Hakubi
Kyoto Univ.

「**Paraiso**」project
for automated generation of partial differential equations solvers for parallel computers

Takayuki Muranush　@nushio
Astrophisicist, Assistant professor at
The Hakubi Center, Kyoto University（2010-2015）

Hakubi Kyoto Univ.

Chapter 1.
An Introduction to a Problem we (numerical astronomers) have.

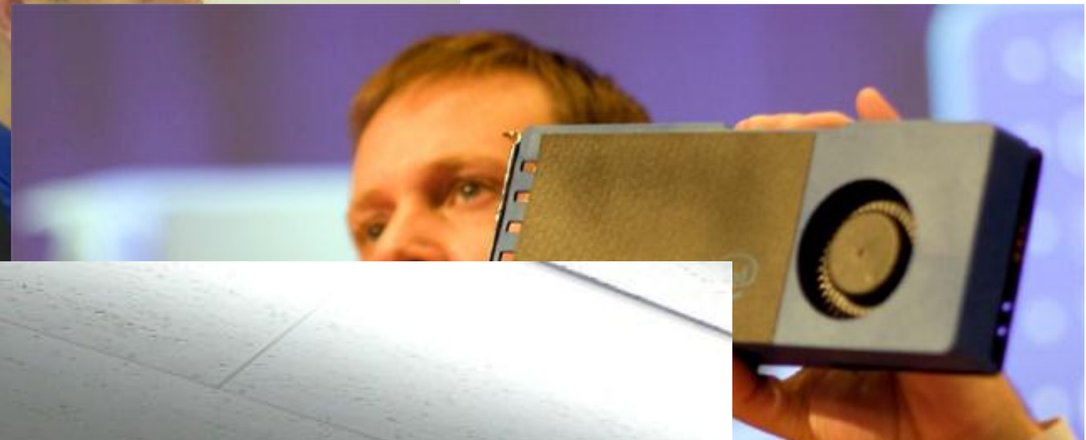
Hakubi

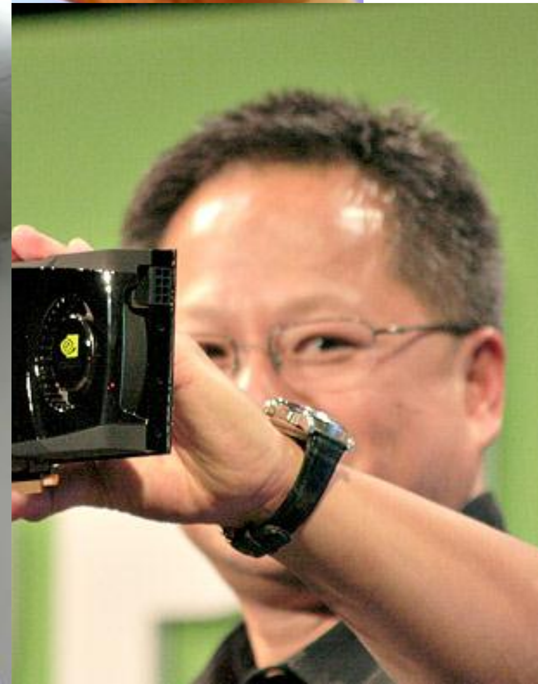
Hakubi

Acceleration

is

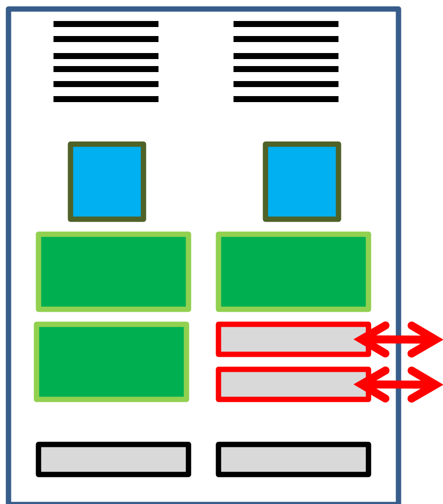
Parallelization





TSUBAME 2.0　Memory Hierarchy

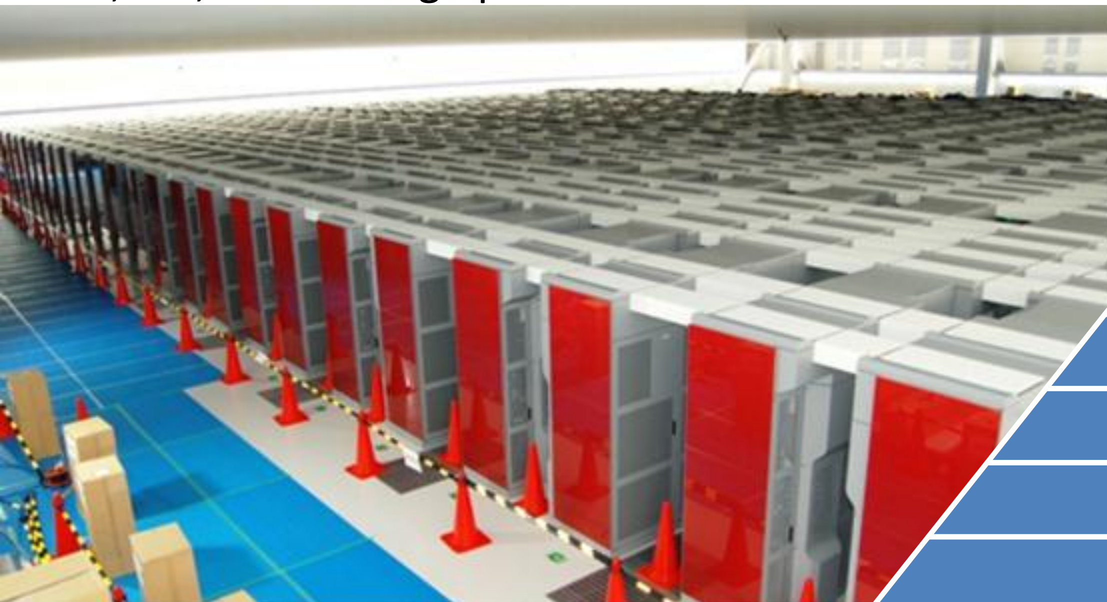
1 node



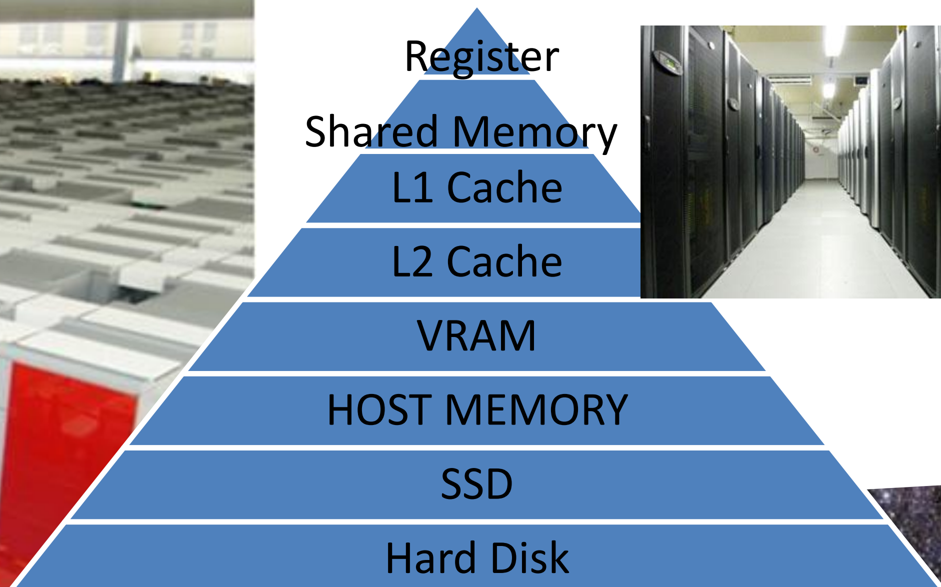
メモリ: (4GBx6) + (8GBx3 + 2GBx3)
CPU : **Westmere EP** x2
GPU : **Tesla 2050** (515Gflops + 3GB) x3
通信 : Infiniband QDR 10GB/s
ローカルディスク : SSD x2 RAID0 (460MB/s read)

- Thanks to the architects, today we have access to huge amount of memory and compute capability.
- **That doesn't come for free to the programmers. anymore.**

4,386,816 floating operations in Parallel



90'832'896 CUDA Thread in Parallel

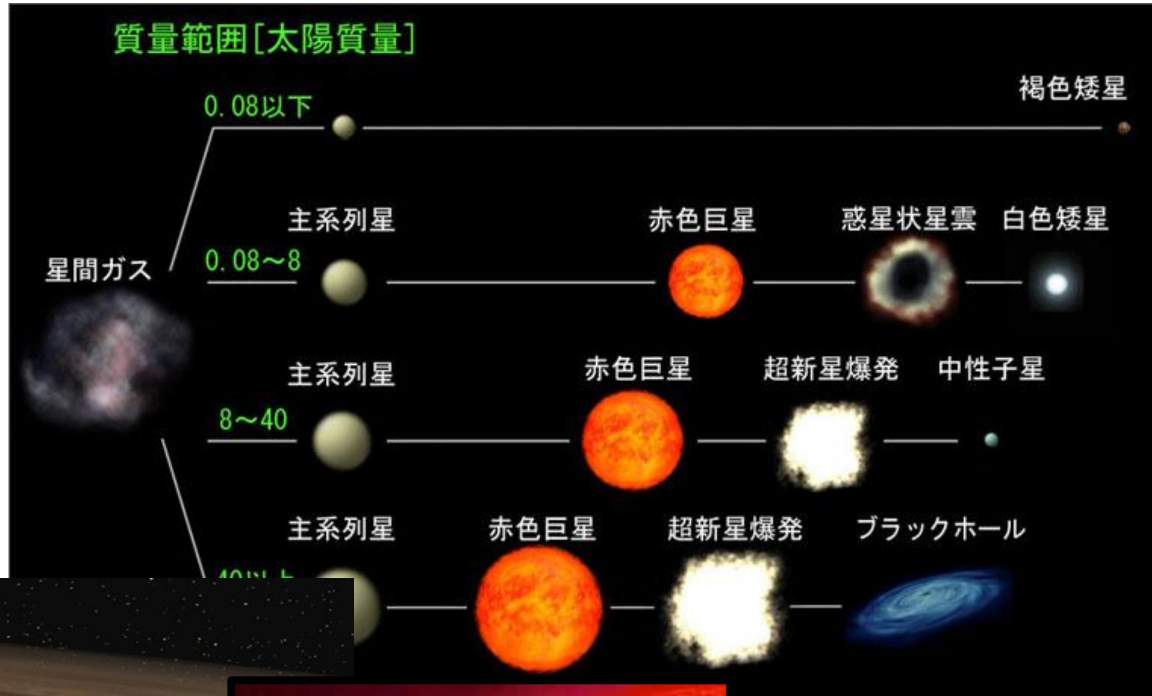


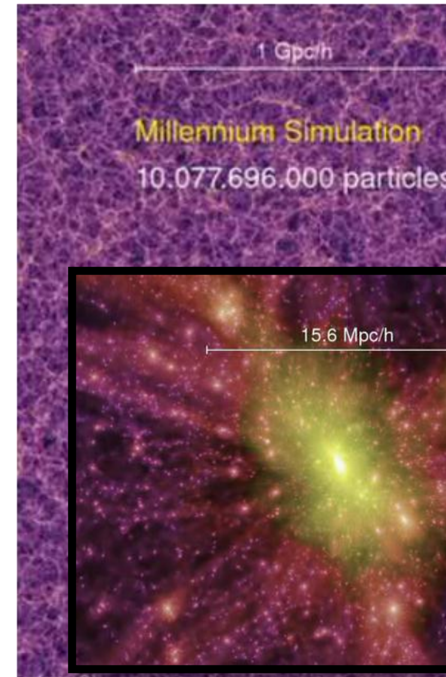
Hakubi

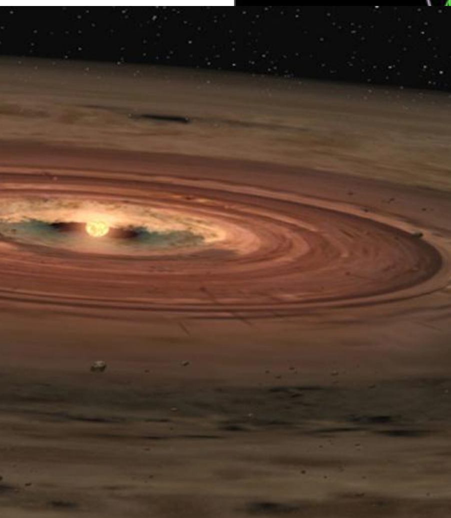
What
kind of computations
I’d like to do
with such hardwares?

