Stencil code generation from Latex

Plan

- Problem description
- Approach to the problem
- Concepts
- Example : Lattice Boltzmann Method (LBM)
- Conclusion

THE PROBLEM

The problem

- Numerical simulation : **two languages**
  - **Math**
  - **Code**
- More **complex architectures** and **codes**
- **Optimization** => **collaboration**
- I focus on “trivial case” : **cartesian meshes** (**stencils**)
- I consider a **PhD. student developing a new** numerical scheme

Decisions for performance

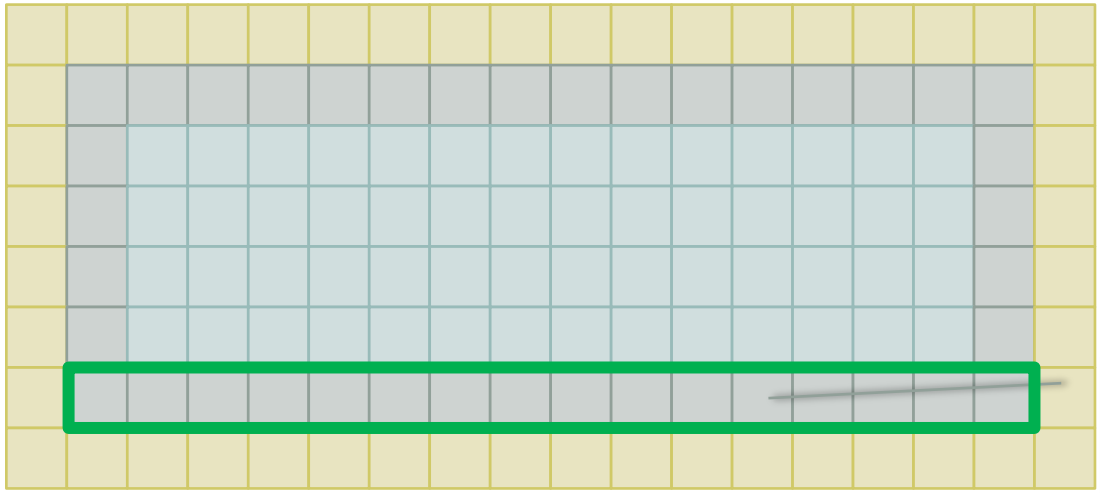
- Fine grain **AND global** optimizations !
- **Data layout**
  - **Arrays of struct / struct of arrays**
  - **Cells ordering** (row/col – major)
- Ways to run over the mesh (**loops**)
- **Dependencies / communications**

Fixed decisions

- **Decisions** are made at **development beginning**
- **Not** necessary **correct**
- They can **change over time**
- On manual codes => **fastidious to change**
- The main issue : **loop adaptations** and **data access** representation

