HIR
Specification

Version 0.1

MeteoSwiss - C2SM

March, 2018

Contents

List of Figures

List of Tables

1 Introduction

This document provides a full specification of the HIR

1.1 Conventions

Some of the elements will allow to have different children type of nodes depending on the scope of the node.

There are two scopes:

1. `control_flow` defines the scope of nodes where one or more of the parallel dimensions of the Domain are not yet resolved within a Computation
2. `domain_computation` is the scope of nodes where all the parallel dimensions of the Domain are resolved within a Computation

The **ContentsModel** section of each node describes, using regular expression, that supported children nodes. Some of the children are named nodes in order to identify in the specification of the node. The following example shows a node with two children, identified by `lhs` and `rhs`, where the `rhs` can be either a FieldAccess or a Literal

ContentsModel

(FieldAccess , (FieldAccess | Literal))

2 General HIR elements

2.1 `Program` element

Description of the element `Program` element:

ContentsModel

(GridDimension +, Domain , FieldDecl +, VarDecl *, (ScopedProgram | ExternalKernel)+)

Child elements

name	description	R/O
`GridDimension`	Definition of all the dimensions used within the program	R
`Domain`	specifies a domain that serves as hints to the compiler	R
`FieldDecl`	Definition of all the fields used by the program	R
`VarDecl`	Definition of scalar variables arguments to the program	O
`ScopedProgram`	a scoped program with computational patterns supported by the concepts of the HIR	O
`ExternalKernel`	describes a kernel that contains computational patterns non supported by the concepts of the HIR	O

Attributes

name	type	description	R/O
`HIRversion`	text	version of the HIR	R
`DomainPolicy`	text	policy that defines the domain of the HIR	R
`time`	text	Date and time of translation	O
`language`	text	source language information	O
`source`	text	source code information	O

2.2 `Domain` element

The domain provides domain information of the application that is used as hints to the compiler toolchain.

ContentsModel

(DomainParallelDimensions , VerticalDimension , ParallelDimensions)

Child elements

name	description	R/O
`DomainParallelDimensions`	list of of one or more GridDimension + representing the dimensions over the domain is parallelized	R
`VerticalDimension`	contains the GridDimension element representing the vertical dimension	R
`ParallelDimensions`	list of GridDimension * on which computations are embarrassingly parallel	R

2.3 `ScopedProgram` element

The `ScopedProgram` defines all the computations performed using concepts of the HIR

ContentsModel

(BlockStmt)

Child elements

name	description	R/O
`BlockStmt`	Block statement containing the sequence of statements that forms the computation of the program	R

2.4 `ExternalKernel` element

The `ExternalKernel` defines a call to an external kernel, for which computations description is not provided

ContentsModel

(Inputs , Outputs)

Child elements

name	description	R/O
`Inputs`	list of FieldDecl that represents the input fields	R
`Outputs`	list of FieldDecl that represents the output fields	R

2.5 `GridDimension` element

The `GridDimension` elements defines a dimension of a multidimensional space where fields are discretized and over which Computation s iterate.

Contents model

()

Attributes

name	type	description	R/O
`name`	text	Name of the dimension	R

2.6 `DimensionLevel` element

The `DimensionLevel` it is used to specify a position (that is specified as a runtime argument to the Program) in a given dimension

ContentsModel

(VarAccess , Literal)

Child elements

name	description	R/O
`VarAccess`	It uses a scalar variable of rank N where each element act as a marker, whose runtime values will store the positions within the extent of the dimension.	R
`Literal`	It is a integer offset that shifts the position of the level with respect to the value of the VarAccess	R

2.7 `DimensionInterval` element

The `DimensionInterval` defines an interval on a dimension.

ContentsModel

(DimensionLevel , DimensionLevel)

Child elements

name	description	R/O
`DimensionLevel`	position placeholder on a dimension that defines the begin and end of the interval	R

2.8 `Type` element

The `Type` defines the type of storage declarations

Contents model

()

Attributes

name	type	description	suppoted values	R/O
`name`	text	any of the supported types	(double,int,float)	R

2.9 `FieldDecl` element

The `FieldDecl` element defines a multidimensional field storage

ContentsModel

(Type , GridDimension +)

Child elements

name	description	R/O
GridDimension	dimensions of the multidimensional space where the storage is defined	R
Type	value type of the grid elements of the field	R

Attributes

name	type	description	R/O
name	text	name of the field	R

2.10 `Offset` element

The `Offset` is the relative distance in a given GridDimension to a neighbor grid point.

