

Copyright (C) <2020> Yulan Fang

Palabos 的数据处理器

Data Processors in Palabos

目录

1	数据处理器	1
1.1	介绍	1
1.2	源程序	1
1.3	版权声明	2
2	数据分析	3
2.1	总结	5
2.2	dataAnalysisFunctional3D.h/hh	8
2.2.1	第一部分: block-lattice 的分析 Analysis of the block-lattice	8
2.2.1.1	BlockLattice 的简化版数据处理功能 Reduc- tive Data Functionals for BlockLattice	8
2.2.1.2	BlockLattice 的数据处理功能 Data Functionals for BlockLattice	10
2.2.2	第二部分: scalar-field 的分析 Analysis of the scalar-field	16
2.2.2.1	ScalarField 的简化版数据处理功能 Reduc- tive Data Functionals for ScalarField	16
2.2.2.2	scalar-fields 的数据处理功能 Data Functionals for scalar-fields	17
2.2.3	第三部分: tensor-field 的分析 Analysis of the tensor-field	20
2.2.4	第三部分: NTensor-field 的分析 Analysis of the NTensor-field	24
2.3	dataAnalysisGenerics3D.h	25
		vii

2.4	dataAnalysisWrapper3D.h/hh	25
2.4.1	第一部分: block-lattice 的 Atomic-block warppers Atomic-block wrappers: Block-Lattice	25
2.4.1.1	简化版函数 Reductive functions	25
2.4.2	第二部分: Scalar-Field 的 Atomic-block warppers Atomic-block wrappers: Scalar-Field	27
2.4.2.1	简化版函数 Reductive functions	27
2.4.3	第三部分: Tensor-Field 的 Atomic-block warppers Atomic-block wrappers: Tensor-Field	29
2.4.3.1	简化版函数 Reductive functions	29
2.4.4	第四部分: Block-Lattice 的 Multi-block warppers Multi-block wrappers: Block-Lattice	31
2.4.4.1	简化版函数 Reductive functions	31
2.4.5	第五部分: Scalar-Field 的 Multi-block warppers Multi-block wrappers: Scalar-Field	36
2.4.5.1	简化版函数 Reductive functions	36
2.4.6	第六部分: Tensor-field 的 Multi-block warppers Multi-block wrappers: Tensor-field	37
2.4.6.1	简化版函数 Reductive functions	37
2.4.7	第七部分: NTensor-field 的 Multi-block warppers Multi-block wrappers: NTensor-field	38
2.4.7.1	简化版函数 Reductive functions	38
2.5	dataInitializerFunctional3D.h/hh	38
2.5.1	第一部分: block-lattice 的初始化 Initialization of the block-lattice	38
2.5.2	第二部分: scalar 和 tensor-fields 的初始化 Initialization of scalar- and tensor-fields	43
2.6	dataInitializerGenerics3D.h	45

<i>Contents</i>	ix
2.6.1 第一部分: block-lattice 的初始化 Initialization of the block-lattice	45
2.6.2 第一部分: scalar 和 tensor 的初始化 Initialization of the scalar- and tensor-field	45
2.7 dataInitializerWrapper3D.h/hh	46
2.7.1 第一部分: block-lattice 的初始化: atomic-blocks Initialization of the block-lattice: atomic-blocks	46
2.7.2 第二部分: block-lattice 的初始化: multi-blocks Initialization of the block-lattice: multi-block	47
2.7.3 第三部分: scalar 和 tensor-fields 的初始化: Atomic-Block Initialization of scalar- and tensor-fields: Atomic-Block	50
2.7.4 第四部分: scalar 和 tensor-fields 的初始化: Multi-Block Initialization of scalar- and tensor-fields: Multi-Block	50



1

数据处理

1.1 介绍

在 Palabos 的算例中，我们运用到大量数据处理器来处理不同区块的信息传递。在 User Guide 里，数据处理器按照用途被分为了三类，分别为初始化，物理模型，数据分析。在这里我们不过多探讨基础的介绍，而是去深入了解其源代码背后的逻辑结构，程序源文件位于 `src/dataProcessors`。

关于 Palabos 的其他内容，欢迎去我的博客阅读

https://blog.csdn.net/qq_40259141

对于 Palabos 的探讨，欢迎来这里的 Issues 讨论

<https://github.com/Yulan-Fang/PalabosCodeExplanation/issues>

开源解决问题，我们才能进步地更快。关于错误，请读者联系我改正，我的邮箱是：ahdhfang@hotmail.com，谢谢。

1.2 源程序

这里我们主要看下面 1 至 6 的源码，从其命名我们可以看出，这是关于初始化和数据分析的数据处理器。源程序分为 2D 和 3D 的部分，但我们只看 3D 的程序。

1. `dataAnalysisFunctional`
2. `dataAnalysisGenerics`
3. `dataAnalysisWrapper`

4. `dataInitializerFunctional`
5. `dataInitializerGenerics`
6. `dataInitializerWrapper`
7. 其他

1.3 版权声明

Palabos's Data Processors 的文本为 Yulan Fang 的 Copyright (C) 2020。
The GNU General Public License 在本 pdf 文件同目录下。

2

数据分析

在解读程序之前，先贴上它的版权声明，Palabos 应当是可以免费传播，修改。在这里我倾向于工具书的目的来解读这些函数的功能，便于读者检索。

在介绍 Palabos 源代码功能之前，我想先介绍一下数据处理器的主要构成，它的代码定义一定是先从 template 开始定义 Descriptor，接着是 class XXXXX3D: public XXXprocessingfunctional，然后 public 里面含有相关的操作，包括 process 相关的，clone 和 get.TypeOfModification。

当然数据处理器调用还会涉及到 integrateProcessingFunctional 和 applyProcessingFunctional，这里不过多介绍，可去我的博客查阅相关介绍。

我将总结放到代码介绍之前，是希望读者对数据处理器有一个大概的认识，并且在不同数据处理器相同功能上我不过多作重复解释，相应的重复描述会随着介绍逐渐减少。

This file is part of the Palabos library.

The Palabos software is developed since 2011 by FlowKit-Numeca Group Sarl(Switzerland) and the University of Geneva (Switzerland), which jointly own the IP rights for most of the code base. Since October 2019, the Palabos project is maintained by the University of Geneva and accepts source code contributions from the community.

Contact:

Jonas Latt

Computer Science Department

University of Geneva

7 Route de Drize

1227 Carouge, Switzerland

jonas.latt@unige.ch

The most recent release of Palabos can be downloaded at
<<https://palabos.unige.ch/>>

The library Palabos is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

The library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details. You should have received a copy of the GNU Affero General Public License along with this program. If not, see
<<http://www.gnu.org/licenses/>>.

2.1 总结

```
template<typename T, template<typename U> class Descriptor>
void PackedRhoBarJfunctional3D<T,Descriptor>::process
    (Box3D domain, BlockLattice3D<T,Descriptor>& lattice, NTen-
sorField3D<T>& rhoBarJField)
    {.....}
    ... ..

template<typename T, template<typename U> class Descriptor>
void PackedRhoBarJfunctional3D<T,Descriptor>::
getTypeOfModification(std::vector<modif::ModifT>& modified) const
{
    modified[0] = modif::nothing; // lattice
    modified[1] = modif::staticVariables; // rhoBarJ
}
```

在编写数据处理器时，我们注意到 process 后的括号内，第一个为 lattice，第二个为 rhoBarJField，所以在后侧的 getTypeOfModification 内 modified[0] 即表示第一个 lattice，modif::nothing 则表示本质上不修改格点数据，而 modif::staticVariables 则表示修改格点数据。PackedRhoBarJfunctional3D 与其他涉及 rhoBarJ 的源码不同，它没有使用 processGenericBlocks 来作相应展开，不过其代码实现思路仍然与那些使用 GenericBlocks 的数据处理器相同。

它的选项总共有：nothing, staticVariables, dynamicVariables, allVariables, dataStructure, undefined。

staticVariables 表示修改分布函数，ExternalField 的量。

dynamicVariables 表示修改了 dynamics object 而不修改分布函数等量。

allVariables 表示 static 和 dynamic 的数据都被改变。

dataStructure 表示重建 dynamics 并获得 static 和 dynamics 的数据。

undefined 一般用于 debug。

在数据处理器介绍中,有涉及到多个场的,如 4 个场的 BoxRhoBarJPiNeq-functional3D,就有 “modified[3] = modif::staticVariables;// piNeq”, 即从 0 数到 3 分别定义其 4 个 fields 涉及的数据变动。

因为大多数数据处理器都是仅仅作用在列举的 fields 上,而在这些数据处理器中的 public 函数中,基本上都是使用 process。如下两例,为一个标量场,和一个 lattice 与张量场。

```
virtual void process(Box3D domain,
ScalarField3D<T>& scalarField);

void BoxSumForcedEnergyFunctional3D<T,Descriptor>::process
( Box3D domain, BlockLattice3D<T,Descriptor>& lattice,
TensorField3D<T,Descriptor<T>::d>& force )
```

