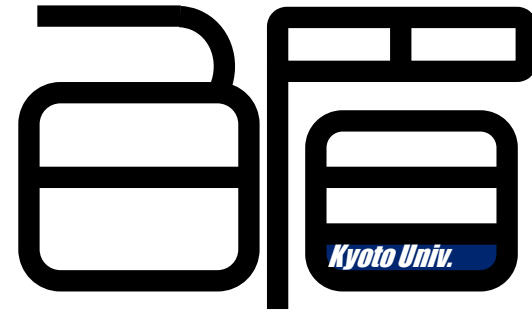
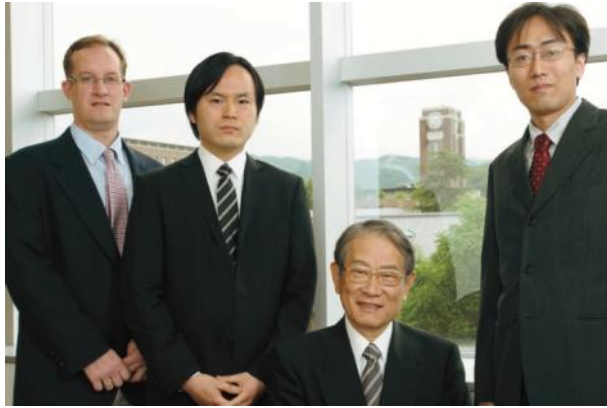


京都大学 白眉センター

Since 2010



学問の新たな潮流を拓く、
広い視野、柔軟な発想を求む！

- どんな分野でもOK
- 5年間、好きな研究をしてOK
- 毎年20人採用

第24回理論懇親会：理論天文学・宇宙物理学の革新
2011/11/07



プログラミングという作業の革新

京都大学白眉センター
村主崇行

check!



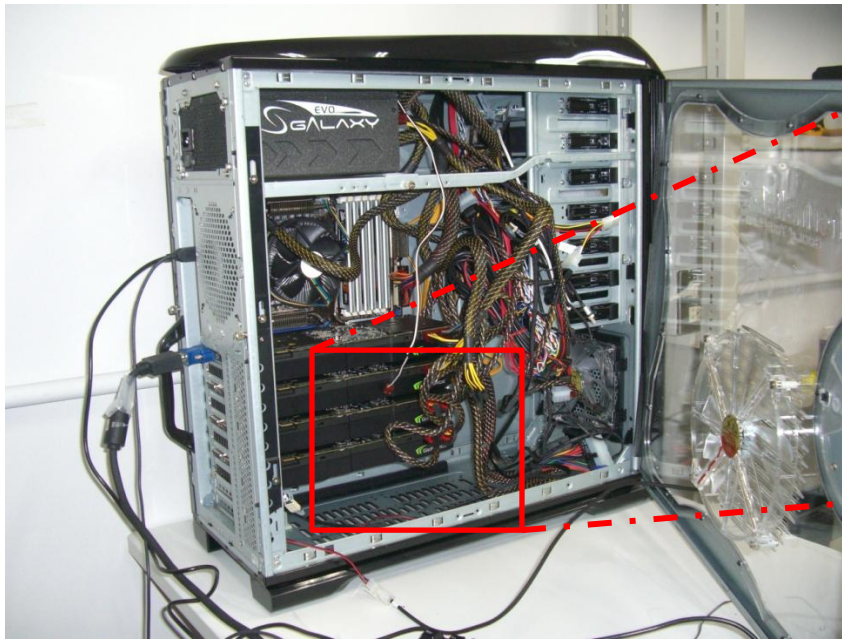
<http://paraiso-lang.org/wiki/>

quick start guide

Install [Haskell Platform](#) and [git](#), then type

```
> git clone git@github.com:nushio3/Paraiso.git
> cd Paraiso/
> cabal install
> cd examples/Life/           #Conway's game of life example
> make lib
> ls output/OM.txt
output/OM.txt                 #this is analysis result for dataflow graph
> ls dist/
Life.cpp Life.hpp            #an OpenMP implementation
> ls dist-cuda/
Life.cu Life.hpp             #a CUDA implementation
> cd ../Hydro/                #hydrodynamics simulator example
> make lib                    #this takes half a minute or so
> ls output/; ls dist/; ls dist-cuda/    #same as above
```

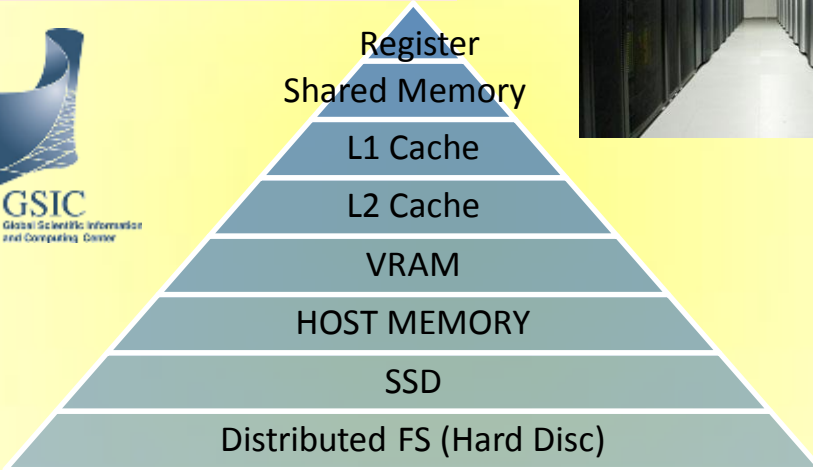
2009年8月@うちの教室
8個のGPUを搭載した1台のパソコン
単精度演算性能**7.3Tflops**



GPGPU: General-Purpose Computation on GPUs

M. Harris et al (2002) who coined the name

演算器数 1'892'352
言語: CUDA



Green500 4位 958MFlops/W
Top500 5位 1192TFlops


NACC
Nagasaki Advanced Computing Center

演算器の数 = 115,200
プログラミング言語: OpenCL

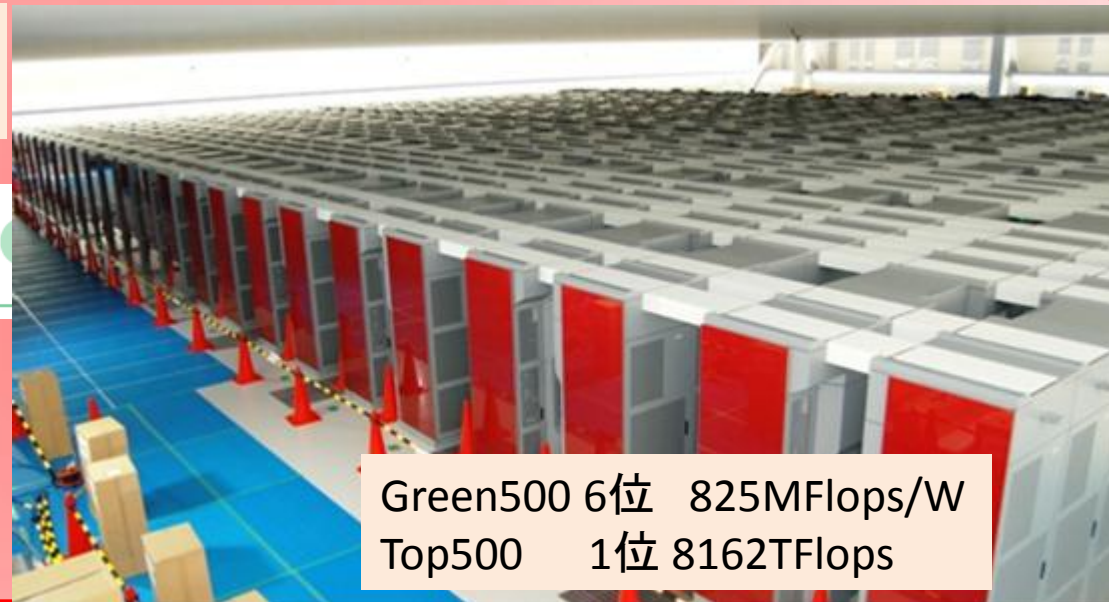


Green500 3位 1376MFlops/W
(47TFlops)

浮動小数演算器の数 = 4,386,816
プログラミング言語: FORTRAN

 計算科学研究機構
Advanced Institute for Computational Science

Top500/Green500 as of June 2011



Green500 6位 825MFlops/W
Top500 1位 8162TFlops

HP Project Moonshot



2800枚ものARM(携帯電話向けCPU)を組み込むことで、従来のサーバーシステムと比べ、最大で89%のエネルギー節約、94%の設置面積縮小、63%のコスト削減を目指す。

