「たいてい」は倍精度で十分
    - 多くのアルゴリズムは倍精度であれば安定
  - プログラムが容易
  - 倍精度まではハードウエア実装
    - 半導体プロセスの進化にしたがって高速化してきた
    - 最新のCPU ~ 96 Gflop/s (12 cores)
    - 最新のGPU ~ 544 Gflop/s (320 cores)
  - ただし単精度以下でも十分な問題はある
    - 例：重力多体問題 (GRAPEの成功の理由)

高精度演算の実現

- 整数演算でエミュレーション
  - 2進数による浮動小数点演算をそのまま実装
  - 仮数部を整数配列で表現
  - 単純な実装では分岐命令が多くなる
- FP演算でエミュレーション
  - Knuth (1969), Dekkar (1971)
  - 数値をFP変数の和と考える

$$A = \sum a_i : (a_1 > a_2 > \ldots > a_n)$$

$$1.0000000004 = a + b$$

$$\Rightarrow a = 1.0, b = 0.0000000004$$

高精度演算ライブラリ

- 整数演算器を利用するエミュレーション
  - GNU Multiple Precision Arithmetic Library
    - C, C++, Fortran
  - 藤原 (2005-2009)
    - exflib : C++, Fortran 90/95, OpenMP
  - 濱口 (2007a,b)
    - HMLIB : Fortran
- 倍精度演算器を利用するエミュレーション
  - Shewchuk (1997) : Adaptive MP
  - Hida, Li, Bailey (2000,2001)
    - MPFUN, qd : C, C++, Fortran

行列用の高精度演算ライブラリ

- XBLAS (Liら)
  - http://www.netlib.org/xblas/
  - BLAS-1,2,3の一部を実装
  - M4マクロプロセッサによるコード生成
- mpack (中田さん@理研)
  - http://mplapack.sourceforge.net/
  - BLASとLAPACKの多くを実装
  - GMP, QDを利用
  - f2cとperlによるfortranからの変換
- メタプログラミングは重要

高精度演算の問題点

- 遅い！
  - 整数演算/浮動小数点エミュレーションどちらもCPUでは遅い。
    - マイクロベンチマークの結果

	加算秒数	乗算秒数	加算 lat.	乗算 lat.	加算性能	乗算性能	clock	MA
Core2	4.545	5.829	40.6	52.2	59	46	2.4	Core MA
Xeon	4.502	6.002	39.0	52.2	60	45	2.33	Core MA
Core i7	3.617	4.621	36.0	46.0	73	58	2.67	Nehalem
Opteron	3.830	4.049	31.3	33.1	70	66	2.2	K8

表 2　DD 加算と乗算の各種 x86 アーキテクチャCPU における演算性能: 性能のコラムの単位は MFLOPS であり, clock 周波数の単位は GHz である. 最後のコラムはマイクロアーキテクチャを示す.

    - Mpackによるベンチマーク (中田 2010)
      - DGEMM by DD(32桁)　　~ 141 Mflop/s = ~ 1.3%
      - DGEMM by GMP (308桁) ~ 14.4 Mflop/s