不过有时候我们涉及到 rhoBarJ 这种组合场,我们则使用 process-GenericBlocks 来以 fields 处理,在数据处理器内部,fields 的值会通过指针运算符被代入到计算中。

```
void BoxRhoBarJfunctional3D<T,Descriptor>::
processGenericBlocks (Box3D domain, std::vector<AtomicBlock3D*>
fields )
```

最后是 Box3D domain 的存在,它主要是提供了诸如 iX=domain.x0 这样的定位功能,方便我们遍历该 field 的所有格点。

那么在 class 那边的定义中,我们见到:

对于单个 lattice

```
public ReductiveBoxProcessingFunctional3D_L<T,Descriptor>
public BoxProcessingFunctional3D_L<T,Descriptor>
```

对于两个 lattice

```
public BoxProcessingFunctional3D_LL<T,Descriptor,T,Descriptor>
public BoxProcessingFunctional3D_LL<T1,Descriptor1,T2,Descriptor2
```

对于一个 lattice 和一个 scalarField

```

public BoxProcessingFunctional3D_LS<T,Descriptor,T>
对于 rhoBarJ 这样的组合场
public BoxProcessingFunctional3D
对于一个 lattice 和一个 TensorField
public BoxProcessingFunctional3D_LT<T,Descriptor,T,Descriptor<T>::d>
对于一个 lattice 和组合场 rhoBarJ
public BoxProcessingFunctional3D_LN<T,Descriptor,T>

```

像 BoxRhoBarJfunctional3D 这种常用的数据处理器，其 rhoBarJ 的定义是 `std::vector<AtomicBlock3D*> fields`。而在 PackedRhoBarJfunctional3D 里，它的定义是 `NTensorField3D<T>& rhoBarJ`。

```

对于一个 scalarField 和组合场 rhoBarJ
BoxProcessingFunctional3D_SN
如 virtual void process(Box3D domain, ScalarField3D<T>& density,NTensorField3D<T>& rhoBarJ);

```

这些例子是想说明，BoxProcessingFunctional3D 后面下划线接字母代表了需要操作的场的性质，L 表示 BlockLattice，S 为 scalarField，T 是 TensorField 等。

最后一点是关于 Box3D domain 和 “Dot3D offset = computeRelativeDisplacement (lattice, xxx)”。我们通过 domain.x0 至 domain.x1 等参数可以遍历 lattice 上所有 cell。但同时我们也需要知道对应位置的其他场。比如我们耦合了两个场，一个是 Blocklattice，一个是 TensorField，我们通过 computeRelativeDisplacement 得到 offset，表示这两个场的距离。在 C++ 里面我们通过地址访问变量的值，那么 (iX,iY,iZ) 反映的是 lattice 的格点坐标，现在我们知道距离 offset，那么 (iX+offset.x, iY+offset.y, iZ+offset.z) 是不是就意味着另一个场对应的格点坐标呢？（我没有学过 c++，这么说明是为了帮助读者理解，如有错误希望懂行的指出。）

2.2 dataAnalysisFunctional3D.h/hh

2.2.1 第一部分: block-lattice 的分析

Analysis of the block-lattice

2.2.1.1 BlockLattice 的简化版数据处理功能

Reductive Data Functionals for BlockLattice

BoxSumRhoBarFunctional3D 作用于 lattice 上, 本质上不修改格点数据, 会遍历该 lattice 上所有 cell, 进行如是操作: “statistics.gatherSum(sumRhoBarId, cell.getDynamics().computeRhoBar(cell));”。其中 “gatherSum” 位于 core/blockStatistics.h/cpp 文件, 起到的作用是将当下 cell 的 rhoBar 值添加到一个临时的”tmpSum[whichSum]” 中。

因为在代码里存在 “tmpSum[whichSum] += value;”, 即该 tmpSum 会累计所有遍历到的 cell 所计算得到的值。关于 “gatherSum” 下文同理所以不再重复

关于 statistics 的信息, 可以点击该网址 [plb::BlockStatistics Class Reference](#) 跳转查看 Palabos Developer Guide 中的描述。

BoxSumEnergyFunctional3D 作用于 lattice 上, 本质上不修改格点数据, 会遍历该 lattice 上所有 cell, 读取该 cell 的速度并储存为 velocity, 并计算向量 velocity 的 norm-square, 进行如是操作: “statistics.gatherSum(sumEnergyId, uNormSqr);”。起到的作用是将当下 cell 的 uNormSqr 值添加到一个临时的”tmpSum[whichSum]” 中。

BoxSumForcedEnergyFunctional3D 作用于 lattice 和 force 上, 本质上不修改两个 block-lattice 的格点数据, 会计算 lattice 和 force 的 offset (用 computeRelativeDisplacement(lattice, force) 来实现) 以得到 lattice 对应位置的 force 数据为 f, 遍历 lattice 上的所有 cell, 计算速度为 velocity, 接着判断该 cell 的 dynamics, 默认情况下 “lattice.get(iX,iY,iZ).getDynamics().hasMoments()” 会返回 true, 而对于 BounceBack, SpecularReflection, NoDynamics 都是返回 false。如果为 true 则让向量速度的三个分量加上 0.5 乘以 force 的分量, 即 velocity[分

量] += (T) 0.5 * f[分量]; (分量为 0, 1, 和 2, 表示三个方向)。随后计算新的 velocity 的 norm-square, 通过 gatherSum 累计起来到一个临时的"tmpSum[whichSum]"中。

BoxSumConstForcedEnergyFunctional3D 作用于 lattice 上, 本质上不修改格点数据, 会遍历该 lattice 上所有 cell, 作用与上一个功能相似, 差别在于这里 force 是常数, 恒定值。

BoxSumCustomForcedEnergyFunctional3D 作用于 lattice 上, 本质上不修改格点数据, 会遍历该 lattice 上所有 cell, 作用与上一个功能相似, 差别在于这里 force 是用户自定义值。

CountLatticeElementsFunctional3D 它在 dataAnalysisWrapper3D 和 dataAnalysisGenerics3D 中均出现, 这个以后说明。

ScalarFieldSingleProbe3D 作用于 scalarField 上, 本质上不修改格点数据, 根据预先设定好的 position 来线性插值得到 cell 坐标 (这里每个 position 的坐标插值计算了周围的 9 个 cell)。这里的源码涉及到了 linearInterpolationCoefficients, 它的定义位于 src/finiteDifference/interpolations3D.hh。主要是根据 position 计算 cell-Pos 和 weights, 再通过 block 将 cell 位置转换为局部坐标。通过 gatherSum 来累计 cell 的 scalar 值。其中通过 scalarIds 来计数。

DensitySingleProbe3D 作用于 lattice 上, 本质上不修改格点数据, 根据预先设定好的 position 来线性插值得到 cell 坐标, 对 cell 进行密度计算并通过 gatherSum 进行累加。

InternalDensitySingleProbe3D 作用与上一个功能相似, 但多出一个 ids, 即在 gatherSum 中仅累加特定 id 的值。

VelocitySingleProbe3D 作用于 lattice 上, 本质上不修改格点数据, 根据预先设定好的 position 来线性插值得到 cell 坐标, 计算速度, 并且因为速度有三个分量而分别通过 gatherSum 累加。

InternalVelocitySingleProbe3D 作用与上一个功能相似，但多出一个 `velIds`，即在 `gatherSum` 中仅累加特定 `velIds` 的值。

VorticitySingleProbe3D 作用于 `lattice` 上，本质上不修改格点数据，根据预先设定好的 `position` 来线性插值得到 `cell` 坐标，计算涡量，并且因为速度有三个分量而分别通过 `gatherSum` 累加。

InternalVorticitySingleProbe3D 作用与上一个功能相似，但多出一个 `vorticityIds`，即在 `gatherSum` 中仅累加特定 `vorticityIds` 的值。

2.2.1.2 BlockLattice 的数据处理功能

Data Functionals for BlockLattice

CopyPopulationsFunctional3D 作用于两个 `lattice` 上，复制第一个 `lattice` 至第二个 `lattice` 数据，本质上不修改第一个 `lattice` 的格点数据，会修改第二个 `lattice` 上的格点数据，会计算 `offset` 得到对应位置，第一个 `lattice` 坐标由 `iX`, `iY`, `iZ` 决定，第二个 `lattice` 的坐标则为 `iX + offset.x`，以此类推。通过 `attributeValues` 来赋值即可，其中 `lattice.get(iX,iY,iZ)` 得到的是该点的分布函数。

CopyConvertPopulationsFunctional3D 作用与上一个功能相同，但两个 `lattice` 的 `Descriptor` 不同。

LatticeCopyAllFunctional3D 作用与上上一个功能相同，但第二个 `lattice` 的分布函数和 `dynamics` 都会被修改，在总结可以找到有关“`modif::allVariables;`”的解释。