今までも、これからも

- より並列度の高い計算機を使っていく
- 新しいプログラミング言語を覚えさせられる
- 新しい並列プログラミングのスタイルを覚えさせられる

The Problem

- 僕らが書くコードは美しく長い

```

#ifdef USE_MP:[]
global__ void communicate_gather_kernel_y
(int displacement_int_inc, Real displacement_real_inc, Real relative_velocity_inc,
 int displacement_int_dec, Real displacement_real_dec, Real relative_velocity_dec,
 Real *buf_inc, Real *buf_dec, Real *density, Real *velocity_x, Real *velocity_y, Real *velocity_z,
 Real *pressure, Real *magnet_x, Real *magnet_y, Real *magnet_z ) {
const int kUnitSizeY = gSizeX * gMarginSizeY * gSizeZ;

CUSTOM_CRYSTAL_MAP(addr, kUnitSizeY) {
int sx, sy, sz;
depack(addr, gSizeX, gMarginSizeY, sx, sy, sz);
int inc_x0 = (sx + displacement_int_inc ) % gSizeX;
int inc_x1 = (sx + displacement_int_inc + 1) % gSizeX;
int dec_x0 = (sx - displacement_int_dec - 1 + gSizeX) % gSizeX;
int dec_x1 = (sx - displacement_int_dec + gSizeX) % gSizeX;
Real val_inc0 = density[ enpack(gSizeX, gSizeY, inc_x0, gSizeY - 2 * gMarginSizeY + sy, sz) ];
Real val_inc1 = density[ enpack(gSizeX, gSizeY, inc_x1, gSizeY - 2 * gMarginSizeY + sy, sz) ];
Real val_dec0 = density[ enpack(gSizeX, gSizeY, dec_x0, gMarginSizeY + sy, sz) ];
Real val_dec1 = density[ enpack(gSizeX, gSizeY, dec_x1, gMarginSizeY + sy, sz) ];
buf_inc[0 * kUnitSizeY + addr] = (Real(1)-displacement_real_inc) * val_inc0 + displacement_real
inc * val_inc0
;
buf_dec[0 * kUnitSizeY + addr] = displacement_real_dec * val_dec0 + (Real(1)-displacement_real
dec) * val_dec0
;
}

CUSTOM_CRYSTAL_MAP(addr, kUnitSizeY) {
int sx, sy, sz;
depack(addr, gSizeX, gMarginSizeY, sx, sy, sz);
int inc_x0 = (sx + displacement_int_inc ) % gSizeX;
int inc_x1 = (sx + displacement_int_inc + 1) % gSizeX;
int dec_x0 = (sx - displacement_int_dec - 1 + gSizeX) % gSizeX;
int dec_x1 = (sx - displacement_int_dec + gSizeX) % gSizeX;
Real val_inc0 = velocity_x[ enpack(gSizeX, gSizeY, inc_x0, gSizeY - 2 * gMarginSizeY + sy, sz) ];
Real val_inc1 = velocity_x[ enpack(gSizeX, gSizeY, inc_x1, gSizeY - 2 * gMarginSizeY + sy, sz) ];
Real val_dec0 = velocity_x[ enpack(gSizeX, gSizeY, dec_x0, gMarginSizeY + sy, sz) ];
Real val_dec1 = velocity_x[ enpack(gSizeX, gSizeY, dec_x1, gMarginSizeY + sy, sz) ];
buf_inc[1 * kUnitSizeY + addr] = (Real(1)-displacement_real_inc) * val_inc0 + displacement_real
inc * val_inc0
- relative_velocity_inc ;
buf_dec[1 * kUnitSizeY + addr] = displacement_real_dec * val_dec0 + (Real(1)-displacement_real
dec) * val_dec0
+ relative_velocity_dec ;
}

CUSTOM_CRYSTAL_MAP(addr, kUnitSizeY) {
int sx, sy, sz;
depack(addr, gSizeX, gMarginSizeY, sx, sy, sz);
int inc_x0 = (sx + displacement_int_inc ) % gSizeX;
int inc_x1 = (sx + displacement_int_inc + 1) % gSizeX;
int dec_x0 = (sx - displacement_int_dec - 1 + gSizeX) % gSizeX;
int dec_x1 = (sx - displacement_int_dec + gSizeX) % gSizeX;
Real val_inc0 = velocity_y[ enpack(gSizeX, gSizeY, inc_x0, gSizeY - 2 * gMarginSizeY + sy, sz) ];
Real val_inc1 = velocity_y[ enpack(gSizeX, gSizeY, inc_x1, gSizeY - 2 * gMarginSizeY + sy, sz) ];
Real val_dec0 = velocity_y[ enpack(gSizeX, gSizeY, dec_x0, gMarginSizeY + sy, sz) ];
Real val_dec1 = velocity_y[ enpack(gSizeX, gSizeY, dec_x1, gMarginSizeY + sy, sz) ];
buf_inc[2 * kUnitSizeY + addr] = (Real(1)-displacement_real_inc) * val_inc0 + displacement_real
inc * val_inc0
;
buf_dec[2 * kUnitSizeY + addr] = displacement_real_dec * val_dec0 + (Real(1)-displacement_real
dec) * val_dec0
;
}

```

- 美しい繰り返しパターン

- 結晶質シリケートのような美
- このような美しいものをたくさん読み書きできるのは一種の幸福、時代性的快感でさえある

- それはプログラマーの求めるべき幸福ではない

check!



<http://paraiso-lang.org/wiki/>

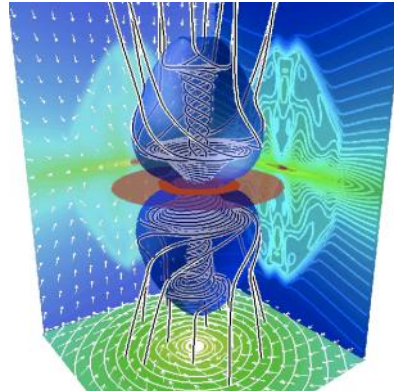
Paraiso計画

偏微分方程式の陽解法のための
コード生成 &
自動チューニングライブラリ

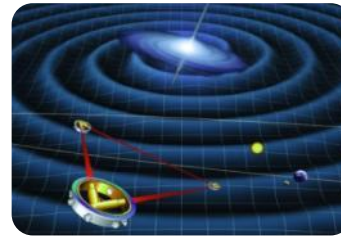
どのような計算が(Paraisoの)対象か？



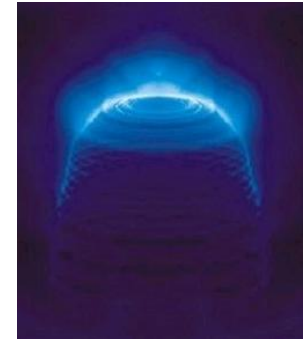
流体力学
(気体・液体)



磁気流体力学
(プラズマ)



一般相対性理論
(時空・座標)



輻射輸送
(光の放出・反射・
吸収・伝搬)

- 宇宙物理の問題のうち「双曲型・楕円型偏微分方程式」に分類され、「陽解法」に属するアルゴリズムで解くことができるもの。
- まあ、空間をマス目に切って、隣同士と情報をやりとりする手法。
- 計算機の並列度があがっても(weak scalingは)やりやすい



ある分野の計算に特化した 自動コード生成や自動チューニング

Problem

Code Generator & Automated Tuning

高速フーリエ変換

FFTW

行列計算

ATLAS

信号処理

SPIRAL

偏微分方程式
の陽的解法

Paraiso

偏微分方程式の陽解法(の一樣メッシュ部分) は少数の基本操作の組み合わせで書ける

5種類、9つくらい

```

data Inst vector gauge
= Imm Dynamic
| Load Name
| Store Name
| Reduce R.Operator
| Broadcast
| Shift (vector gauge)
| LoadIndex (Axis vector)
| Arith A.Operator

instance Arity (Inst vector gauge) where
arity a = case a of
  Imm _      -> (0,1)
  Load _     -> (0,1)
  Store _    -> (1,0)
  Reduce _   -> (1,1)
  Broadcast -> (1,1)
  Shift _    -> (1,1)
  LoadIndex _ -> (0,1)
  Arith op   -> arity op
    
```

Imm

即値をロードする

LoadIndex, LoadSize

配列の添字やサイズを取得

Load (グラフの始点)

名前付き変数からロードする

Store (グラフの終点)

名前付き変数に書きこむ

Reduce

配列をスカラー値に

Broadcast

スカラー値を配列に

Shift

配列全体を少しずらす

Arith

演算を行う

arith:配列の同じ位置の要素どうしの演算を行う

```
c <- arith add <- a,b
```

```
do j = 1, N-5
  do i = 1, M-1
    c(i,j) = a(i,j)+b(i,j)
  end do
end do
```

```
b <- arith sin <- a
```

```
do j = 1, N-5
  do i = 1, M-1
    b(i,j) =sin(a(i,j))
  end do
end do
```

```
d <- arith select <- a,b,c
```

```
do j = 1, N-5
  do i = 1, M-1
    if(a(i,j))then
      d(i,j)=b(i,j)
    else
      d(i,j)=c(i,j)
    end if
  end do
end do
```

shift:配列に入っているデータの 位置をずらす

```
b <- shift (1,5) <- a
```

```
do j = 1, N-5  
  do i = 1, M-1  
    b(i+1,j+5) = a(i,j)  
  end do  
end do
```

reduce:配列をひとつの値に変える
broadcast:値を配列にばらまく

```
b <- reduce MIN <- a
```

```
c <- broadcast <- b
```

```
b = a(0,0)
do j = 1, N
  do i = 1, M
    b = min(b, a(i,j))
  end do
end do
```

```
do j = 1, N
  do i = 1, M
    c(i,j) = b
  end do
end do
```