Coordinates nightmare

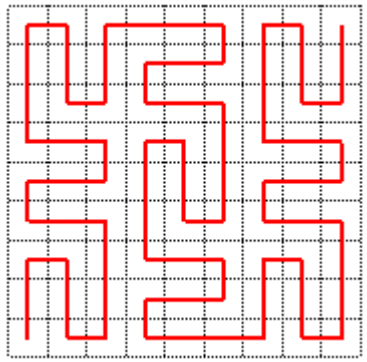
- Ghost cells coordinates



!/@$*%

[1 : H-1 , W-2 : 1]
Rec. de MPI_Rank+Mw

- Cache aware loops



!/@$*%

- Face and vertex IDs $f_{i+\frac{1}{2}}$

APPROACH

Use libraries

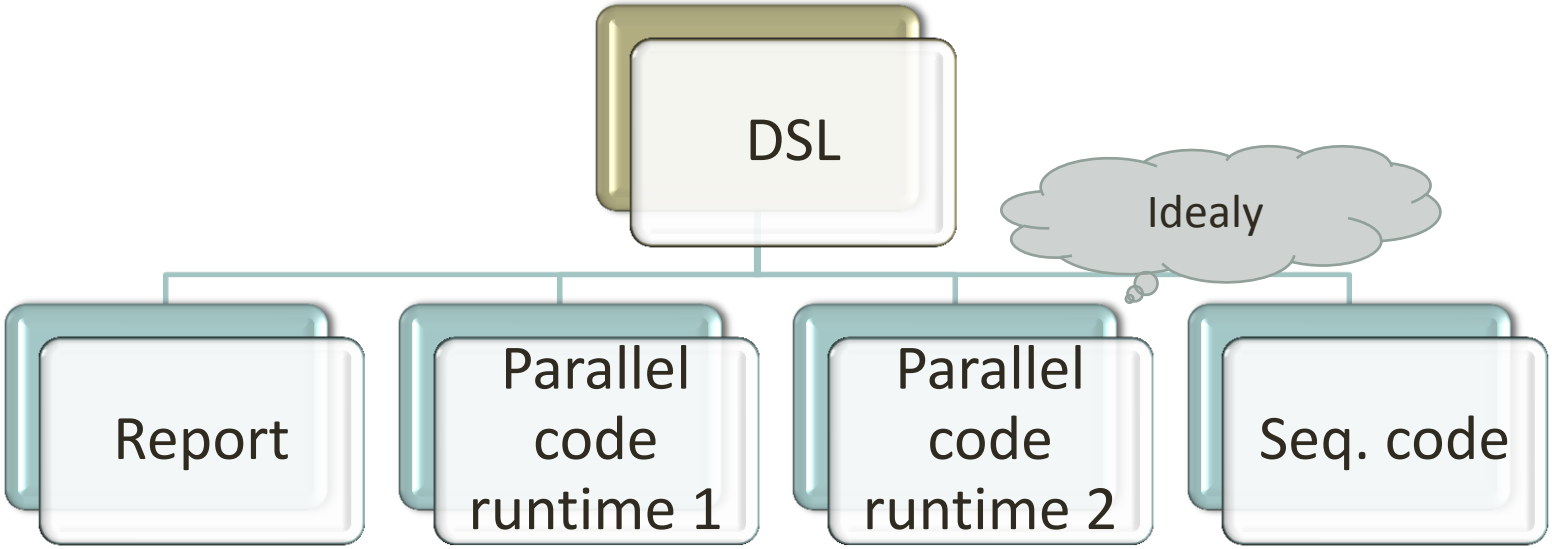
- **Easier** to write code
- **Hide** mesh management complexity
- Example : 

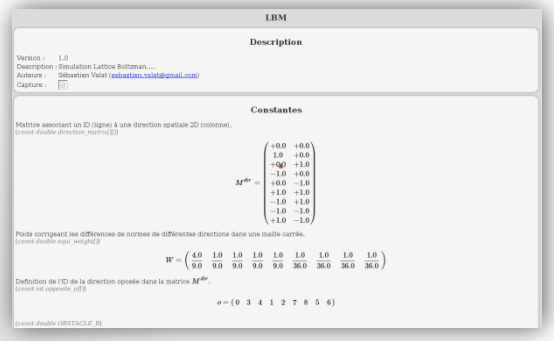
- Limitations :
  - **Abstraction** can **afraid the user**
  - **Maintain** a separation between **paper equations** and the **code**
  - **Maintain deeper details** depending on the **code implementation**

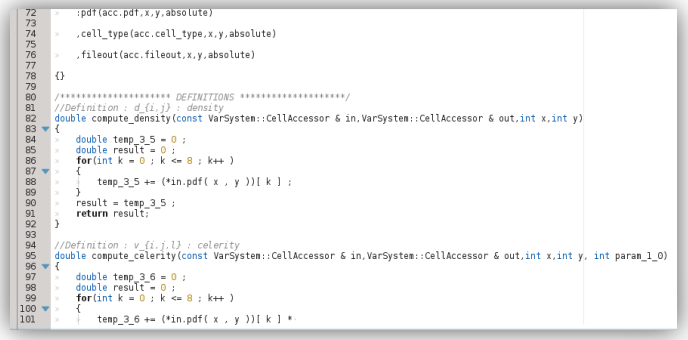
DSL / MDA

- **MDA** : Model Driven Architecture
- Use **multiple abstract levels**
- Goes from **abstract to concrete** representation (**UML** to code)
- DSL : Domain **Specific** Language
- Examples : **pochoir**, **patus**, **qiral**,…
- I retain a language based on **Latex**

Code generation







Declaration & optimization

- The **user declare <u>what</u>** he want to compute
- The user declare elsewhere **<u>how</u> to optimize** source code
  - **Data layout**
  - **Loops**
- The user **select kind of parallelism at runtime**
- **Manual tuning** VS auto-tunings
- Now the **user can experiment** easily to **understand** the **performance properties** of his scheme

CONCEPT

A scheme, for computer scientist

- **Variables** attached to each **cells**
- Some **constants**
- **Derivate values** from **cells variables**
- **Operations** to apply to **each cells** (DSL, ways to run over the mesh, **no loops**)
- **Loop** of **steps** to repeat on each **mesh areas**

Generated code = unreadable code ?