ContentsModel

(GridDimension)

Child elements

name	description	R/O
GridDimension	Identifies the dimension where the offset is computed	R

Attributes

name	type	description	R/O
distance	int	relative distance in the given dimension of the offset	R

2.11 `Computation` element

The `Computation` defines an iteration loop over the specified GridDimension s of the domain.

ContentsModel

((GridDimension | DimensionInterval)+ ,(BlockStmt))

Child elements

name	description	R/O
GridDimension	Specifies the dimensions where the computation is defined, convering the whole extent of the grid for that dimension	O
DimensionInterval	Provides a specific range on a dimension to iterate over	O
BlockStmt	Specifies the block with the list of statements that form the computation	R

2.12 `BoundaryCondition` element

The `BoundaryCondition` defines the strategy to apply a boundary condition to a field, if required.

ContentsModel

(FieldDecl , BlockStmt)

Child elements

name	description	R/O
`FieldDecl`	field subject of boundary condition	R
`BlockStmt`	BlockStmt with statement that implement the boundary condition computation	R

Attributes

name	type	description	R/O
`distance`	int	relative distance in the given dimension of the offset	R

3 Statement elements

3.1 `VarDecl` element

The `VarDecl` represents a N-dimensional scalar.

Contents model

```
(Type)
```

Child elements

name	description	R/O
`Type`	type of the variable n-dimensional variable	R

Attributes

name	type	description	R/O
`name`	text	name of the variable	R
`ndims`	int	number of dimensions of the variable	R
`isarg`	bool	specifies if the variable is argument to the main program	R
`initialization`	string	operation to initialize the variable	0

3.2 `IfStmt` element

The `IfStmt` is a statement element that defines an if condition.

ContentsModel

```
( Condition , Then , Else? )
```

```
where if (scope == domain_computation)
Condition = ( unaryOpModel | binaryOpModel | TernaryOp | FieldAccess | VarAccess | Literal )
else
Condition = ( unaryOpModel | binaryOpModel | TernaryOp | VarAccess | Literal )
```

Child elements

name	description	R/O
`Condition`	contains expression that defines the condition of the if block	R
`Then`	contains a sequence of stmtModel that compose the `then` computation of the block	R
`Else`	contains a sequence of stmtModel that compose the `Else` computation of the block	O

3.3 `BlockStmt` element

The `BlockStmt` is a statement element that defines an block (of statements).

ContentsModel

(stmtModel +)

Child elements

name	description	R/O
`stmtModel`	statement that composes the `block` computation	R

3.4 `AssignmentStmt` element

The `AssignmentStmt` defines an assignment operation expression.

ContentsModel

(lValueModel , exprModel)

Child elements

name	description	R/O
`lValueModel`	Specifies the left-hand side expression. Refer to lValueModel .	R
`exprModel`	Specifies the right-hand side expression. Refer to exprModel .	R

4 Expression elements

4.1 `Literal` element

The `Literal` defines the specification of a literal.

Contents model

`(Type)`

Child elements

name	description	R/O
`Type`	type of the literal	R

Attributes

name	type	description	R/O
`value`	string	value of the literal	R

4.2 `Binary operations` element

The elements representing binary operators are described below and follows the binaryOpModel :

element	operator	operation
plusOp	+	addition
minusOp	-	subtraction
mulOp	*	multiplication
divOp	/	division
powerOp	**	power
logicalAnd	&&	logical AND
logicalOr	——	logical OR
logicalEqual	==	logical equality
logicalNotEqual	!=	logical unequality
logicalGt	¿	logical greater than
logicalLt	¡	logical less than
logicalGe	¿=	logical greater or equal than
logicalLe	¡=	logical less or equal than

4.3 Unary operations element

The elements representing unary operators are described below and follows the unaryOpModel :

element	operator	operation
unaryMinus	-	sign inversion
logNot	!	logical not
incrementOp	++	increment by one
decrementOp	–	decrement by one

ContentsModel

(exprModel)

Child elements

name	description	R/O
exprModel	Specifies the operand expression. Refer to exprModel	R

4.4 TernaryOp element

The TernaryOp defines an ternary operator expression.

ContentsModel

(exprModel , exprModel , exprModel)

Child elements

name	description	R/O
exprModel	Specifies the condition of the operation as the first operand, the left-hand expression of the ternary operation as the second operand, the right-hand expression of the ternary operation as the third operand. Refer to exprModel .	R

Attributes

name	type	description	R/O
operator	string	operator being applied to the operands	R

4.5 FctCall element

Built-in function call for mathematical functions.

