

# 天体物理学シミュレーション におけるFPGAの利用について

中里直人(会津大学)

台坂博(一橋大), 石川正, 湯浅富久子, 元木信二(KEK), KFCR社

2014年10月17-18日天体形成研究会  
筑波大学 計算科学研究センター ワークショップ室

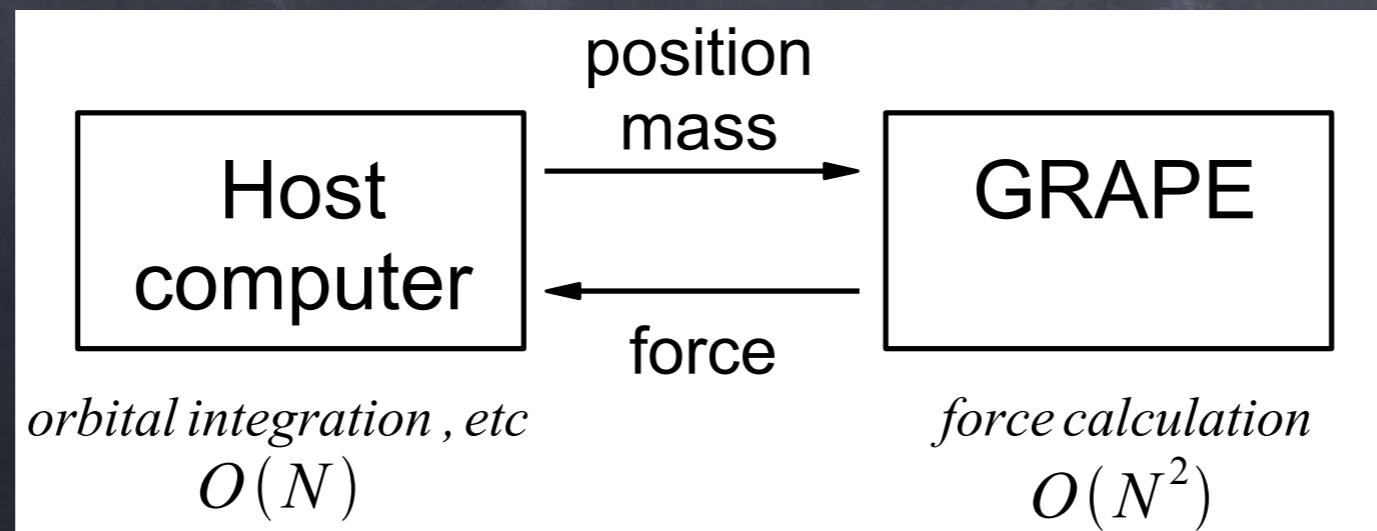
# 並列計算 = 高速計算

- 伝統的な高速計算(HPC) = Fortran
  - この言語自体が数値計算のために設計された
  - 最適化がやりやすい言語仕様
- 現代の高速計算 = 並列計算
  - プログラミングが本質的に困難
  - 自動並列化は絶望的
    - Fortranのベクトル化だけはうまくいっていたけれど。。。
  - 上から下まで異なる粒度の並列化
  - SSE/AVX, スレッド, CPU-GPU, MPI....