- If we do **not** want to **afraid users**
  he must **understand** the **generated code** !
- **No obfuscated** code.
- As possible **1 to 1** matching between **DSL and code**
- Help to **understand** the **DSL abstraction**
- Possibility to **inject manual C code**

DETAILED EXAMPLE : LBM

Mesh variables

```
<!-- ********************* MESH ******************* -->
<mesh>
    <var mathname='f_{i,j,k}' longname='pdf' type='double' ghost='1' memory="CMRMemoryModelRowMajor" doc="
        <extradims>
            <extradim mathname='k' longname='direction' size='9' start='0' doc="Direction à l'interieur de
        </extradims>
    </var>
    <var mathname='T_{i,j}' longname='cell_type' type='LBMCellType'  memory="CMRMemoryModelRowMajor"   ghc
    <var mathname='F_{i,j}' longname='fileout'   type='LBMFileEntry'  memory="CMRMemoryModelColMajor" doc=
</mesh>
```

Maillage

- $f_{i,j,k}$: Densité de particule dans différentes directions. (*const double pdf*)
- $T_{i,j}$: Type de cellule. (*const LBMCellType cell_type*)
- $F_{i,j}$: Structure de stokage pour la sortie. (*const LBMFileEntry fileout*)

```
/********************* VARIABLES *********************/
class VarSystem : public CMRVarSystem
{
    public:
        struct CellAccessor
        {
            CellAccessor(CMRVarSystem & sys,int tstep,int x,int y,bool absolute = true);
            CellAccessor(CellAccessor & acc,int x,int y,bool absolute = false);
            CMRCellAccessor<double[9],CMRMemoryModelRowMajor> pdf;
            CMRCellAccessor<LBMCellType,CMRMemoryModelRowMajor> cell_type;
            CMRCellAccessor<LBMFileEntry,CMRMemoryModelColMajor> fileout;
        };
    public:
        VarSystem(CMRDomainBuilder * builder = NULL);
};
```

Definitions

```
<def mathname='v_{i,j,l}' longname='celerity' doc="Vitesse (au sens vecteur) macroscopique de la maille
    <defparameter mathname='l' longname='l' type='int' doc='Coordonnée spatiale (1 ou 2) considérée.'/>
    <mathstep>v_{i,j,l} = \frac{\sum_k{f_{i,j,k} M^{dir}_{k,l}}} {d_{i,j}}</mathstep>
</def>
```

Definition : $v_{i,j,l}$

Vitesse (au sens vecteur) macroscopique de la maille (somme des vitesses pondérées par la densité et les poids (W)).

$$v_{i,j,l} = \frac{\sum_k f_{i,j,k} M^{dir}_{k,l}}{d_{i,j}}$$

```
94  //Definition : v_{i,j,l} : celerity
95  double compute_celerity(const VarSystem::CellAccessor & in,VarSystem::CellAccessor & out,
96                          int x,int y, int param_1_0)
97  {
98      double temp_3_6 = 0 ;
99      double result = 0 ;
100     for(int k = 0 ; k <= 8 ; k++ )
101     {
102         temp_3_6 += (*in.pdf( x , y ))[ k ] * direction_matrix[ k ][ param_1_0 ] ;
103     }
104     result = ( temp_3_6 / compute_density(in,out,x,y) ) ;
105     return result;
106 }
```

Steps

Opération sur chaque cellule : zou_he_const_dentity

Applique des conditions de Zou He en considérant une densité constante (fluide sortant par la droite).