ContentsModel

(exprModel+)

Child elements

name	description	R/O
`exprModel`	Arguments of the function call.	R

Attributes

name	type	description	R/O
`name`	string	function name: (abs, sqrt, sin, cos, tan, asin, acos, atan, exp, log)	R

4.6 `VarAccess` element

The `VarAccess` is a expression that defines an access to a VarDecl

ContentsModel

(Literal *)

Child elements

name	description	R/O
`Literal`	access index of the var, when it is declared with more than 1 dimension	O

Attributes

name	type	description	R/O
`name`	string	The var declaration that is being accessed in this expression	R

4.7 `FieldAccess` element

The `FieldAccess` is a expression that defines an access to a field

ContentsModel

(Offset +)

Child elements

name	description	R/O
`Offset`	An offset (relative to current grid position) used to de-reference the field access	O

Attributes

name	type	description	R/O
`name`	string	The name of the field declaration that is being accessed in this expression	R

5 Common elements

The definitions commonly used in an arbitrary element are shown in this Section.

5.1 `stmtModel` model

The `stmtModel` is commonly used for elements that refer statements.

ContentsModel

(VarDecl | IfStmt | BlockStmt)

5.2 `exprModel` model

The `exprModel` is commonly used for elements that refer expression.

ContentsModel

```
where if (scope == domain_computation)
    ( unaryOpModel | binaryOpModel | TernaryOp | Literal | FieldAccess | VarAccess | FctCall )
else if (scope == control_flow)
    ( unaryOpModel | binaryOpModel | TernaryOp | VarAccess | Literal | FctCall | FctCall )
```

5.3 `lValueModel` model

The `lValueModel` is commonly used for elements that refer left-hand side expression.

ContentsModel

```
where if (scope == domain_computation)
    ( VarDecl | VarAccess | FieldAccess )
else
    ( VarDecl | VarAccess )
```

5.4 `binaryOpModel` model

The `binaryOpModel` is used for elements that refer to binary operations.

ContentsModel

```
(exprModel, exprModel)
```

Child elements

name	description	R/O
`exprModel`	Specifies the left-hand expression as the first operand, the right-hand expression as the second operand. Refer to exprModel .	R

5.5 `unaryOpModel` model

The `unaryOpModel` is used for elements that refer to unary operations.

ContentsModel

```
(exprModel)
```

Child elements

name	description	R/O
`exprModel`	Specifies the right-hand expression. Refer to exprModel .	R

5.6 `Common attributes` model

Some elements may have the following attributes.

Attributes

name	type	description	R/O
`lineno`	text	Specifies the line number in the source program	R
`file`	text	Specifies the source code file name	R