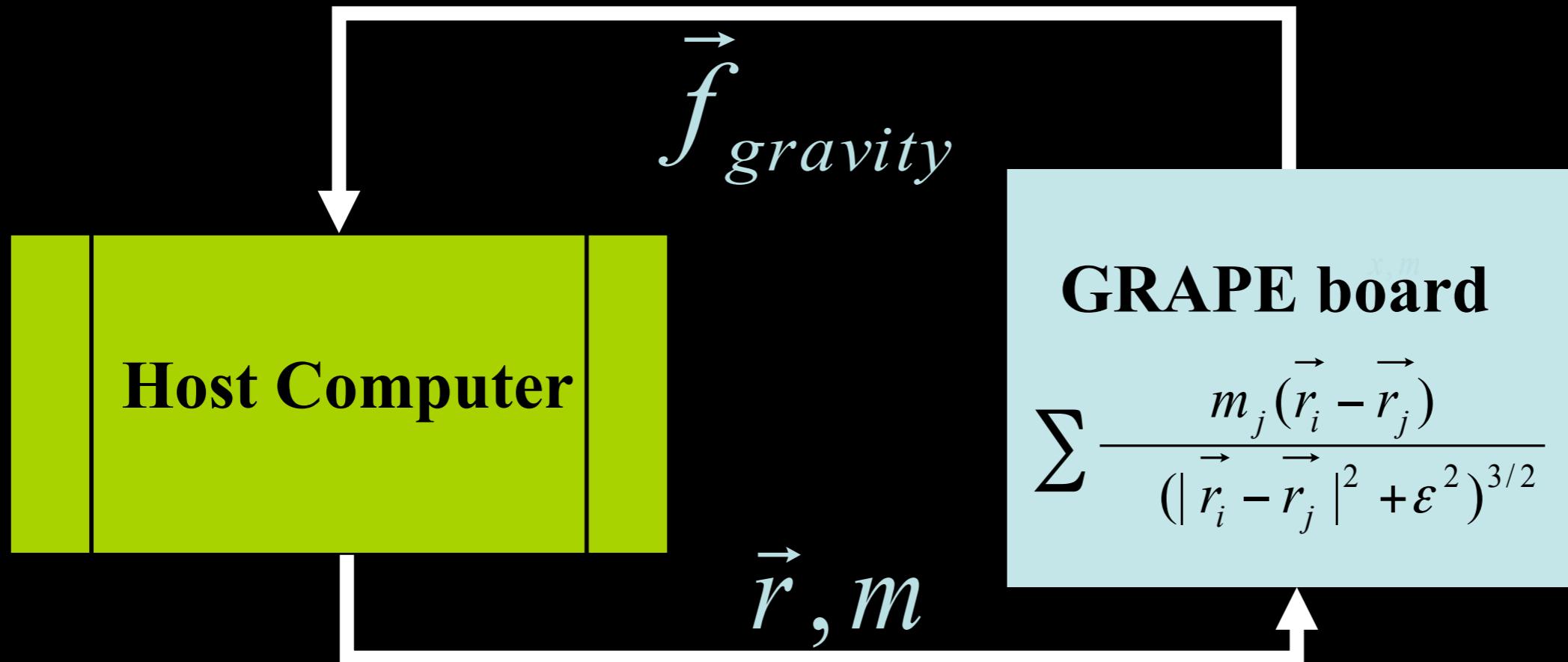
# 発展的な粒子法のためのDSL

- 必要なこと

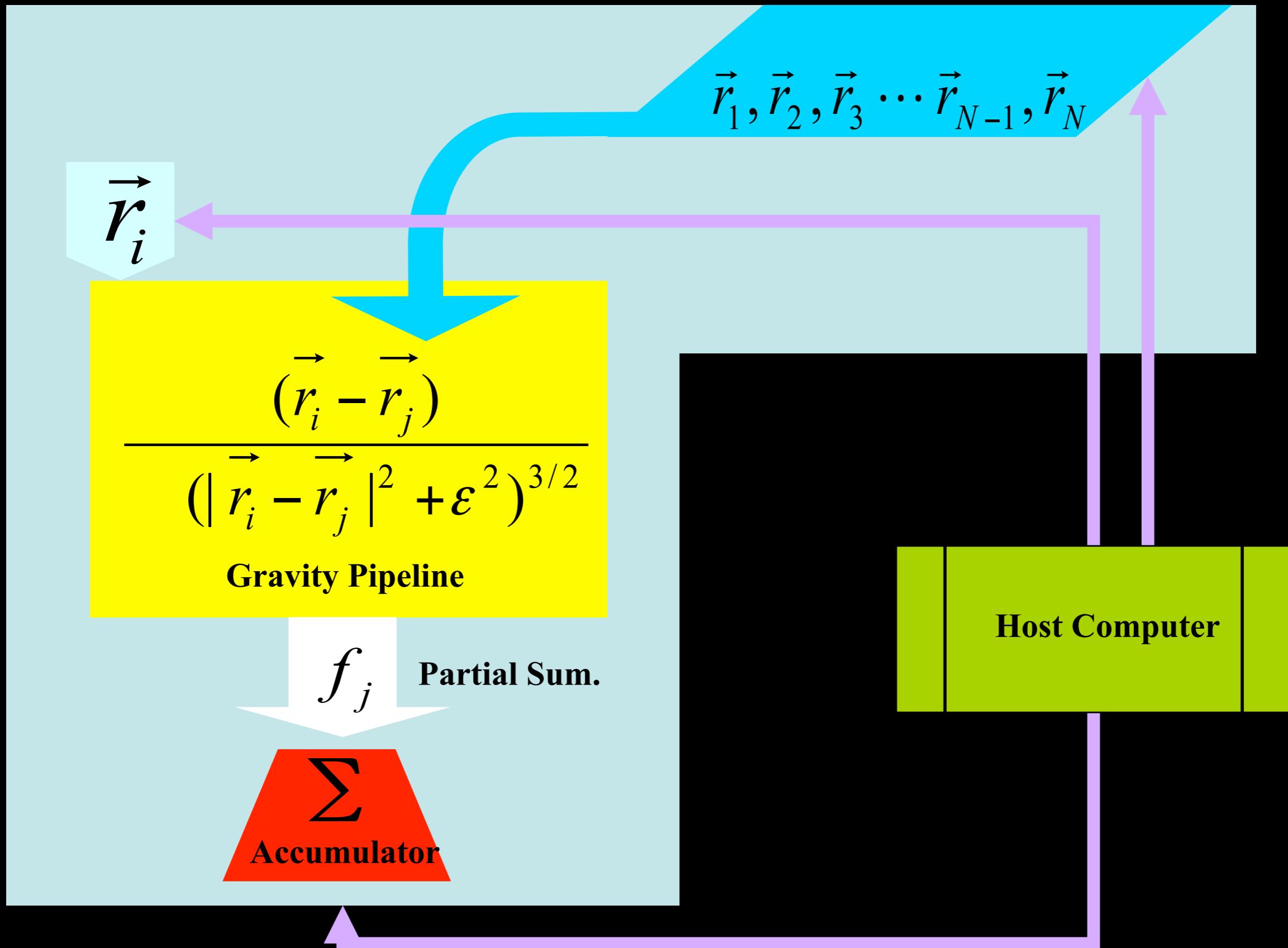
- やりたいのは「カスタムGRAPE」の実装
  - FPGAベース(PGPG, PGR), GRAPE-DR, LSUMP....
- 粒子間相互作用の表現 (=演算カーネル)
- メモリアクセスパターンの抽象化
- 近接や長距離をうまく使い分けられたら
- そもそもGRAPEは計算機の「関数化」



# Hardware Accelerator : GRAPE

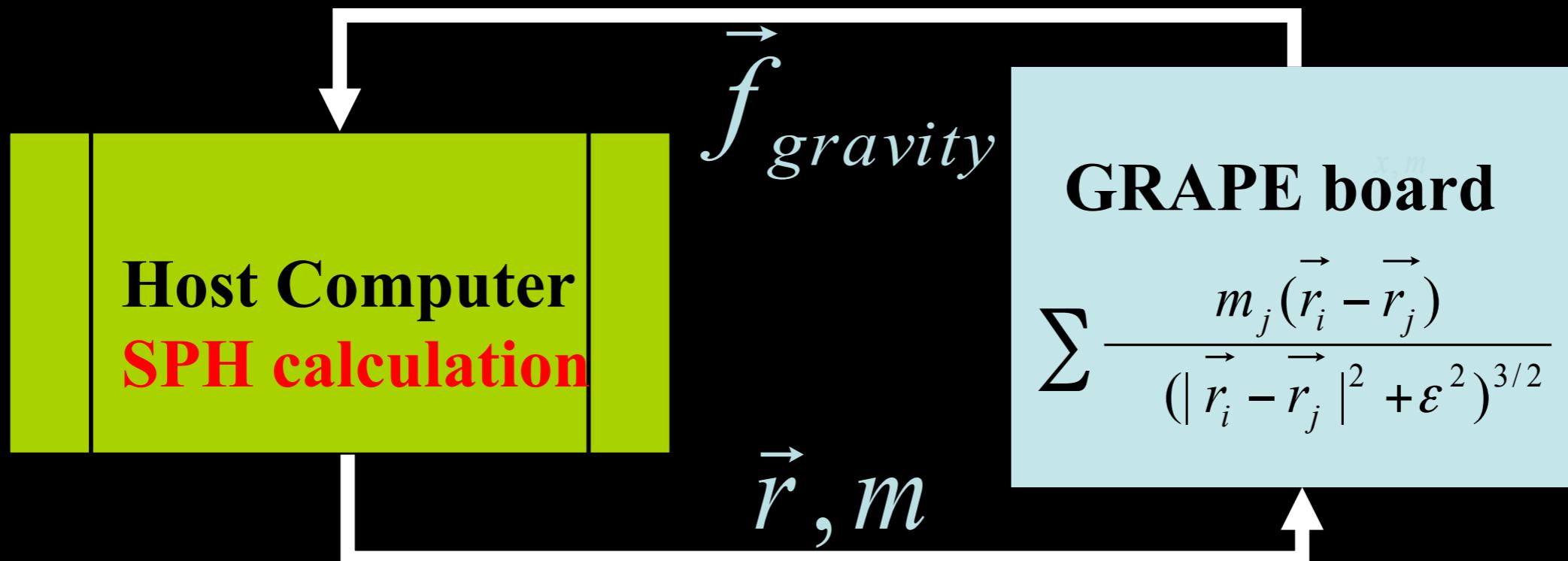


# GRAPE: custom LSI



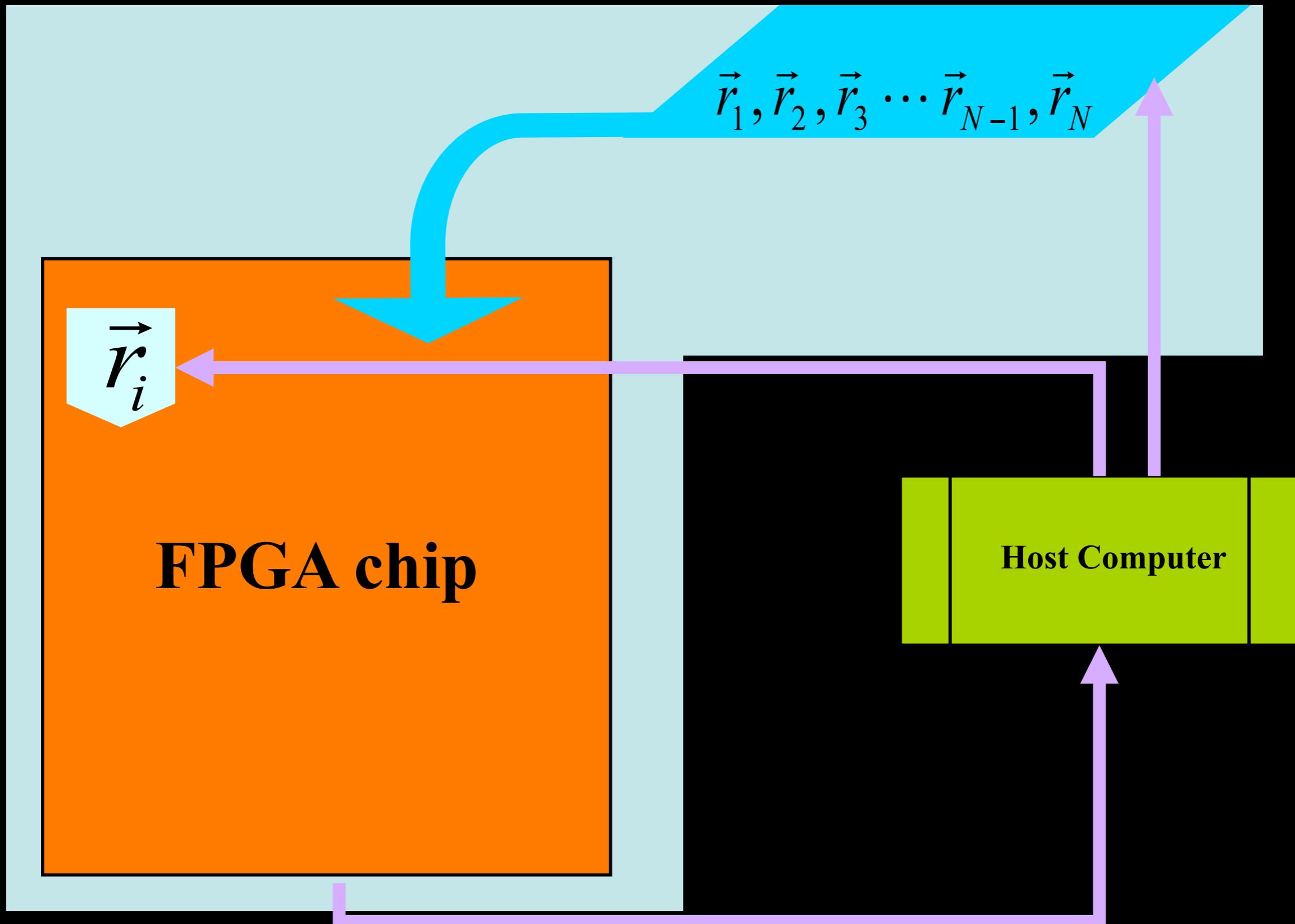
# Problem is...