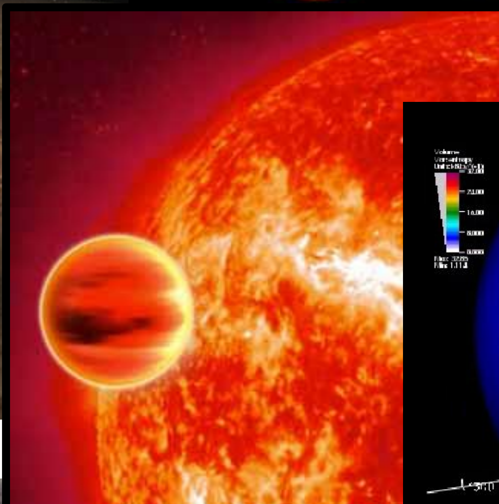
Hakubi

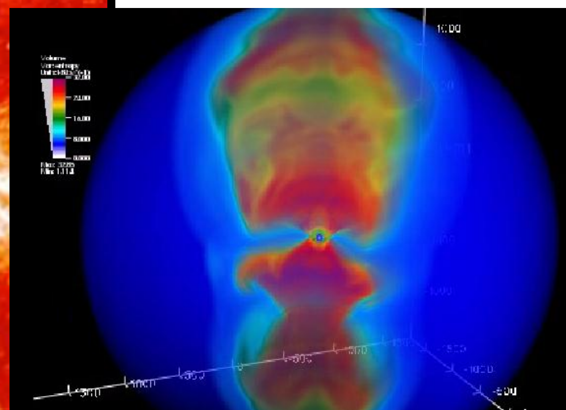
many categories of problems

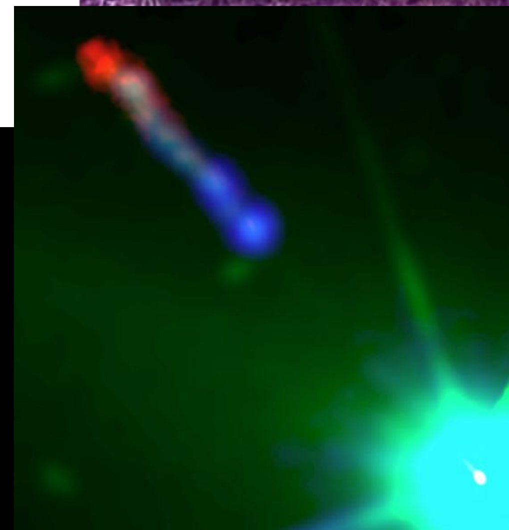






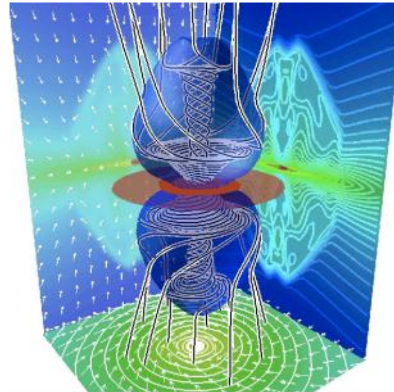




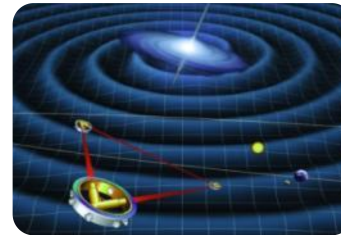


Target Problem of *Paraiso*: Partial Differential Equations, Explicit Solvers, on Uniform Mesh

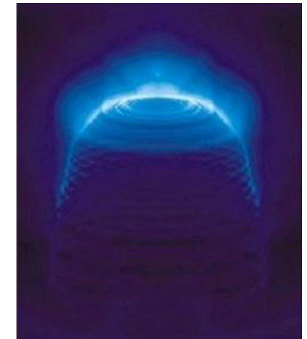
Hydrodynamics



Magneto-Hydrodynamics



General Relativity



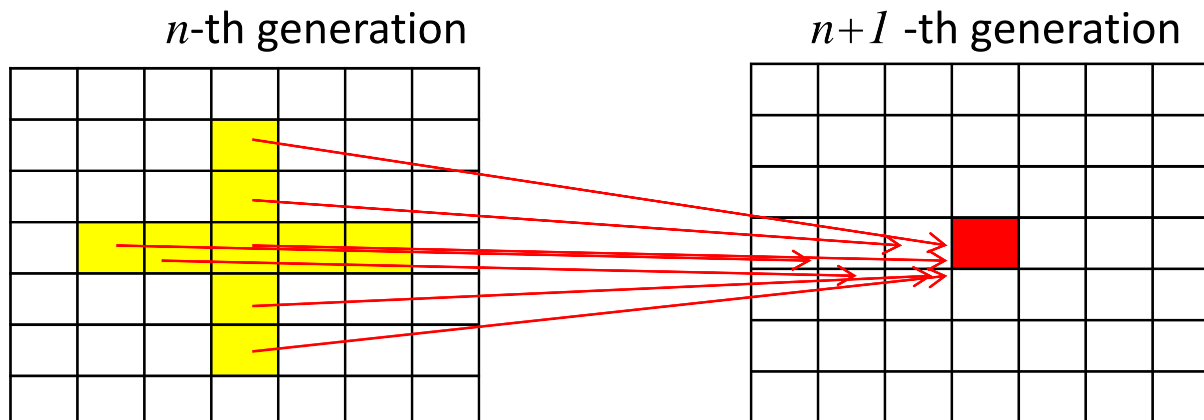
Radiative Transfer (Relativistic)

- Hyperbolic PDEs that appers in astrophysics
- combinations of these equations
- combinations with chemistry etc..

Partial Differential Equations, Explicit Solvers, on Uniform Mesh

From computational point of view:

- They are *d*-Dimensional, real-number cell automata.
- The state of each cell is a tuple of real numbers.
- The state of the cell at generation (n+1) is defiend as function of the states of its neighbor cells at generation (n).

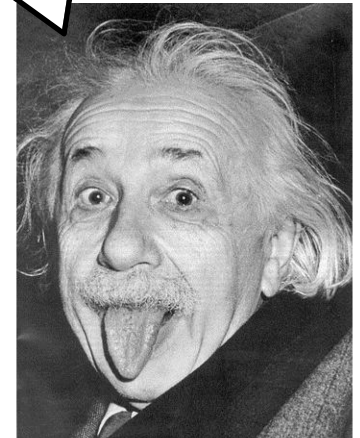


What kind of equations?

- For example, the General Theory of Relativity which only two man on the earth truly understood, is as follows

$$R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu} = \frac{8\pi G}{c^4}T_{\mu\nu}$$

it’s easy, isn’t it?



Hakubi

What kind of algorithms?

- A description of BSSN algorithm, to solve the Einstein's equations

B. Einstein's equation

Our formulation for Einstein's equations is the same as in [6] in three spatial dimensions and in [31] in axial symmetry. Here, we briefly review the basic equations in our formulation. Einstein's equations are split into constraint and evolution equations. The Hamiltonian and momentum constraint equations are written as

$$R_k^k - \tilde{A}_{ij}\tilde{A}^{ij} + \frac{2}{3}K^2 = 16\pi\rho_{\rm H}, \tag{26}$$

$$D_i\tilde{A}^i{}_j - \frac{2}{3}D_jK = 8\pi J_j, \tag{27}$$

or, equivalently

$$\tilde{\Delta}\psi = \frac{\psi}{8}\tilde{R}_k{}^k - 2\pi\rho_{\rm H}\psi^5 - \frac{\psi^5}{8}\left(\tilde{A}_{ij}\tilde{A}^{ij} - \frac{2}{3}K^2\right), \tag{28}$$

$$\tilde{D}_i(\psi^6\tilde{A}^i{}_j) - \frac{2}{3}\psi^6\tilde{D}_jK = 8\pi J_j\psi^6, \tag{29}$$

where $\psi = e^{\phi}$. These constraint equations are solved to set initial conditions. A method in the case of GRMHD is presented in Sec. IV.

In the following of this subsection, we assume that Einstein's equations are solved in the Cartesian coordinates (x, y, z) for simplicity. Although we apply the implementation described here to axisymmetric issues as well as nonaxisymmetric ones, this causes no problem since Einstein's equations in axial symmetry can be solved using the so-called Cartoon method in which an axisymmetric boundary condition is appropriately imposed in the Cartesian coordinates [31–33]: In the Cartoon method, the field equations are solved only in the $y = 0$ plane, and grid points of $y = \pm\Delta x$ (Δx denotes the grid spacing in the uniform grid) are used for imposing the axisymmetric boundary conditions.

We solve Einstein's evolution equations in our latest BSSN formalism [6,29]. In this formalism, a set of variables $(\tilde{\gamma}_{ij}, \phi, \tilde{A}_{ij}, K, F_i)$ are evolved. Here, we adopt an auxiliary variable $F_i = \delta^{jl}\partial_l\tilde{\gamma}_{ij}$ that is the one originally proposed and different from the variable adopted in [10] in which $\partial_i\tilde{\gamma}^{ij}$ is used. Evolution equations for $\tilde{\gamma}_{ij}$, ϕ, $\tilde{A}_{ij}$, and K are

$$(\partial_t - \beta^l\partial_l)\tilde{\gamma}_{ij} = -2\alpha\tilde{A}_{ij} + \tilde{\gamma}_{ik}\beta^k{}_{,j} + \tilde{\gamma}_{jk}\beta^k{}_{,i} - \frac{2}{3}\tilde{\gamma}_{ij}\beta^k{}_{,k}, \tag{30}$$

$$\begin{aligned}(\partial_t - \beta^l\partial_l)\tilde{A}_{ij} = {}& e^{-4\phi}\left[\alpha\left(R_{ij} - \frac{1}{3}e^{4\phi}\tilde{\gamma}_{ij}R_k{}^k\right) - \left(D_iD_j\alpha - \frac{1}{3}e^{4\phi}\tilde{\gamma}_{ij}\Delta\alpha\right)\right] \\ & + \alpha(K\tilde{A}_{ij} - 2\tilde{A}_{ik}\tilde{A}_j{}^k) + \beta^k{}_{,i}\tilde{A}_{kj} \\ & + \beta^k{}_{,j}\tilde{A}_{ki} - \frac{2}{3}\beta^k{}_{,k}\tilde{A}_{ij} \\ & - 8\pi\alpha\left(e^{-4\phi}S_{ij} - \frac{1}{3}\tilde{\gamma}_{ij}S_k{}^k\right),\end{aligned} \tag{31}$$

$$(\partial_t - \beta^l\partial_l)\phi = \frac{1}{6}(-\alpha K + \beta^k{}_{,k}), \tag{32}$$

$$(\partial_t - \beta^l\partial_l)K = \alpha\left[\tilde{A}_{ij}\tilde{A}^{ij} + \frac{1}{3}K^2\right] - \Delta\alpha + 4\pi\alpha(\rho_{\rm H} + S_k{}^k). \tag{33}$$

For a solution of ϕ, the following conservative form may be adopted [6]:

$$\partial_t e^{6\phi} - \partial_l(\beta^l e^{6\phi}) = -\alpha K e^{6\phi}. \tag{34}$$

For computation of R_{ij} in the evolution equation of $\tilde{A}_{ij}$, we decompose

$$R_{ij} = \tilde{R}_{ij} + R^{\phi}_{ij}, \tag{35}$$

where

$$R^{\phi}_{ij} = -2\tilde{D}_i\tilde{D}_j\phi - 2\tilde{\gamma}_{ij}\tilde{\Delta}\phi + 4\tilde{D}_i\phi\tilde{D}_j\phi - 4\tilde{\gamma}_{ij}\tilde{D}_k\phi\tilde{D}^k\phi, \tag{36}$$

$$\tilde{R}_{ij} = \frac{1}{2}\left[\delta^{kl}(-h_{ij,kl} + h_{ik,lj} + h_{jk,li}) + 2\partial_k(f^{kl}\Gamma_{l,ij}) - 2\Gamma^l_{kj}\Gamma^k_{il}\right]. \tag{37}$$

In Eq. (37), we split $\tilde{\gamma}_{ij}$ and $\tilde{\gamma}^{ij}$ as $\delta_{ij} + h_{ij}$ and $\delta^{ij} + f^{ij}$, respectively. Γ^k_{ij} is the Christoffel symbol with respect to $\tilde{\gamma}_{ij}$, and $\Gamma_{k,ij} = \tilde{\gamma}_{kl}\Gamma^l_{ij}$. Because of the definition $\det(\tilde{\gamma}_{ij}) = 1$ (in the Cartesian coordinates), we use $\Gamma^k_{ki} = 0$.