ALIAS $L_k = f_{i,j,k}$

double d(d)

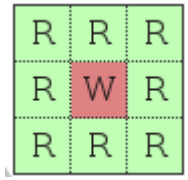
double v(v)

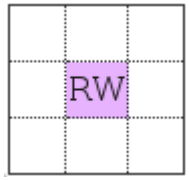
$$v = -1 + \frac{1}{d}\left(L_0 + L_2 + L_4 + 2(L_1 + L_5 + L_8)\right)$$

$$f_{i,j,3} = L_1 - \frac{2.0}{3.0}\,dv$$

$$f_{i,j,7} = L_5 - \frac{1.0}{2.0}\left(L_2 - L_4\right) + \frac{1.0}{6.0}\,dv$$

$$f_{i,j,6} = L_8 + \frac{1.0}{2.0}\left(L_2 - L_4\right) + \frac{1.0}{6.0}\,dv$$






```
193  struct Actionzou_he_const_dentity
194  {
195      void cellAction(const VarSystem::CellAccessor & in,VarSystem::CellAccessor& out,
196                      const CMRCellPosition & pos,int x,int y) const
197      {
198          double v = 0 ;
199          double d = 0 ;
200          v = - 1 + ( 1 / d ) * ( (*in.pdf( x , y ))[ 0 ] + (*in.pdf( x , y ))[ 2 ]
201              + (*in.pdf( x , y ))[ 4 ] + 2 * ( (*in.pdf( x , y ))[ 1 ] + (*in.pdf( x , y ))[ 5 ]
202              + (*in.pdf( x , y ))[ 8 ] ) ) ;
203          (*out.pdf( x , y ))[ 3 ] = (*in.pdf( x , y ))[ 1 ] - ( 2.0 / 3.0 ) * d * v ;
204          (*out.pdf( x , y ))[ 7 ] = (*in.pdf( x , y ))[ 5 ] - ( 1.0 / 2.0 )
205              * ( (*in.pdf( x , y ))[ 2 ] - (*in.pdf( x , y ))[ 4 ] ) + ( 1.0 / 6.0 ) * d * v ;
206          (*out.pdf( x , y ))[ 6 ] = (*in.pdf( x , y ))[ 8 ] + ( 1.0 / 2.0 )
207              * ( (*in.pdf( x , y ))[ 2 ] - (*in.pdf( x , y ))[ 4 ] ) + ( 1.0 / 6.0 ) * d * v ;
208      }
209
210      typedef CMRMeshOperationSimpleLoopWithPos<VarSystem,Actionzou_he_const_dentity> LoopType;
211  };
```

Mixing C & Latex code

```
129    <cellaction name='SpecialCells'  loop='CMRMeshOperationSimpleLoopWithPos'
130        <ccode>switch($T_{i,j}$)                                    </ccode>
131        <ccode>{                                                    </ccode>
132        <ccode>case CELL_FLUID:                                     </ccode>
133        <ccode>    $\cmrsubaction{simple_copy}$;                    </ccode>
134        <ccode>    break;                                           </ccode>
135        <ccode>case CELL_BOUNCE_BACK:                               </ccode>
136        <ccode>    $\cmrsubaction{simple_copy}$;                    </ccode>
137        <ccode>    $\cmrsubaction{bounce\_back}$;                   </ccode>
138        <ccode>    break;                                           </ccode>
139        <ccode>case CELL_LEFT_IN:                                   </ccode>
140        <ccode>    $\cmrsubaction{zou\_he\_poiseuil}$;              </ccode>
141        <ccode>    break;                                           </ccode>
142        <ccode>case CELL_RIGHT_OUT:                                 </ccode>
143        <ccode>    $\cmrsubaction{zou\_he\_const_dentity}$;         </ccode>
144        <ccode>    break;                                           </ccode>
145        <ccode>default:                                             </ccode>
146        <ccode>    warning("Bad cell type, ignore");                </ccode>
147        <ccode>    break;                                           </ccode>
148        <ccode>}                                                    </ccode>
```

Loop & output

```
<mainloop>
    <!-- ************** -->
    <callaction name="SpecialCells">
        <zone>global.expended(1)</zone>
    </callaction>
    <!-- ************** -->
    <callaction name="Collision">
        <zone>local</zone>
    </callaction>

    <!-- ************** -->
    <callaction name="Propagation">
        <zone>global.expended(-1)</zone>
    </callaction>
</mainloop>
```

```
<output>
    <entry name="v"       type="float">v_{i,j}</entry>
    <entry name="density" type="float">d_{i,j}</entry>
</output>
```