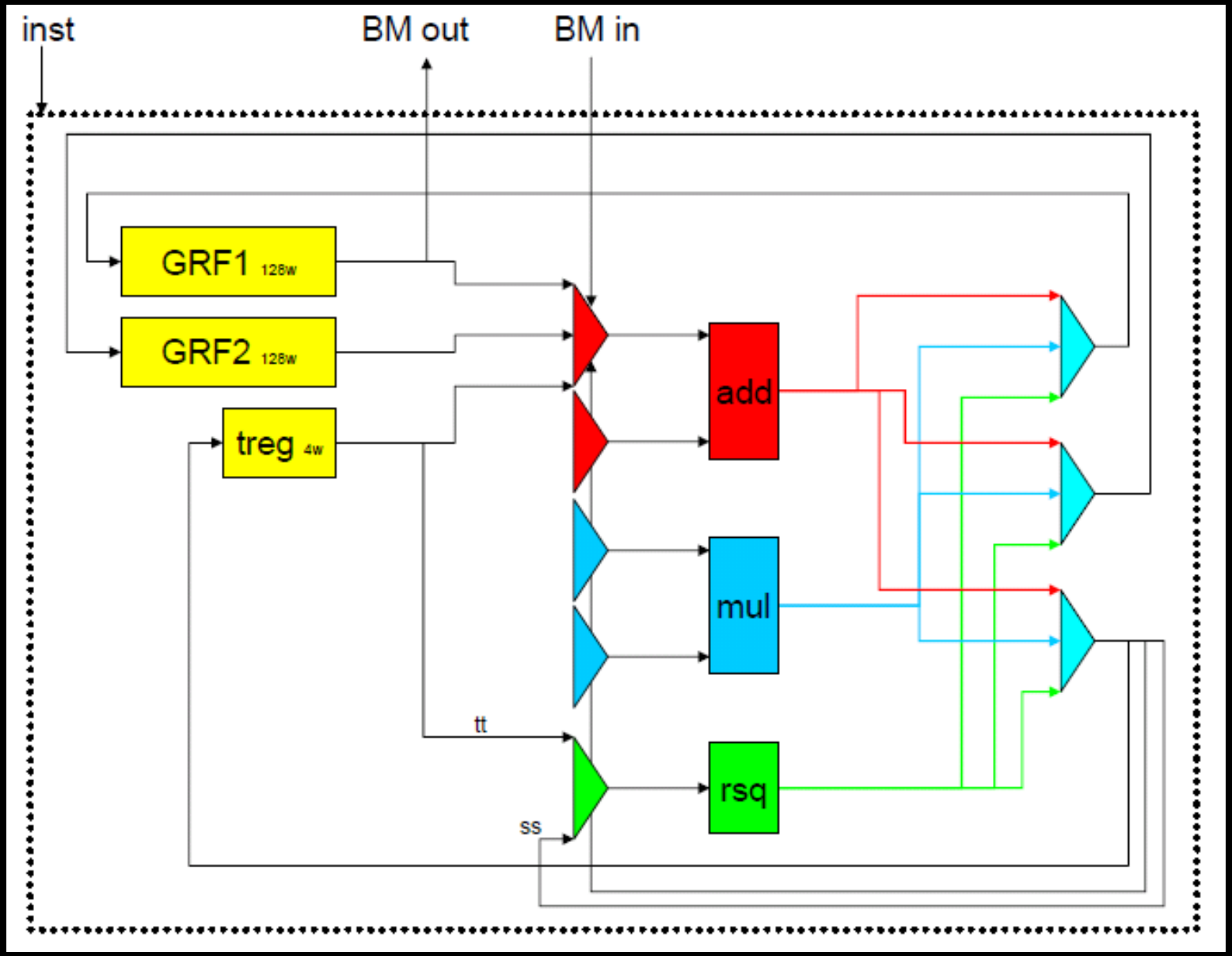
GRAPE-MPとは

- GRAPE-DRのアーキテクチャにもとづき、演算器を四倍精度とした専用プロセッサ
  - GRAPE-DRとは
    - HPC専用に拡張されたプログラマブルなGRAPE
    - SIMD型プロセッサ (512 PE)
    - 省メモリ帯域設計
    - 縮約専用回路
    - DGEMMや重力計算に向いた設計
  - GRAPE-MP
    - 128 bit浮動小数点演算に特化したプログラマブルSIMD型プロセッサ
    - 縮約回路なし。外部メモリなし。