In addition to a flat Laplacian of h_{ij}, $\tilde{R}_{ij}$ involves terms linear in h_{ij} as $\delta^{kl}h_{ik,lj} + \delta^{kl}h_{jk,li}$. To perform numerical simulation stably, we replace these terms by $F_{i,j} + F_{j,i}$. This is the most important part in the BSSN formalism, pointed out originally by Nakamura [26]. The evolution equation of F_i is derived by substituting Eq. (30) into the momentum constraint as

$$\begin{aligned}(\partial_t - \beta^l\partial_l)F_i = {}& -16\pi\alpha J_i + 2\alpha\left\{f^{kj}\tilde{A}_{ik,j} + f^{kj}{}_{,j}\tilde{A}_{ik} - \frac{1}{2}\tilde{A}^{jl}h_{jl,i} + 6\phi_{,k}\tilde{A}^k{}_i - \frac{2}{3}K_{,i}\right\} \\ & + \delta^{jk}\left\{-2\alpha_{,k}\tilde{A}_{ij} + \beta^l{}_{,k}h_{ij,l} + \left(\tilde{\gamma}_{il}\beta^l{}_{,j} + \tilde{\gamma}_{jl}\beta^l{}_{,i} - \frac{2}{3}\tilde{\gamma}_{ij}\beta^l{}_{,l}\right)_{,k}\right\}.\end{aligned} \tag{38}$$

We also have two additional notes for handling the evolution equation of $\tilde{A}_{ij}$. One is on the method for evaluation of $R_k{}^k$ for which there are two options, use of the Hamiltonian constraint and direct calculation by

$$R_{ij}\gamma^{ij} = e^{-4\phi}(\tilde{R}_k{}^k + R^{\phi}_{ij}\tilde{\gamma}^{ij}). \tag{39}$$

We always adopt the latter one since with this, the conservation of the relation $\tilde{A}_{ij}\tilde{\gamma}^{ij} = 0$ is much better preserved. The other is on the handling of a term of $\tilde{\gamma}^{ij}\delta^{kl}h_{ij,kl}$ which appears in $\tilde{R}_k{}^k$. This term is written by

$$\tilde{\gamma}^{ij}\delta^{kl}h_{ij,kl} = -\delta^{kl}h_{ij,k}f^{ij}{}_{,l}, \tag{40}$$

where we use $\det(\tilde{\gamma}_{ij}) = 1$ (in the Cartesian coordinates).

As the time slicing condition, an approximate maximal slice condition $K \approx 0$ is adopted following previous papers (e.g., [34]). As the spatial gauge condition, we adopt a hyperbolic gauge condition as in [6,35]. Successful numerical results for merger of binary neutron stars and stellar core collapse in these gauge conditions are presented in [6–8,24]. We note that these are also different from those in [10].

What kind of code we use?

Hakubi

＊BSSN algorithm

Fortran＋OpenMP

* My MHD Solver in CUDA+MPI

Hakubi

What’s wrong?

- what makes our programs *this* long?

Programming is to choose

algebraic concepts	tensors, its symmetry…
physical equations	HD, MHD, GR, …
time integration methods	1st order, 2nd order, 4th order, more…
space interpolation methods	1st, 2nd, 3rd, TVD, Shock…
data structures	SoA, AoS, distribution,
optimization techniques	communication timing…
and hardware designs	CPU, GPU, what next, …

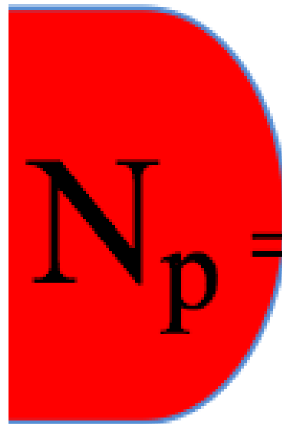
Hakubi

To make things worse...

- We write more than one program.
- We make trial & error in search for better programs
- Suddenly the architecture changes and we are forced to use SSE, CUDA, AVX ...
- We need to insert communications correctly

To make a single point of change, we have to grep all over the codes

Modern Parallel Programming is like this



The amount of programs we write in our life is the product of the factors mentioned

$$N_p = N_{eq} \times N_{int} \times N_{math} \times N_{hard} \cdots$$

I want it like this

Specify each of the sufficient knowledge modules, and programs like above are automatically generated

$$N_p = N_{eq} + N_{int} + N_{math} + N_{hard} \cdots$$

Hakubi

What a code generator aims for

Hakubi

- Generally you write $N_f \times N_{math} \times N_{eq} \times N_{int} \times N_{hw}$... lines of code
- You find a bug / improvement and want $N_{eq} = N_{eq} + 1$; then you need to re-write $N_f \times N_{math} \times 1 \times N_{int} \times N_{hw}$... lines

- With code generator you only have to write

  $N_f + N_{math} + N_{eq} + N_{int} + N_{hw}$... lines
- You want $N_{eq} = N_{eq} + 1$; then just add 1 line
- *You can concentrate on physics*

Can’t you do that in existing language

- possibly

- we can define vectors and tensors as classes
- we can overload operators
- we have templates
- we have accumulated sophisticated techniques such as expression templates

as a result ...

```
thrusting::transform(
  n_particle,
  thrust::make_transform_iterator(
    thrusting::make_zip_iterator(
      thrust::make_transform_iterator(
        thrust::counting_iterator<size_t>(0),
        thrusting::compose(
          thrusting::make_uniform_real_distribution<real>(0,1),
          thrusting::make_fast_rng_generator(seed))),
      thrust::make_transform_iterator(
        thrust::counting_iterator<size_t>(n_particle),
        thrusting::compose(
          thrusting::make_uniform_real_distribution<real>(0,1),
          thrusting::make_fast_rng_generator(seed))),
      thrust::make_transform_iterator(
        thrust::counting_iterator<size_t>(2 * n_particle),
        thrusting::compose(
          thrusting::make_uniform_real_distribution<real>(0,1),
          thrusting::make_fast_rng_generator(seed))),
      thrust::make_transform_iterator(
        thrust::counting_iterator<size_t>(3 * n_particle),
        thrusting::compose(
          thrusting::make_uniform_real_distribution<real>(0,1),
          thrusting::make_fast_rng_generator(seed))),
      thrust::make_transform_iterator(
        thrust::counting_iterator<size_t>(4 * n_particle),
        thrusting::compose(
          thrusting::make_uniform_real_distribution<real>(0,1),
          thrusting::make_fast_rng_generator(seed))),
      thrust::make_transform_iterator(
        thrust::counting_iterator<size_t>(5 * n_particle),
        thrusting::compose(
          thrusting::make_uniform_real_distribution<real>(0,1),
          thrusting::make_fast_rng_generator(seed)))),
    detail::maxwell_rand(m, T, BOLTZMANN, PI)),
  thrusting::make_zip_iterator(u, v, w),
  thrust::identity<real3>());
```

I want a language that
is modular
(easy to reuse components)
transplantable
fast
beautiful

- hopeless

or is it...?

if you limit the problem domain

Hakubi

Chapter 2.

The overall design of

Paraiso

PARallel **A**utomated **I**ntegration **S**cheme **O**rganizer

Input: Discretized Algorithms for solving Partial Differential Equations, in mathematical notations

Output: Implementations on Distributed, Manycore Machines.

8 building blocks of PDE solvers

```
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`
  load constant value

`Load (graph starts here)`
  read from named array

`Store (graph ends here)`
  write to named array

`Reduce`
  array to scalar value

`Broadcast`
  scalar to array

`Shift`
  copy each cell to neighbourhood

`LoadIndex & LoadSize`
  get coordinate of each cell
  get array size

`Arith`
  various mathematical operations

$$\frac{\partial U}{\partial t} + \nabla \cdot \boldsymbol{F} = 0$$

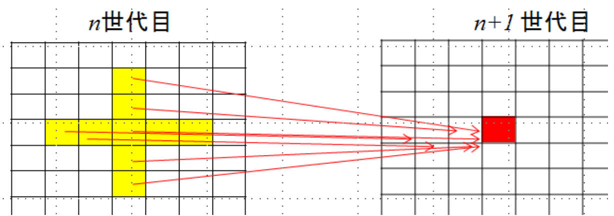
Basic Equations

Manually

Discretized Form

Discrete PDE Language

```
a_i_j <- … …
q_i <- dt *
   a_i_j * f_j
```

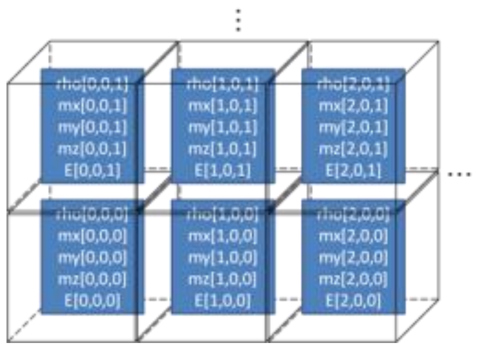


OM Builder

OM Dataflow Graph

Orthotope Machine Code

```
ld  r2, g2[0,0,0]
ld  r1, g2[0,0,1]
add r1,r2,r3
st  r3,g1
```



OM Compiler

Native Code

```
*q=cudaMalloc(…);
__shared__ a,b;
a=q[idx];
b=q[idx+1];
p[idx]=a+b;
```

Native Compiler

Executable Files

result

Hakubi

サーベイしたところ出てきた先行研究

Vol. 26 No. 1　　情報処理学会論文誌　　Jan. 1985

数値シミュレーション用プログラミング言語 DEQSOL[†]

梅谷 征雄[††]　辻 みちる[††]　岩沢 京子[††]

DEQSOL (Differential EQuation SOlver Language) は数値シミュレーション用に設計された問題向きプログラミング言語であり，偏微分方程式の解法を数値アルゴリズムのレベルで記述することを目的とする．言語設計の狙いは二つあり，第1はこの分野におけるプログラム生産性を飛躍的に向上させることであり，第2は数値シミュレーションに通常用いられるベクトル計算機や並列計算機に向けたプログラムに対しシステム側の最適化の余地を保持することである．このために言語の設計にあわせて，DEQSOL で書かれたプログラムをベクトル/並列計算機向き FORTRAN コードに自動変換するトランスレータを検討・開発した．言語の設計にあたっては，領域形状，メッシュ分割，微分演算子，ベクトル，境界条件など偏微分方程式系の数値シミュレーションに固有の概念を導入して，差分法に基づく計算手順を簡潔・柔軟に記述できるように工夫した結果，典型的な3次元流体シミュレーションプログラムを FORTRAN の10分の1以下の行数で作成する実績を得た．また差分法や線形解法に内在する並列性を生かした FORTRAN コードの生成に努めた結果，M 200 H IAP にて90% 以下のベクトル化を達成することができた．DEQSOL は解法記述言語である点で PDEL や ELLPACK などのソルバとは異なり，むしろ PDELAN のアプローチを拡張・発展させたものである．

Hakubi

かなり似ている・・・

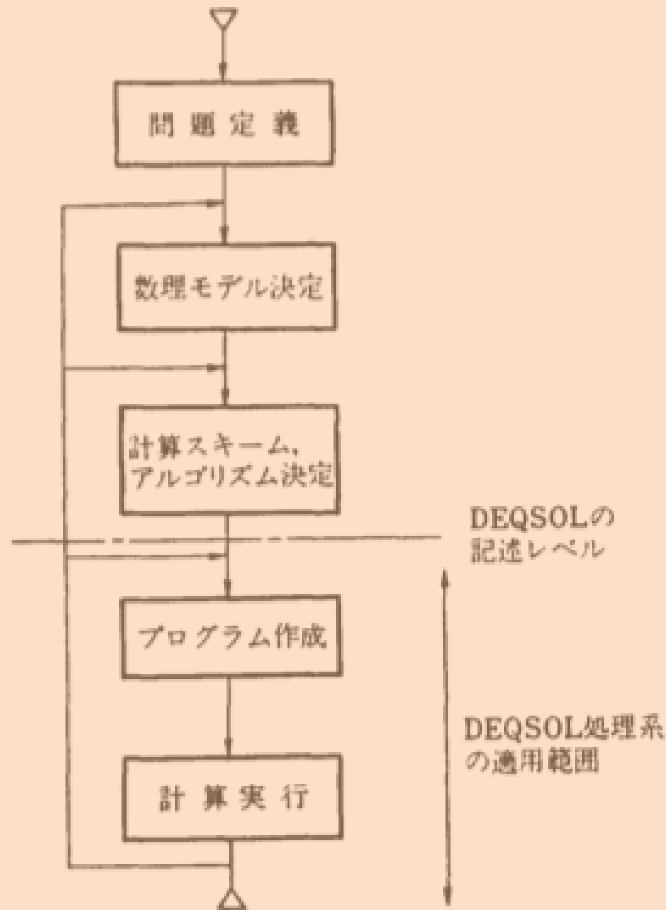


図 1 数値シミュレーション過程と DEQSOL の役割
Fig. 1 Numerical simulation process and the role of DEQSOL system.

- 基礎方程式
- さしあたり人手
- 離散化形
- 自動
- VVM上のコード
- 自動
- 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

Hakubi

DEQSOLの末路・・・

Hi Takayuki,

DEQSOL was originally developped on Hitachi vector machines (S810/820). It seems Hitachi continued to provide it as a commercial product, at least untill around 2000, but only on shared-memory machines. See:

http://www.cc.u-tokyo.ac.jp/publication/news/VOL4/No2/tetsuzuki-gaiyou.pdf
http://www.hucc.hokudai.ac.jp/sokuho/2001/08.html

So it seems they could not port the DEQSOL program to distributed memory machines (or nobody was using it anyway and there was no demand?)

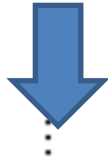
Discretized PDE Language

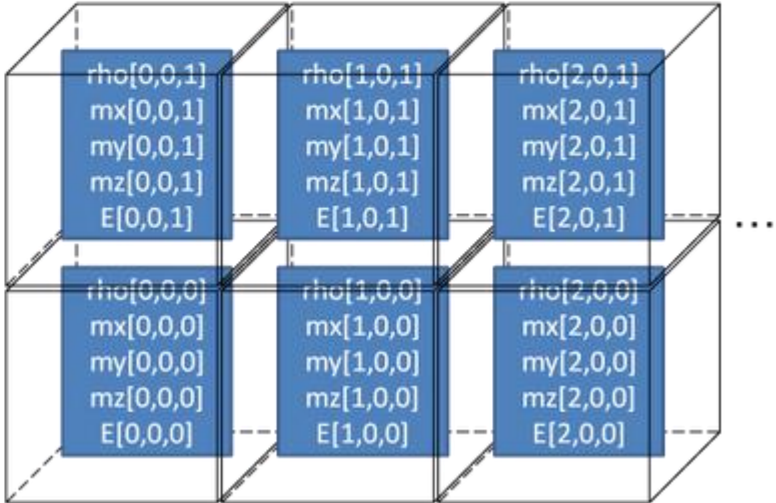
```
cell2 <- proceedSingle 1 (dt/2) dR cell  cell
cell3 <- proceedSingle 2  dt    dR cell2 cell
```