- SPH for gas dynamics
- GRAPE for gravity



- At that time the HOST is so slow for SPH calculations.

# PROGRAPE: FPGA



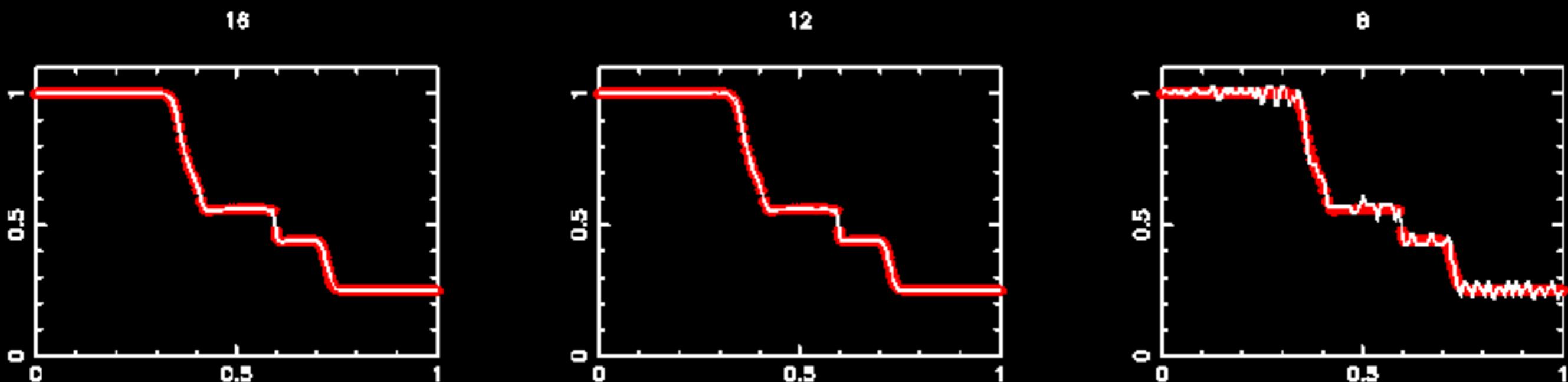
# Accuracy test (1)

- How numerical accuracy affect SPH calculations of the Sod's shock tube problem?
  - Floating point format:  $s \times f \times 2^{\text{exp}}$ 

sign	exponent	Fraction/mantissa
------	----------	-------------------
  - On CPU: single or double precision
    - Single precision: exp 8 bit, fraction 24 bit
    - Double precision: exp 11 bit, fraction 53 bit
  - On FPGA, we can use any size of exponent and fraction. Smaller, faster calculations!

# Accuracy test (2)

- Changing the size of the exponent to 16, 12, 8 bit (red: double precision result)



- 16bit: very similar to double prec.
- 12bit: mostly similar
- 8bit: different with oscillations
  - Reasons are in under investigation

# Recent Examples: SO/PHI

- J.P. Cobos Carrascosa et al. MCSoC 2014
- Solar Orbiter Polarimetric and Helioseismic Imager
  - 太陽プラズマの磁場と視線速度の観測装置
  - Solar Orbiter <http://sci.esa.int/solar-orbiter/>
  - Solving the inversion of the radiative transfer equation (RTE)
- 非線形の最小二乗法に帰着される  $2048^2 \times 24$  frames
- 特異値分解に  $\sim 10$ GFlops が必要
- オンライン処理のためFPGAによる実装を評価した