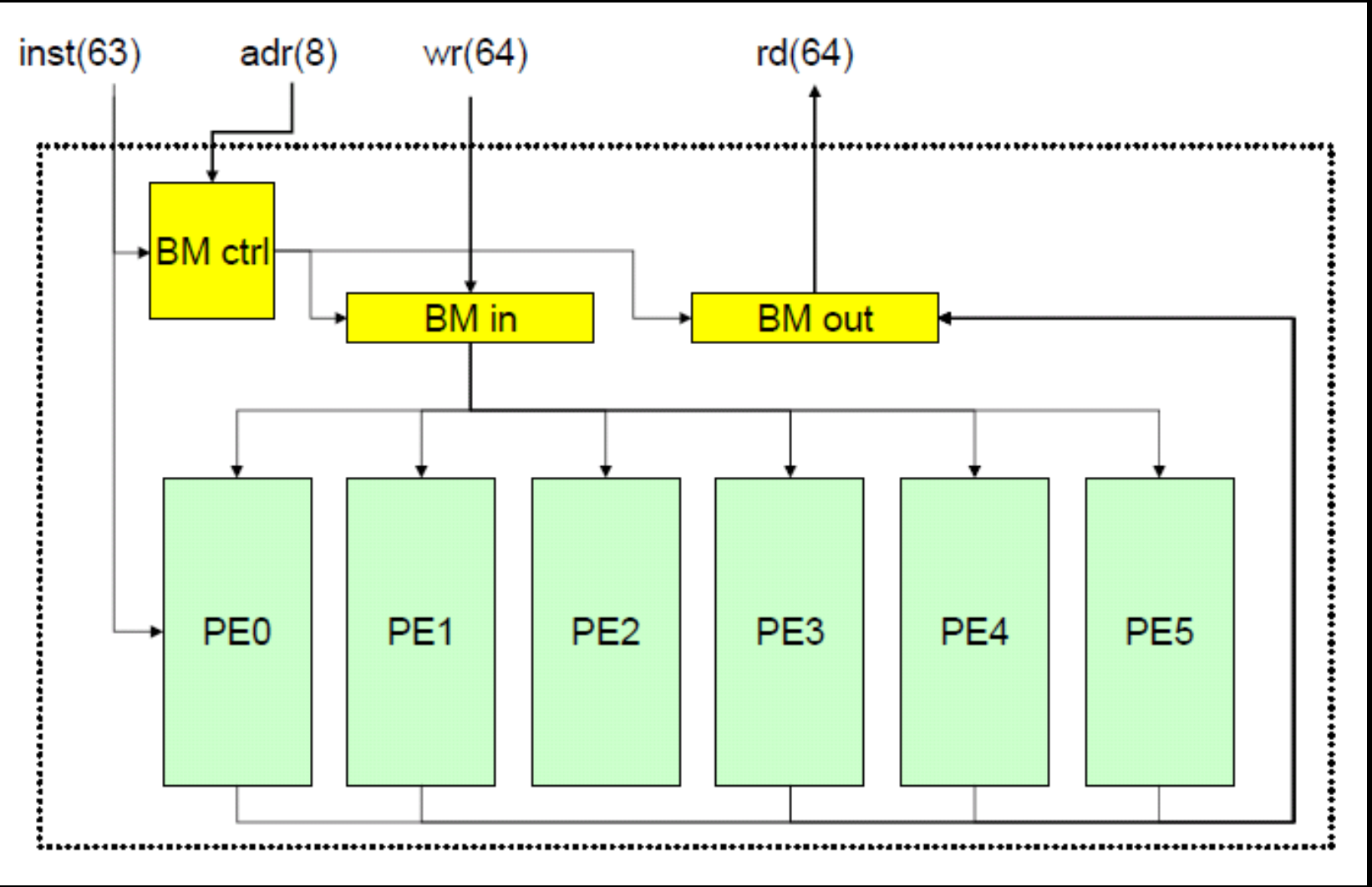
GRAPE-MPができるまで(1)

- 2008年8月
  - eASICでのプロセッサ開発の検討 (牧野)
- 2008年10月
  - 三つの設計案を検討 (中里, 牧野)
    - GRAPE-DRを拡張 (捨てられているビットを保持する)
      - DDエミュレーションの高速化
    - 整数のALUをたくさん並べる
      - 整数エミュレーション専用プロセッサ
    - 128bit FP演算器のはいったGRAPE-DR
      - 四倍精度専用プロセッサ
  - 第三案を採用 (12月頃に決定)

PEブロック図



chipブロック図


inst(63)
adr(8)
wr(64)
rd(64)
BM ctrl
BM in
BM out
PE0
PE1
PE2
PE3
PE4
PE5

GRAPE-MPができるまで(2)

- 2008年12月 – 2009年1月
  - eASICでデバイスを作ることを決定
  - アーキテクチャの設計などにとりかかる (中里)
    - C言語でのエミュレーションプログラムの作成
      - 問題:128bit整数演算をどうするか？ (GMPを利用)
    - VHDLで演算回路およびプロセッサの設計・テスト
- 2009年1月 – 4月
  - 1月17日「GRAPE-MP」という名前がつく
  - 引き続き回路設計 (中里)
    - 他, システムソフトウエアの作成 (アセンブラ)
  - 設計担当会社との打ち合わせ (中里,牧野)
  - 4月28日 Final Design Review

四倍精度演算器の設計 (1)

- 我々が開発したFP回路生成フレームワーク
  - 指数部, 仮数部等を指定してFP演算器を生成
    - CORE Generator/Mega Functionと同等
      - ただしこれらは必要最低限のflexibility
      - 128bitのFP演算器などは生成できない
    - 自前で設計することで、すべてflexible！
    - ただし、FP乗算器は一部再設計した
      - bit数が大きいため
  - メタプログラミングはやっぱり重要
    - RubyによりCのソースとVHDLを生成
      - Cのソースは FP演算のエミュレーションコード
    - GRAPE的なパイプライン生成も含む

四倍精度演算器の設計 (2)

- VHDLの面倒なところ
  - 記述が冗長。
  - 同じようなことを繰り返し記述する必要がある
  - 「レジスタ」の暗黙的な宣言
  - VHDLでの繰り返し処理等を覚えるのが大変

- HTMLの面倒なところを改善するのに
  - HTML用のテンプレートエンジン
    - Formの繰り返し記述をソースから生成
    - VHDLの繰り返し記述に応用できる？