```
putStrLn "testing for theorems on Levi Civita tensor"
print $ s_(\i -> s_(\j -> s_(\k -> eps!i!j!k * eps!i!j!k))) == 6
print $ c_(\i -> c_(\j -> s_(\m -> s_(\n -> eps!i!m!n * eps!j!m!n)))) ==
  c_(\i -> c_(\j -> 2 * del!i!j))
print $ c_(\i -> c_(\j -> s_(\k -> c_(\l -> c_(\m -> eps!i!j!k * eps!k!l!m))))) ==
  c_(\i -> c_(\j -> c_(\l -> c_(\m -> del!i!l * del!j!m - del!i!m * del!j!l))))
```

Algorithm notation

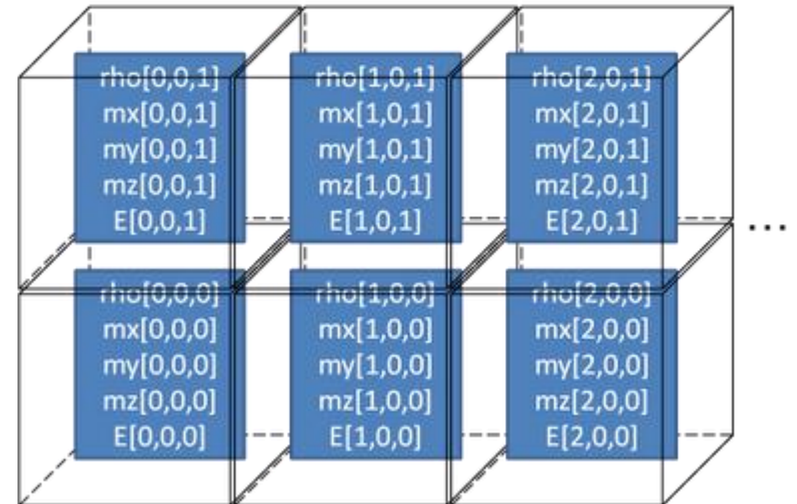
- Describe your numerical algorithm using natural notations e.g. tensor notations, difference operators.
- Translates to Orthotope Machine





Orthotope Machine

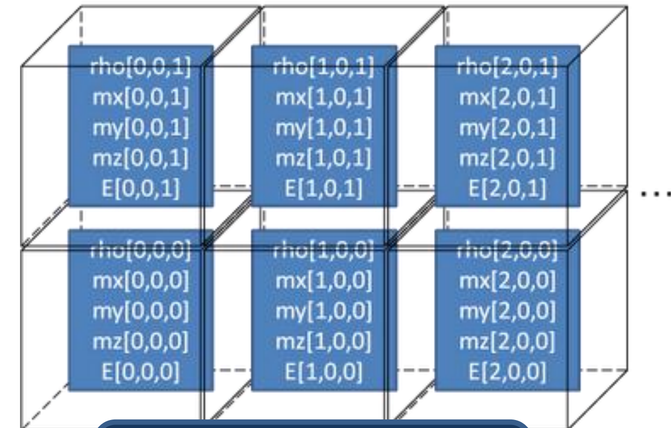
- A Real number cellular automata
- A virtual machine with multidimensional array vector register of infinite size
- arithmetic operations work in parallel on each mesh, loads from neighbour cells,



No intention of buiding a real hardware:
a thought object to construct a dataflow graph

Orthotope Machine Compiler

- convert dataflow graph to real codes
- Divide OM registers onto distributed memories
- Generate Communication codes
- The code generator has wide choice on data layout, granularity of communication, computation

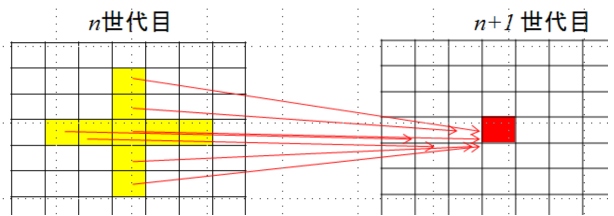


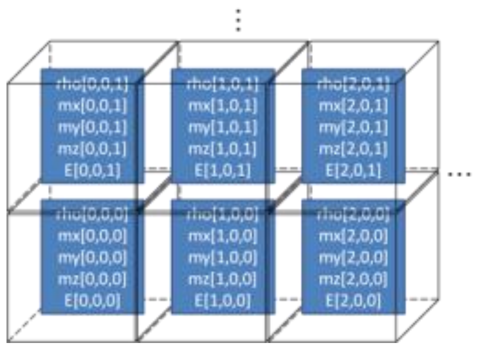
OM Compiler

Native Codes

Chapter 3.

$$\frac{\partial U}{\partial t} + \nabla \cdot \boldsymbol{F} = 0$$





result

Basic Equations

Manually

Discretized Form

OM Builder

OM Dataflow Graph

OM Compiler

Native Code

Native Compiler

Executable Files

Discrete PDE Language

```
a_i_j <- … …
q_i <- dt *
   a_i_j * f_j
```

Orthotope Machine Code

```
ld  r2, g2[0,0,0]
ld  r1, g2[0,0,1]
add r1,r2,r3
st  r3,g1
```

```
*q=cudaMalloc(…);
__shared__ a,b;
a=q[idx];
b=q[idx+1];
p[idx]=a+b;
```

Hakubi

More Details on Orthotope Machine

```
data OM vector gauge anot
  = OM { omName :: Name, setup :: Setup vector gauge anot,
         kernels :: [Kernel vector gauge anot] }

data Kernel vector gauge anot
  = Kernel { kernelName :: Name,
             dataflow :: Graph vector gauge anot }

type Graph vector gauge anot
  = FGL.Gr (Node vector gauge anot) Edge

data Node vector gauge anot
  = NValue DynValue anot
  | NInst (Inst vector gauge) anot
```

Orthotope Machine

```
data OM vector gauge anot
```

POM is Primordial Orthotope Machine.

Constructors

```
OM
  omName :: Name
  setup :: Setup vector gauge anot
  kernels :: [Kernel vector gauge anot]
```

Hakubi

Three Type parameters OM components have

Hakubi

Language.Paraiso.OM.Graph

the components for constructing Orthotope Machine data flow draph. Most components take three arguments:

Type Constructor for
The dimension of the Orthotope Machine

`vector :: * -> *`

The array dimension. It is a `Vector` that defines the dimension of the Orthotope on which the OM operates.

Type of the Array Index

`gauge :: *`

The array index. The combination `vector gauge` needs to be an instance of `Algebra.Additive.C` if you want to perform `Shift` operation.

Type of the Annotation you can apply at graph node

`anot :: *`

The annotations put on each node. If you want to use Annotation, `anot` needs to be an instance of `Data.Monoid`.

Hakubi

```
data Kernel vector gauge anot
```

A Kernel for OM perfor a block of calculations on OM.

Constructors

```
Kernel
    kernelName :: Name
    dataflow :: Graph vector gauge anot
```

```
type Graph vector gauge anot = Gr (Node vector gauge anot) Edge
```

The dataflow graph for Orthotope Machine. anot is an additional annotation.

bipartile graph consisting of value nodes and inst nodes

```
data Node vector gauge anot
```
Source

The `Node` for the dataflow `Graph` of the Orthotope machine. The dataflow graph is a 2-part graph consisting of `NValue` and `NInst` nodes.

Constructors

`NValue DynValue anot`	A value node. An `NValue` node only connects to `NInst` nodes. An `NValue` node has one and only one input edge, and has arbitrary number of output edges.
`NInst (Inst vector gauge) anot`	An instruction node. An `NInst` node only connects to `NValue` nodes. The number of input and output edges an `NValue` node has is specified by its `Arity`.

8 building blocks of PDE solvers

```haskell
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`
  load constant value

`Load (graph starts here)`
  read from named array

`Store (graph ends here)`
  write to named array

`Reduce`
  array to scalar value

`Broadcast`
  scalar to array

`Shift`
  copy each cell to neighbourhood

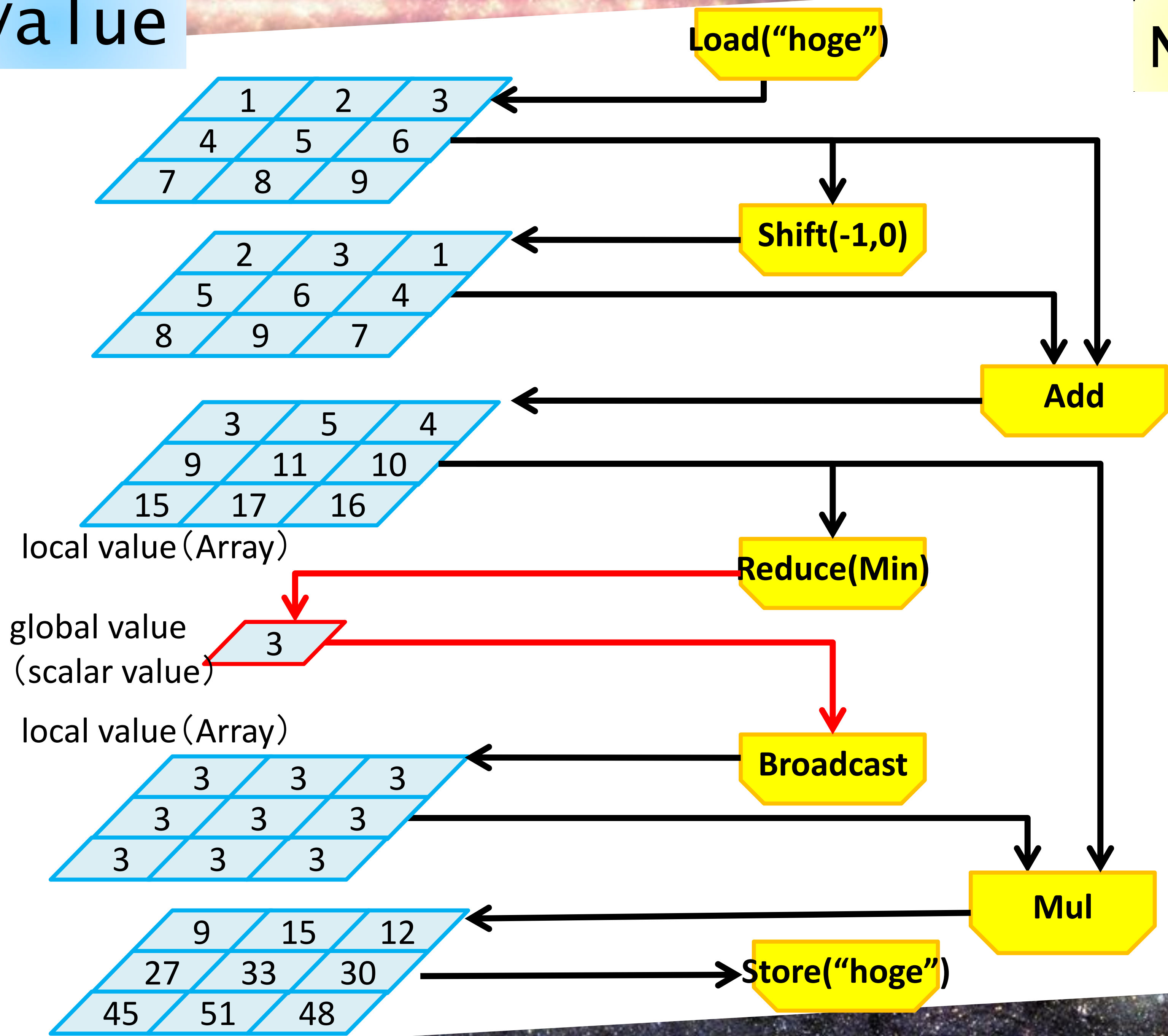
`LoadIndex & LoadSize`
  get coordinate of each cell
  get array size