# Recent Examples: SO/PHI

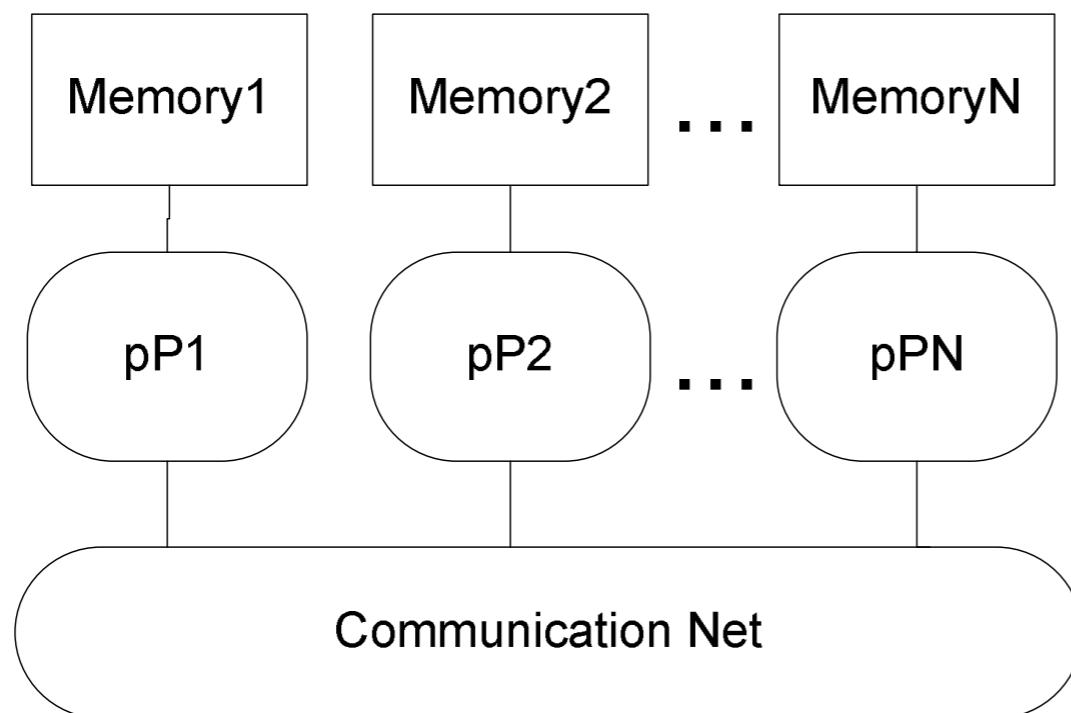


Fig 1. Distributed memory, MIMD multicomputer

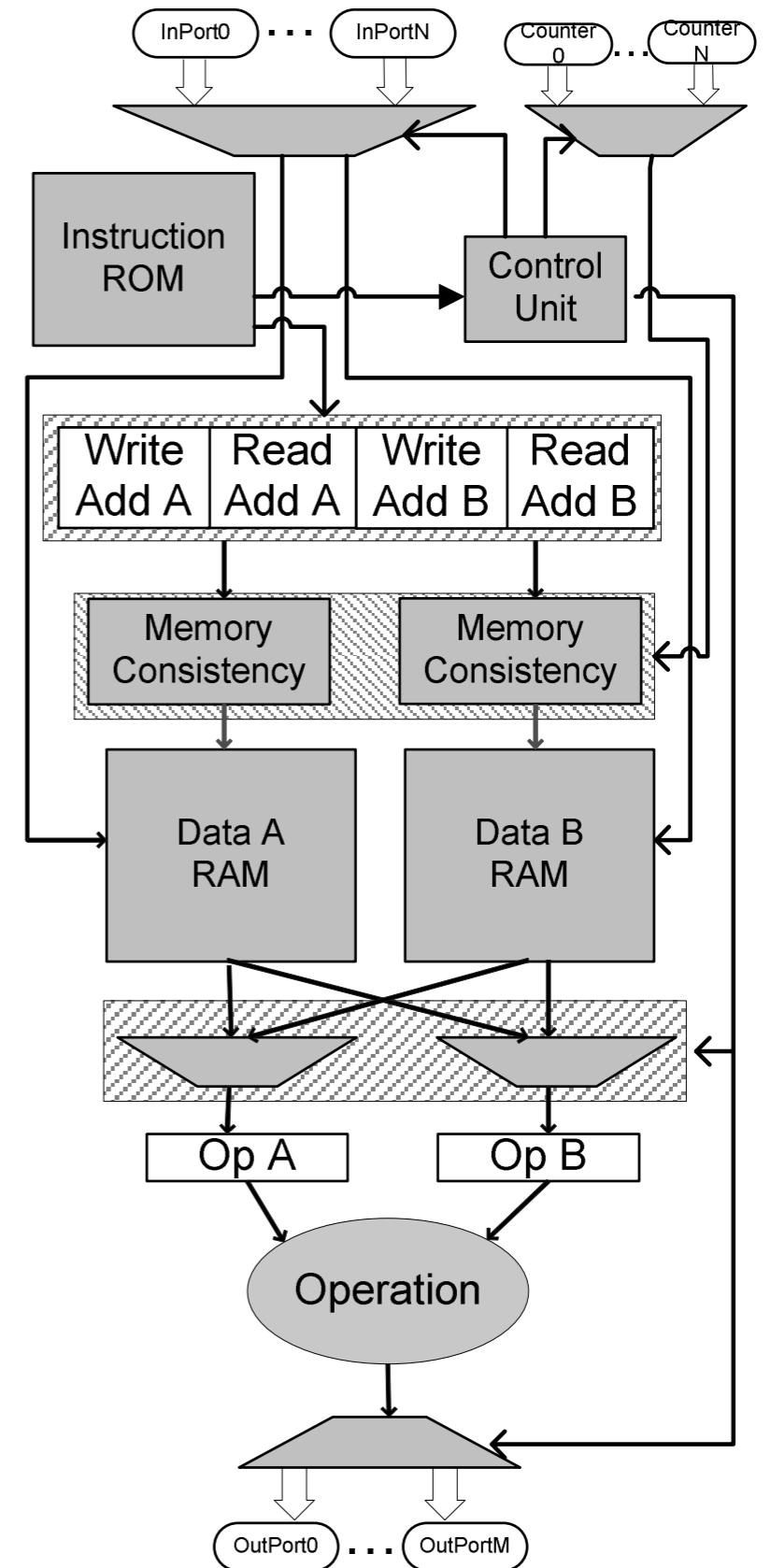
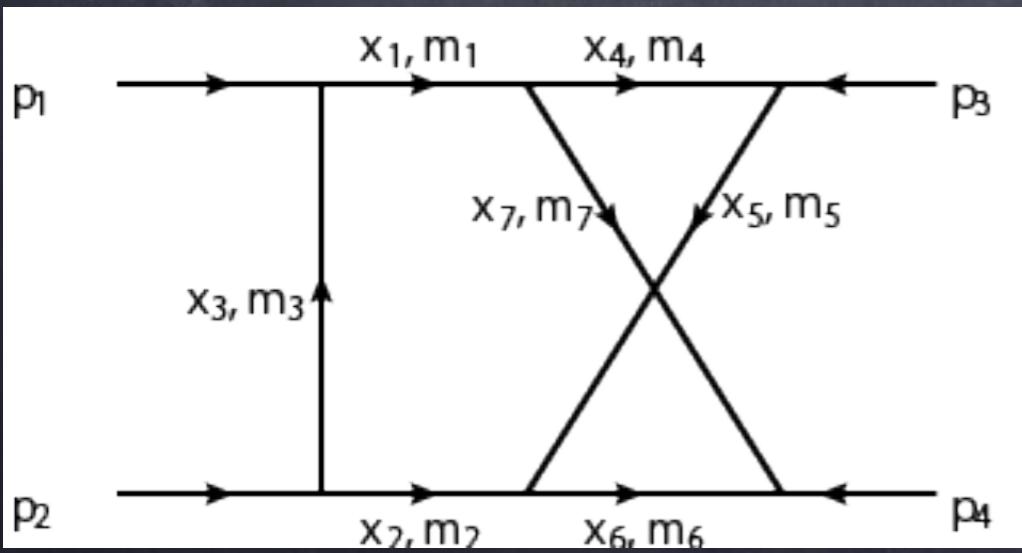


TABLE III. DEVICE COMPARATIVE TIME. FPGA VS PC

Resource	FPGA	C-MILOS
Device	XC5VFX130T	Intel Xeon
Clock	200 MHz	2,6 GHz
Execution time	1,1 $\mu$ s	43.35 $\mu$ s
Speedup	39.41	1

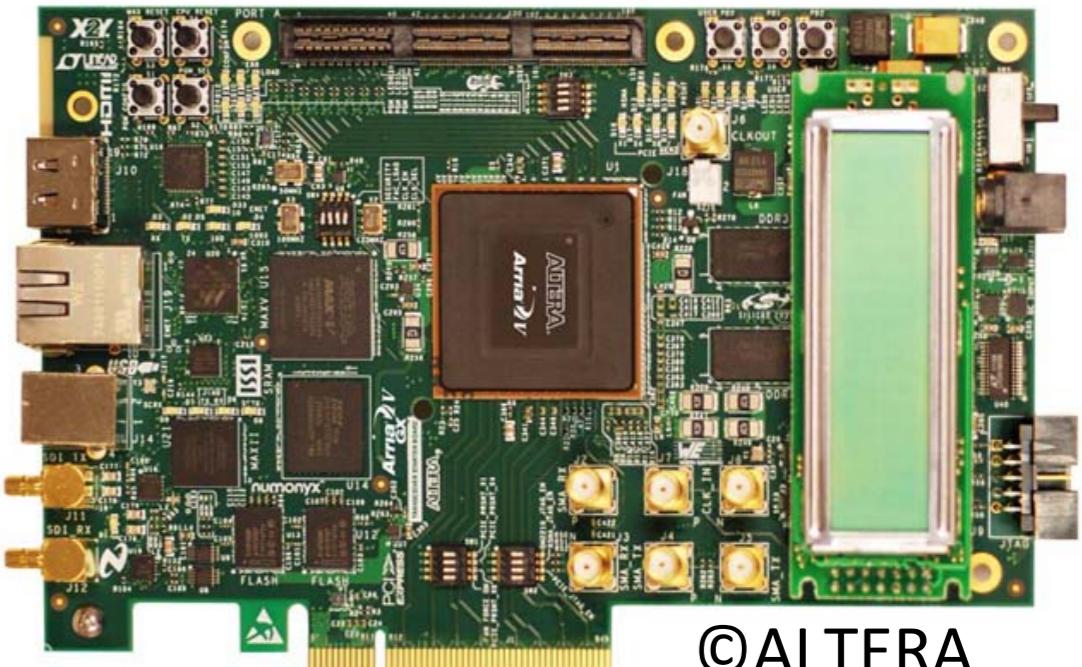
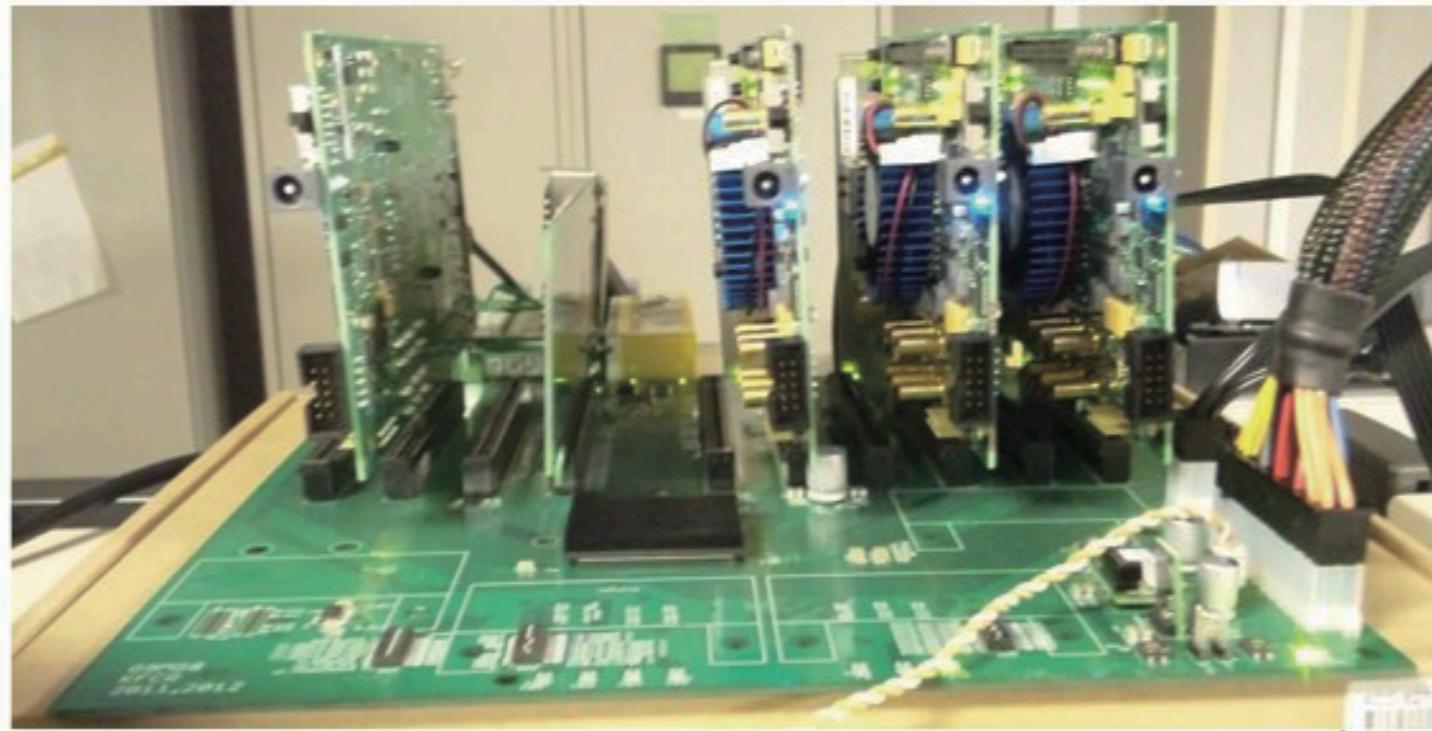
# GRAPE9-MPX (1)