四倍精度演算器の設計 (3)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity int_lshift_27_8_0 is
  port (
    ss : in std_logic_vector(7 downto 0);
    i : in std_logic_vector(26 downto 0);
    o : out std_logic_vector(26 downto 0)
  );
end int_lshift_27_8_0;

architecture source of int_lshift_27_8_0 is
begin
  with ss select
      o <= i(26 downto 0) when "00000000", -- 0
           i(25 downto 0)&"0" when "00000001", -- 1
           i(24 downto 0)&"00" when "00000010", -- 2
           i(23 downto 0)&"000" when "00000011", -- 3
           i(22 downto 0)&"0000" when "00000100", -- 4
           i(21 downto 0)&"00000" when "00000101", -- 5
           i(20 downto 0)&"000000" when "00000110", -- 6
           i(19 downto 0)&"0000000" when "00000111", -- 7
           i(18 downto 0)&"00000000" when "00001000", -- 8
           i(17 downto 0)&"000000000" when "00001001", -- 9
           i(16 downto 0)&"0000000000" when "00001010", -- 10
           i(15 downto 0)&"00000000000" when "00001011", -- 11
           i(14 downto 0)&"000000000000" when "00001100", -- 12
           i(13 downto 0)&"0000000000000" when "00001101", -- 13
           i(12 downto 0)&"00000000000000" when "00001110", -- 14
           i(11 downto 0)&"000000000000000" when "00001111", -- 15
           i(10 downto 0)&"0000000000000000" when "00010000", -- 16
           i(9 downto 0)&"00000000000000000" when "00010001", -- 17
           i(8 downto 0)&"000000000000000000" when "00010010", -- 18
           i(7 downto 0)&"0000000000000000000" when "00010011", -- 19
           i(6 downto 0)&"00000000000000000000" when "00010100", -- 20
           i(5 downto 0)&"000000000000000000000" when "00010101", -- 21
           i(4 downto 0)&"0000000000000000000000" when "00010110", -- 22
           i(3 downto 0)&"00000000000000000000000" when "00010111", -- 23
           i(2 downto 0)&"000000000000000000000000" when "00011000", -- 24
           i(1 downto 0)&"0000000000000000000000000" when "00011001", -- 25
           i(0)&"00000000000000000000000000" when others; -- 26
end source;
```

四倍精度演算器の設計 (3)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

with ss select
    o <= i(26 downto 0) when "00000000", -- 0
         i(25 downto 0)&"0" when "00000001", -- 1
         i(24 downto 0)&"00" when "00000010", -- 2
         i(23 downto 0)&"000" when "00000011", -- 3
         i(22 downto 0)&"0000" when "00000100", -- 4
         i(21 downto 0)&"00000" when "00000101", -- 5
         i(20 downto 0)&"000000" when "00000110", -- 6
         i(19 downto 0)&"0000000" when "00000111", -- 7
         i(8 downto 0)&"00000000000000000" when "00010010", -- 18
         i(7 downto 0)&"00000000000000000" when "00010011", -- 19
         i(6 downto 0)&"00000000000000000" when "00010100", -- 20
         i(5 downto 0)&"00000000000000000" when "00010101", -- 21
         i(4 downto 0)&"00000000000000000" when "00010110", -- 22
         i(3 downto 0)&"00000000000000000" when "00010111", -- 23
         i(2 downto 0)&"00000000000000000" when "00011000", -- 24
         i(1 downto 0)&"00000000000000000" when "00011001", -- 25
         i(0)&"00000000000000000" when others; -- 26
end source;
```

四倍精度演算器の設計 (4)

```
architecture source of <%= @name %> is
begin
% s = n-1
   with ss select
%(0..(n-1)).each { |i|
%    u = s - i
%    l = u - n + 1
%    if l < 0 then
%      z = 0.to_b(-l)
%      zero = "&¥"#{z}¥""
%      l = 0
%    else
%      zero = ''
%    end
%    case i
%    when 0
       o <= i(<%= u %> downto <%= l %>) when "<%= i.to_b(@nexp) %>", -- <%= i %>
%    when n-1
           i(<%= u %>)<%= zero %> when others; -- <%= i %>
%    else
           i(<%= u %> downto <%= l %>)<%= zero %> when "<%= i.to_b(@nexp) %>", -- <%= i %>
%    end
%}
end source;
```

eRubyによるメタプログラミング
(VHDL + 埋め込みRuby)

四倍精度演算器の設計 (5)

- 回路設計とテスト
  - テストベンチもeRubyにより生成
  - GHDL (VHDLのコンパイラ)により実行ファイルの自動生成と実行
  - 結果をシミュレータと比較
  - テスト駆動回路設計

- 演算回路の動作はこの手法で検証可能だが
  - デバイス特有部分(メモリなど)のシミュレーションはできない

GRAPE-MPができるまで(3)