`Arith`
  various mathematical operations

Hakubi

NValue

NInst



Hakubi

DynValue = Array (data container) with realm and element type information

Hakubi

```
data DynValue                                                    Source
```

dynamic value type, with its realm and content type informed as values

Constructors

```
DynValue
  realm :: Realm
  typeRep :: TypeRep
```

Local Realm = Array

Global Realm = Scalar data

```haskell
-- | Type-level representations
class TRealm a where
  tRealm :: a -> Realm
  unitTRealm :: a
data TGlobal = TGlobal
data TLocal = TLocal

-- | Value-level representations
data Realm = Global | Local
```

Type-level representation of Array

```
data (TRealm rea, Typeable con) => Value rea con
```

Source

value type, with its realm and content type discriminated in type level

Constructors

`FromNode`	data obtained from the dataflow graph. `realm` carries a type-level realm information, `content` carries only type information and its ingredient is nonsignificant and can be `undefined`.

```
realm :: rea
content :: con
node :: Node
```

`FromImm`	data obtained as an immediate value. `realm` carries a type-level realm information, `content` is the immediate value to be stored.

```
realm :: rea
content :: con
```

Value : type-level
DynValue : value-level

```haskell
-- value type, with its realm and content type
-- discriminated in type level
data (R.TReal rea, Typeable con) =>
  Value rea con
    = FromNode {realm :: rea, content :: con, node :: G.Node}
    | FromImm  {realm :: rea, content :: con}

-- dynamic value type, with its realm and content type
-- informed as values
data DynValue
  = DynValue {realm :: Realm, typeRep :: TypeRep}

-- Convert 'Value' to 'DynValue'
toDyn :: (TRealm r, Typeable c) => Val.Value r c -> DynValue
toDyn x = mkDyn (Val.realm x) (Val.content x)
```

Hakubi

User space

Value TLocal Float

Value TGlobal Int

Value TLocal Float
-> Value TGlobal Int
-> Value Tlocal Float

Internal Dataflow Graph

DynValue

DynValue

DynValue
-> DynValue
-> DynValue

- User interface is in Type-level
  - The type-checker helps user
  - and assures type-consistency for the backend
- Dataflow graph under cover is Value-level
  - can handle the graph in one type.

Chapter 4.

$$\frac{\partial U}{\partial t} + \nabla \cdot \boldsymbol{F} = 0$$

Hakubi

Basic Equations

Manually

Discretized Form

OM Builder

OM Dataflow Graph

OM Compiler

Native Code

Native Compiler

Executable Files

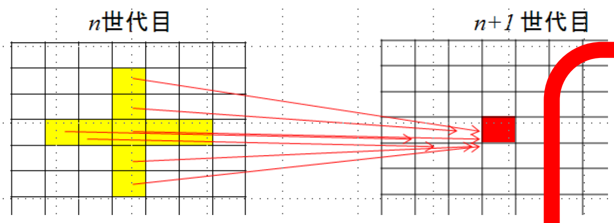
Discrete PDE Language

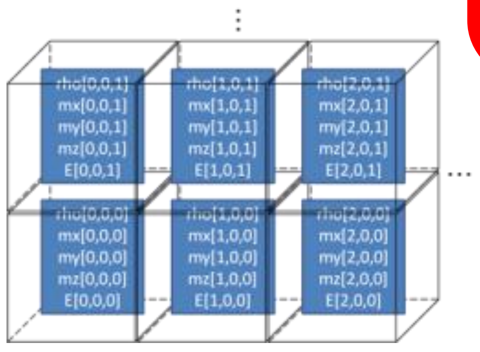
```
a_i_j <- … …
q_i <- dt *
   a_i_j * f_j
```

Orthotope Machine Code

```
ld  r2, g2[0,0,0]
ld  r1, g2[0,0,1]
add r1,r2,r3
st  r3,g1
```

```
*q=cudaMalloc(…);
__shared__ a,b;
a=q[idx];
b=q[idx+1];
p[idx]=a+b;
```





result

Builder Monads constructs dataflow graph

the interface has type-level info on Realm and Type
but internal representations are value-level on Realm and Type

```
-- | The 'Builder' monad is used to build 'Kernel's.
type Builder (vector:: * -> *) (gauge:: *) (anot:: *) (val:: *)
  = State.State (BuilderState vector gauge anot) val

data BuilderState vector gauge anot = BuilderState
    { setup   :: Setup vector gauge anot,
      context :: BuilderContext anot,
      target  :: Graph vector gauge anot} deriving (Show)

data BuilderContext anot =
  BuilderContext
  { currentAnnotation :: anot } deriving (Show)
```

Hakubi

a helper function to define binary operators for Builder Monad

```
-- | Make a binary operator
mkOp2 :: (TRealm r, Typeable c) =>
         A.Operator                          -- ^The operator
      -> (Builder v g a (Value r c)) -- ^Input 1
      -> (Builder v g a (Value r c)) -- ^Input 2
      -> (Builder v g a (Value r c)) -- ^Output
mkOp2 op builder1 builder2 = do
  v1 <- builder1
  v2 <- builder2
  let
      r1 = Val.realm v1
      c1 = Val.content v1
  n1 <- valueToNode v1
  n2 <- valueToNode v2
  n0 <-  addNodeE [n1, n2] $ NInst (Arith op)
  n01 <- addNodeE [n0] $ NValue (toDyn v1)
  return $ FromNode r1 c1 n01
```

Hakubi

Builder monad being an Additive
Builder monad being a Ring

```
-- | Builder is Additive 'Additive.C'.
-- You can use 'Additive.zero', 'Additive.+', 'Addi
instance (TRealm r, Typeable c, Additive.C c)
 => Additive.C (Builder v g a (Value r c)) where
  zero = return $ FromImm unitTRealm Additive.zero
  (+) = mkOp2 A.Add
  (-) = mkOp2 A.Sub
  negate = mkOp1 A.Neg
```

```
-- | Builder is Ring 'Ring.C'.
-- You can use 'Ring.one', 'Ring.*'.
instance (TRealm r, Typeable c, Ring.C c) => Ring.C (Builder v g a (Value r c)) where
  one = return $ FromImm unitTRealm Ring.one
  (*) = mkOp2 A.Mul
```

- programing language *Paraiso* lacks frontend (lexer, parser, AST analyzer ...)

- Instead, Paraiso components are **first-class objects** of a widely-used language (Haskell)

```
(the (ultimate (source (of (programmability)))))
```

Example : Implement Hydro Solver

Hakubi

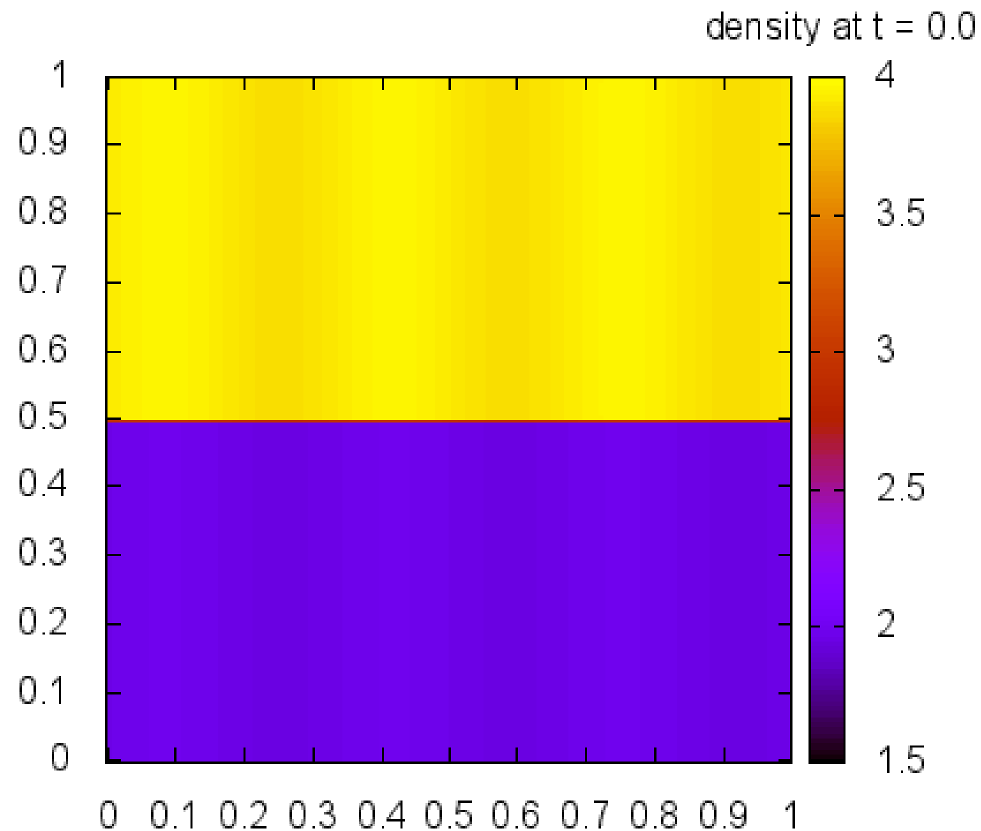
An HLLC Riemann-solver in Builder Monad

It looks like usual mathematics, but every terms appear here are Builder Monads, and the expressions are as themselves code generators for Orthotope Machine.

```
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
```

Hakubi

sample hydrodynamics solver in Paraiso



Chapter 5.

$$\frac{\partial U}{\partial t} + \nabla \cdot \boldsymbol{F} = 0$$

Basic Equations

Manually

Discretized Form

OM Builder

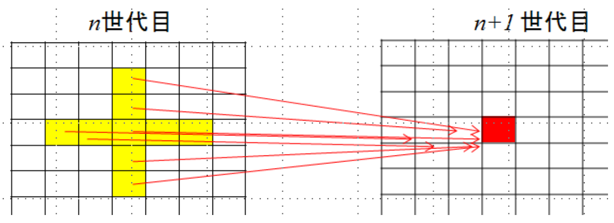
OM Dataflow Graph

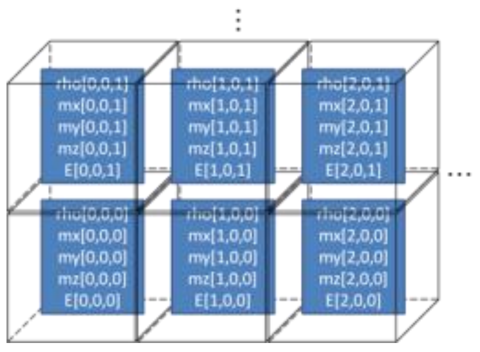
OM Compiler

Native Code

Native Compiler

Executable Files





result

Discrete PDE Language

```
a_i_j <- … …
q_i <- dt *
   a_i_j * f_j
```

Orthotope Machine Code

```
ld  r2, g2[0,0,0]
ld  r1, g2[0,0,1]
add r1,r2,r3
st  r3,g1
```

```
*q=cudaMalloc(…);
__shared__ a,b;
a=q[idx];
b=q[idx+1];
p[idx]=a+b;
```

Hakubi

old code generator...

```
runBinder graph0 n0 binder = unlines $ header ++  [bindStr] ++ footer
 where
   (header,footer) = case context state of
     CtxGlobal -> ([],[])
     CtxLocal loopIndex ->
      ([loop (symbol Cpp loopIndex) ++ " {"], ["}"])
   loop i =
     "for (int " ++ i ++ " = 0 ; "
                 ++ i ++ " < " ++ symbol Cpp sizeName ++ "() ; "
                 ++  "++" ++ i ++ ")"
```

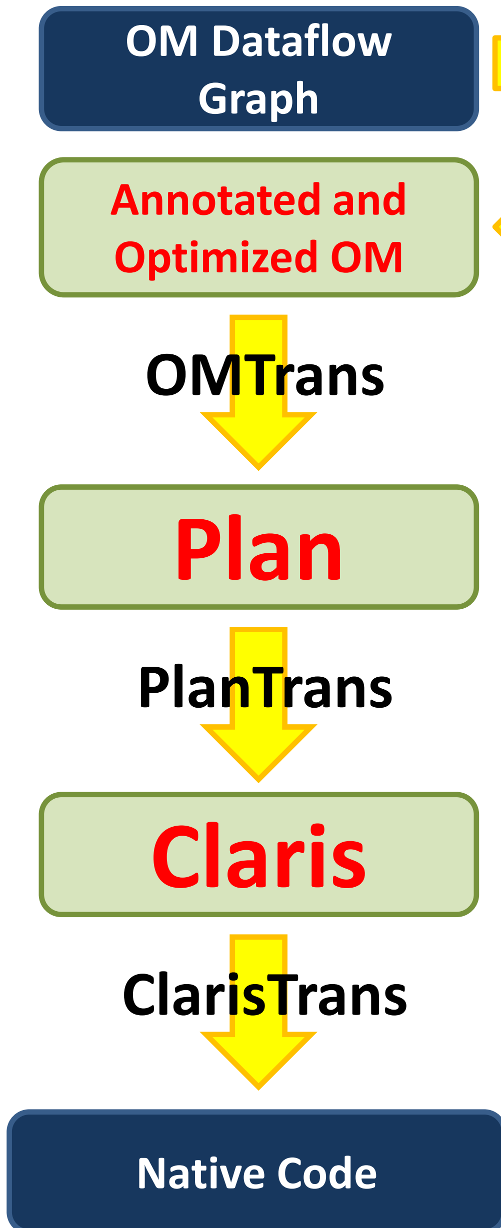
OM Dataflow Graph

directly

Native Code

- not so productive

code generator ver. 2



Optimization/Analysis

Optimization :: OM -> OM

Analysis = add annotation

Plan = decisions made upon

- how much memory to allocate
- which part of calculation to take place in same subroutine

Claris

- a C++ -like syntax tree with CUDA extension.