- Motoki et al. <http://arxiv.org/abs/1410.3252>
- 多倍長浮動小数点演算アクセラレータ
- ファインマン・ダイアグラムの直接計算の高速化
- binary128フォーマットと同じ128ビットFP
- 指数部が15ビットになっている

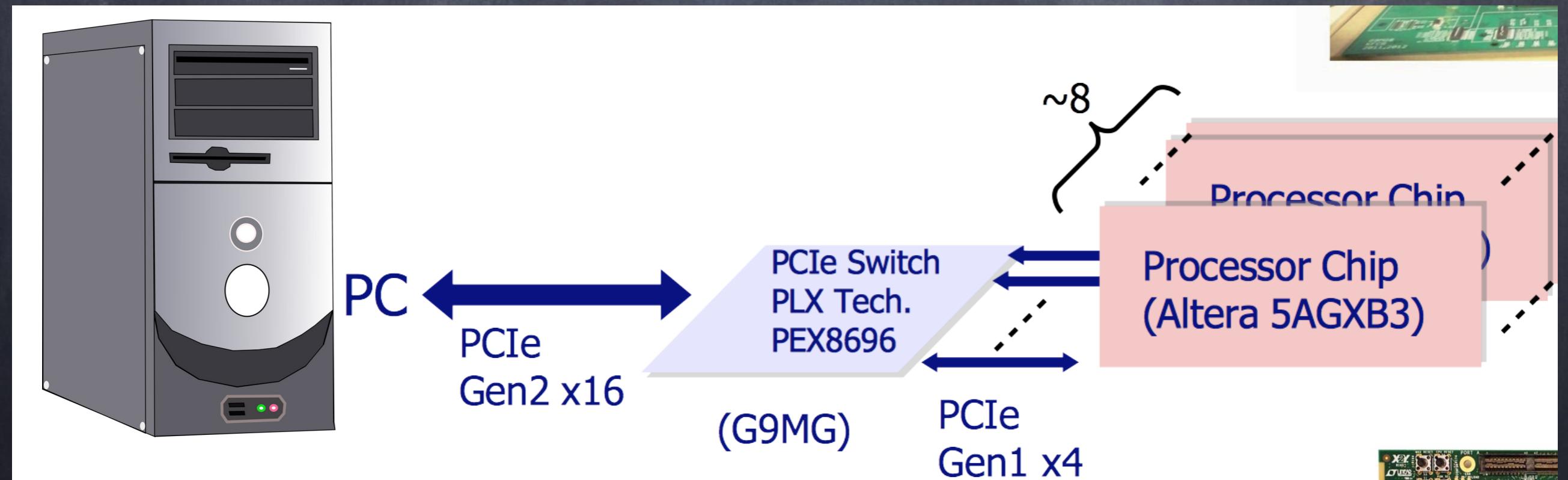


$$\begin{aligned}
 I &= -2 \int_0^1 dx_1 dx_2 dx_3 dx_4 dx_5 dx_6 dx_7 \delta(1 - \sum_{\ell=1}^7 x_\ell) \frac{C}{(D - i\epsilon C)^3} \\
 C &= (x_1 + x_2 + x_3 + x_4 + x_5)(x_1 + x_2 + x_3 + x_6 + x_7) - (x_1 + x_2 + x_3)^2 \\
 D &= C \sum_{\ell=1}^7 x_\ell m_\ell^2 \\
 &\quad - \{s(x_1 x_2 x_4 + x_1 x_2 x_5 + x_1 x_2 x_6 + x_1 x_2 x_7 + x_1 x_5 x_6 + x_2 x_4 x_7 - x_3 x_4 x_6) \\
 &\quad + t(x_3(-x_4 x_6 + x_5 x_7)) \\
 &\quad + p_1^2(x_3(x_1 x_4 + x_1 x_5 + x_1 x_6 + x_1 x_7 + x_4 x_6 + x_4 x_7)) \\
 &\quad + p_2^2(x_3(x_2 x_4 + x_2 x_5 + x_2 x_6 + x_2 x_7 + x_4 x_6 + x_5 x_6)) \\
 &\quad + p_3^2(x_1 x_4 x_5 + x_1 x_5 x_7 + x_2 x_4 x_5 + x_2 x_4 x_6 + x_3 x_4 x_5 + x_3 x_4 x_6 + x_4 x_5 x_6 + x_4 x_5 x_7) \\
 &\quad + p_4^2(x_1 x_4 x_6 + x_1 x_6 x_7 + x_2 x_5 x_7 + x_2 x_6 x_7 + x_3 x_4 x_6 + x_3 x_6 x_7 + x_4 x_6 x_7 + x_5 x_6 x_7)\}
 \end{aligned}$$