`imm` :定数値を設定

`loadIndex`:配列の添え字を取得

`loadSize` :配列のサイズを取得

```
a <- imm 4.2
```

```
do j = 1, N
  do i = 1, M
    a(i,j) = 4.2
  end do
end do
```

```
b <- loadIndex
```

```
do j = 1, N
  do i = 1, M
    b_0(i,j) = i
    b_1(i,j) = j
  end do
end do
```

```
c <- loadSize
```

```
c_0 = M
c_1 = N
```


load:名前つき変数から読み込み
save:名前つき変数へ書き込み

```
a <- load "density"
```

```
save "density" <- b
```

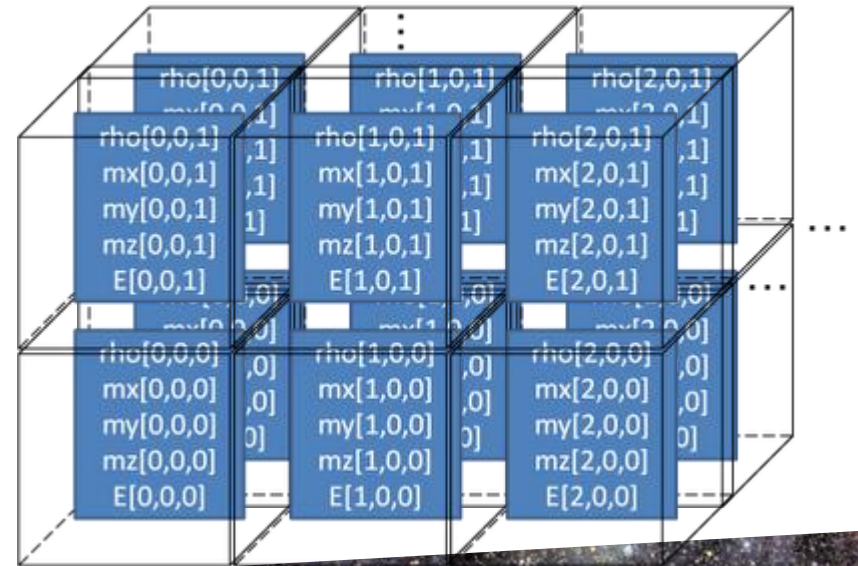
```
do j = 1, N  
  do i = 1, M  
    a(i,j) = density(i,j)  
  end do  
end do
```

```
do j = 1, N  
  do i = 1, M  
    density(i,j) = b(i,j)  
  end do  
end do
```

Orthotope Machine

- 多次元配列状にならんだレジスタを持つ巨大なベクトル計算機
- 実数テンソル計算に対応する仮想マシン
- 演算命令は基本的に全セルに並列に作用
- 隣のセルからロードする命令なども

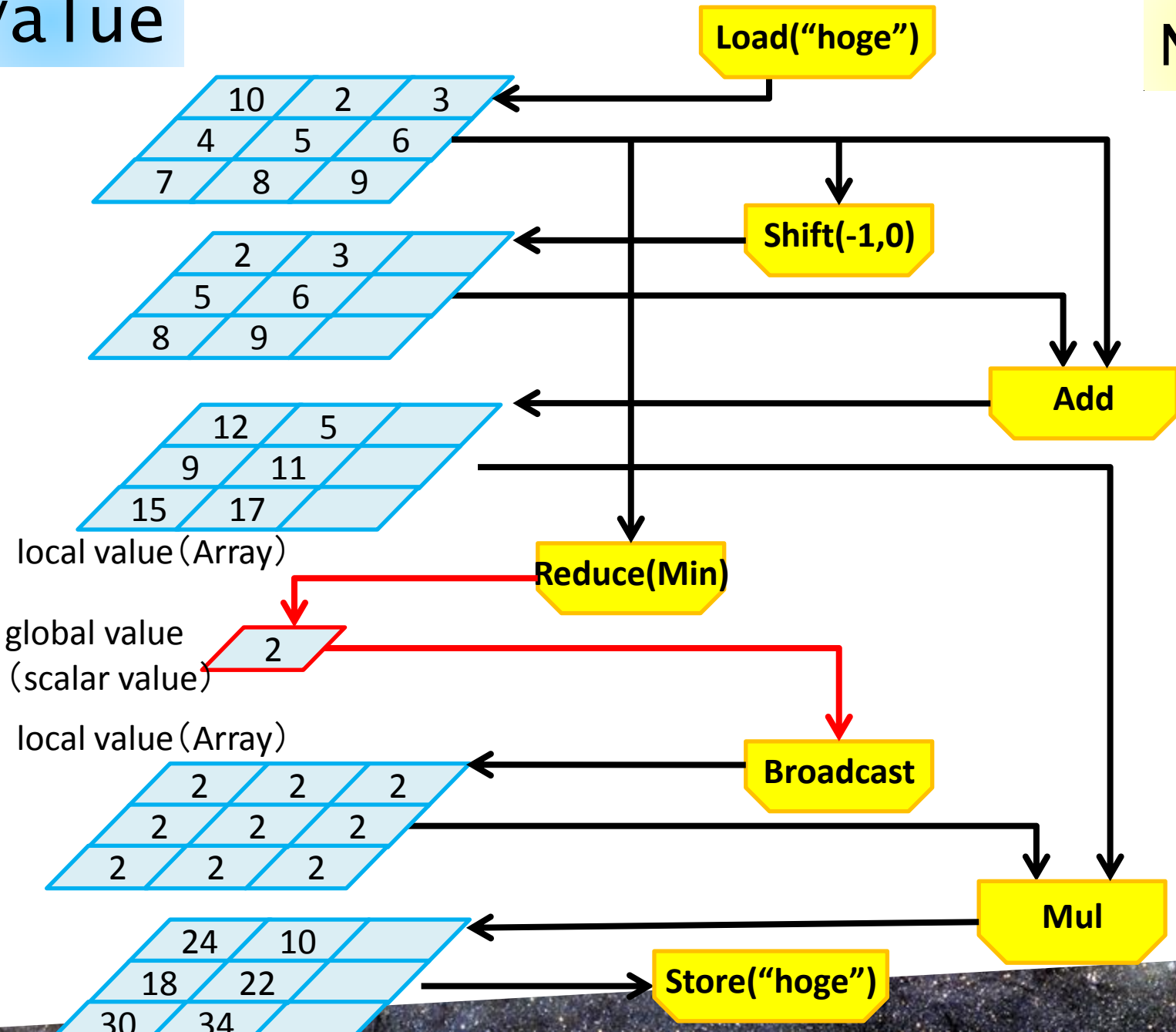
実行するための仮想マシンではなく、
データフローグラフを構築するための仮想マシン



データフローグラフの例

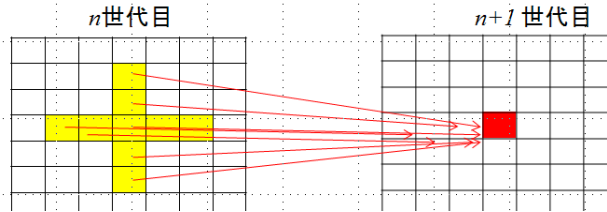
Nvalue

NInst

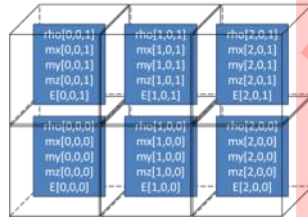


Paraisoの全貌

離散化方程式記述言語：
Discrete PDE Language



多次元配列仮想マシン：
Orthotope Machine (OM)