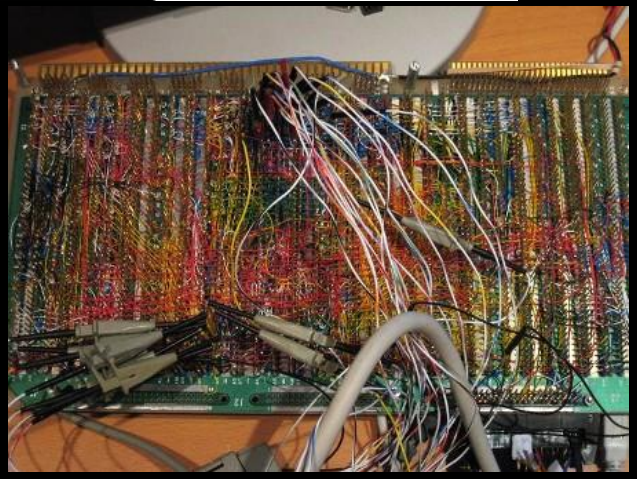
- 2009年4月-
  - 4月30日テープアウト
    - イノテック → eASIC
  - 7月下旬 チップが届く
  - 7月 – 12月
    - GRAPE-MPボードの設計 (台坂)
    - 9月- FPGA回路の設計開始 (台坂, 中里)
    - アセンブラ, コンパイラの整備など (中里)
- 20010年4月 -
  - GRAPE-MPボードが届く
  - FPGA回路の本格的な実装 (台坂, 中里)

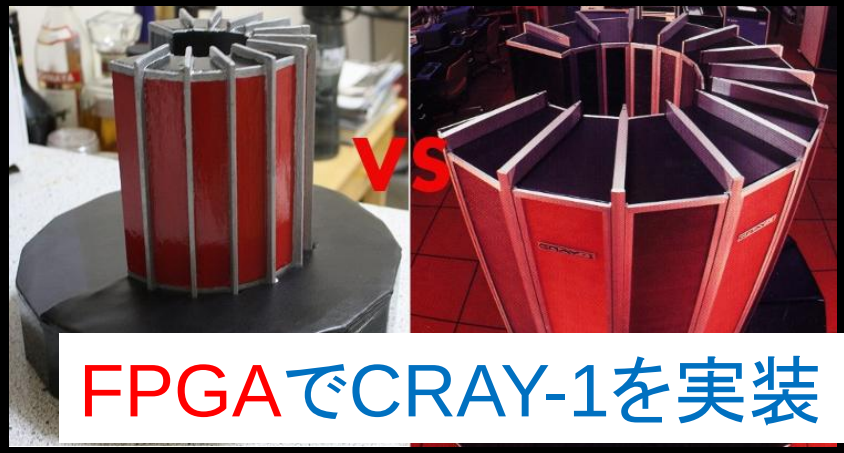
GRAPE-MPの開発とは

- 計算機を(ほぼ)full scratchで作ること
  - アーキテクチャ定義
  - 回路の論理設計
  - シミュレーションコードの作成
  - アセンブラの設計と実装
  - システムソフトウエアの実装
  - コンパイラの実装
- 私がやってないこと
  - 物理設計, チップの作成 (イノテック&eASIC)
  - ボードの設計や制御回路の設計 (台坂さん)

趣味としてのDIY計算機

CPUとOSを自作





FPGAでCRAY-1を実装

http://www.stevechamberlin.com/cpu/category/bmow1/

http://chrisfenton.com/homebrew-cray-1a/

GRAPE-MPボード

- FPGAとGRAPE-MPチップからなる

GRAPE-MPシステム(1)

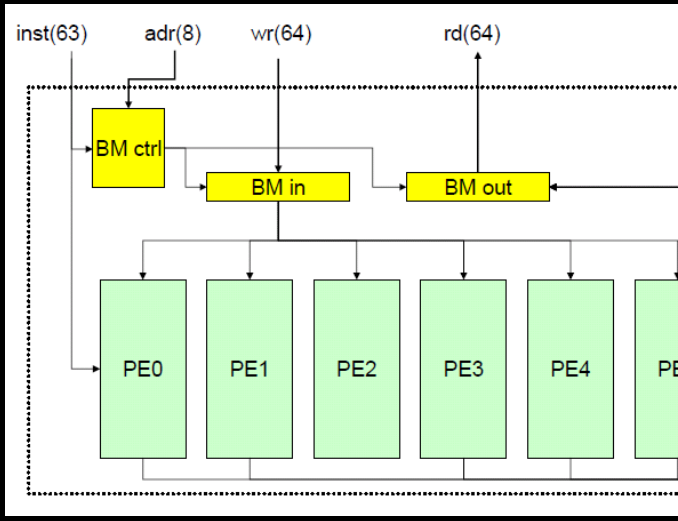
- アーキテクチャ
  - PEが6個からなるSIMDプロセッサ
    - 4 cycleのSIMTベクトル処理
    - 実効ベクトル長は24
  - PEは四倍精度演算器を持つ
    - 仮数部 116 bit 指数部 11 bit の演算器
      - 加算, 乗算
    - 逆数平方根の初期値回路 (26 bit精度)
      - フル精度の除算と平方根はニュートン法でソフトウエア処理
    - レジスタは256語
    - Forwarding pathあり
  - PEはブロードキャストメモリ(BM)と接続