LatticeRegenerateFunctional3D 作用与上一个功能相同，即将第一个 `lattice` 的值复制到第二个 `lattice` 中，同时也赋值了第一个 `lattice` 的 `dynamics`。由此 `dynamics object` 必须要重新生成，即“`modif::dataStructure`”。

BoxDensityFunctional3D 作用于一个 `lattice` 和一个 `scalarField` 上，本质上不修改 `lattice` 的格点数据，修改 `scalarField` 上的格点数据。为

scalarField 上对应的位置设置成 cell 的密度 Density。

BoxRhoBarFunctional3D 与上相同，赋值 rhoBar 至 scalarField。

BoxRhoBarJfunctional3D 作用于 fields 上，这里 fields 含有 3 个 fields，第一个为 BlockLattice3D lattice，第二个为 ScalarField3D rhoBar，第三个为 TensorField3D j，本质上不改变 lattice 上的格点数据，改变 rhoBar 和 j 的格点数据，先计算出对应位置，接着通过经典的“Cell<T,Descriptor> const& cell = lattice.get(iX,iY,iZ);”，来对 cell 进行“getDynamics().computeRhoBarJ(.....)”，将得出的 rhoBar 和 j 值返回到对应的 rhoBar 和 j 的 fields 上。

延伸一下：算例 movingWall 中，则定义了 rhoBarJarg 来包含 lattice, rhoBar, 和 j, 后续通过 integrateProcessingFunctional 将这个 rhoBarJarg 集成到每次迭代中，实现 rhoBar 和 j 参与到碰撞迁移中。（因为应用浸入边界法需要 j 进行插值）

MaskedBoxRhoBarJfunctional3D 作用与上相似，含有 4 个 fields，第四个 maskField 起到判断作用，即只对 flag 上的格点进行 BoxRhoBarJfunctional3D 操作。

BoxJfunctional3D 作用于 lattice 和一个 tensorField 上，本质上不修改 lattice 的格点数据，对 tensorField 上返回 j。

BoxRhoBarJPiNeqfunctional3D 与上相似，仅多出 piNeqField，同样是 lattice 格点数据不动，其他赋值。

PackedRhoBarJfunctional3D 这里自行定义了 rhoBar 和 j，随后通过 lattice 计算出 cell 的 rhoBar 和 j 的值。接着通过指针定义 scalarField 的 rhoBarJ，先赋值 rhoBar，后给 rhoBar 的地址加一，再继续赋值 j 至 rhoBar 中。我们可以将 NTensorField3D<T>& rhoBarJField 理解为两个缝合在一起的场，遍历完前一半的位置为 rhoBar，后一半的位置为 J。

DensityFromRhoBarJfunctional3D 标量场 density 值会被改变，直

接通过 `NTensorField3D<T>& rhoBarJField` 得到 `rhoBar` 值，加 1 得到 `density` 并赋值给标量场 `density`。

VelocityFromRhoBarJfunctional3D 这里的 `rhoBarJ` 是以 `std::vector<AtomicBlock3D*> fields` 定义的，所以在 `processGenericBlocks` 内也是先展开来的，在作用前先判定 `bool velIsJ_ = false`。`velIsJ` 则以 `false` 为 1，其他为 0，在计算中，if 判定 `!velIsJ`，为 1 则会除以 `rho`，因为这儿运用了二重指针得到 `rhoBar` 后添加 1。逻辑上总体来说就是判定这儿的速度 `velocity` 是不是等于动量 `j`，相等就直接赋值到 `velocity` 上，不等就处理一下再去赋值。

```

    动量 j 和速度 u 以及 rhoBar 和 rho 的关系如下
    rho = Descriptor<T>::fullRho(rhoBar);
    用 rho 计算 rhoBar
    rhoBar = Descriptor<T>::rhoBar(rho);
    j[::d] = rho * u[::d];
    u[::d] = 1./rho * j[::d];
    u[::d] = Descriptor<T>::invRho(rhoBar) * j[::d];

```

在 Palabos 中 `rhoBar` 和 `rho` 差距为 1，在其用户手册中解释过是为了计算效率的目的而这样做。

VelocityFromRhoBarAndJfunctional3D 基本与上相同，仅多出一个为 `j` 的场。

BoxKineticEnergyFunctional3D 对 `scalarField` 对应位置赋值 `norm-Sqr(velocity) / (T)2`。

BoxVelocityNormFunctional3D 先根据 `lattice` 计算 `cell` 的 `velocity`，随后赋值 `velocity` 的 `norm-square` 的平方根至标量场中。

BoxForcedVelocityNormFunctional3D 根据 `lattice` 计算 `cell` 的 `ve-`

locity, 后判断该 cell 的 dynamics, 如果返回 true 则 $\text{velocity}[:,d] += (T) 0.5 * f[:,d]$, 并计算更正后的 velocity 的 norm-square 再求平方根, 并赋值。

BoxConstForcedVelocityNormFunctional3D 与上同理, 仅 force 来源不从组合场, 而是自行设定常数 f 而来。

BoxCustomForcedVelocityNormFunctional3D 与上同理, 仅 force 来源不从组合场, 而是自行设定 f 函数而来。

BoxVelocityComponentFunctional3D 为标量场赋值 lattice.cell 的速度 velocity。

BoxForcedVelocityComponentFunctional3D 与上相同, 但多出了判断力 f 的影响的过程。

BoxConstForcedVelocityComponentFunctional3D 与上相同, 力为常数。

BoxCustomForcedVelocityComponentFunctional3D 与上相同, 力为函数。

BoxVelocityFunctional3D 为 tensorField 赋值 lattice.cell 的 velocity。

BoxForcedVelocityFunctional3D 为 tensorField 赋值 lattice.cell 的 velocity, 但考虑了力 f 的影响。

BoxConstForcedVelocityFunctional3D 与上相同, 力为常数。

BoxCustomForcedVelocityFunctional3D 与上相同, 力为函数。

BoxTemperatureFunctional3D 为 scalarField 赋值 lattice.cell 的温度 temperature。

BoxPiNeqFunctional3D 为 TensorField 赋值 PiNeq。

BoxShearStressFunctional3D 为 TensorField 赋值 ShearStress。

BoxStressFunctional3D 需声明 rho0 和是否为可压缩流体。先通过 lattice.cell 计算 stress 到 TensorFields 中, 判断是否为可压缩, 如果是可压缩流体, 则 s 除以密度 rho。最后计算 $T_{pressure} = (\rho - \rho_0) * \text{Descriptor} \langle T \rangle :: cs2$; 并让 s 减去它。

BoxStrainRateFromStressFunctional3D 先赋值 TensorField 对应位置的 PiNeq, 计算 omega, 判断 dynamics 来决定是否计算 dynamicOmega 来代替 omega, 计算 prefactor 并乘以 TensorField 中已存在的 PiNeq 完成赋值。

BoxShearRateFunctional3D 先通过 lattice.cell 得到 rhoBar, j, PiNeq 来计算 normPiNeq, rho, dynamicOmega 和对应的 tau, 来完成 ScalarField3D 的 shearRate 的计算。

BoxNTensorShearRateFunctional3D 与上相同, 但定义为 NTensorField3D<T>& shearRate。

BoxQcriterionFunctional3D 计算 vorticity 和 strain 的 normSqr, 并赋值到 qCriterion。

BoxComputeInstantaneousReynoldsStressFunctional3D 通过速度和平均速度计算 reynoldsStress。

BoxLambda2Functional3D 由 vorticity 和 strain 计算 lambda2, 应当是特征根。

BoxPopulationFunctional3D 对 scalarField 赋值 lattice.cell 的分布函数。

BoxEquilibriumFunctional3D 对 scalarField 赋值 lattice.cell 的平衡分布函数。

BoxAllPopulationsFunctional3D 对 tensorField 赋值 lattice.cell 的分布函数。

BoxAllEquilibriumFunctional3D 对 tensorField 赋值 lattice.cell 的平衡分布函数。

BoxAllNonEquilibriumFunctional3D 对 tensorField 赋值 lattice.cell 的非平衡分布函数，即 `cell[iPop]-cell.computeEquilibrium(iPop, rhoBar, j, jSqr)`。

BoxAllPopulationsToLatticeFunctional3D 从 tensorField 得到分布函数，并赋值到 lattice 中。

BoxOmegaFunctional3D 对 scalarField 赋值 lattice.cell 的 omega。

BoxKinematicViscosityFunctional3D 由 lattice.cell 计算 omega 并赋值 scalarField 相应的 Kinematic Viscosity。

BoxNTensorKinematicViscosityFunctional3D 与上相同，但赋值至 NTensorField3D 中。

BoxKinematicEddyViscosityFunctional3D 与上相同，但多出判断 dynamic object 部分，赋值到 scalarField。