Parallelism implementation

```
38  /********************  FUNCTION  **********************/
39  void CMRRunnerOMPForRect::runOperationNode ( CMRMeshOperationNode& opNode )
40  {
41      //if have enougth job, split in sub elements
42      int jobs = nbThreads * multiplier;
43      int cellsPerThread = opNode.rect.surface() / jobs;
44      if (cellsPerThread <= minCells)
45      {
46          //sequential
47          opNode.op->run(system,opNode.rect);
48      } else {
49          //ensure to have the allocation
50          opNode.op->forceMeshAllocation(system,opNode.rect);
51          //split and omp
52          CMRBasicSpaceSplitter splitter(opNode.rect,jobs,0);
53          //splitter.printDebug(0);
54          #pragma omp parallel for
55          for (int i = 0 ; i < jobs ; i++)
56              opNode.op->run(system,splitter.getLocalDomain(i));
57      }
58  }
```

Use and run

- Code generation and usage

```
cmr-generate lbm.cmr.xml
make
./lbm-release –c config.ini
```

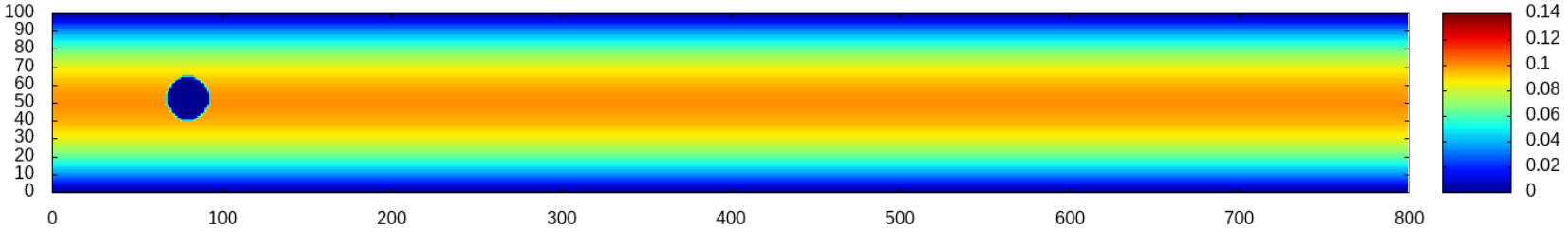
- Configuration file

```
[app]
runner                = CMRRunnerOMPForRect

[debug]
mpi_domains                = false

[mesh]
noout                 = false
output_file              = output.raw
width                 = 800
height                = 100
iterations             = 10000
```

CMRRunnerSeq
CMRRunnerOMPLoops
CMRRunnerOMPForRect
CMRRunnerOMPTask

Performance

Version	Language	Sequential (s)	OpenMP (s)
LBM-c-mihps	C	22,24	8,48
LBM-CMR (manual)	**C++**	**21,26**	**8,30**
LBM-CMR (generated)	C++	25,22	12,37



CONCLUSION

Conclusion

- It is possible to **generate codes from Latex**
- We can reach **performance similar to manual code**
- Its **easier** to **explore** solutions
- Help developer to **understand the performance properties** of the scheme
- Produce **clear definition** (**report**) of the scheme
- Still **need work** to get a fully **usable prototype**

BAKCUP

Parallelism implementation

```
38  /********************  FUNCTION  **********************/
39  void CMRRunnerOMPTask::runOperationNode ( CMRMeshOperationNode& opNode )
40  {
41      //if have enougth job, split in sub elements
42      int jobs = nbThreads * multiplier;
43      int cellsPerThread = opNode.rect.surface() / jobs;
44      if (cellsPerThread <= minCells)
45      {
46          //sequential
47          opNode.op->run(system,opNode.rect);
48      } else {
49          //ensure to have the allocation
50          opNode.op->forceMeshAllocation(system,opNode.rect);
51          //split and omp
52          CMRBasicSpaceSplitter splitter(opNode.rect,jobs,0);
53          //splitter.printDebug(0);
54          #pragma omp parallel
55          {
56              #pragma omp for
57              for (int i = 0 ; i < jobs ; i++)
58              {
59                  #pragma omp task shared(opNode)
60                  {
61                      opNode.op->run(system,splitter.getLocalDomain(i));
62                  }
63              }
64          }
65      }
66  }
```

Abstract representation of derivate variabes

- Depending on **compute cost**, **derivate variables** can be **instantiate**
  - As **a function** called for **each use**
  - ***Cached values*** *for the* ***current cell***
  - *As a new* ***cell variable*** *stored inside mesh*
  - *As cell* ***variable stored*** *for partial mesh areas (cache blocking)*
- **Decision** take at **generation** time