GRAPE-MPシステム(2)

- 現状の物理的仕様
  - 100 MHz動作を想定
    - チップの消費電力 5W
    - 理論演算性能 ~ 1.2 Gflop/s
  - FPGAがGRAPE-MPチップを制御する
    - GRAPE-DRと同様の構成
  - 外部メモリなし (FPGAのbramを利用)
  - FPGAはPCI-Expressインターフェイスを内蔵し、ホストとの通信もおこなう

GRAPE-MPシステム(3)

- 動作のイメージ
  - exec
    - FPGA: 命令を書き込み
  - write
    - FPGA: BMにデータ書き込み
    - exec : BMからPEに読み込み
  - read
    - exec: PEからBMに書き出し
    - FPGA: BMからデータ読み出し



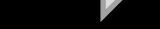
GRAPE-MPのソフトウエア(1)

- C/C++言語によるシミュレータ
  - 演算器レベル
    - GMPによる128bit整数演算
  - PEレベル
    - 演算命令やレジスタのシミュレーション
  - チップレベル (BM等を含めたシミュレーション)
    - アセンブラが生成した命令を読み込んで計算
    - デバッグ命令の処理
    - 変数変換のライブラリ
      - QDとGRAPE-MPフォーマットの変換処理など

GRAPE-MPのソフトウエア(2)

- アセンブラ
  - ソースコードをマイクロコード (63bit)に変換
  - 読み書きの衝突は検出
    - 他のエラー処理は今はなし

```
sub bm16v ra0v rb40v
sub bm20v ra4v rb44v
sub bm24v ra8v rb48v
mul rb40v rb40v ra36v
mul rb44v rb44v tt
add ra36v ts ra32v
mul rb48v rb48v tt
add ra32v ts tt
```

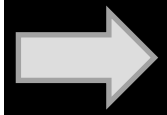


```
1006600214000003        0001000000000110011000000000001000010100
1106600214800007        0001000100000110011000000000001000010100
120660021500000b        0001001000000110011000000000001000010101
130660021580000f        0001001100000110011000000000001000010101
1406600216000013        0001010000000110011000000000001000010110
1506600216800017        0001010100000110011000000000001000010110
160660021700001b        0001011000000110011000000000001000010111
170660021780001f        0001011100000110011000000000001000010111
1806600218000023        0001100000000110011000000000001000011000
1906600218800027        0001100100000110011000000000001000011000
1a0660021900002b        0001101000000110011000000000001000011001
1b0660021980002f        0001101100000110011000000000001000011001
7a24000245001   0000000000000111101000100100000000000000001001000
7a24000255201   0000000000000111101000100100000000000000001001010
7a24000265401   0000000000000111101000100100000000000000001001100
7a24000275601   0000000000000111101000100100000000000000001001110
3e24000005801   0000000000000011111000100100000000000000000000000
3e24040005a01   0000000000000011111000100100000100000000000000000
3e24080005c01   0000000000000011111000100100000100000000000000000
3e240c0005e01   0000000000000011111000100100000110000000000000000
7802000200091   0000000000000011110000000010000000000000001000000
7802000210095   0000000000000011110000000010000000000000001000010
7802000220099   0000000000000011110000000010000000000000001000100
780200023009d   0000000000000011110000000010000000000000001000110
3e24000006001   0000000000000011111000100100000000000000000000000
3e24040006201   0000000000000011111000100100000100000000000000000
3e24080006401   0000000000000011111000100100000100000000000000000
3e240c0006601   0000000000000011111000100100000110000000000000000
1e02000000081   0000000000000001111000000010000000000000000000000
1e02040000085   0000000000000001111000000010000010000000000000000
```

GRAPE-MPのソフトウエア(3)