BoxNTensorKinematicEddyViscosityFunctional3D 与上相同，但赋值到 NTensorField 中。

BoxExternalForceFunctional3D 从 ExternalField::forceBeginsAt 获取 lattice.cell 上的 force，并赋值到 tensorField 中

BoxExternalScalarFunctional3D 从 `lattice.cell.getExternal(whichScalar)` 得到该标量场值，赋值到 `scalarField` 中。

BoxExternalVectorFunctional3D 从 `lattice.cell.getExternal(vectorBeginsAt)` 得到该标量场值，赋值到 `tensorField` 中。

BoxDynamicParameterFunctional3D 为 `scalarField` 赋值 `dynamic parameters`。

BoxDynamicViscosityFunctional3D 为 `scalarField` 赋值 `Dynamic Viscosity`。

TagLocalDynamicsFunctional3D 需先声明 `dynamicsId` 和 `tags`，判断 `dynamicsId`，若是则并为赋值 `tags`。

2.2.2 第二部分：scalar-field 的分析

Analysis of the scalar-field

2.2.2.1 ScalarField 的简化版数据处理功能

Reductive Data Functionals for ScalarField

BoxScalarSumFunctional3D 由 `gatherSum` 累计标量场的值。

MaskedTextBoxScalarSumFunctional3D 由 `gatherSum` 累计标记 `flag` 的标量场的值。

BoxScalarIntSumFunctional3D 先将标量场的值转为 `int` 型，再由 `gatherSum` 累计。

MaskedTextBoxScalarAverageFunctional3D 需声明 `flag`，用 `mask` 得到位置，并通过 `gatherAverage` 累计值。这里不知括号中是否丢失 `averageScalarId`。

BoxScalarMinFunctional3D 因为 `BlockStatistics` 只会判断并保留最

大值，所以先得到 scalarField 的最大值，然后取负号，即最大值变成最小值，最小值变成最大值，这样可由 gatherMax 得到实际上的最小值。

MaskedTextBoxScalarMinFunctional3D 与上相同，需声明 flag。

BoxScalarMaxFunctional3D 与上相同，但取得为实际最大值。

MaskedTextBoxScalarMaxFunctional3D 与上相同，需声明 flag。

BoundingBoxScalarSumFunctional3D 得到该 field 的内部，外表面，棱角边，角落的值，并由不同的权重通过 gatherSum 累计。

CopyConvertScalarFunctional3D 标量场 2 复制标量场 1 的值。

2.2.2.2 scalar-fields 的数据处理功能

Data Functionals for scalar-fields

ExtractScalarSubDomainFunctional3D 为标量场 2 赋值标量场 1 的值。

ComputeScalarSqrtFunctional3D 为标量场 2 赋值标量场 1 的平方根。

ComputeAbsoluteValueFunctional3D 为标量场 2 赋值标量场 1 的绝对值。

ComputeTensorSqrtFunctional3D 为 TensorField 2 赋值 TensorField 1 的平方根。

A_lt_alpha_functional3D 需声明 alpha 的值，并判断标量场 A 与 alpha 的大小，输出判断结果至标量场 result。(a<alpha, lt 为 less than 缩写)

A_gt_alpha_functional3D 与上相同。(a>alpha)

A_plus_alpha_functional3D 与上相同, result 为 $a+\alpha$ 。

A_minus_alpha_functional3D 与上相同, result 为 $a-\alpha$ 。

Alpha_minus_A_functional3D 与上相同, result 为 $\alpha-a$ 。

A_times_alpha_functional3D 与上相同, result 为 $\alpha*a$ 。

A_dividedBy_alpha_functional3D 与上相同, result 为 a/α 。

Alpha_dividedBy_A_functional3D 与上相同, result 为 α/a 。

A_plus_alpha_inplace_functional3D 没有了 result, $A.get(iX,iY,iZ) += \alpha$ 。

A_minus_alpha_inplace_functional3D $A.get(iX,iY,iZ) -= \alpha$;

A_times_alpha_inplace_functional3D $A.get(iX,iY,iZ) *= \alpha$;

A_dividedBy_alpha_inplace_functional3D $A.get(iX,iY,iZ) /= \alpha$;

A_lt_B_functional3D 判断 $A < B$, 返回 0 和 1 至 result。

A_gt_B_functional3D 与上相同, 判断 $A > B$, 至 result。

A_plus_B_functional3D 与上相同, $A+B$, 至 result。

A_minus_B_functional3D 与上相同, $A-B$, 至 result。

A_times_B_functional3D 与上相同, $A*B$, 至 result。

A_dividedBy_B_functional3D 与上相同, A/B , 至 result。

A_plus_B_inplace_functional3D 没有了 result, $A += B$ 。

A_minus_B_inplace_functional3DA $-= B$ 。

A_times_B_inplace_functional3DA $*= B$ 。

A_dividedBy_B_inplace_functional3DA $/= B$ 。

UniformlyBoundScalarField3D 需声明 bound, data 的绝对值大于 bound 则继续潘丹 $\text{val} > T()$, 不清楚 $T()$ 意味着什么。

BoundScalarField3D 需声明 lowerBound 和 upperBound, data 的数据里小于 lowerBound 的会被赋值为 lowerbound 的值, 大于 upperBound 的值会被赋值为 upperbound 的值。

LBMsmoothen3DD::t 是权重, $D::c$ 是方向, result 得到得到周围所有的 q 个 data 值乘以权重, 再除以权重之和。

LBMsmoothenInPlace3D, 与上相同, 只是没有了 result, 结果直接赋值到 data 上。

Smoothen3D 以 iX, iY, iZ 为中心求和周围 data 的值, 并求均值, 赋值到 result 中。

MollifyScalar3D 需声明 mollifier 的特征长度“ $T l$,” 小于 envelope 的 Mollifying neighborhood “ $\text{plint } d$,” “Box3D globalDomain;” 如果为非循环边界, 则尺寸必须至多和算域一样大, 如果是循环边界, 则需要大于算域尺寸加上循环方向的 envelope。以及 “int exclusionFlag”, flag 的部分不会被 mollify。该数据处理器会遍历以 d 长度 scalar, 对于 flag 标记的部分, result 将直接被赋值为 scalar。对于其他部分, 将以 (iX, iY, iZ) 为中心, d 尺寸的区域计算 sum, 并后续将值转入 result。

LBMcomputeGradient3D 获取 scalarField 的 gradient, 并赋值到 TensorField 中。

UpdateMinScalarTransientStatistics3D 遍历 scalar 的值, 和 statistics 值对比, 如果 scalar 该点的值更小, 则 statistics 被赋值该更小的值。

UpdateMaxScalarTransientStatistics3D 与上相同, 但 statistics 保留最大值。

UpdateAveScalarTransientStatistics3D 需声明 n, 将 $\frac{1}{n}*(n-1)*statistics+scalar$ 的值写入 statistics 中。

UpdateRmsScalarTransientStatistics3D 需声明 n, 从 statistics 中获得 oldRms, 从 scalar 中获得 newData, 向 statistics 中写入 $\frac{1}{n}*((n-1)*oldRms^2+newData^2)$ 的平方根。

UpdateDevScalarTransientStatistics3D 与上相似, 不同的是从 statistics 中获得 oldDev。

2.2.3 第三部分: tensor-field 的分析

Analysis of the tensor-field

BoxTensorSumFunctional3D 同样是通过 gatherSum 求和, 但因为是 TensorField, 所以遍历坐标后, 还需要遍历 nDim。

MaskedTextBoxTensorSumFunctional3D 判断是否为 flag 位置, 然后求和。

MaskedTextBoxTensorAverageFunctional3D 同样是通过 gatherAverage 求和, 但因为是 TensorField, 所以遍历坐标后, 还需要遍历 nDim。

CopyConvertTensorFunctional3D 为 field2 赋值 field1 上的值, 但因为是 TensorField, 所以遍历坐标后, 还需要遍历 nDim。field1 格点数

据不变，field2 上的格点数据改变，这两个 fields 的 type name 不同。

ExtractTensorSubDomainFunctional3D 与上作用相似，但两个 fields 的 type name 相同。

ExtractTensorComponentFunctional3D 从 TensorField 上获取值，赋值到 scalarField 上，因为是 scalarField，所以不需要遍历 nDim。

ComputeNormFunctional3D 计算 TensorField 的 normSqr 的平方根，赋值到 scalarField 上。

ComputeNormSqrFunctional3D 计算 TensorField 的 normSqr，赋值到 scalarField 上。

BoxLocalMaximumPerComponentFunctional3D 遍历所有格点，将对应位置 tensorField 上的最大量的值赋值到 scalarField 上。（因为 scalarField 不像 TensorField 有维度上的分量，所以一个坐标点有多个分量）