Annotation

- Allocation, Ballon, Boundary, Comment, Dependency

```
-- | a type that represents valid region of computation.
newtype Valid g = Valid [Interval (NearBoundary g)]
-- | the displacement around either side of the boundary.
data NearBoundary a
  = NegaInfinity | LowerBoundary a
  | UpperBoundary a | PosiInfinity
               deriving (Eq, Ord, Show, Typeable)


data Allocation
  = Existing -- ^ This entity is already allocated as a static variable.
  | Manifest -- ^ Allocate additional memory for this entity.
  | Delayed  -- ^ Do not allocate, re-compute it whenever if needed.
  deriving (Eq, Show, Typeable)
```

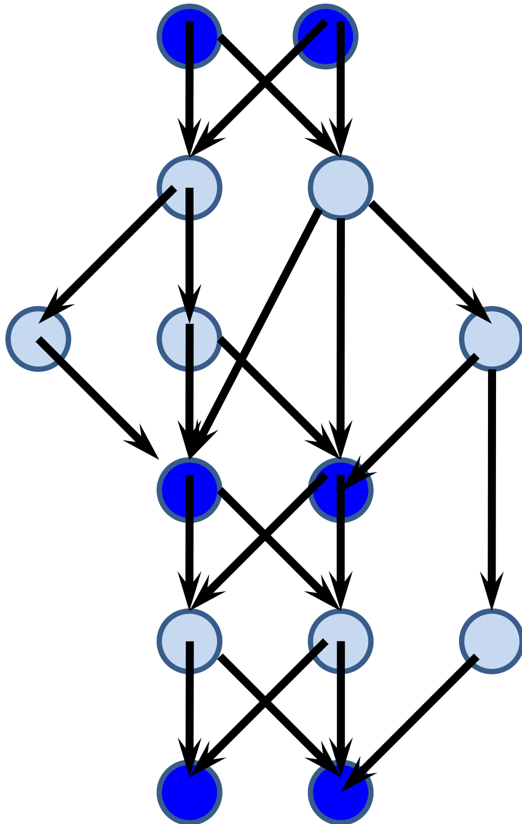
Analysis

```
optimize level = case level of
  O0 -> gmap identity . writeGrouping . gmap boundaryAnalysis . 
gmap decideAllocation
  _  -> optimize O0
```

- decide allocation
- boundary analysis
- dependency analysis and write grouping

Allocation

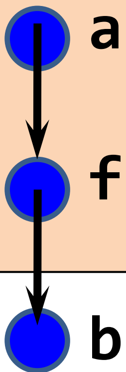
```
data Allocation
  = Existing -- ^ This entity is already allocated as a static variable.
  | Manifest -- ^ Allocate additional memory for this entity.
  | Delayed  -- ^ Do not allocate, re-compute it whenever if needed.
  deriving (Eq, Show, Typeable)
```



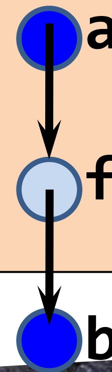
- some of the dataflow graph nodes are marked 'Manifest.'
- Manifest nodes are stored in memory.
- Delayed nodes are re-computed as needed.

- Less computation
- Less storage consumption & access

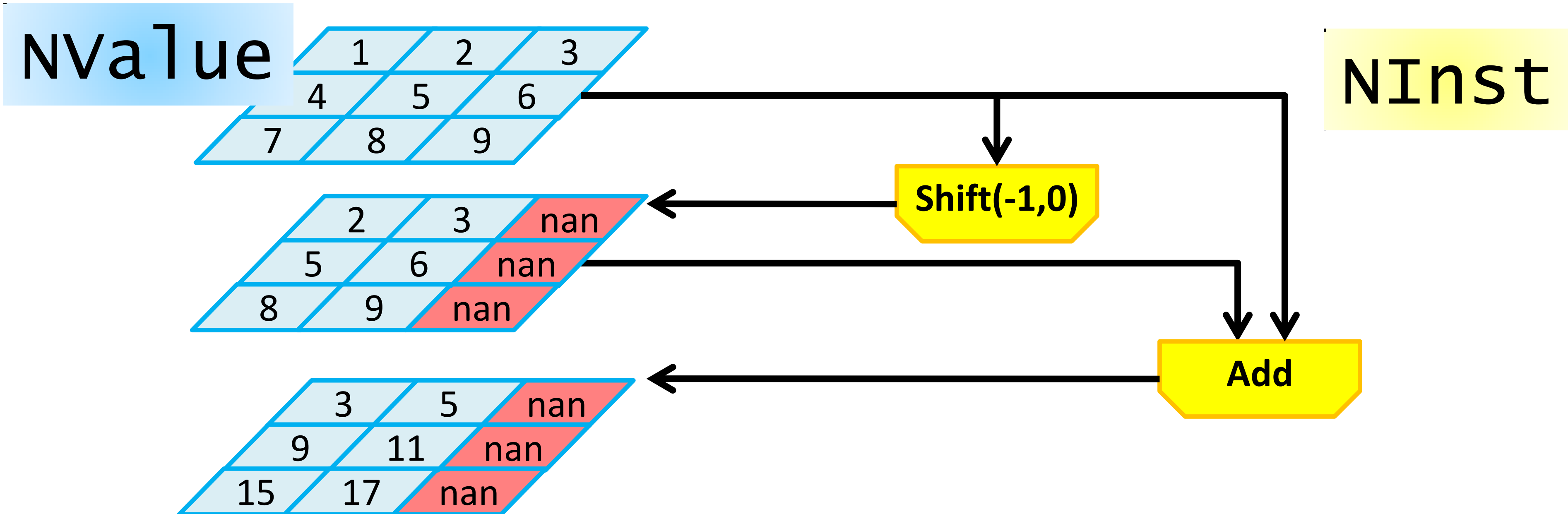
```
for(;;){
   f[i] = calc_f(a[i], a[i+1]);
}
for (;;){
   b[i] += f[i] – f[i-1];
}
```




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



Valid Boundary Analysis



- Shift operations create undefined regions in value.
- Boundary analysis trace this to find out valid regions for every node in the graph.
- How many additional mesh we need to obtain valid answers for desired region.
- To generate boundary-region communications.

write grouping

Kernel

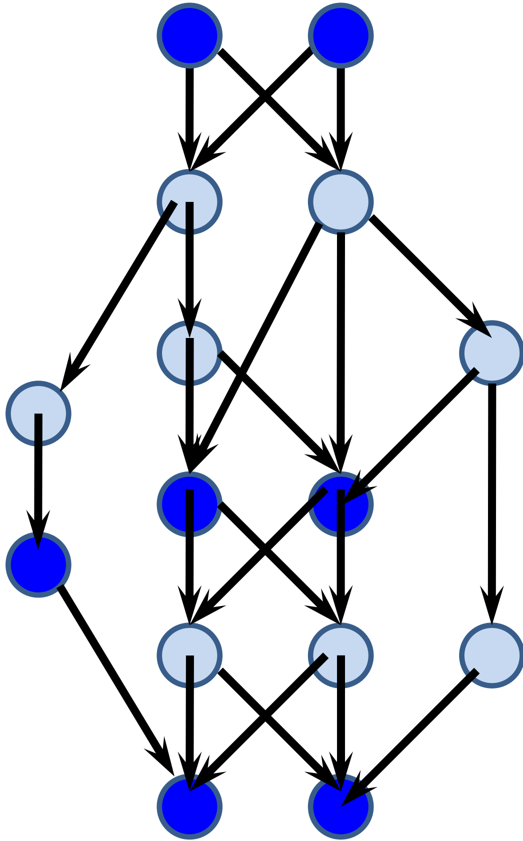
- a user-defined API of the generated class

Subkernel

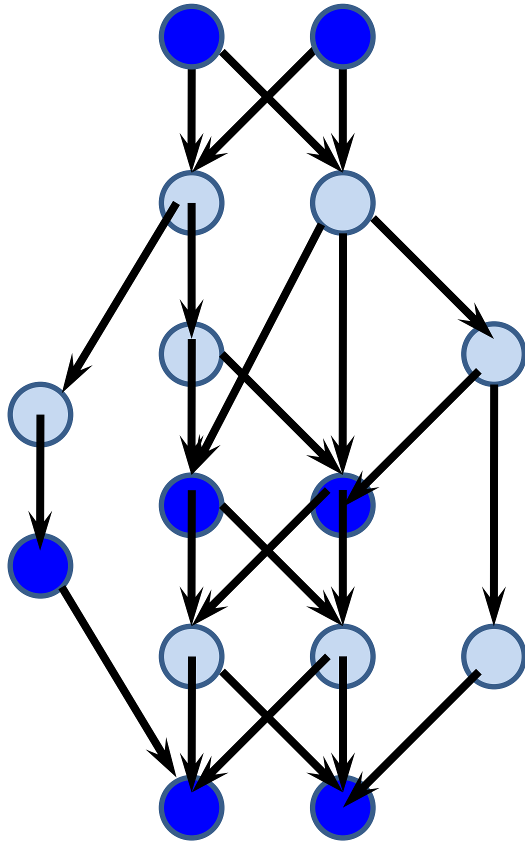
- a set of calculation executed in a loop
- = Fortran subroutine
- = CUDA __global__ kernel

```
void Life::proceed ()  {
  Life_sub_2(static_2_cell, manifest_1_67);
  Life_sub_3(static_1_generation, manifest_1_67, manifest_1_69,
manifest_1_74);
  (static_0_population) = (manifest_1_69);
  (static_1_generation) = (manifest_1_74);
  (static_2_cell) = (manifest_1_67);
}
```