- コンパイラ (18行 → 63行のアセンブリ言語)

```
VARI xi, yi, zi, e2;
VARJ xj, yj, zj, mj;
VARF ax, ay, az, pt;

dx = xj - xi;
dy = yj - yi;
dz = zj - zi;

r1i = rsqrt(dx**2 + dy**2 + dz**2 + e2);

pf = mj*r1i;
pt += pf;

af = pf*r1i**2;

ax += af*dx;
ay += af*dy;
az += af*dz;
```



```
bm_in bm12v ra12v pe0
bm_in bm8v ra8v pe0
bm_in bm4v ra4v pe0
bm_in bm0v ra0v pe0
mov zz ra16v
mov zz ra28v
mov zz ra24v
mov zz ra20v
sub bm16v ra0v rb40v
sub bm20v ra4v rb44v
sub bm24v ra8v rb48v
mul rb40v rb40v ra36v
mul rb44v rb44v tt
add ra36v ts ra32v
mul rb48v rb48v tt
add ra32v ts tt
add ts ra12v tt
mul ts bm126 ra120v
rsq tt tt
 ....
```

GRAPE-MPの応用(1)

- 現在までに実機で以下のカーネルが動作
  - ファイマンループ積分

$$
\begin{aligned}
I &= \int_0^1 dx \int_0^{1-x} dy \int_0^{1-x-y} dz F(x,y,z), \\
F(x,y,z) &= D(x,y,z)^{-2} \\
D &= -xys - tz(1-x-y-z) + (x+y)\lambda^2 \\
&\quad +(1-x-y-z)(1-x-y)m_e^2 \\
&\quad +z(1-x-y)m_f^2. \qquad (2)
\end{aligned}
$$

  - 重力とポテンシャルの計算

$$
\boldsymbol{f}_i = \sum_{j=1}^{N} \frac{m_j(\boldsymbol{x}_i - \boldsymbol{x}_j)}{(|\boldsymbol{x}_i - \boldsymbol{x}_j|^2 + \epsilon^2)^{3/2}},
$$

GRAPE-MPの応用(2)

- 重力とポテンシャル計算

```
daisaka@cabernet[NBODY]%./mytest
BMEM read 0 words
INST read 228 instructions
        emu             QD              err
16      3.572036e-01    3.572036e-01    1.066375e-35    3fd6dc6cac3464b2        76d9ed965e148af4
17      1.115395e+00    1.115395e+00    -7.050021e-36   3ff1d8a84409ea35        9565b3349cf6ad2f
18      -1.029145e+00   -1.029145e+00   -3.086344e-35   bff07761299b503c        315cace2225276d3
19      -6.640516e+00   -6.640516e+00   2.397415e-35    c01a8fe363542ce8        13c7e4379258409a
20      9.701022e-01    9.701022e-01    2.214752e-35    3fef0b13d917dc18        62c85d1f716f668f
21      -3.523966e-01   -3.523966e-01   -4.343786e-35   bfd68daa9a8a205f        9e69ab1ddce5ff4c
22      6.196427e-02    6.196427e-02    -1.419997e-35   3fafb9c7f71859e2        45509e9beeaccd75
23      -6.774808e-02   -6.774808e-02   -1.608934e-33   bfb157f03bfbd2b6        9635d8144ed48713
```

シミュレータでの結果

実機の結果

```
daisaka@g7host02[NBODY]%../../../lib/betest 2 0
argv[0] = ../../../lib/betest
argv[1] = 2
argv[2] = 0
hib[0] opened.
#before DMAW:  size = 128,  datacnt=1024
-----------------------------------------------------------
32:  76d9ed965e148af4
33:  3fd6dc6cac3464b2
34:  9565b3349cf6ad2f
35:  3ff1d8a84409ea35
36:  315cace2225276d3
37:  bff07761299b503c
38:  13c7e4379258409a
39:  c01a8fe363542ce8
40:  62c85d1f716f668f
41:  3fef0b13d917dc18
42:  9e69ab1ddce5ff4c
```

GRAPE-MPの応用(3)

- ファイマンループ積分
  - より複雑なものへの適用
  - 演算性能の評価
  - 複数枚システムの開発

- N体計算での応用の可能性
  - 2, 3体問題を高精度に解くのに最適
  - 惑星系の安定性を調べる
  - GPUボードと組み合わせて、球状星団のコアの計算に利用する？