ComputeSymmetricTensorNormFunctional3D 对其 6 个值进行处理，赋值到 scalarField 中。

ComputeSymmetricTensorNormSqrFunctional3D 同上。

ComputeSymmetricTensorTraceFunctional3D 对其 6 个值中的 xx, yy, zz 三个值进行处理，赋值到 scalarField 中。

BoxBulkGradientFunctional3D 分别求得三个方向的导数，并赋值到 TensorField 中。

BoxGradientFunctional3D 需要处理内部，外表面，棱角处，角落处的导数计算，并赋值到 TensorField 中。

BoxBulkVorticityFunctional3D 经 velocity 求 vorticity, 两个都是 TensorFields。

BoxBulkVorticityOrderSixFunctional3D 与上相似, 但调用的 vorticity 计算为 fdDataField::bulkVorticityXOrderSix。

BoxBulkVorticityOrderEightFunctional3D 与上相似, 但调用的 vorticity 计算为 fdDataField::bulkVorticityXOrderEight。

BoxVorticityFunctional3D 需要处理内部, 外表面, 棱角便, 角落处的 vorticity 计算, 并赋值到 TensorField 中。

BoxBulkHelicityFunctional3D 需要处理内部, 外表面, 棱角便, 角落处的 Helicity 计算, 并赋值到 TensorField 中。

BoxBulkStrainRateFunctional3D 通过 velocity 求 StrainRate。

BoxStrainRateFunctional3D 需要处理内部, 外表面, 棱角便, 角落处的 StrainRate 计算, 并赋值到 TensorField 中。

BoxBulkDivergenceFunctional3D 通过 velocity 求 divergence, 并赋值到 scalarField 中。

Tensor_A_plus_B_functional3DTensorField 版本的 result=A+B。

Tensor_Aminus_B_functional3DTensorField 版本的 result=A-B。

Tensor_A_times_B_functional3DTensorField 版本的 result=A*B。

IndexContraction_SymmetricTensor_A_SymmetricTensor_B_functional3D
对 A 和 B 进行 SymmetricTensorImpl<T,nDim>::contractIndexes 操作,
并反馈结果到 result 中。

TensorProduct_A_A_functional3Dresult 为张量 A 与张量 A 的乘积。

InterpolateTensorFieldFunctional3D 需声明 N 和 t, 用以计算 w, 乘以 field 的值并赋值到 result 中。

Scalar_A_times_Tensor_B_functional3Dresult 等于 scalarField 乘以 TensorField。

Tensor_A_dividedBy_B_functional3Dresult 等于 A/B。

Tensor_A_plus_B_inplace_functional3D 没有 result, $A=A+B$ 。

Tensor_A_minus_B_inplace_functional3D $A-=B$ 。

Tensor_A_times_B_inplace_functional3D $A*=B$ 。

Tensor_A_times_alpha_inplace_functional3D 需声明 alpha 值, $A*=alpha$ 。

Tensor_A_times_alpha_functional3D 同上, $result=alpha*A$ 。

Tensor_A_dividedBy_B_inplace_functional3D $A/=B$ 。

Tensor_A_dividedBy_Scalar_B_inplace_functional3D $A/=B$, B 是 scalarField。

Masked_Tensor_A_dividedBy_Scalar_B_inplace_functional3D 通过 flag 确认需要 $tensor/=scalar$ 的格点位置, 并计算赋值。

Normalize_Tensor_functional3D 判断 $norm(a)$ 是否为 0, 是 0 则赋值 0, 不是 0 则计算得 $result=a/norm(a)$ 。

Normalize_Tensor_inplace_functional3D 同上，但没有 result。非零格点直接 $A /= \text{norm}(A)$ 。

LBMsmoothenTensor3D 遍历求和邻域值并求平均，赋值至 result。

LBMsmoothenTensorInPlace3D 同上，但没有 result。

SmoothenTensor3D 遍历求和，再求平均，赋值至 result。

MollifyTensor3DTensor 版本的 Mollify，并赋值至 result。

LBMcomputeDivergence3D 偏向格子玻尔兹曼法，通过向量场来计算收敛。遍历格点遍历 q 个方向，c 为方向，t 为权重。

UpdateAveTensorTransientStatistics3D $\frac{1}{n} * (n-1) * \text{tensor} + \text{tensor}$ 。

UpdateDevTensorTransientStatistics3D 连续遍历两次 nDim，对 statistics 赋值 $\frac{1}{n} * (n-1) * \text{oldDev} + (\text{tensor}[iA] - \text{ave}) * (\text{tensor}[iB] - \text{ave})$ 。

2.2.4 第三部分：NTensor-field 的分析

Analysis of the NTensor-field

ExtractNTensorSubDomainFunctional3D 为 field2 赋值 field1 的值。

MaskedNTensorNeumannInLayersFunctional3D 判断是否为 toflag，不是则判断是否为 noflag，若为 noflag 则赋值 fromflag。如果是 toflag 则继续判断是否为 fromflag，如果是则读取 data 计算 sum。继续判断 n 不为 0 则通过 toData 对 Data 赋值，并将该 mask 区域赋值为 noflag。

2.3 dataAnalysisGenerics3D.h

CountLatticeElementsFunctional3D 作用于 lattice 上, 根据 boolMask 判断是否加 1, 累计 boolMask 反馈为 true 的格点数量。

CountScalarElementsFunctional3D 同上累计求数量, 但作用于 scalarField 上。

CountTensorElementsFunctional3D 同上累计求数量, 但作用于 TensorField 上。

ApplyScalarFunctional3D 需预先声明函数 f, 然后遍历 scalarField 上的所有格点, 每个格点代入函数 f 计算, 并赋值到 scalarField 上。

EvaluateScalarFunctional3D 作用同上, 但代入函数 f 计算后, 赋值到 scalarField result 上。

2.4 dataAnalysisWrapper3D.h/hh

2.4.1 第一部分: block-lattice 的 Atomic-block warppers Atomic-block wrappers: Block-Lattice

2.4.1.1 简化版函数

Reductive functions

computeAverageDensity 对 lattice 进行一次 BoxSumRhoBarFunctional3D 数据处理器操作, statistics 会累计求和 lattice 的 rhoBar, 并将总和 rhoBar 除以 cell 数量得到平均值 rhoBar, 通过 Descriptor<T>::fullRho 得到对应的平均 Density。

computeAverageRhoBar 同上, 但没有通过 Descriptor<T>::fullRho

来计算 Density，所以返回的是 rhoBar 平均值。

computeAverageEnergy 对 lattice 进行一次 BoxSumEnergyFunctional3D 数据处理器操作，statistics 会累计求和 velocity 的 norm-square 值，并除以 cell 数量得到平均值。

count 对 lattice 进行一次 CountLatticeElementsFunctional3D 数据处理器操作，statistics 会通过 boolMask 累计 cell 数量并返回。

computeDensity 对 lattice 进行 BoxDensityFunctional3D 数据处理器操作，返回 density。

computeRhoBar 对 lattice 进行 BoxRhoBarFunctional3D 数据处理器操作，返回 rhoBar。

computeKineticEnergy 对 lattice 进行 BoxKineticEnergyFunctional3D 数据处理器操作，返回 $\text{normSqr}(\text{velocity}) / (T)^2$ 至 scalarField energy。

computeVelocityNorm 对 lattice 进行 BoxVelocityNormFunctional3D 数据处理器操作，返回 velocity 的 norm-square 的平方根至 scalarField velocityNorm 中。

computeVelocityComponent 对 lattice 进行 BoxVelocityComponentFunctional3D 数据处理器操作，返回值至 scalarField velocityComponent。

computeVelocity 对 lattice 进行 BoxVelocityFunctional3D 数据处理器操作，返回 TensorField velocity。

computePiNeq 对 lattice 进行 BoxPiNeqFunctional3D 数据处理器操作，返回 TensorField PiNeq。

computeShearStress 对 lattice 进行 BoxShearStressFunctional3D 数据处理器操作，返回 TensorField ShearStress。

computeStrainRateFromStress 对 lattice 进行 BoxStrainRateFromStressFunctional3D 数据处理器操作，返回 TensorField StrainRate。

computePopulation 对 lattice 进行 BoxPopulationFunctional3D 数据处理器操作，返回分布函数至 scalarField 中。

computeAllPopulations 对 lattice 进行 BoxAllPopulationsFunctional3D 数据处理器操作，返回 TensorField StrainRate。一般是 q 个，以 D3Q19 为例则是从 f[0] 数到 f[18]。

copyPopulations 对 lattice 进行 CopyPopulationsFunctional3D 数据处理器操作，实现 lattice1 的分布函数值被复制到 lattice2 上。

copyAll 对 lattice 进行 LatticeCopyAllFunctional3D 数据处理器操作，实现 lattice1 的分布函数和 dynamic object 都被复制到 lattice2 上。

copyRegenerate 对 lattice 进行 LatticeRegenerateFunctional3D 数据处理器操作，实现 lattice1 的分布函数和 dynamics 都被复制到 lattice2 上，同时重新生成了 dynamics object。