a Kernel



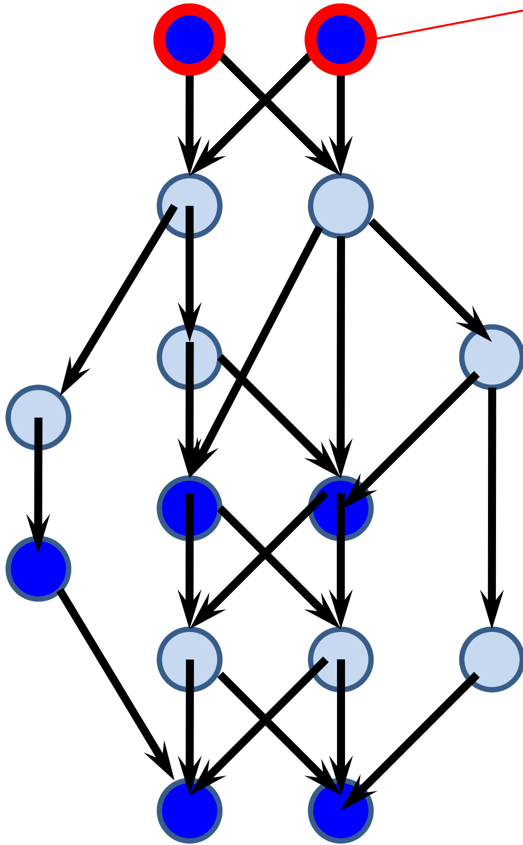
write grouping = a Kernel -> subkernels



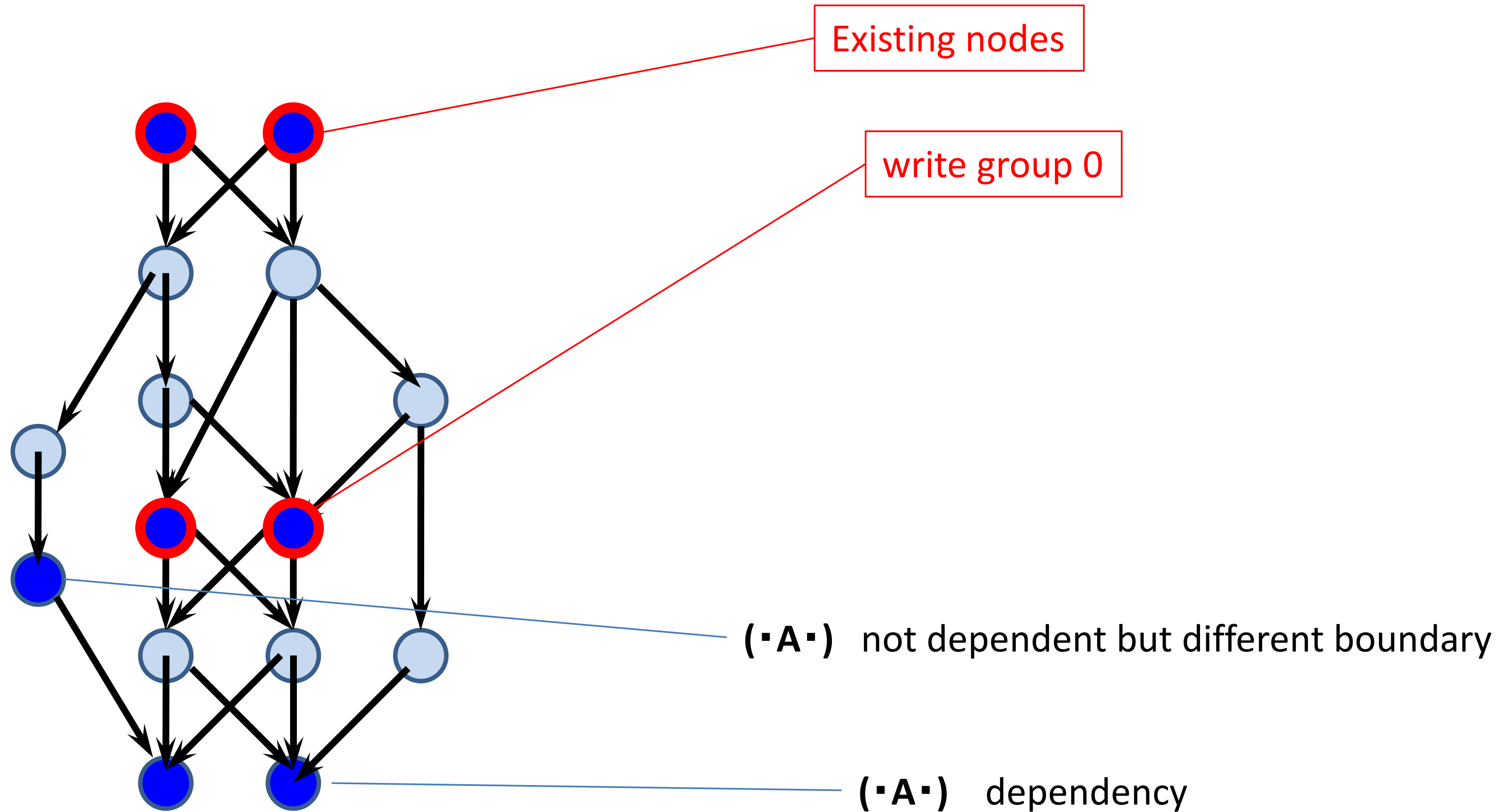
- all node written by one subkernel must have the same valid region
- nodes written by one subkernel must not depend on each other
- greedy