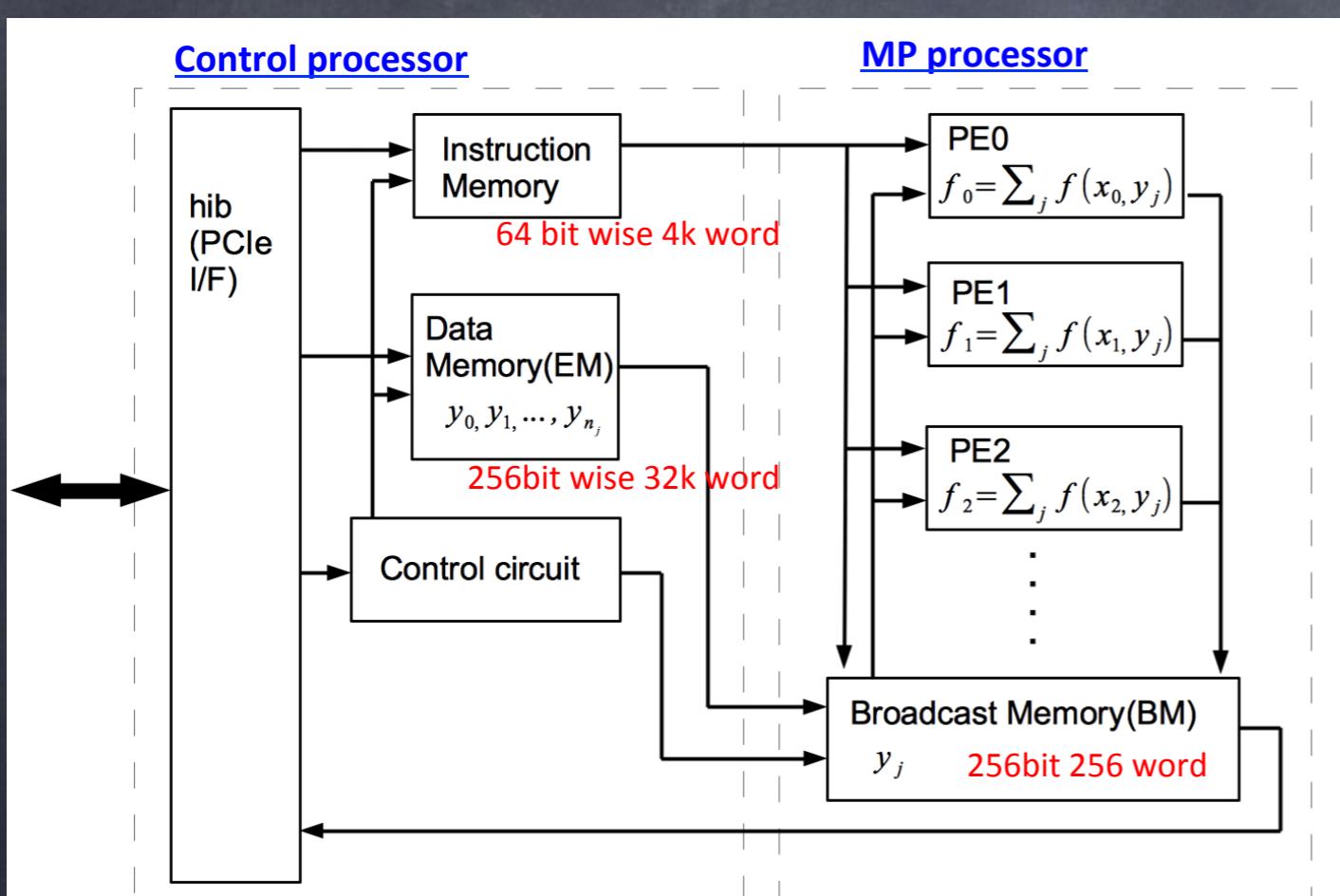
# GRAPE9-MPX (2)



©ALTERA



# GRAPE9-MPX(3)



## Sample code: one-loop box

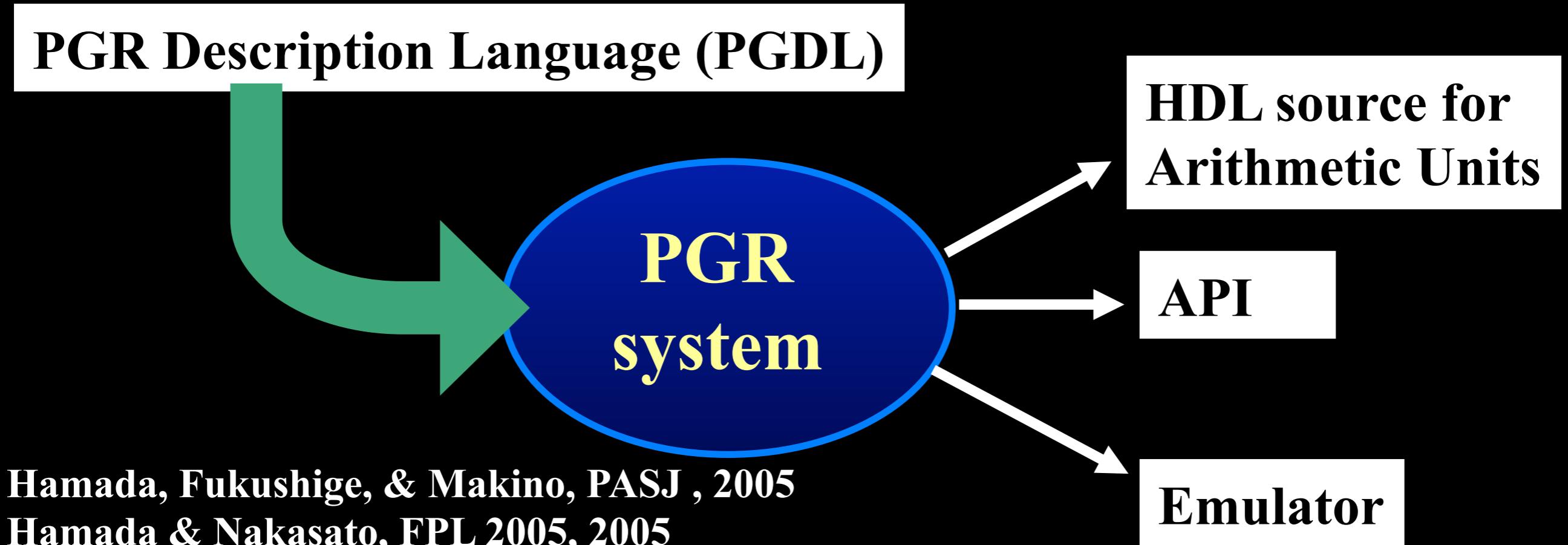
```
#pragma goose parallel for loopcounter(ixy, iz)
for(ixy = 0; ixy < ni; ixy++) {
    sumzG[ixy] = 0.0;
    for (iz = 0; iz < nj; iz++) {
        xx = dev_xx[ixy];
        yy = dev_yy[ixy];
        zz = x30_1[iz] * dev_cnt4[ixy];
        d = -xx * yy * s
        tt * zz * (one - xx - yy - zz) +
        (xx + yy) * lambda2 +
        (one - xx - yy - zz) * (one - xx - yy) * fme2+
        zz * (one - xx - yy) * fmfc2 ;
        sumzG[ixy] += gw30[iz] / (d * d);
    }
}
```

GPAPE9-MPX	quadmath	gmp-mpfr
------------	----------	----------

21.3	1074.7	6486.7
------	--------	--------

[sec]

# How PGR works?



PGR is a “first” compiler for numerical calculations on FPGA chips.

- With PGR, we can implement  $N^2$  loops appeared in many-body simulations on FPGA chips.

# // PGDL sample for gravity calculations

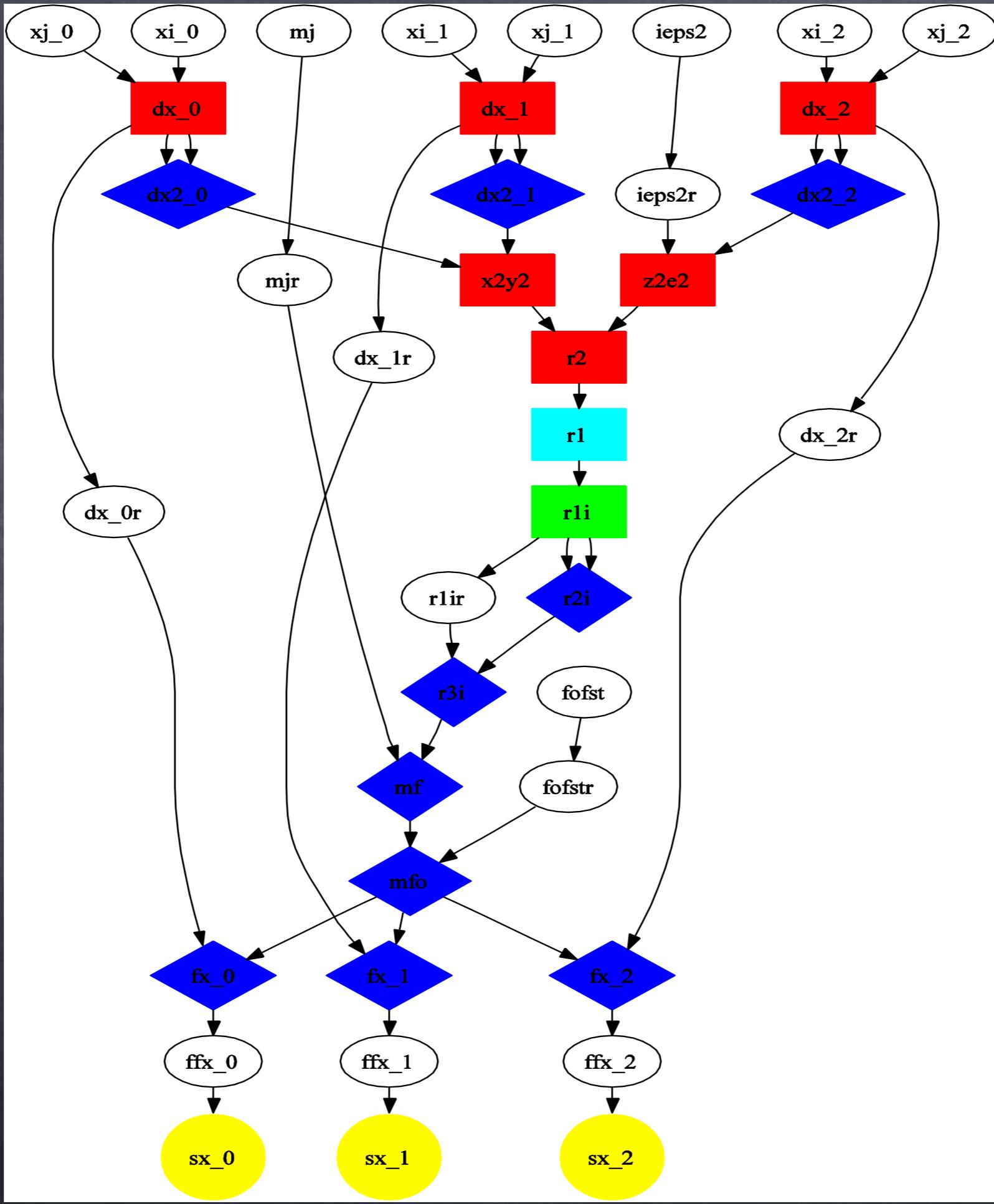
/NPIPE 16;