6 Example

```
Program {
  [ GridDimension {ncol}, GridDimension {nlay}, GridDimension {ngpt}],
  Domain {
    parallel_domain_dim : ncol,
    vertical_dim : nlay,
    parallel_dim : ngpt
  },
  FieldDecl {
    real, [GridDimension {ncol}],
    name : mu0
  },
  FieldDecl {
    real, [GridDimension {ncol}, GridDimension{nlay}, GridDimension{ngpt}],
    name : tau
  },
  FieldDecl {
    real, [GridDimension {ncol}, GridDimension {nlay}, GridDimension{ngpt}]
    name : w0
  },
  FieldDecl {
    real, [GridDimension {ncol}, GridDimension {nlay}, GridDimension {ngpt}],
    name : g
  },
  FieldDecl {
    real, [GridDimension {ncol}, GridDimension{nlay}],
    name : Rdif
  },
  FieldDecl {
    real, [GridDimension {ncol}, GridDimension {nlay}],
    name : Tdif
  },
  FieldDecl {
    real, [GridDimension {ncol}, GridDimension {nlay}],
    name : Rdir
  },
  FieldDecl {
    real, [GridDimension {ncol}, GridDimension {nlay}],
    name : Tdir
  },
  FieldDecl {
    real, [GridDimension {ncol}, GridDimension {nlay}],
    name : Tnoscat
  },
  VarDecl {
    int,
    name : ncolbounds,
    isarg : true
  },
  VarDecl {
    int,
    name : nlaybounds,
    isarg : true
  },
  ScopedProgram {
    // two_stream (Missing arg list, name of what is called, shouldn't inline stuff at this point)
    // adding_sw (same as up)
    // additional_step (same as up)
    // two_stream
    VarDecl {
      real,
```

```
      name : mu0_inv
    },
    VarDecl {
      real,
      name : mu0_inv
    },
    VarDecl {
      real,
      name : gamma1
    },
    VarDecl {
      real,
      name : gamma2
    },
    VarDecl {
      real,
      name : gamma3
    },
    VarDecl {
      real,
      name : gamma4
    },
    VarDecl {
      real,
      name : alpha1
    },
    VarDecl {
      real,
      name : alpha2
    },
    VarDecl {
      real,
      name : k
    },
    VarDecl {
      real,
      name : RT_term
    },
    VarDecl {
      real,
      name : exp_minusktau
    },
    VarDecl {
      real,
      name : exp_minus2ktau
    },
    VarDecl {
      real,
      name : k_mu
    },
    VarDecl {
      real,
      name : k_gamma3
    },
    VarDecl {
      real,
      name : k_gamma4
    },
    BlockStmt {
      // Computation, this is the only stmt that we will express as a tree.
      // From this on, pseudocode for the statements will be used
      AssignmentStmt {
        VarAccess {
          name : mu0_inv
        },
        divOp {
          VarAccess {
            name : mu0
          },
          Literal {
            real,
```

```
          value : 1.0
        }
      }
    },
    Computation {
      [ GridDimension {name: ngpt}, GridDimension {name: ncol} ],
      DimensionInterval {
        GridDimension {name : nlay},
        DimensionLevel {
          VarAccess {
            Literal {0},
        name : nlaybounds
          },
          offset : 0
        },
        DimensionLevel {
          VarAccess {
            Literal {1},
        name : nlaybounds
          },
          offset : 0
        }
      },
      BlockStmt {
        AssignmentStmt {
          VarAccess {
            name : gamma1
          },
          mulOp {
            minusOp {
              Literal { 8.0 },
              mulOp {
                VarAccess { name : w0 },
                addOp {
                  Literal { 5.0 },
                  mulOp {
                    Literal { 3.0 },
                    VarAccess { name : g }
                  }
                }
              }
            },
            Literal { 0.25 }
          },
          lineno : 654,
          file : sw_solver.f90
        },
        AssignmentStmt {
          VarAccess { name : gamma2},
          mulOp {
            Literal { 3.0 },
            mulOp {
              mulOp {
                VarAccess { name : w0 },
                minusOp {
                  Literal { 1.0 },
                  VarAccess { name : g }
                }
              },
              Literal { 0.25 }
            }
          },
          lineno : 655,
          file : sw_solver.f90
        },
        AssignmentStmt {
          VarAccess { name : gamma3 },
          mulOp {
            minusOp {
              Literal { 2.0 },
              mulOp {
```

```
                    mulOp {
                      Literal { 3.0 },
                      VarAccess { name : mu0 }
                    },
                    VarAccess { name : g }
                  }
                },
                Literal { 0.25 }
              }
            }
            AssignmentStmt {
              VarAccess { name : gamma4 },
              minusOp {
                Literal { 1.0 },
                VarAccess { name : gamma3 }
              }
            }

            % Rest of the program not represented as HIR
            % alpha1 = gamma1 * gamma4 + gamma2 * gamma3
            % alpha2 = gamma1 * gamma3 + gamma2 * gamma4
            % k = sqrt(max((gamma1 - gamma2) * (gamma1 + gamma2), 1.e-12_wp))
            % exp_minusktau = exp(-tau * k)
            % exp_minus2ktau = exp_minusktau * exp_minusktau
            % RT_term = 1._wp / (k * (1._wp + exp_minus2ktau)  + gamma1 * (1._wp - exp_minus2ktau) )
            % Rdif = RT_term * gamma2 * (1._wp - exp_minus2ktau)
            % Tdif = RT_term * 2._wp * k * exp_minusktau
            % Tnoscat = exp(-tau * mu0_inv)
            % k_mu      = k * mu0
            % k_gamma3 = k * gamma3
            % k_gamma4 = k * gamma4
            % RT_term =  w0 * RT_term/merge(1._wp - k_mu*k_mu, epsilon(1._wp), abs(1._wp - k_mu*k_mu) >= epsilor
            % Rdir = RT_term  * ((1._wp - k_mu) * (alpha2 + k_gamma3) -  (1._wp + k_mu) * (alpha2 - k_gamma3) *
* exp_minusktau  * Tnoscat)
            % Tdir = Tnoscat - RT_term * ((1._wp + k_mu) * (alpha1 + k_gamma4) * Tnoscat - (1._wp - k_mu) * (alp
* exp_minusktau)
            % Tdir -= Tnoscat
          }
        }
      }
    }
  }
```