結果



基礎方程式

さしあたり人手

アルゴリズム
の記述

OMビルダー

OM上の
コード

OMコンパイラ

実マシン上の
コード

既存コンパイラ

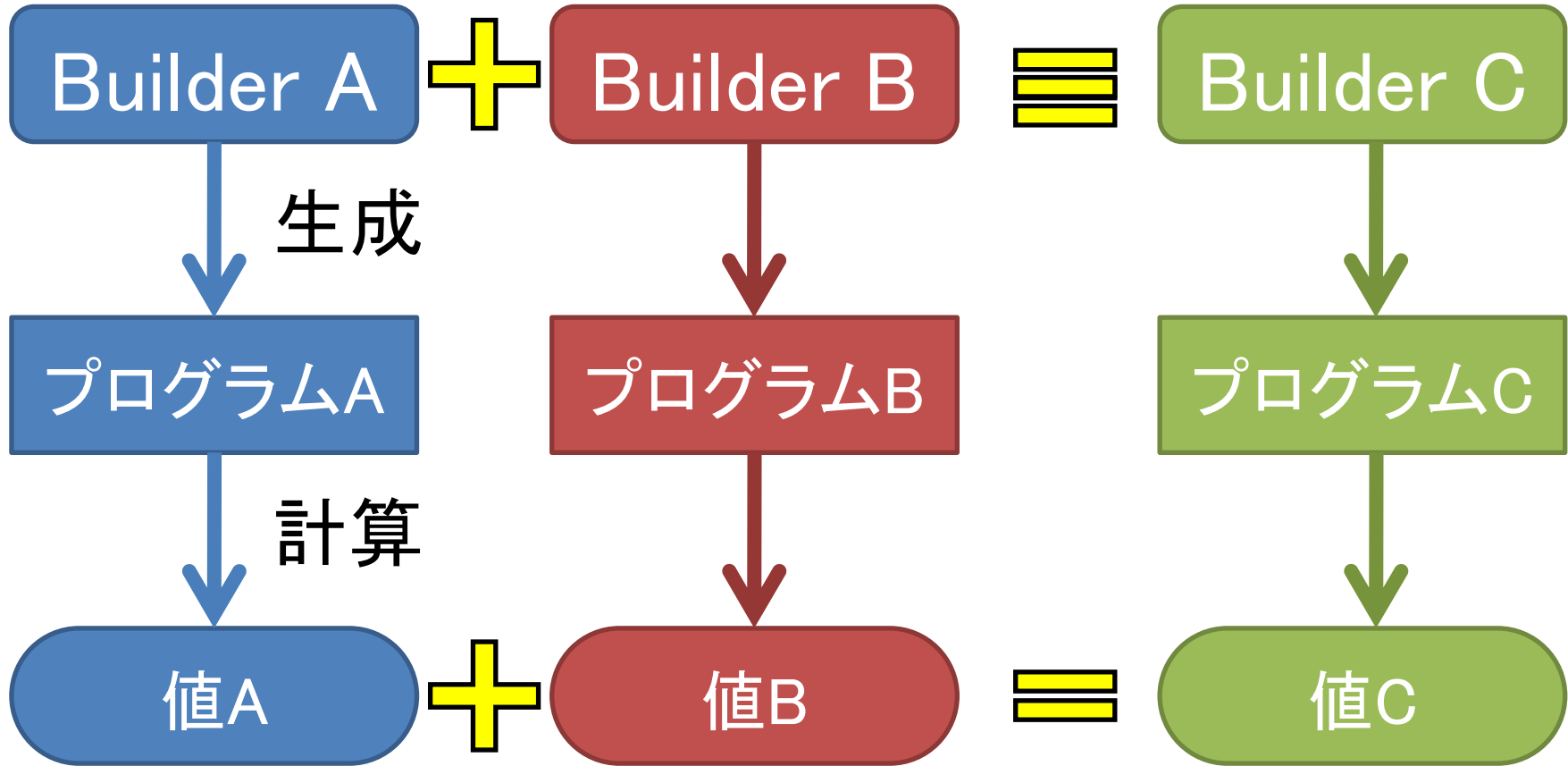
実マシン上の
実行ファイル

- Paraisoプログラムはどうやって書くのか？

プログラミング言語Paraisoには 通常の意味でのフロントエンドがない

- Paraiso言語のソースは文字列じゃない
- LexerやParserはない
- Paraisoは関数型言語Haskellの中に埋め込まれたライブラリ
- **Builderモナド**とその**コンビネーター**でプログラムを記述する。

Builder同士の演算の定義とは？



「Builder A と builder B の和であるBuilder」とは、それが生成するプログラムが計算する値が、「Builder Aの生成するプログラムが計算する値」と「Builder Bの生成するプログラムが計算する値」の和になるようなものである。

typelevel-tensor

Einstein's notation

$$C_{ik} = A_{ij}B_{jk}$$

notation in standard
mathematics terminology

$$C_{ik} = \sum_{j=1}^3 A_{ij}B_{jk}$$

Notation in Haskell
using typelevel-tensor

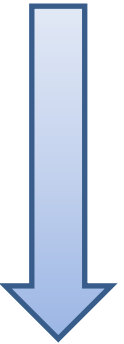
```
a :: Vec4 (Vec3 Double)
b :: Vec3 (Vec4 Double)
c = compose $ \i ->
      contract $ \j ->
        compose $ \k ->
          a!i!j * b!j!k
```

Implementation in C++

```
double a[4][3], b[3][4];
double c[4][4];
for (int i = 0; i < 4; ++i) {
  for (int k = 0; k < 4; ++k) {
    c[i][k] = 0;
    for (int j = 0; j < 3; ++j) {
      c[i][k] += a[i][j] * b[j][k];
    }
  }
}
```


実際に使っているところ

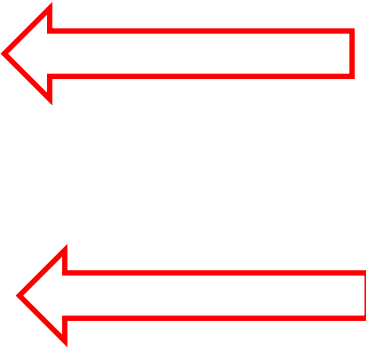
ただの数式に見えるが、各項はBuilderモナドであり、全体がOMグラフのジェネレータになっている



```
hllc :: Axis Dim -> Hydro BR -> Hydro BR -> B (Hydro BR)
hllc i left right = do
  densMid <- bind $ (density left + density right) / 2
  soundMid <- bind $ (soundSpeed left + soundSpeed right) / 2
  let
    speedLeft = velocity left !i
    speedRight = velocity right !i
  presStar <- bind $ max 0 $ (pressure left + pressure right) / 2 -
    densMid * soundMid * (speedRight - speedLeft)
  shockLeft <- bind $ velocity left !i -
    soundSpeed left * hllcQ presStar (pressure left)
  shockRight <- bind $ velocity right !i +
    soundSpeed right * hllcQ presStar (pressure right)
  shockStar <- bind $ (pressure right - pressure left
    + density left * speedLeft * (shockLeft - speedLeft)
    - density right * speedRight * (shockRight - speedRight)
  )
  / (density left * (shockLeft - speedLeft) -
    density right * (shockRight - speedRight) )
  lesta <- starState shockStar shockLeft left
  rista <- starState shockStar shockRight right
```

「流体っぽいもの」型クラスを定義

```
class Hydrable a where
  density    :: a -> BR
  velocity   :: a -> Dim BR
  velocity x =
    compose (\i -> momentum x !i / density x)
  pressure   :: a -> BR
  pressure x = (kGamma-1) * internalEnergy x
  momentum  :: a -> Dim BR
  momentum x =
    compose (\i -> density x * velocity x !i)
  energy     :: a -> BR
  energy x = kineticEnergy x + 1/(kGamma-1) * pressure x
  enthalpy   :: a -> BR
  enthalpy x = energy x + pressure x
  densityFlux :: a -> Dim BR
```



- 必要そうな物理量の定義を全部用意
- あとでDead Code Eliminationが消すから大丈夫

「流体っぽいもの」をApplicativeにする

```
instance Applicative Hydro where
  pure x = Hydro
  {densityHydro = x, velocityHydro = pure x, pressureHydro = x,
   momentumHydro = pure x, energyHydro = x, enthalpyHydro = x,
   densityFluxHydro = pure x, momentumFluxHydro = pure (pure x),
   energyFluxHydro = pure x, soundSpeedHydro = x,
   kineticEnergyHydro = x, internalEnergyHydro = x}
  hf <*> hx = Hydro
  {densityHydro          = densityHydro          hf $ densityHydro          hx,
   pressureHydro         = pressureHydro         hf $ pressureHydro         hx,
   energyHydro           = energyHydro           hf $ energyHydro           hx,
   enthalpyHydro         = enthalpyHydro         hf $ enthalpyHydro         hx,
   soundSpeedHydro      = soundSpeedHydro      hf $ soundSpeedHydro      hx,
   kineticEnergyHydro   = kineticEnergyHydro   hf $ kineticEnergyHydro   hx,
   internalEnergyHydro  = internalEnergyHydro  hf $ internalEnergyHydro  hx,
   velocityHydro        = velocityHydro        hf <*> velocityHydro        hx,
   momentumHydro        = momentumHydro        hf <*> momentumHydro        hx,
   densityFluxHydro     = densityFluxHydro     hf <*> densityFluxHydro     hx,
   energyFluxHydro      = energyFluxHydro      hf <*> energyFluxHydro      hx,
   momentumFluxHydro    =
      compose(\i -> compose(\j -> (momentumFluxHydro hf!!i!j)
                               (momentumFluxHydro hx!!i!j)))
```

- 結構たくさんある流体変数全体に一つの演算を施せるように！

隣り合う4マスを補間して 間の量を求める関数

```
interpolate :: Int -> Axis Dim -> Hydro BR -> B (Hydro BR, Hydro BR)
interpolate order i cell = do
  let shifti n = shift $ compose (\j -> if i==j then n else 0)
      a0 <- mapM (bind . shifti ( 2)) cell
      a1 <- mapM (bind . shifti ( 1)) cell
      a2 <- mapM (bind . shifti ( 0)) cell
      a3 <- mapM (bind . shifti (-1)) cell
      intp <- sequence $ interpolateSingle order <$> a0 <*> a1 <*> a2 <*> a3
```