/NVMP 1;

/MEM xj[3] <= x[][] : fix(NPOS);  
/MEM mj <= m[] : log(NLOG,NMAN);  
/REG xi[3] <= x[][] : fix(NPOS);  
/REG ieps2 <= eps2 : log(NLOG,NMAN);  
/REG sx[3] => a[][] : fix(NACC);

/CONST zero <= 0.0 : log(NLOG,NMAN);

pg\_fix\_addsub(SUB,xi,xj,xij, NPOS, D\_FSU);  
pg\_conv\_ftol(xij,dx, NPOS,NLOG,NMAN, D\_FTL);  
pg\_log\_shift(1,dx,x2, NLOG);  
pg\_log\_unsigned\_add\_itp(x2[0],x2[1], x2y2, NLOG,NMAN,D\_LAD,NCUT);  
pg\_log\_unsigned\_add\_itp(x2[2],ieps2, z2e2, NLOG,NMAN,D\_LAD,NCUT);  
pg\_log\_unsigned\_add\_itp(x2y2,z2e2, r2, NLOG,NMAN,D\_LAD,NCUT);  
pg\_log\_shift(-1,r2, r1, NLOG);  
pg\_log\_muldiv(MUL,r2,r1,r3, NLOG, D\_LMU);  
pg\_log\_muldiv(SDIV,mj,r3,mf, NLOG, D\_LMU);  
pg\_log\_muldiv(MUL,mf,dx,fx, NLOG, D\_LMU);  
pg\_log\_expadd(fx,fxa, -NEXAD,NLOG,NMAN, D\_LMU);  
pg\_conv\_ltof(fxa,ffx, NLOG,NMAN,NFOR, D\_LTF);  
pg\_fix\_smaccum(ffx,sx, NFOR,NACC, D\_FAC);



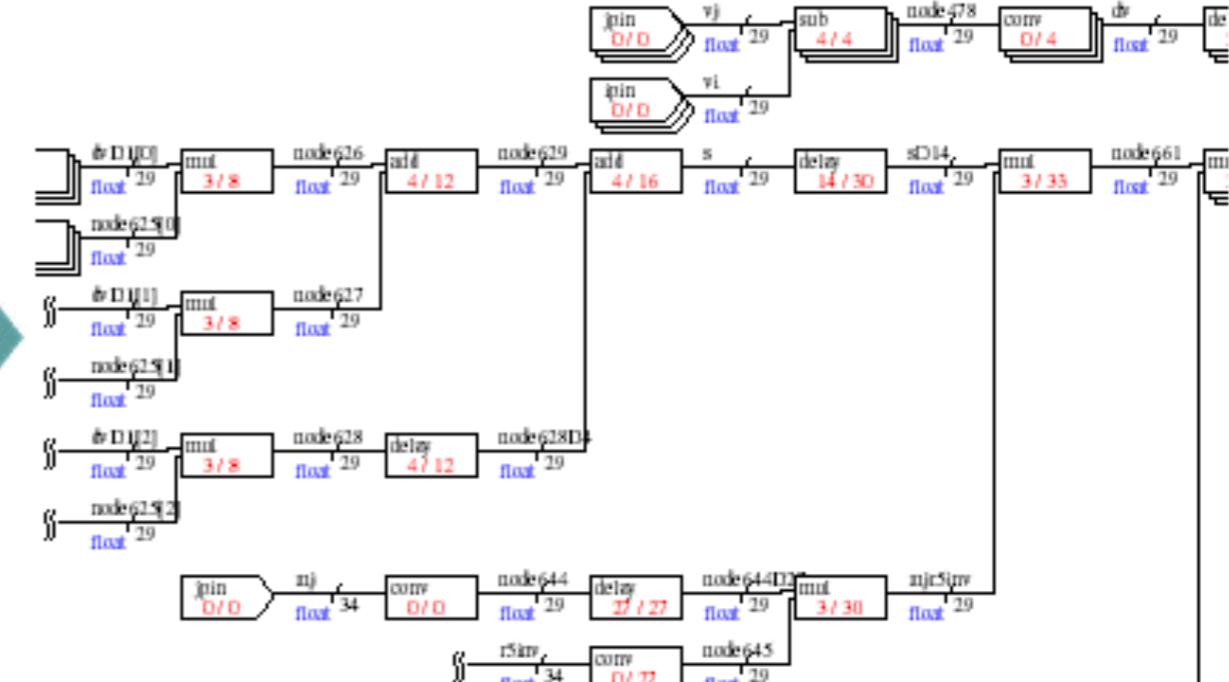
# 相互作用記述によるパイプライン実装

- やりたいのは「カスタムGRAPE」の実装
  - FPGAベース(PGPG, PGR), GRAPE-DR, LSUMP….
  - 粒子間相互作用の表現 (=演算カーネル)
- FPGAを専用計算機として利用
- MPのようにプログラマブル化?

```
int10      jshift (int)ipin; // (the library handle
int64      acc[3]  fout;
int32      jerk[3] fout;

// force calculation
dx       = (float34.23)(xj - xi);
r2       = dx * dx + epsi2;
r5inv   = (float34.23)pg_pow(r2, -5, 2, 9);
r3inv   = r5inv * r2;
r1inv   = r3inv * r2;
acc     += (int64)((mj * r3inv * dx) << fshift);

// jerk calculation
dv       = (float29.18)(vj - vi);
s        = (dv * (float29.18)dx);
mjr5inv = (float29.18)mj * (float29.18)r5inv;
j1      = s * mjr5inv * (float29.18)dx;
j1a     = j1 << 1;
```



# OpenCLについて



- 並列計算APIの標準規格
  - AMD, Apple, IBM, Intel, NVIDIA etc...
  - CPU, GPU, CellBE, DSP, FPGA etc...
- アクセラレータのプログラミングモデル
  - ホスト計算機から「デバイス」の機能を呼び出す
    - デバイスで動作するコードを「カーネル」と呼ぶ
    - CUDAと本質的な差はない
  - SIMD的な動作するスレッド群の並列化
  - 演算の明示的なベクトル化

# OpenCLの利点と欠点

## • 利点

- 各社のデバイスで動作する
- CPUとGPUを同時扱うことが可能
  - 従来: SSEベクトル化 + OpenMP/pthread + MPI
  - OpenCL時代:
    - OpenCL(ベクトル変数 + スレッド) + MPI
    - (OpenCL+OpenMP) + MPI