a Kernel

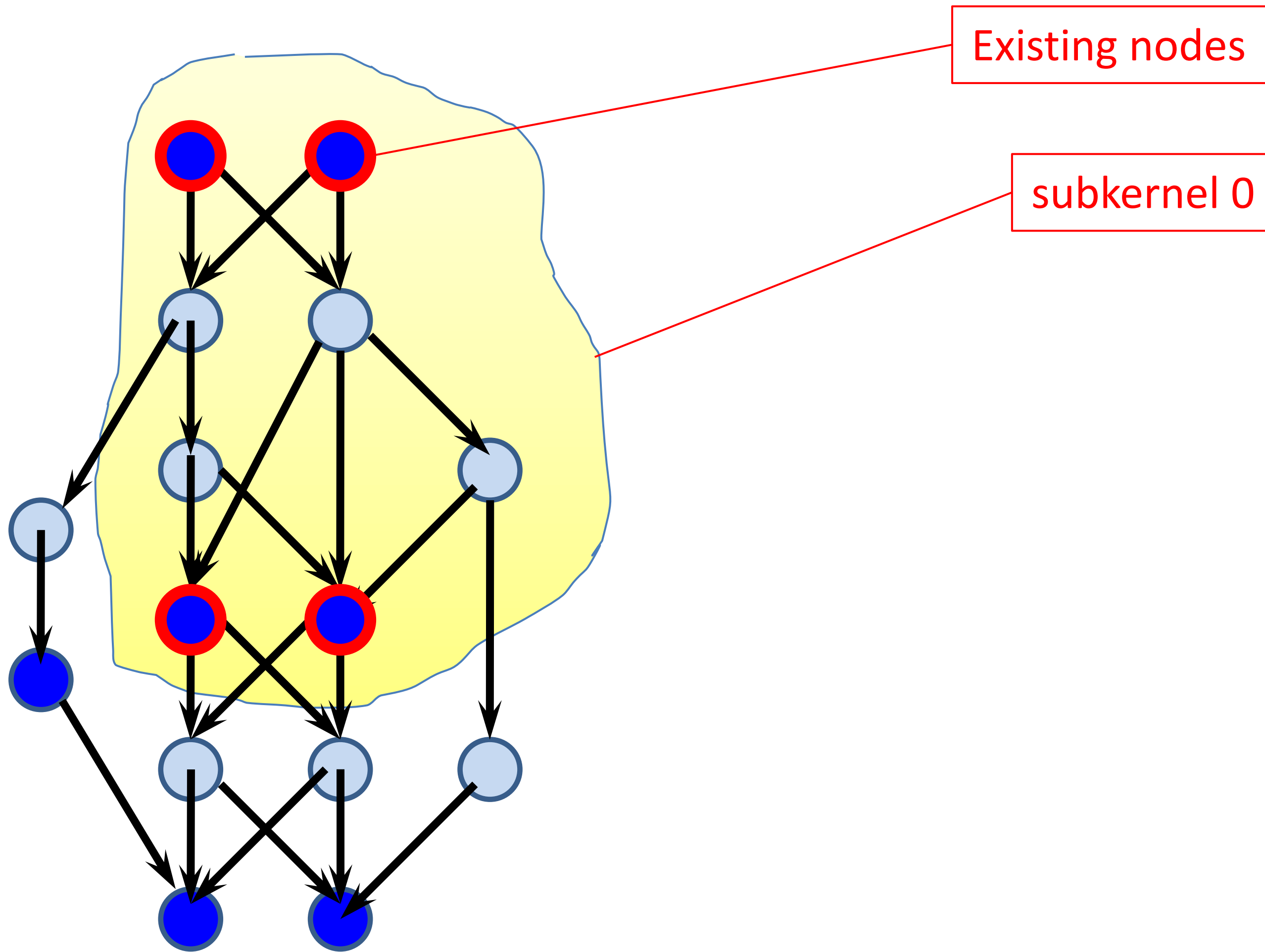
Existing nodes



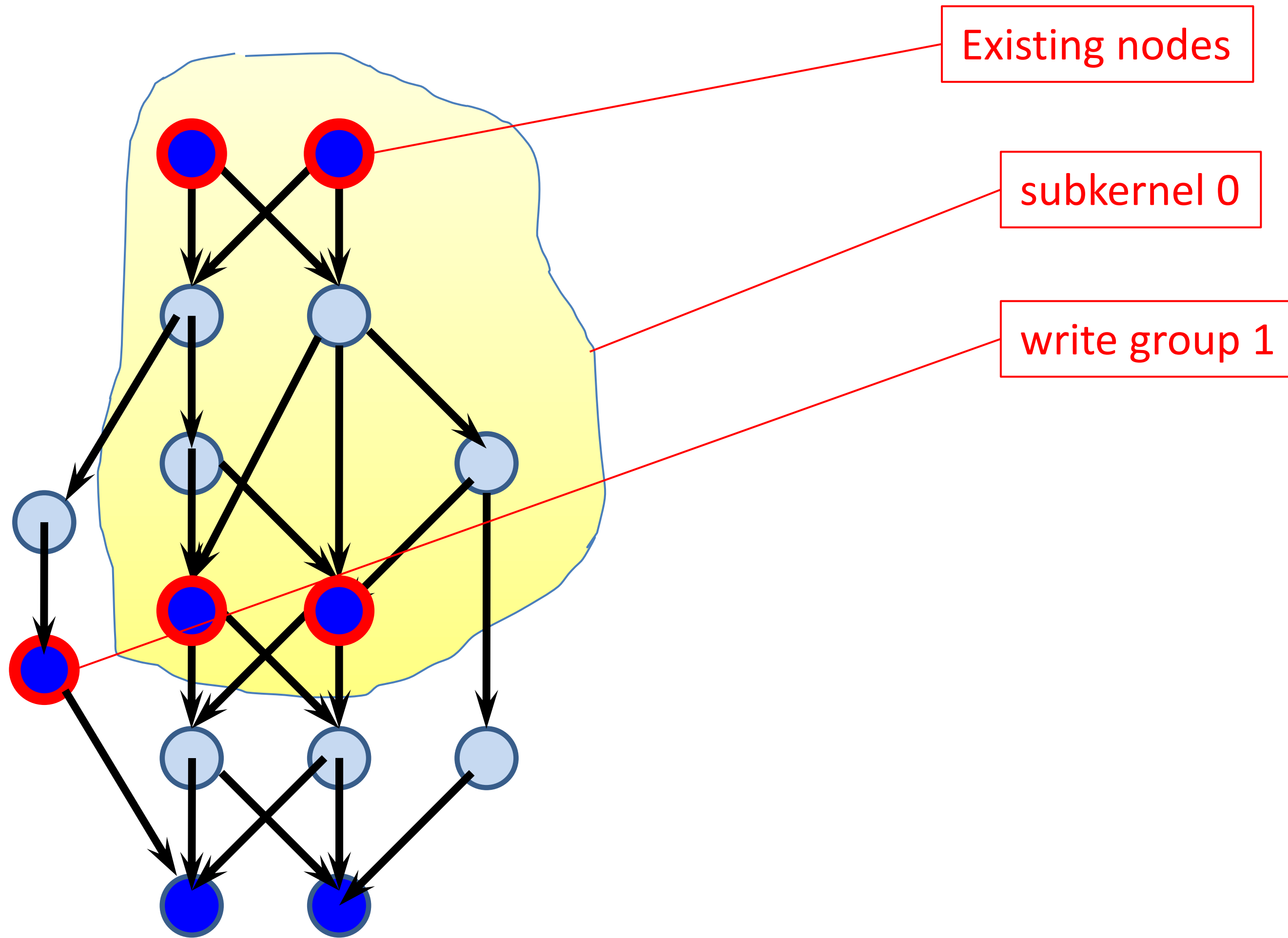
a Kernel



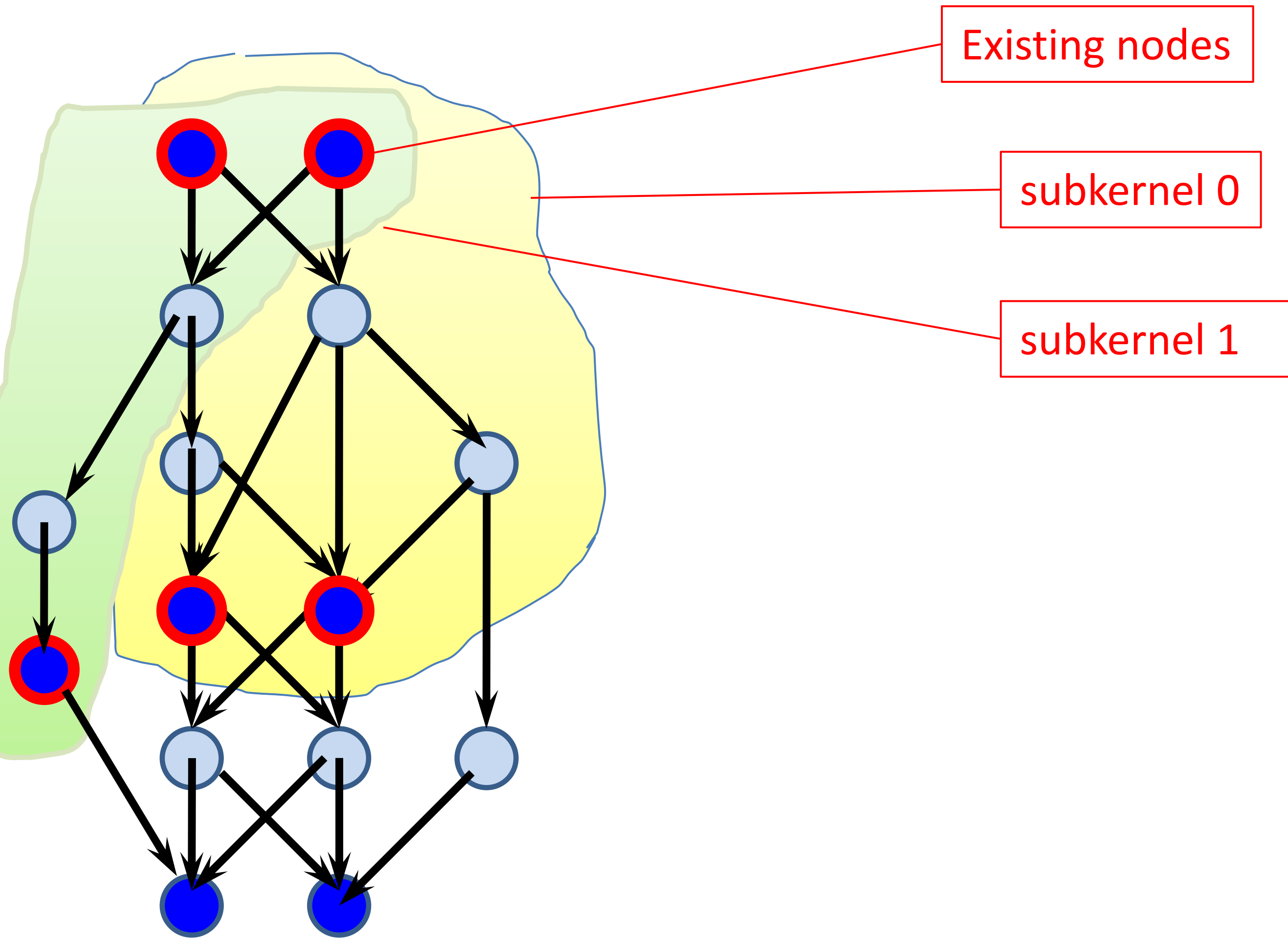
a Kernel



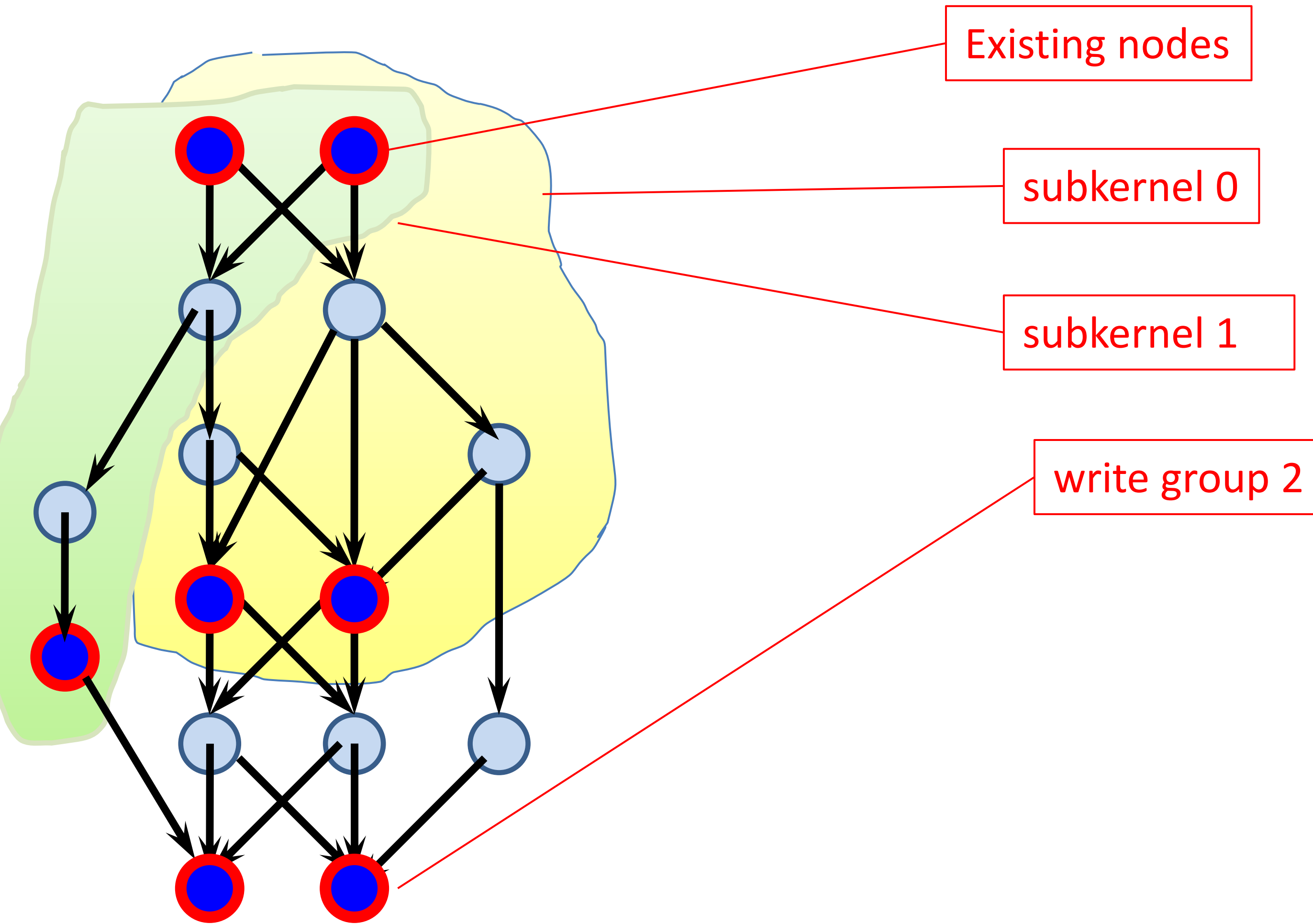
a Kernel



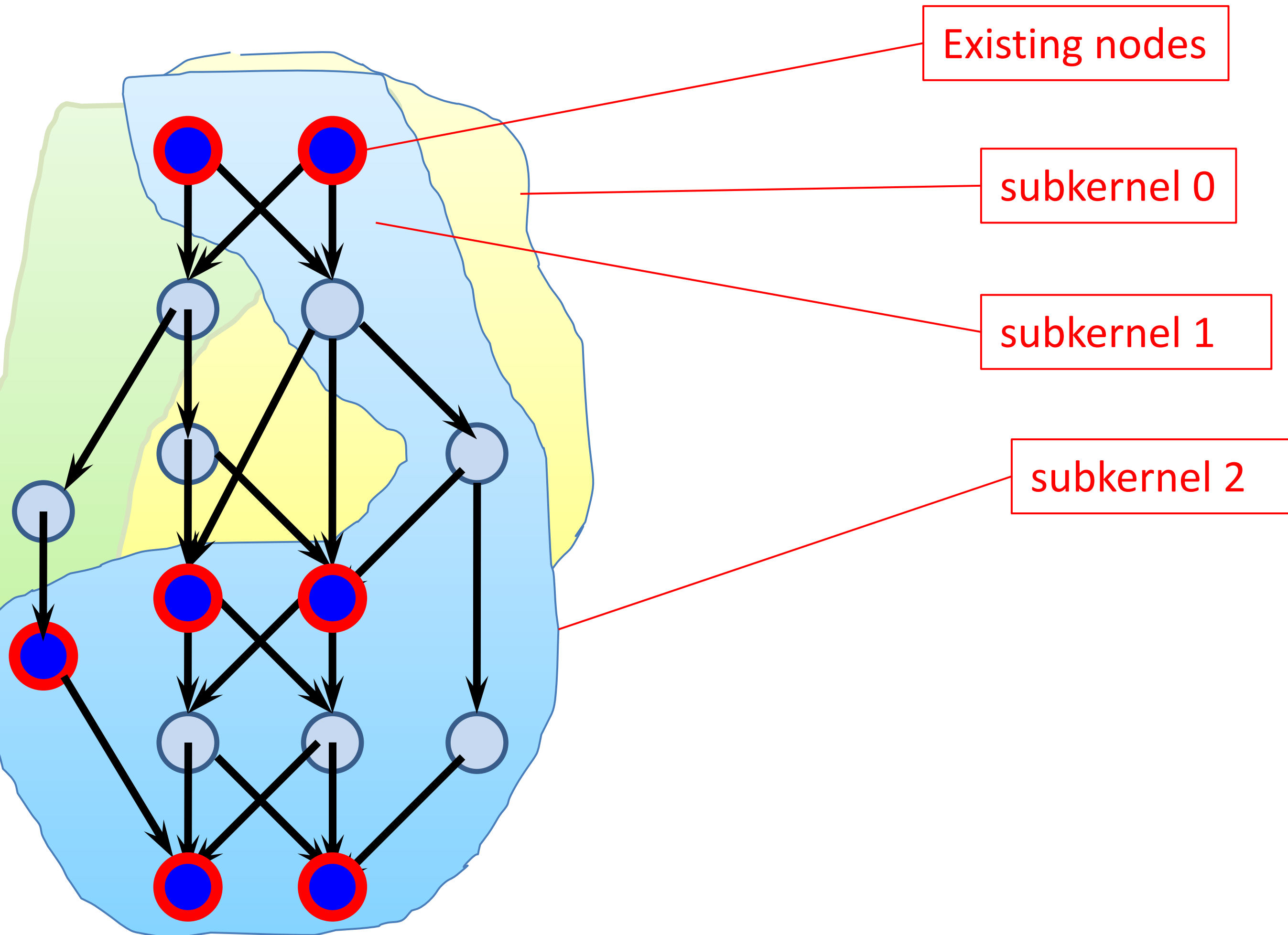
a Kernel



a Kernel



a Kernel



Diffusion Equation

- an example of code manipulation

```
diffuse :: BR -> B BR
diffuse field = do
  neighbours <- fmap (map return) $ forM adjVecs (\v -> shift v field)
  num <- bind $ foldl1 (+) neighbours
  ret <- bind $ ((1/6) * num )
  return $ ret
```

```
buildProceed :: B ()
buildProceed = do
  pink  <- bind $ load Rlm.TLocal  (undefined::Real) $ mkName "pink"
  black <- bind $ load Rlm.TLocal  (undefined::Real) $ mkName "black"

  pink1  <- diffuse black
  black1 <- diffuse pink

  pink2  <- diffuse black1
  black2 <- diffuse pink1

  pink3  <- diffuse black2
  black3 <- diffuse pink2

  store (mkName "pink") $ pink3
  store (mkName "black") $ black3
```

Diffusion Equation Example

Annotation	size of .cpp file	number of subkernels	memory consumption
no annotation	1019 lines	4	5 x N
Manifest	301 lines	6	9 x N

```
diffuse :: BR -> B BR
diffuse field = do
  neighbours <- fmap (map return) $ forM adjVecs (\v -> shift v field)
  num <- bind $ foldl1 (+) neighbours
  ret <- bind $ ((1/6) * num )
  return $ ret
```

```
diffuse :: BR -> B BR
diffuse field = do
  neighbours <- fmap (map return) $ forM adjVecs (\v -> shift v field)
  num <- bind $ foldl1 (+) neighbours
  ret <- bind $ (Anot.add Alloc.Manifest <?> (1/6) * num )
  return $ ret
```

homework
「内職」

```
-- | implement the loop for each subroutine
loopMaker :: Opt.Ready v g => Env v g -> Realm.Realm -> Plan.SubKernelRef v g AnAn -> [C.Statement]
loopMaker env@(Env setup plan) realm subker = case realm of
  Realm.Local ->
    [ C.StmtPrpr $ C.PrprPragma "omp parallel for",
      C.StmtFor
      (C.VarDefSub loopCounter (intImm 0))
      (C.Op2Infix "<" (C.VarExpr loopCounter) (C.toDyn (product boundarySize)))
      (C.Op1Prefix "++" (C.VarExpr loopCounter)) $
      [C.VarDefSub addrCounter codecAddr] ++
      loopContent
    ]
  Realm.Global -> loopContent
```

all Paraiso-generated code
become OpenMP Compatible
in one line!

x8 ~~faster!~~

cpu consumption

```
top - 10:15:37 up 56 days, 23:55,  6 users,  load average: 6.79, 3.86, 1.7
Tasks: 172 total,   2 running, 165 sleeping,   5 stopped,   0 zombie
Cpu(s):100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.
Mem:  12324684k total,  8167996k used,  4156688k free,   508816k buffers
Swap:   265068k total,     4032k used,   261036k free,  6746332k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6233 nushio    20   0  312m 240m 1452 R  795  2.0  27:20.49 kh.out
 6252 nushio    20   0 10788 1168  824 R    0  0.0   0:00.10 top
    1 root      20   0  3920  516  432 S    0  0.0   0:21.46 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.04 kthreadd
    3 root      20   0     0    0    0 S    0  0.0   0:02.32 ksoftirqd/0
    6 root      RT   0     0    0    0 S    0  0.0   0:00.00 migration/0
    7 root      RT   0     0    0    0 S    0  0.0   0:04.39 watchdog/0
```

Hakubi

Hakubi

糸冬