- これ1つで、無数の流体変数全体を一気に処理
- 任意の次元、任意の方向に対応！
- 一発で書ける

4つの解の候補のなかから 場合分けに応じて正しいものを選ぶ

```
let selector a b c d =  
  select (0 `lt` shockLeft) a $  
  select (0 `lt` shockStar) b $  
  select (0 `lt` shockRight) c d  
mapM bind $ selector <$> left <*> lesta <*> rista <*> right
```

- これ1つで、無数の流体変数全体を一気に処理
- 任意の次元、任意の方向に対応！
- 一発で書ける

各方向ごとの計算結果を足し合わせ 全体の解を求める処理

```
proceedSingle :: Int -> BR -> Dim BR -> Hydro BR -> Hydro BR -> B (Hydro B
proceedSingle order dt dR cellF cells = do
  let calcWall i = do
        (lp,rp) <- interpolate order i cellF
        hllc i lp rp
    wall <- sequence $ compose calcWall
  foldl1 (.) (compose (\i -> (>=> addFlux dt dR wall i))) $ return cells
```

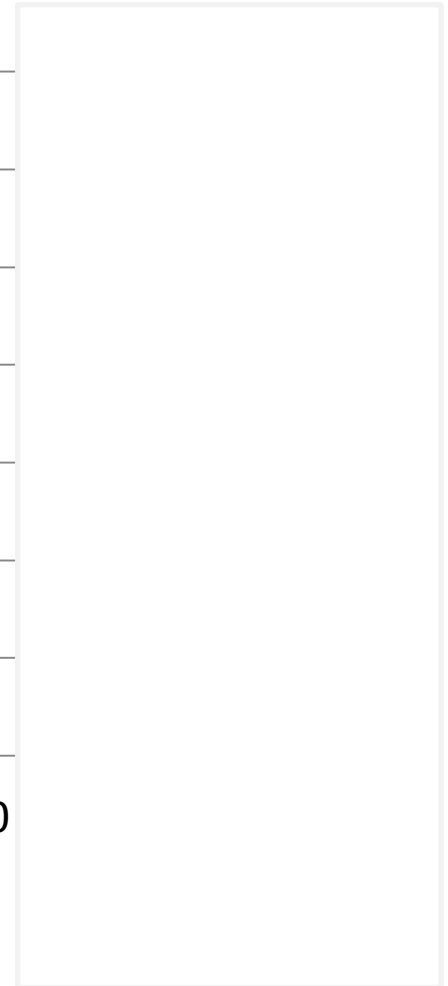
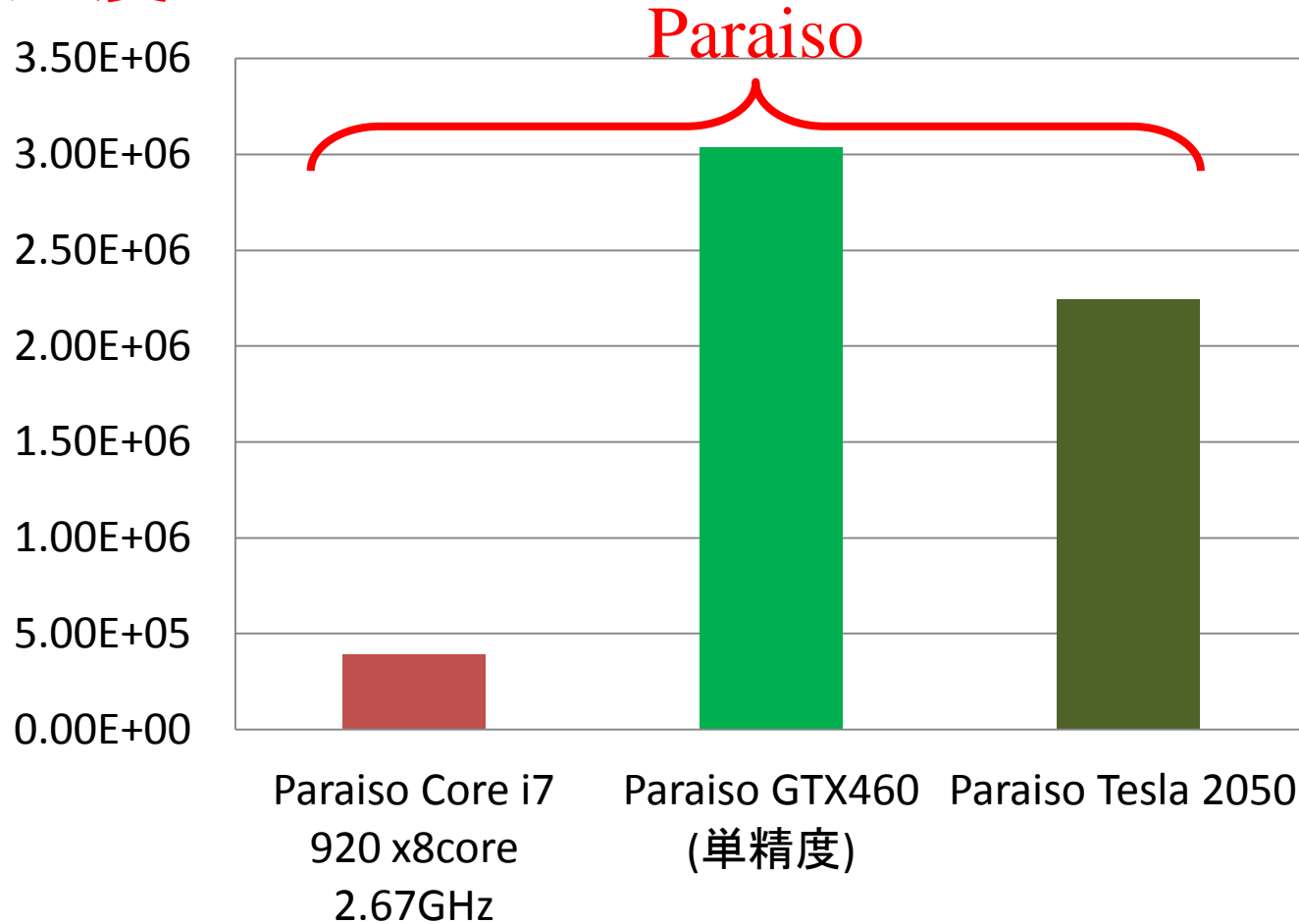
- これ1つで無数の流体変数全体を (ry
- 任意の次元、任意の方向に (ry
- モナド、Fold、演算子の部分適用などすごい
Haskellの楽しい機能を駆使
- 自分で後からみても正直読めない
- でもこんなに少ない行数で書ける！

Don't Repeat Yourself

- Paraisoには文字列フロントエンドがない
- コード生成器Builder自体が言語の第一級の対象
- **関数型言語の強力な利点！**
- コード生成器を自由に操れる
- DRY(同じことは2度書かない)原則をとことん追求できる