2.4.2 第二部分：Scalar-Field 的 Atomic-block warppers

Atomic-block wrappers: Scalar-Field

2.4.2.1 简化版函数

Reductive functions

computeSum 运行 BoxScalarSumFunctional3D，累计了 scalarField 的值。在有 flag 标记时则运行 MaskedTextBoxScalarSumFunctional3D 累计那些标记的值。

computeBoundedSum 运行 BoundingBoxScalarSumFunctional3D，求和 scalarField 内值。

computeAverageBoxScalarSumFunctional3D 对 scalarField 求和，再除以 cell 数量得到平均值，若存在 flag 标记则运行 MaskedBoxScalarAverageFunctional3D。

computeMin 运行 BoxScalarMinFunctional3D，得到 scalarField 的最小值。

computeMax 运行 BoxScalarMaxFunctional3D。得到 scalarField 的最大值。

computeBoundedAverage 求 scalarField 通过 BoundedBoxScalarSumFunctional3D 求和，再除以 cell 数量以求平均值。

count 运行 CountScalarElementsFunctional3D 累计 scalarField 格点数量。

add

subtract

multiply

divide

addInPlace

subtractInPlace

multiplyInPlace

divideInPlace

运行 A_plus_alpha_functional3D 等函数进行运算。

copy 运行 CopyConvertScalarFunctional3D，标量场 2 复制标量场 1。

computeSqrt 运行 ComputeScalarSqrtFunctional3D，为 scalarField2 赋值 scalarField1 的平方根。

computeAbsoluteValue 运行 ComputeAbsoluteValueFunctional3D，为 scalarField2 赋值 scalarField1 的绝对值。

2.4.3 第三部分: Tensor-Field 的 Atomic-block wrappers

Atomic-block wrappers: Tensor-Field

2.4.3.1 简化版函数

Reductive functions

重复内容不过多叙述, 可 Ctrl+F 检索 Functionals 找到解释。

count

CountTensorElementsFunctional3D

computeSum

BoxTensorSumFunctional3D

computeAverage

BoxTensorSumFunctional3D

MaskedBoxTensorAverageFunctional3D

extractComponent

ExtractTensorComponentFunctional3D

computeNorm

ComputeNormFunctional3D

computeSqrt

ComputeTensorSqrtFunctional3D

computeNormSqr

ComputeNormSqrFunctional3D

computeMaximumElement

BoxLocalMaximumPerComponentFunctional3D

computeSymmetricTensorNorm

ComputeSymmetricTensorNormFunctional3D

computeSymmetricTensorNormSqr

ComputeSymmetricTensorNormSqrFunctional3D

computeSymmetricTensorTrace

ComputeSymmetricTensorTraceFunctional3D

computeVorticity

BoxVorticityFunctional3D

computeBulkVorticity

BoxBulkVorticityFunctional3D

computeHelicity

BoxHelicityFunctional3D

computeBulkHelicity

BoxBulkHelicityFunctional3D

computeBulkDivergence

BoxBulkDivergenceFunctional3D

computeStrainRate

BoxStrainRateFunctional3D

computeBulkStrainRate

BoxBulkStrainRateFunctional3D

copy

CopyConvertTensorFunctional3D

add

subtract

multiply

divide

addInPlace

subtractInPlace

multiplyInPlace

divideInPlace

Tensor_A_plus_B_functional3D 等

2.4.4 第四部分: Block-Lattice 的 Multi-block warppers

Multi-block wrappers: Block-Lattice

2.4.4.1 简化版函数

Reductive functions

computeAverageDensity

BoxSumRhoBarFunctional3D

computeAverageRhoBar

BoxSumRhoBarFunctional3D

computeAverageEnergy

BoxSumEnergyFunctional3D

computeAverageForcedEnergy

BoxSumForcedEnergyFunctional3D

BoxSumConstForcedEnergyFunctional3D

BoxSumCustomForcedEnergyFunctional3D

count

CountLatticeElementsFunctional3D

scalarSingleProbes

ScalarFieldSingleProbe3D

densitySingleProbes

DensitySingleProbe3D

velocitySingleProbes

VelocitySingleProbe3D

vorticitySingleProbes

VorticitySingleProbe3D

extractSubDomain

LatticeRegenerateFunctional3D

computeDensity

BoxDensityFunctional3D

computeRhoBar

BoxRhoBarFunctional3D

computeRhoBarJ

BoxRhoBarJfunctional3D

maskedComputeRhoBarJ

MaskedBoxRhoBarJfunctional3D

computeRhoBarJPiNeq

BoxRhoBarJPiNeqfunctional3D

computeKineticEnergy

BoxKineticEnergyFunctional3D

computePackedRhoBarJ

PackedRhoBarJfunctional3D

computeDensityFromRhoBarJ

DensityFromRhoBarJfunctional3D

computeVelocityFromRhoBarJ

VelocityFromRhoBarJfunctional3D

computeVelocityFromRhoBarAndJ

VelocityFromRhoBarAndJfunctional3D

computeVelocityNorm

BoxVelocityNormFunctional3D

computeForcedVelocityNorm

BoxForcedVelocityNormFunctional3D

BoxConstForcedVelocityNormFunctional3D

BoxCustomForcedVelocityNormFunctional3D

computeVelocityComponent

BoxVelocityComponentFunctional3D

computeForcedVelocityComponent

BoxForcedVelocityComponentFunctional3D

BoxConstForcedVelocityComponentFunctional3D

BoxCustomForcedVelocityComponentFunctional3D

computeVelocity

BoxVelocityFunctional3D

computeForcedVelocity

BoxForcedVelocityFunctional3D

BoxConstForcedVelocityFunctional3D

BoxCustomForcedVelocityFunctional3D

computeTemperature

BoxTemperatureFunctional3D

computePiNeq

BoxPiNeqFunctional3D

computeShearStress

BoxShearStressFunctional3D

computeStress

BoxStressFunctional3D

computeStrainRateFromStress

BoxStrainRateFromStressFunctional3D

computeShearRate

BoxShearRateFunctional3D

computeShearRate__N

BoxNTensorShearRateFunctional3D

computePopulation

BoxPopulationFunctional3D

computeEquilibrium

BoxEquilibriumFunctional3D

BoxAllEquilibriumFunctional3D

computeNonEquilibrium

BoxAllNonEquilibriumFunctional3D

computeAllPopulations

BoxAllPopulationsFunctional3D

computeAllPopulationsFromTensorField

BoxAllPopulationsToLatticeFunctional3D

copyPopulations

CopyPopulationsFunctional3D

copyAll

LatticeCopyAllFunctional3D

copyRegenerate

LatticeRegenerateFunctional3D

copyConvertPopulations

CopyConvertPopulationsFunctional3D

computeOmega

BoxOmegaFunctional3D

computeKinematicViscosity

BoxKinematicViscosityFunctional3D

computeKinematicViscosity__N

BoxNTensorKinematicViscosityFunctional3D

computeKinematicEddyViscosity

BoxKinematicEddyViscosityFunctional3D

computeKinematicEddyViscosity__N

BoxNTensorKinematicEddyViscosityFunctional3D

computeExternalForce

BoxExternalForceFunctional3D

computeExternalScalar

BoxExternalScalarFunctional3D

computeExternalVector

BoxExternalVectorFunctional3D

computeDynamicParameter

BoxDynamicParameterFunctional3D

computeDynamicViscosity

BoxDynamicViscosityFunctional3D

2.4.5 第五部分: Scalar-Field 的 Multi-block warppers

Multi-block wrappers: Scalar-Field

2.4.5.1 简化版函数

Reductive functions

其内容与之前所述大部分相同, 不再列举。

lessThan

A_lt_B_functional3D

greaterThan

A_gt_B_functional3D

uniformlyBoundScalarField

UniformlyBoundScalarField3D

boundScalarField

BoundScalarField3D

mollifyScalar

MollifyScalar3D

lbmComputeGradient

LBMcomputeGradient3D

2.4.6 第六部分: Tensor-field 的 Multi-block wrappers

Multi-block wrappers: Tensor-field

2.4.6.1 简化版函数

Reductive functions

computeSymmetricTensorNorm

ComputeSymmetricTensorNormFunctional3D

computeSymmetricTensorNormSqr

ComputeSymmetricTensorNormSqrFunctional3D

computeSymmetricTensorTrace

ComputeSymmetricTensorTraceFunctional3D

computeGradient

BoxGradientFunctional3D

computeBulkGradient

BoxBulkGradientFunctional3D

computeInstantaneousReynoldsStress

BoxComputeInstantaneousReynoldsStressFunctional3D

computeQcriterion

BoxQcriterionFunctional3D

computeLambda2

BoxLambda2Functional3D

normalize

Normalize__Tensor__functional3D

symmetricTensorProduct

TensorProduct__A__A__functional3D

fullIndexContractionOfSymmetricTensors

IndexContraction__SymmetricTensor__A__SymmetricTensor__B__functional3D

2.4.7 第七部分: NTensor-field 的 Multi-block warppers**Multi-block wrappers: NTensor-field****2.4.7.1 简化版函数****Reductive functions****extractSubDomain**