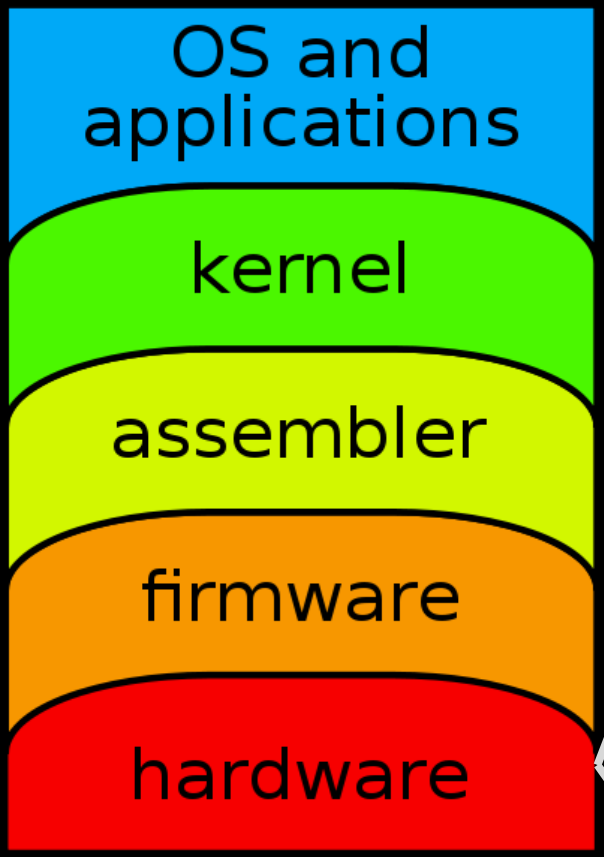
まとめ

- 四倍精度専用プロセッサGRAPE-MP
  - (たぶん)世界初の四倍精度プロセッサ
  - GRAPE-DR的なプログラマブル計算機
  - 計算機をfull scratchで作ること
    - Do it yourself !
- さらにGRAPE-MPシステムを開発中
  - すでに実機でのテストを行った
  - 今後本格的な計算の実装

- メタプログラミングはますます重要

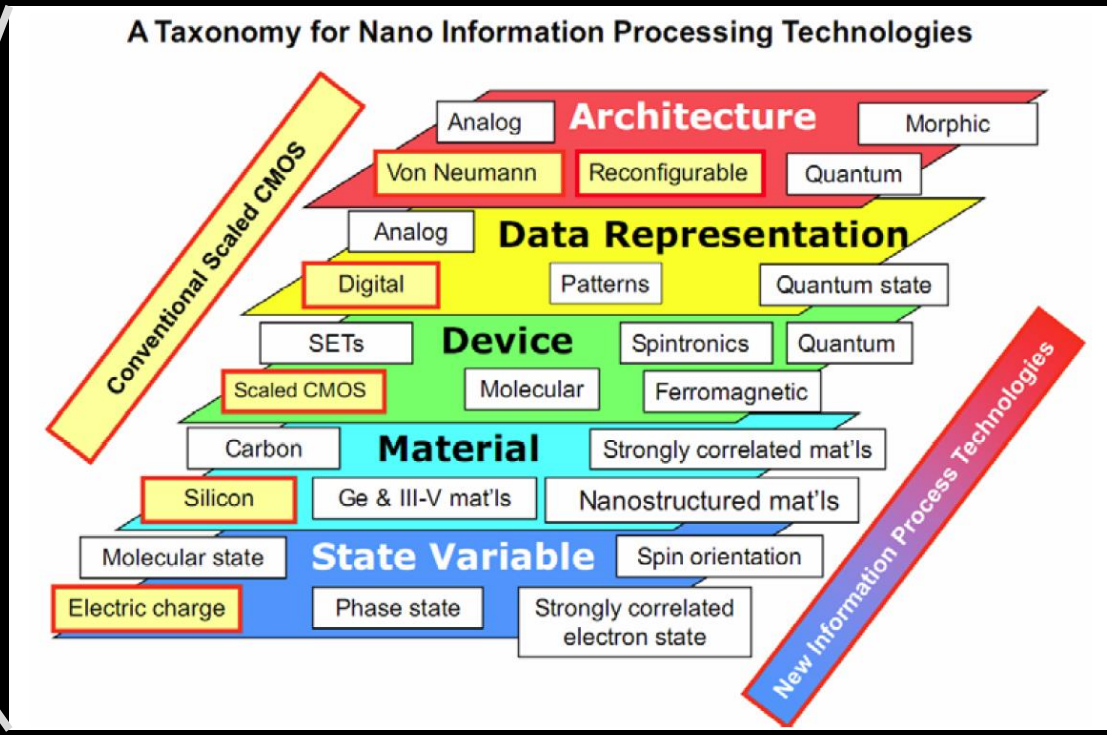
レイヤー思考を突き抜ける

計算機の抽象レイヤー構造



GRAPE-MPの開発で我々がDIYした部分

計算デバイスのレイヤー構造



DIY精神の成功例

- “Parallel Computing Works“ Fox, Williams, Messina 1994, 20章より
  - Caltech Cosmic Cube(CCC)を作ったFoxらによる、並列計算プロジェクトC3Pについてまとめた大著
- 逸話：CCCの実働者であったOttoは、QCDの物理を理解していただけでなく、その定式化、そしてCCCの実現に必要な計算機アーキテクチャやハードウエアも理解し、CCCを実現した --- 計算科学者の誕生

- 詳しくは私訳(http://galaxy.u-aizu.ac.jp/trac/note/blog/)を参照してください