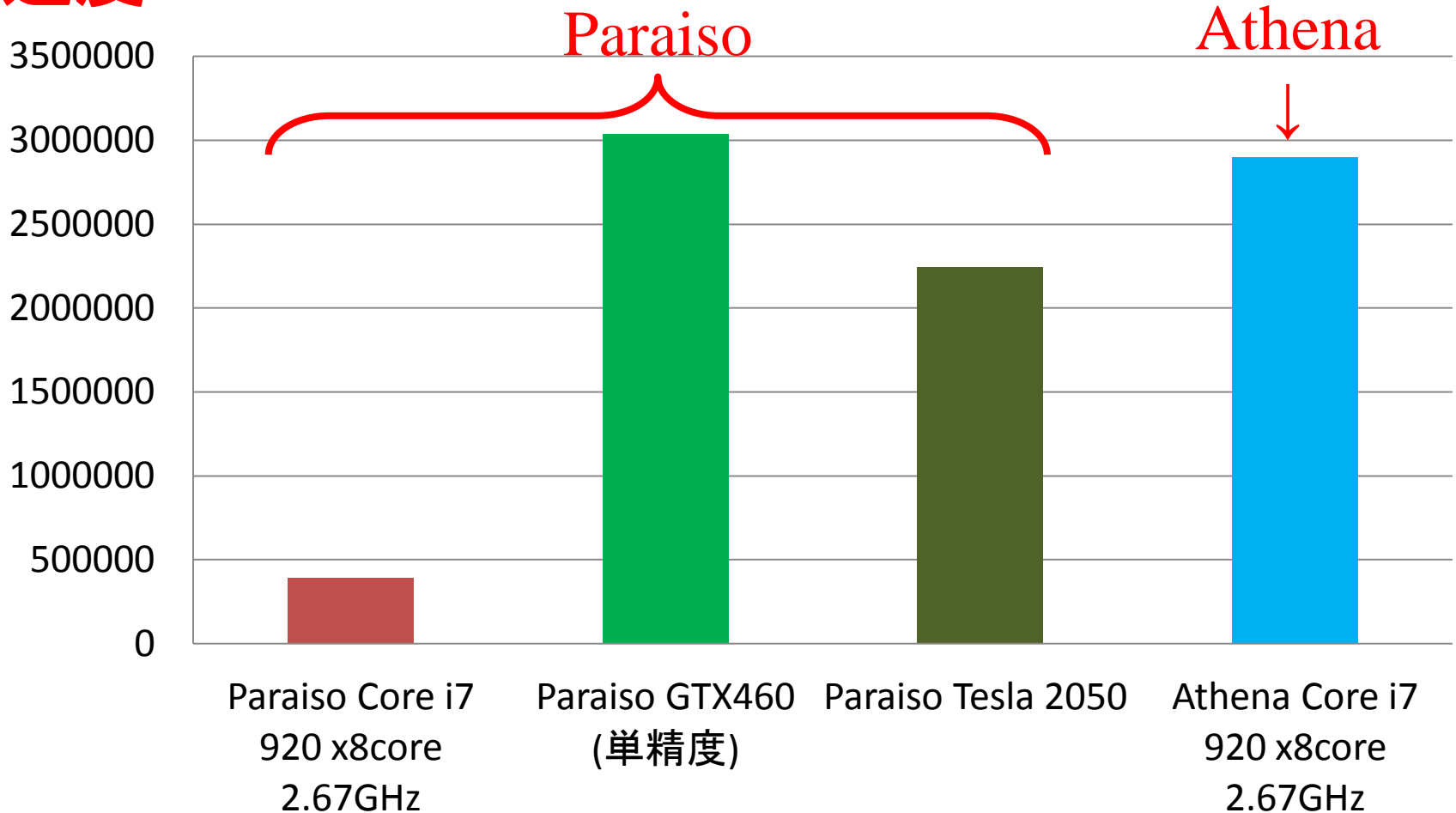
ベンチマーク結果

速度



ベンチマーク結果

速度



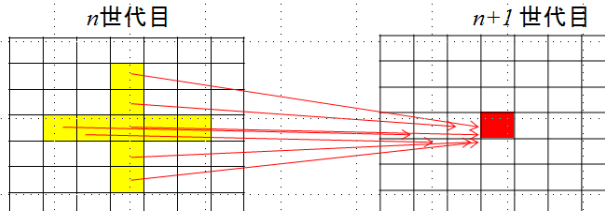
Athena : 惑星業界をはじめ広く使われている既存のコード。
同じ条件の2次元、流体HLLC、2次精度に設定して比較。

残念

自動生成しただけではだめで
自動チューニングが必要

解きたい式: $\frac{\partial U}{\partial t} + \nabla \cdot F = 0$

流体計算記述言語:
Discrete PDE Language



多次元配列仮想マシン:
Orthotope Machine (OM)
からのNativeコード生成



基礎方程式

さしあたり人手

アルゴリズム
の記述

OMビルダー

OM上の
コード

OMコンパイラ

実マシン上の
コード

既存コンパイラ

実マシン上の
実行ファイル

結果



コード生成部

OM グラフ

解析・最適化
済みOM

OMTrans

Plan

PlanTrans

Clariss

ClarissTrans

Native Code

解析と最適化

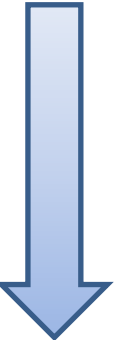
Analysis = 解析して注釈
(Annotation)をグラフにつける
Optimization 注釈にもとづきグラフ
を変換

Plan = コード生成戦略の決定

- どのくらいメモリを使うか？
- どこまでの計算を1つのサブ
ルーチンに収めるか？

Clariss

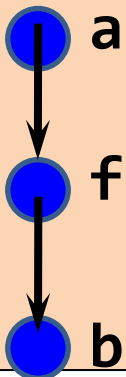
- C++やCUDAをカバーする 抽
象構文



Annotationの一例：計算結果をメモリに保持するか、捨てて再計算するか？

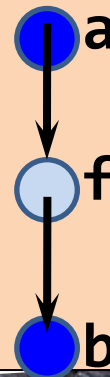
- メモリを食ってでも計算を減らしたほうが速い？
- 計算を多くしてでもメモリと帯域を節約した方がいい？

```
for(;;){
    f[i] = calc_f(a[i], a[i+1]);
}
for (;;){
    b[i] += f[i] - f[i-1];
}
```



演算量 $\sim N$
メモリ $\sim 3N$

```
for(;;){
    f0 = calc_f(a[i-1], a[i]);
    f1 = calc_f(a[i], a[i+1]);
    b[i] += f1 - f0;
}
```



演算量 $\sim 2N$
メモリ $\sim 2N$

例：Paraisoで実装された流体計算

- グラフのノード数 = 3958個
- レイアウトを自由に選べるValueは = 1908個
- 可能な実装は、それだけでも

→ 2^{1908} 通り

=2318631474140359897594479094137816650163390396354617107978538972914676911296289889528
94988789846447793390988399384716551223336856806783982602912691606248364445770172335039
54535729241917880311363490383137914861274921255128950712734788397408670521950919714209
83222926979177135181119534352143339906235134472215632092222013464750709343628667288853
94848451529803078779559205459073953255482226948670514566096452159327589352442445790848
16176470059329340736642337222850662358951938698298215645717772808920891115086440342006
478637177469672403326343875446350241918444483542305006944256通り

2011/09/08 00:08

```

interpolateSingle :: Int -> BR -> BR -> BR -> BR -> B (BR, BR)
interpolateSingle order x0 x1 x2 x3 =
  if order == 1
  then do
    return (x1, x2)
  else if order == 2
  then do
    d01 <- bind $ x1-x0
    d12 <- bind $ x2-x1
    d23 <- bind $ x3-x2
    let absmaller a b = select ((a*b) `le` 0) 0 $ select (abs a `lt` abs b) a b
        d1 <- bind $ absmaller d01 d12
        d2 <- bind $ absmaller d12 d23
        l <- bind $ x1 + d1/2
        r <- bind $ x2 - d2/2
    return ( Anot.add Alloc.Manifest <?> l, Anot.add Alloc.Manifest <?> r)
  else error $ show order ++ "th order spatial interpolation is not yet implemented"

```

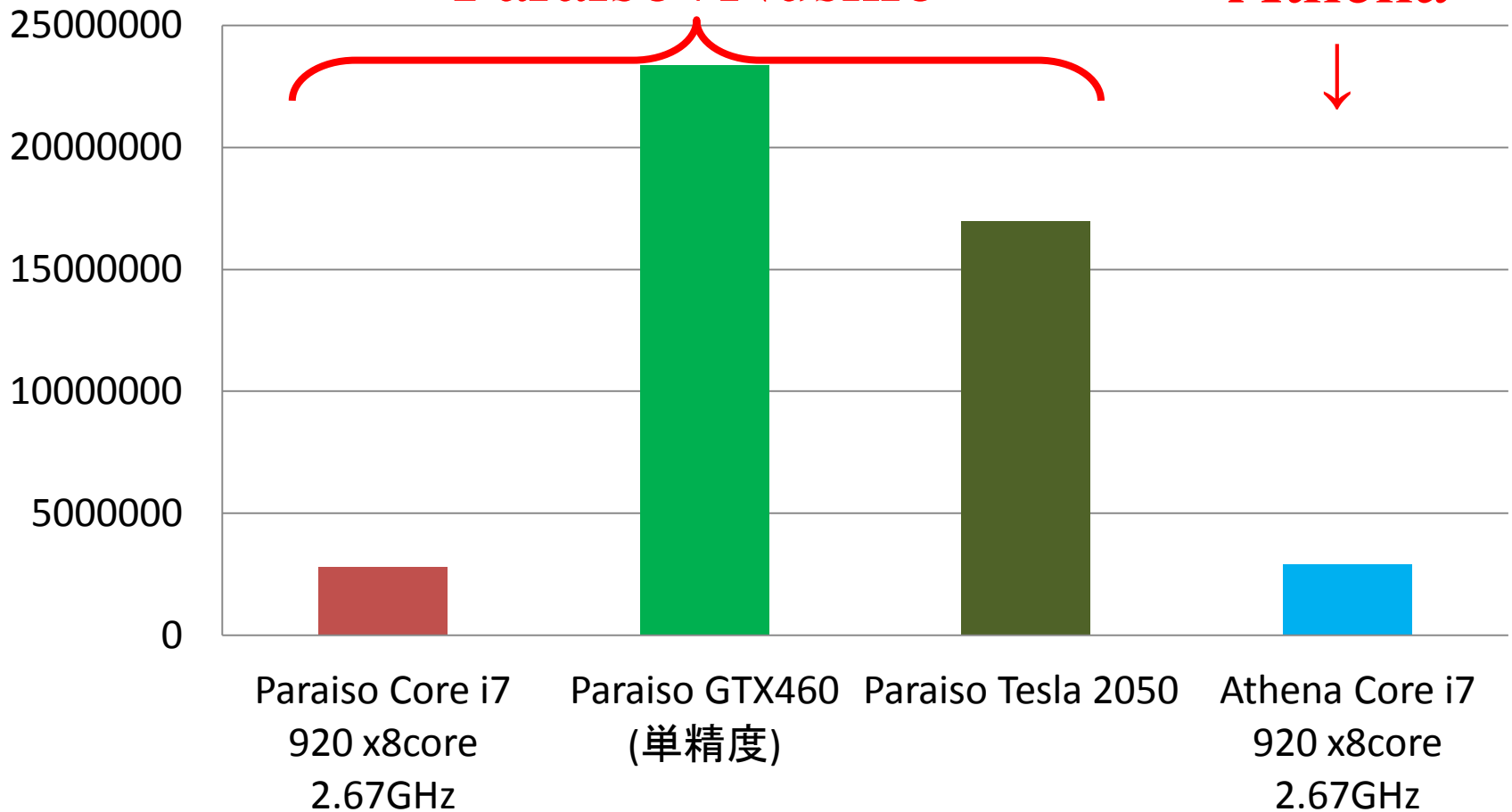
```

hllc :: Axis Dim -> Hydro BR -> Hydro BR -> B (Hydro BR)
hllc i left right = do
  densMid <- bind $ (density left + density right ) / 2
  soundMid <- bind $ (soundSpeed left + soundSpeed right) / 2
  let
    speedLeft = velocity left !i
    speedRight = velocity right !i
  presStar <- bind $ max 0 $ (pressure left + pressure right ) / 2 -
    densMid * soundMid * (speedRight - speedLeft)
  shockLeft <- bind $ velocity left !i -
    soundSpeed left * hllcQ presStar (pressure left)
  shockRight <- bind $ velocity right !i +
    soundSpeed right * hllcQ presStar (pressure right)
  shockStar <- bind $ (pressure right - pressure left
    + density left * speedLeft * (shockLeft - speedLeft)
    - density right * speedRight * (shockRight - speedRight) )
    / (density left * (shockLeft - speedLeft) -
    density right * (shockRight - speedRight) )
  lesta <- starState shockStar shockLeft left
  rista <- starState shockStar shockRight right
  let selector a b c d =
    (Anot.add Alloc.Manifest <?> ) $
    select (0 `!t` shockLeft) a $
    select (0 `!t` shockStar) b $
    select (0 `!t` shockRight) c d
  mapM bind $ selector <$> left <*> lesta <*> rista <*> right
  where