ExtractNTensorSubDomainFunctional3D

2.5 dataInitializerFunctional3D.h/hh**2.5.1 第一部分: block-lattice 的初始化****Initialization of the block-lattice****OneCellFunctional3D** 对局部的 cell 进行操作。**OneCellIndexedFunctional3D** 对局部的特定坐标的 cell 进行操作。**OneCellIndexedWithRandFunctional3D** 对局部的特定坐标的 cell 进行伴随 “T randVal” 的操作。

DomainFunctional3D 对 lattice 的一片区域进行操作。

GenericLatticeFunctional3D 需在 `OneCellIndexedFunctional3D` 声明 `f`，对目标区域执行 “`f->execute(lattice.get(iX,iY,iZ));`”。

GenericIndexedLatticeFunctional3D 需在 `OneCellIndexedFunctional3D` 声明 `f`，对目标区域执行同上操作。

GenericIndexedWithRandLatticeFunctional3D 同上。

InstantiateDynamicsFunctional3D 对区域执行 `attributeDynamics`。

InstantiateDynamicsInBulkAndEnvelopeFunctional3D 与上相同，区别在于 `appliesTo` 部分包含了 `envelope`。运用这种底层的数据处理器是因为不规则的稀疏区域可能会让 `envelope` 的值变得不理想，所以需要运用这种数据处理器给 `envelope` 也分配 `dynamics`。

InstantiateComplexDomainDynamicsFunctional3D 对指定区域执行 `attributeDynamics`。

InstantiateDotDynamicsFunctional3D 这里出现了 `DotProcessing-Functional3D_L`，对 lattice 的每个 dot 执行 `attributeDynamics`。

DynamicsFromMaskFunctional3D 通过 `mask` 来判断标记的位置，并执行 `attributeDynamics`。

DynamicsFromIntMaskFunctional3D，同上，但判断 `int` 型的 `mask`。

RecomposeFromOrderZeroVariablesFunctional3D 这儿涉及到了 `dynamics.h/hh` 的代码，D2Q9 的 BGK dynamics 为例，0 阶分解为 `rho`，`u`，和 `fNeq` 是 (1+2+9) 共 12 个变量。这儿以 `rawData[0]` 存储 `rhoBar` 的值，`rawData[1 3]` 存储动量 `j`，继续往后从 4 开始数 `q` 的 `fNeq`，数

完后继续数 External Field 的量。数完再 recompose。它的注释提到可以同时对外围 envelope 也执行这种操作,但当下还是设置为 bulk,有需要时再说。

RecomposeFromFlowVariablesFunctional3D 如上,为 1 阶分解和重建。

AssignOmegaFunctional3D 计算 omega, 执行 setOmega。

AssignScalarFieldOmegaFunctional3D 通过 scalarField 获得值计算 omega, 执行 setOmega。

SetConstBoundaryVelocityFunctional3D 需声明 velocity, 乘以 scaleFactor, 后执行 `lattice.get(iX,iY,iZ).defineVelocity(scaledU)`; 完成赋值。注意 `Array<T,3> scaledU` 是一个具有 3 个分量的变量。

SetConstBoundaryVelocityWithTensorForceFunctional3D 它适用于存在力场的流场初始化中, “`Array<T,3> scaledU = scaleFactor * (u - (T) 0.5 * force.get(iX+ofs.x, iY+ofs.y, iZ+ofs.z));`”, 速度的计算考虑到了力的贡献。

SetConstBoundaryVelocityWithForceFunctional3D 基本同上, 但 u 和 f 来源于相应的函数。值得注意的是, 底下声明 lattice 有哪些数据变动。是 “`modified[0] = modif::allVariables;`”。

SetCustomBoundaryVelocityWithTensorForceFunctional3D 基本同上, u 会根据 VelocityFunction f_ 来计算, force 来源于 TensorField, 然后定义速度。

SetCustomBoundaryVelocityWithForceFunctional3D 基本同上, u 根据函数, f 也根据函数计算, 然后定义。

SetCustomBoundaryVelocityWithCustomForceFunctional3D 基本同上, u 和 f 根于 ForceVelocityFunction f_ 来计算, 然后定义。

SetConstBoundaryDensityFunctional3D 执行“lattice.get(iX,iY,iZ).defineDensity(rho);”

IniConstEquilibriumFunctional3D 需声明密度 T density, 速度 Array<T,Descriptor<T>::d> velocity, 温度 T Temperature, 可计算得 rho, rhoBar, u, thetaBar。定义“Array<T,Descriptor<T>::d> f;”, 接着获取力的三个分量如 “f[0] = getExternalForceComponent(lattice.get(iX,iY,iZ), 0);”, 随后得到 scaledJ 已经是考虑了力的作用, 再通过 “computeEquilibria(lattice.get(iX,iY,iZ).getRawPopulations(),rhoBar, scaledJ, scaledJsqr, thetaBar);”, 以 rhoBar, scaledJ, scaledJsqr 来计算 q 各 fEq, 并通过 lattice.get(iX,iY,iZ).getRawPopulations() 完成对该格点的分布函数 f 赋值 fEq。这儿数据变动是 “modified[0] = modif::staticVariables;”, 作用范围 “BlockDomain::bulkAndEnvelope;” 包括了 envelope。

MaskedIniConstEquilibriumFunctional3D 同上, 仅多出判断 mask 的部分。

IniConstTensorForceEquilibriumFunctional3D 同上, 区别在于力来源于 TensorField。应用层面上, 比如力有个初始的力场, 可以用这个数据处理器初始化 fEq。

IniCustomTensorForceEquilibriumFunctional3D 同上, 但需要 RhoUFunction f_ 函数来计算 rho 和 u。

IniCustomTensorForceRandomEquilibriumFunctional3D 同上, 但 RhoUFunction f_ 的计算涉及到了 randomValues。

IniConstForceEquilibriumFunctional3D 同上, force 为声明的常数。

IniCustomForceEquilibriumFunctional3D 同上, force 为自定义值。

IniCustomForceRandomEquilibriumFunctional3D 同上, force 为自定义值, RhoUFunction f_ 的计算涉及到了 randomValues。

IniConstEquilibriumOnDomainFunctional3D 同上。不过作用域出现了 bbox。

StripeOffDensityOffsetFunctional3D 对 dynamics 进行了 0 阶的 decompose 和 recompose。

InstantiateCompositeDynamicsFunctional3D 通过 cloneAndInsertAtTopDynamics, 将 compositeDynamics 设置为 newTop。

SetExternalScalarFunctional3D 本质上为：“*lattice.get(iX,iY,iZ).getExternal(whichScalar) = externalScalar;”。

SetExternalScalarFromScalarFieldFunctional3D 同上, 仅赋值的来源变成了 scalar.get(oX,oY,oZ)。

MaskedSetExternalScalarFunctional3D 同上, 仅多出判断 mask 标记。

SetGenericExternalScalarFunctional3D 同上, 本质上为 cell 的 external 赋值 functional(iX+absOffset.x,iY+absOffset.y,iZ+absOffset.z);, 需自己声明 functional。

MaskedSetGenericExternalScalarFunctional3D 同上, 仅多出判断 mask 标记。

AddToExternalScalarFunctional3D 本质上为：“*lattice.get(iX,iY,iZ).getExternal(whichScalar) += externalScalar;”。

SetExternalVectorFunctional3D 声明 vectorStartsAt_, 并为 externalVector 赋值。

MaskedSetExternalVectorFunctional3D 同上, 仅多出判断 mask 标

记。

SetGenericExternalVectorFunctional3D 声明 `vectorBeginsAt__`, 并为 `u` 赋值。

MaskedSetGenericExternalVectorFunctional3D 同上, 仅多出判断 `mask` 标记。

SetExternalVectorFromTensorFieldFunctional3D 本质上为: “`*cell.getExternal(vectorStartsAt+i) = externalVector[iD];`”。

InterpolatePopulationsFunctional3D 根据 `lamda` 执行 “`cell1[iPop] = lambda*cell1[iPop] + (1.-lambda)*cell2[iPop];`”。

2.5.2 第二部分: scalar 和 tensor-fields 的初始化

Initialization of scalar- and tensor-fields

IniConstScalarFunctional3D 为 `scalarField` field 赋值 `value`。

MaskedIniConstScalarFunctional3D 判断 `mask`, 为 field 赋值 `value`。

MaskedIniConstScalarFunctional3D_N 同上, 只是 `mask` 不是 `scalarField` 而是 `NTensorField3D<int>` `mask`。