## • 欠点

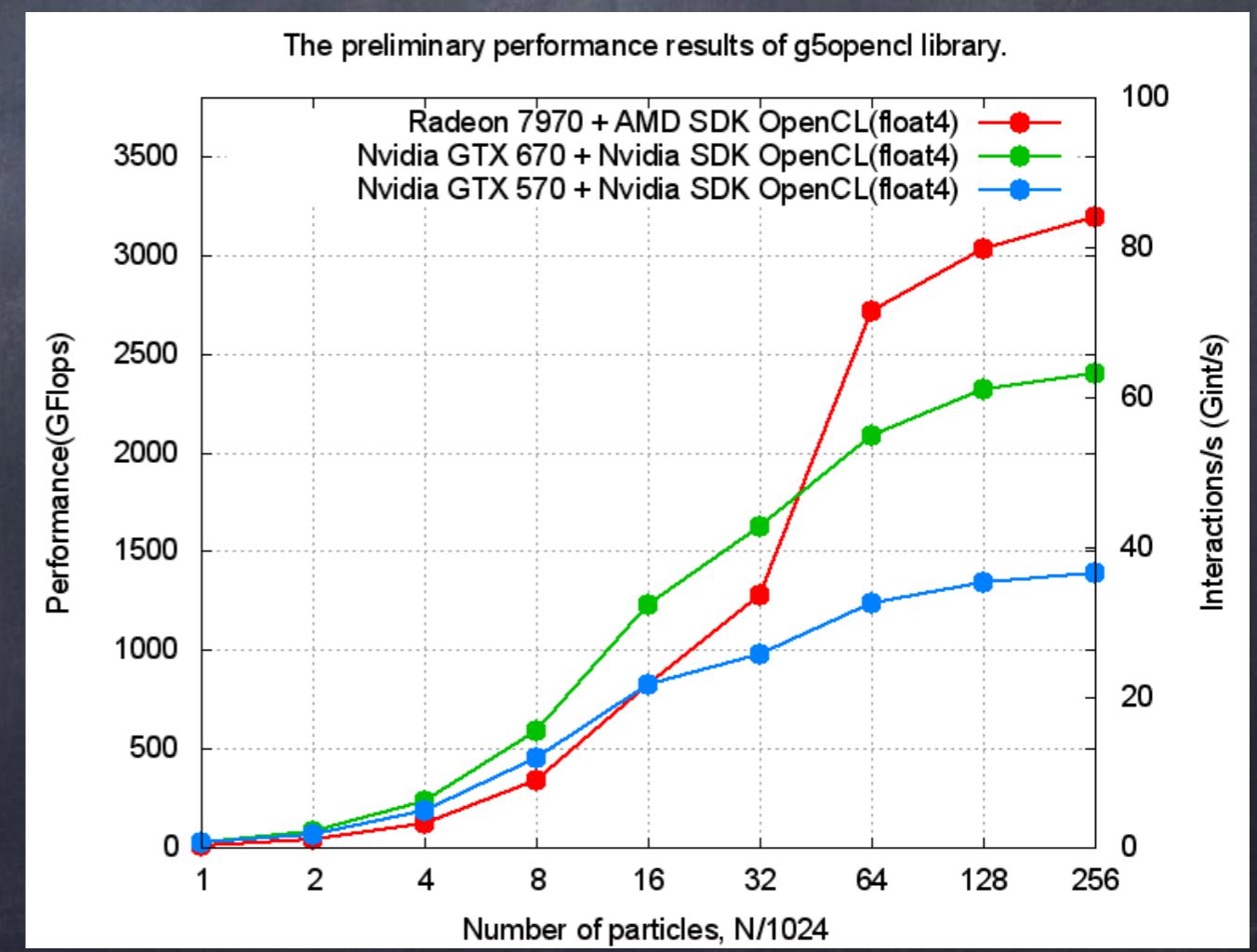
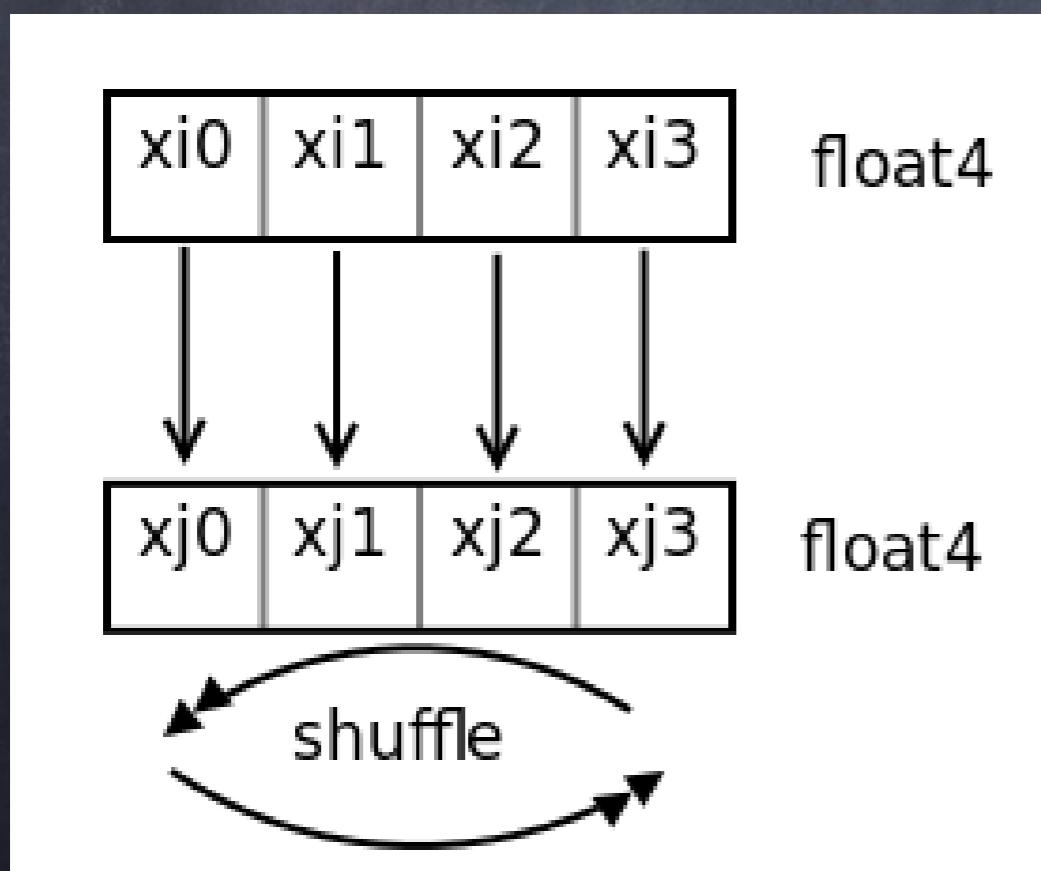
- より煩雑なプログラミング: C++ APIは簡単
- NVIDIAのデバイスでは性能が落ちる場合あり

# OpenCLによるFPGA利用

- Altera社, Xilinx社ともサポートしている
  - A社のほうが早期から
- OpenCLとCUDAの違い
  - カーネル記述が独立
  - OpenCLのほうがよりフレキシブル
- Alteraの場合：“aoc kernel.cl”
  - FPGA回路の生成、論理合成、配置配線
  - 時間が結構かかる
  - 実行には評価用ボードが必要

# Altera SDK for OpenCLテスト(1)

- 単精度で $N^2$ により重力とポテンシャル計算するカーネル
- 粒子の座標データはfloat4で保持 (SoA)
- スレッドあたり4粒子の相互作用を計算



# Altera SDK for OpenCL テスト (2)

- ・ 単純コンパイル：約 140 分
  - ・ LU 26%, DSP 46, clock 240 MHz
- ・ 最適化コンパイル(S1CU2)：約 4 時間
  - ・ LU 59%, DSP 356, clock 230 MHz
- ・ 最適化コンパイル(S2CU1)：約 4 時間30分
  - ・ LU 56%, DSP 354, clock 198 MHz
- ・ 簡単なカーネル：LU 15%, 290MHz

# まとめ

- FPGAの天体物理学計算での応用
  - カスタム“GRAPE”を実現する手段として
  - SPH専用計算機を実現
- プログラマブル計算機の実装
  - GRAPE-MPX, GRAPE9-MPX
- どのようにプログラミングするのか?
  - PGPG, PGR, PGPG2, LSUMP…
- OpenCLによるFPGA設計は可能になりつつ
  - 性能評価についてはこれから