```

ベンチマーク結果(改)

速度

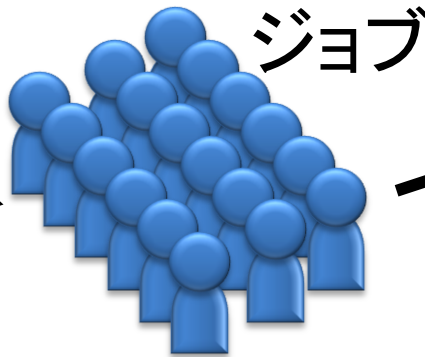


もっといろいろなパターンを試したい！

全自動チューニング全体像

コンピュータ

マネージャー



一定時間ごとに、終了しているジョブの成績を回収
いままでの全ての遺伝子の成績をもとに新種を生成してジョブを作り投入する

1つのジョブは1つの遺伝子をもとにプログラムを生成してコンパイル、実行

プログラムを実行正しく計算できているかを検証したうえでベンチマークを記録する

1秒ごとに1個のジョブを
昼夜休みなく投げる怪しい人

新種を作りだす三つの方法

- mutation

ATATATAAATTATATATATAAAAAAAAAAAAAAT

↓

ATATAGCAATTATATCTATAAAAAAGTGAAAAT

- cross

ATATATAAATTATATATATAAAAAAAAAAAAAAT

GGCCGCGCCCGCGCGCCCGCGCGCCCGGCGG

↓

ATATGCGAATTATATATACGCGCGCCCGGCGT

- triangulation

ATATATAAATTATATATATAAAAAAAAAAAAAAT

ATATATAAATTATATATATAAAAAAGTTAAAT

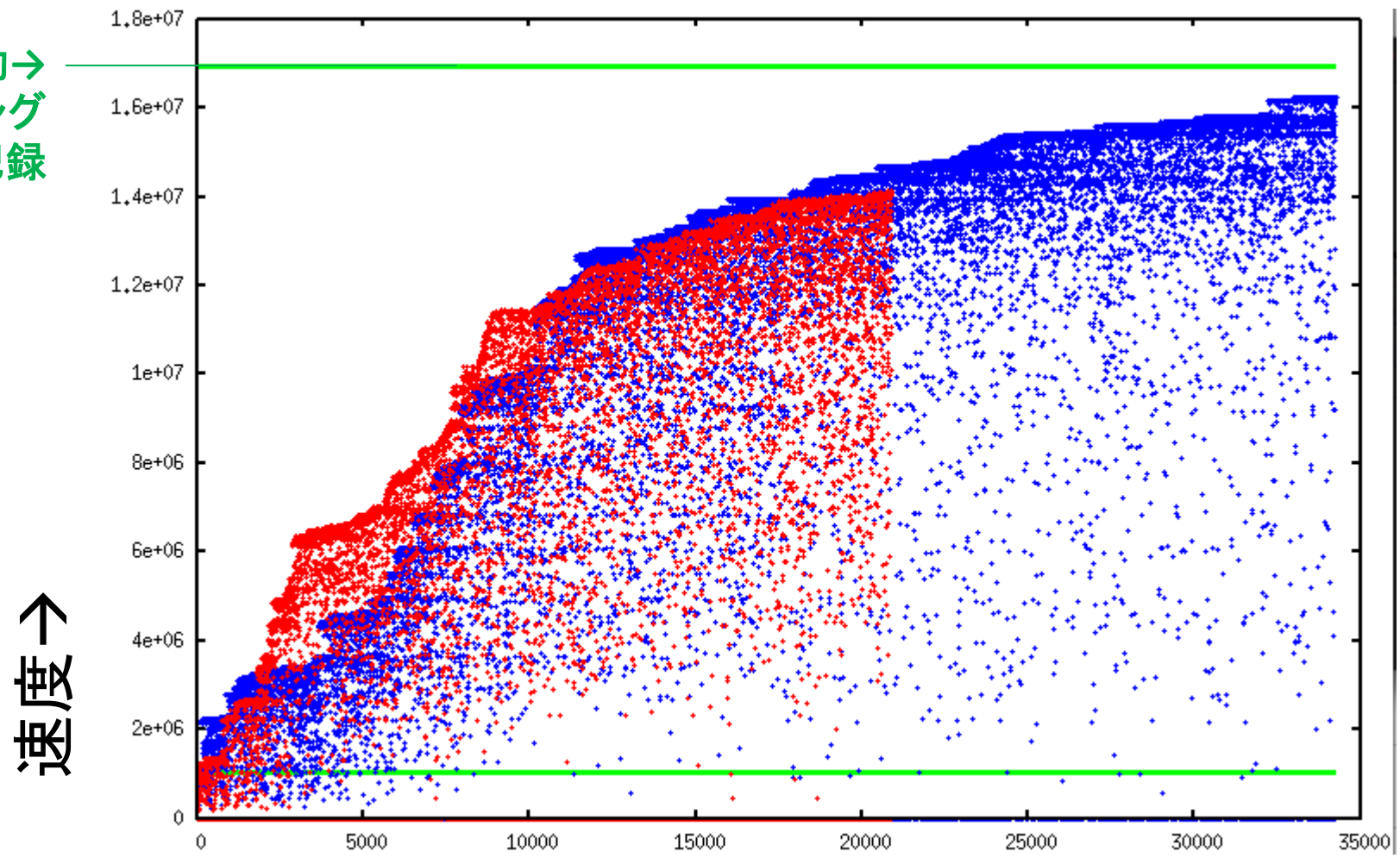
ATATAGCAATTATATCTATAAAAAAAAAAAAAAT

↓

ATATAGCAATTATATCTATAAAAAAGTTAAAT

実験1

手動→
チューニング
の最高記録

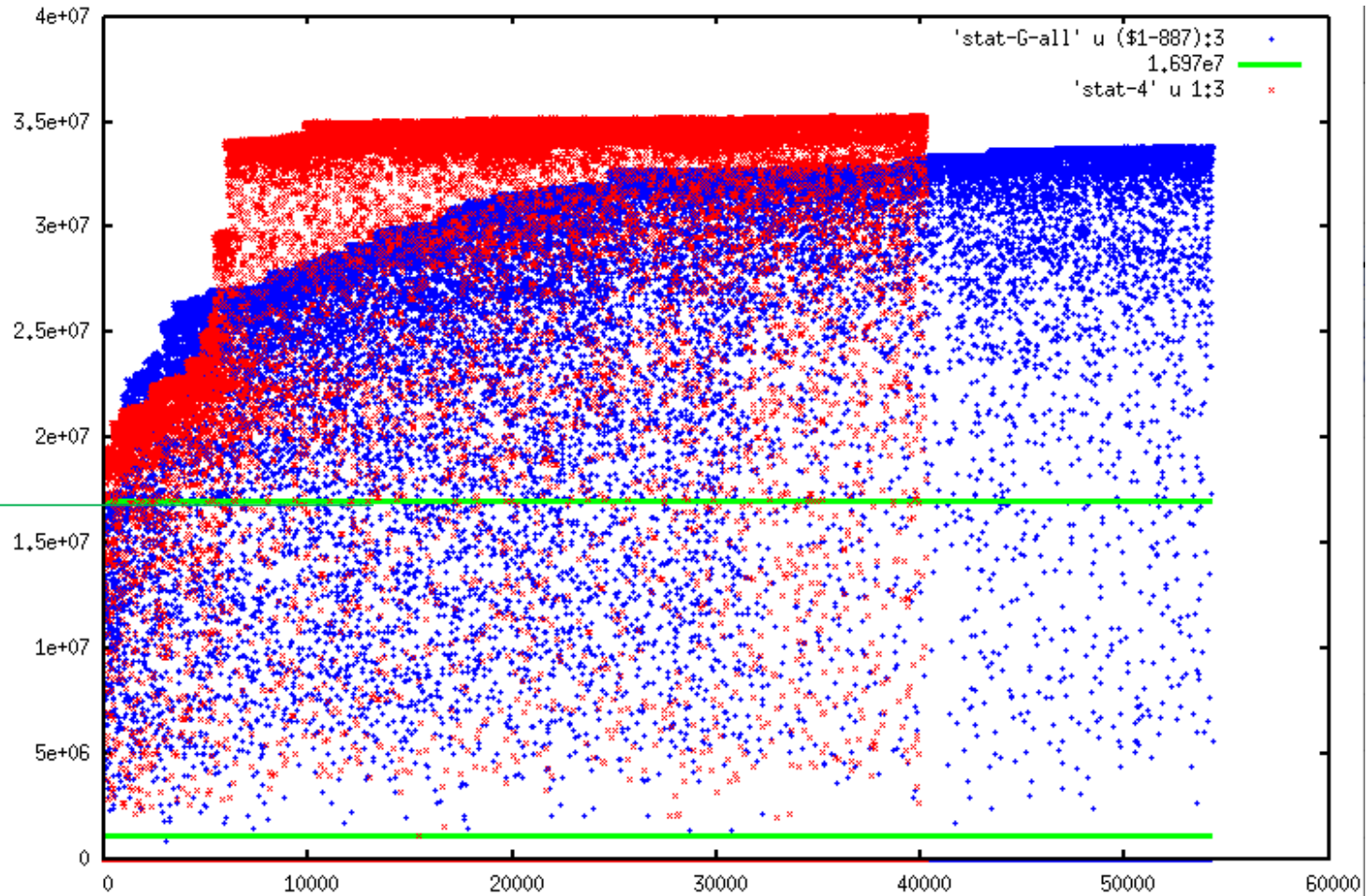


試したプログラムの数→

実験2

↑
速度

手動→
チューニング
の最高記録



試したプログラムの数→

トータルで

- Paraiso 自体のソースコード: 約4300行
- Paraiso で書かれた流体計算: 464行

↓から↓

- 15万0653通りのプログラム
- 5億3353万3650行
- 24.726464400ギガバイトのソースコード

を生成した

一番速いコードの性能

- 43.8GFlops
- 128.2GB/s

M2050のピーク性能

- 515.2GFlops(倍精度)
- 144.8GB/s

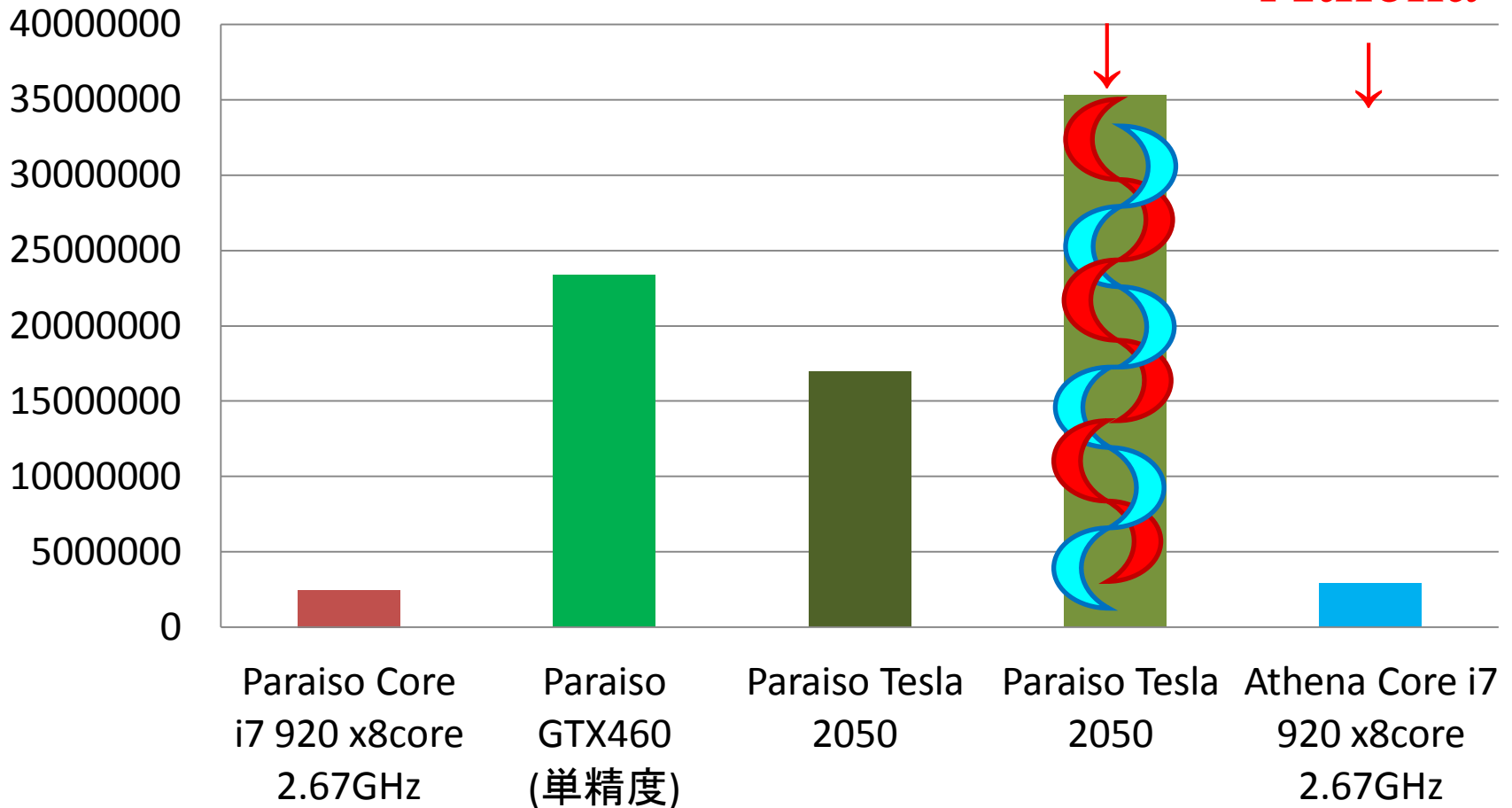
帯域リミットである

ベンチマーク結果(改改)

速度

Paraiso+Nushio+Genome

Athena



統計