IniConstTensorFunctional3D 为 `TensorField` field 赋值 `value`。

MaskedIniConstTensorFunctional3D 同上, 多出了判断 `mask` 的部分。

MaskedIniConstTensorFunctional3D_Nmask 内含于 `std::vector<AtomicBlock3D*>` `atomicBlocks` 中。

SwapValuesBulkAndEnvelope3D_N 对 `A` 和 `B` 两个 `NTensorField`

进行 swap，但 “modified[0] = modif::staticVariables;” 告诉我们 A 的格点数据被改变了。

SetToCoordinateFunctional3D 本质上为 “field.get(pos[0],pos[1],pos[2]) = (T) (pos[index]+ofs[index]);”。

SetToCoordinatesFunctional3D 同上，只是 field 从 scalarField 变成了 TensorField。

SetToRandomFunctional3D 本质为 “field.get(iX,iY,iZ) = (T)eng() / ((T)std::numeric_limits<uint32_t>::max()+1.0);”。

SetTensorComponentFunctional3D 为 TensorField 的某个维度赋值 scalarField。

PropagateInZdirection3D 首先判断 flag 是否为 true，不为 true 则继续判断 iniZ 加上相对位移是否为 0，如果是就 +1，然后获得 z 方向小于 1 的值，如果不小于-0.5 则赋值该点。最后标记 flag 为 true。

GrowDomainFunctional3D 通过 flag 判断为 voxels 赋值 flag。

ScalarNeumannToUnusedEnvelopes3D 先判断该格点不在 bulk 内，再求和在 bulk 内的值，并继续求其均值，再赋值给 field。

TensorNeumannToUnusedEnvelopes3D 同上，只是 field 为 TensorField。

2.6 dataInitializerGenerics3D.h

2.6.1 第一部分：block-lattice 的初始化

Initialization of the block-lattice

SetCustomBoundaryVelocityFunctional3D 由 `VelocityFunction` 得 `u`，乘以 `velocityScale` 并由 `defineVelocity` 完成赋值。

SetCustomBoundaryDensityFunctional3D 由 `DensityFunction` 得 `rho`，并由 `defineDensity` 完成赋值。

SetCustomOmegaFunctional3D 由 `OmegaFunction` 得 `omega`，并由 `setOmega` 完成赋值。

IniCustomEquilibriumFunctional3D 由 `cell` 访问 `External`

`Force`，由 `RhoUFunction` 计算 `rho` 和 `j` 完成 `fEq` 计算和赋值。

IniCustomRandomEquilibriumFunctional3D 同上，但 `RhoUFunction` 计算 `rho` 和 `j` 包含 `randVal`。

IniCustomThermalEquilibriumFunctional3D 基本同上，`RhoVelTempFunction` 包括了温度 `temperature` 计算，并由 `thebaBar` 参与到 `fEq` 的计算中。

2.6.2 第一部分：scalar 和 tensor 的初始化

Initialization of the scalar- and tensor-field

SetToScalarFunctionFunctional3D 为 `field` 赋值由 `Function f_` 计算得的值。

SetToNTensorFunctionFunctional3D 同上，只是 `field` 不为 `scalarField`，而是 `NTensorField3D<T>& field`。

AnalyticalSetRhoBarJFunctional3D 由 `function(坐标, rhoBar, j)` 计算得 `rhoBar` 和 `j`, 再赋值。

SetToTensorFunctionFunctional3D 同上, 为 `TensorField3D` 赋值由函数 `f` 计算出的值。

2.7 dataInitializerWrapper3D.h/hh

2.7.1 第一部分: block-lattice 的初始化: atomic-blocks

Initialization of the block-lattice: atomic-blocks

apply

`GenericLatticeFunctional3D`, 同时需利用 `OneCellFunctional3D`。

applyIndexed

`GenericIndexedLatticeFunctional3D`

defineDynamics

`InstantiateDynamicsFunctional3D`

`InstantiateComplexDomainDynamicsFunctional3D`

`InstantiateDotDynamicsFunctional3D`

`DynamicsFromMaskFunctional3D`

recomposeFromFlowVariables

`DynamicsFromIntMaskFunctional3D`

setOmega

`AssignOmegaFunctional3D`

`AssignScalarFieldOmegaFunctional3D`

setBoundaryVelocity

SetConstBoundaryVelocityFunctional3D
 SetConstBoundaryVelocityWithTensorForceFunctional3D
 SetConstBoundaryVelocityWithForceFunctional3D
 SetCustomBoundaryVelocityWithTensorForceFunctional3D
 SetCustomBoundaryVelocityWithForceFunctional3D
 SetConstBoundaryDensityFunctional3D

initializeAtEquilibrium

IniConstEquilibriumFunctional3D

stripeOffDensityOffset

StripeOffDensityOffsetFunctional3D

setCompositeDynamics

InstantiateCompositeDynamicsFunctional3D

setExternalScalar

SetExternalScalarFunctional3D
 MaskedSetExternalScalarFunctional3D

setExternalVector

SetExternalVectorFunctional3D

2.7.2 第二部分: block-lattice 的初始化: multi-blocks

Initialization of the block-lattice: multi-block

apply

GenericLatticeFunctional3D

applyIndexed

GenericIndexedLatticeFunctional3D
 GenericIndexedWithRandLatticeFunctional3D

defineDynamics

InstantiateDynamicsFunctional3D

defineDynamicsInBulkAndEnvelope

InstantiateDynamicsInBulkAndEnvelopeFunctional3D

defineDynamics

DynamicsFromMaskFunctional3D

DynamicsFromIntMaskFunctional3D

recomposeFromFlowVariables

RecomposeFromFlowVariablesFunctional3D

recomposeFromOrderZeroVariables

RecomposeFromOrderZeroVariablesFunctional3D

setOmega

AssignOmegaFunctional3D

SetCustomOmegaFunctional3D

AssignScalarFieldOmegaFunctional3D

setBoundaryVelocity

SetConstBoundaryVelocityFunctional3D

SetConstBoundaryVelocityWithTensorForceFunctional3D

SetConstBoundaryVelocityWithForceFunctional3D

SetCustomBoundaryVelocityWithTensorForceFunctional3D

SetCustomBoundaryVelocityWithForceFunctional3D

setBoundaryDensity

SetConstBoundaryDensityFunctional3D

initializeAtEquilibrium

IniConstEquilibriumFunctional3D

IniConstTensorForceEquilibriumFunctional3D
 IniCustomTensorForceEquilibriumFunctional3D
 IniCustomTensorForceRandomEquilibriumFunctional3D
 IniConstForceEquilibriumFunctional3D
 IniCustomForceEquilibriumFunctional3D
 IniCustomForceRandomEquilibriumFunctional3D

maskedInitializeAtEquilibrium

IniConstEquilibriumOnDomainFunctional3D
 MaskedIniConstEquilibriumFunctional3D

stripeOffDensityOffset

StripeOffDensityOffsetFunctional3D

setCompositeDynamics

InstantiateCompositeDynamicsFunctional3D

setExternalScalar

SetExternalScalarFunctional3D
 MaskedSetExternalScalarFunctional3D

setGenericExternalScalar

SetGenericExternalScalarFunctional3D
 MaskedSetGenericExternalScalarFunctional3D

setExternalVector

SetExternalVectorFunctional3D
 MaskedSetExternalVectorFunctional3D
 SetExternalVectorFromTensorFieldFunctional3D
 SetGenericExternalVectorFunctional3D
 MaskedSetGenericExternalVectorFunctional3D

interpolatePopulations

InterpolatePopulationsFunctional3D

**2.7.3 第三部分: scalar 和 tensor-fields 的初始化: Atomic-Block
Initialization of scalar- and tensor-fields: Atomic-Block****setToConstant**

IniConstScalarFunctional3D

MaskedIniConstScalarFunctional3D

IniConstTensorFunctional3D

MaskedIniConstTensorFunctional3D

setToCoordinate

SetToCoordinateFunctional3D

setToCoordinates

存在一个没有具体代码的 SetToCoordinatesFunctional3D

assignComponent

SetTensorComponentFunctional3D

**2.7.4 第四部分: scalar 和 tensor-fields 的初始化: Multi-Block
Initialization of scalar- and tensor-fields: Multi-Block****setToConstant**

IniConstScalarFunctional3D

MaskedIniConstScalarFunctional3D

MaskedIniConstScalarFunctional3D_N

IniConstTensorFunctional3D

MaskedIniConstTensorFunctional3D

MaskedIniConstTensorFunctional3D_N

setToCoordinate

SetToCoordinateFunctional3D

setToCoordinates

存在一个没有具体代码的 SetToCoordinatesFunctional3D

setToRandom

SetToRandomFunctional3D

assignComponent

SetTensorComponentFunctional3D

propagateInZdirection

PropagateInZdirection3D

growDomain

GrowDomainFunctional3D