Manifest	Hardware	size of .cu file	number of CUDA kernels	memory consumption	speed (mesh/s)
Athena	Core i7 x8 倍精度				2.90×10^6
none	Core i7 x8 倍精度	13108 lines		52 x N	0.38×10^6
手動チューニング・2	Core i7 x8 倍精度	2978 lines		68 x N	2.48×10^6
none	GTX 460	13108 lines	7	52 x N	3.03×10^6
手動チューニング・2	GTX 460 単精度	2978 lines	11	68 x N	23.37×10^6
自動チューニング	Tesla M2050 倍精度	3103 lines	15	73 x N	35.30×10^6

Manifest	Hardware	size of .cu file	number of CUDA kernels	memory consumption	speed (mesh/s)
none	GTX 460	13108 lines	7	52 x N	3.03×10^6
手動チューニング・1	GTX 460 単精度	3417 lines	15	84 x N	22.38×10^6
手動チューニング・2	GTX 460 単精度	2978 lines	11	68 x N	23.37×10^6
手動チューニング・3	GTX 460 単精度	17462 lines	12	68 x N	0.68×10^6
手動チューニング・2	Tesla M2050 倍精度	2978 lines	11	68 x N	16.97×10^6
自動チューニング	Tesla M2050 倍精度	3103 lines	15	73 x N	35.30×10^6
Athena	Core i7 x8 倍精度				2.90×10^6

Paraisoの現状まとめ

- Paraisoソースから、OpenMPあるいはCUDAにより並列化されたコードを出力できる。
- OpenMP版は、広く使われているAthenaコードに匹敵する速度。
- CUDA版は一桁上の速度が出る。しかもCPU版からソースを全く書き換える必要がない。
- 1, 2行のAnnotationを追加するだけでメモリの使い方やサブルーチンの分割の仕方を、したがってコードの性能をがらりと変えられる。

Paraisoの現状まとめ

- メモリの使い方やサブルーチンの分け方、同期のタイミングなどを、ベンチマークと遺伝的アルゴリズムで自動チューニングしてくれる
- 人間がチューンしたコードをぶちこむとさらに2倍近く速くしてくれる
- これからMPIを使った分散計算に対応していきたい
- OpenCLやFortranのバックエンドも作りたい