EPSN

Fortran 90 Tutorial

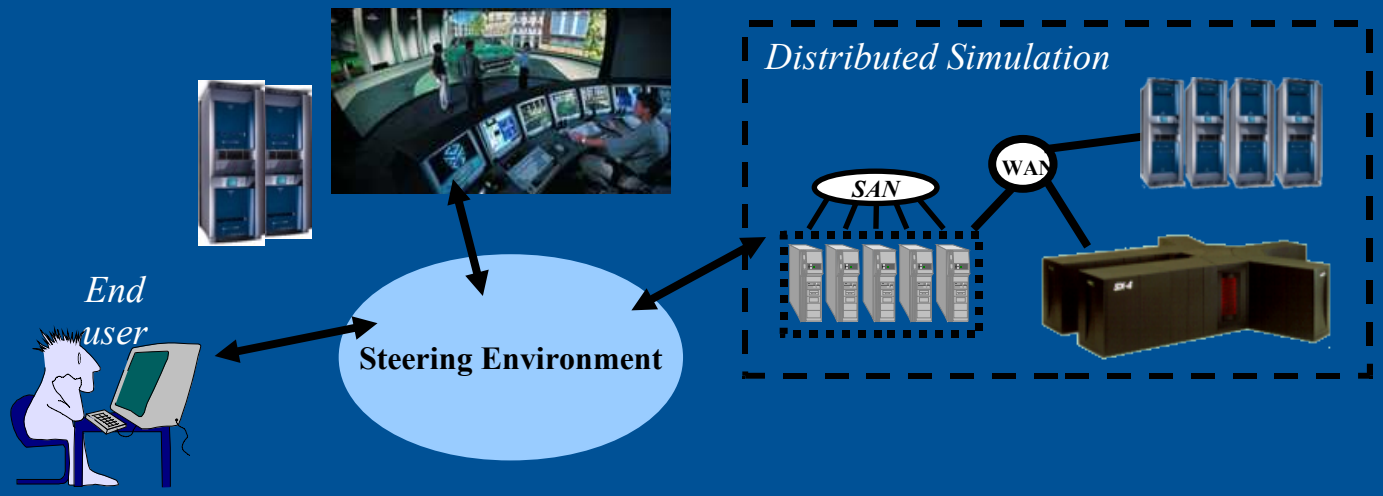
V 0.3

Olivier.Coulaud@inria.fr

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
BORDEAUX - SUD-OUEST

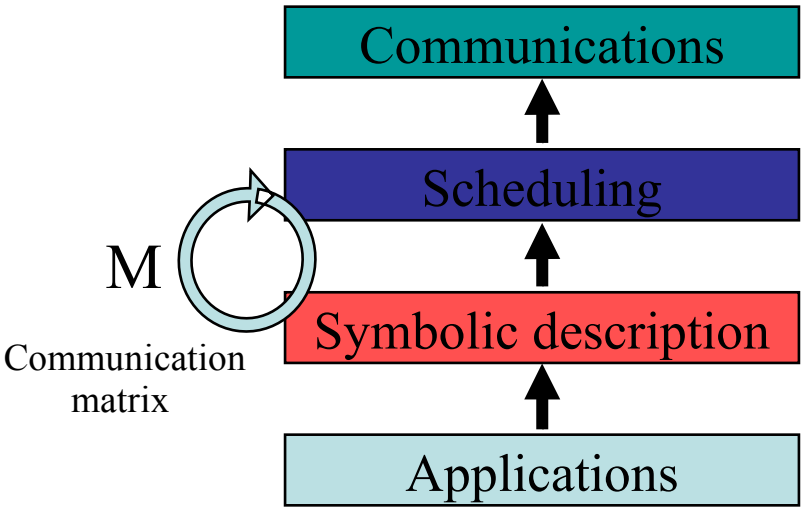


Outline

- EPSN: Steering environment of parallel simulation
  - Model
  - Platform
- Simulation integration
  - F90 API and XML file
  - RedSYM Points and Parameter API
- Examples
  - Heat equation
  - DL_poly
- Tulip plugin first results

Data transferts / Redistribution

Data redistribution generic approach

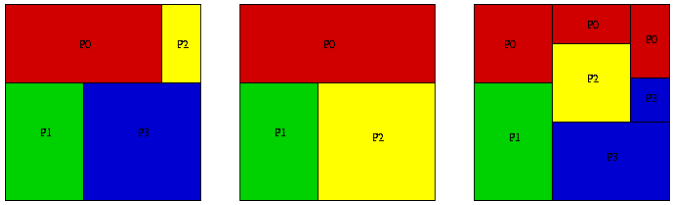


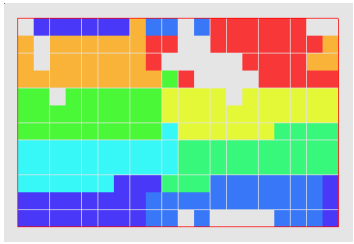
- Parallel flow
- Cutting messages to maximize rate
- Symbolic description of data
- Distribution of data between processors
- Memory organization in memory on a processor.

- Modular and efficient approach

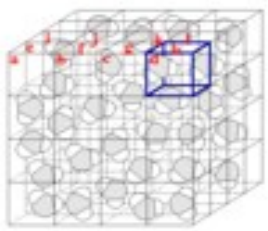
Data model in EPSN

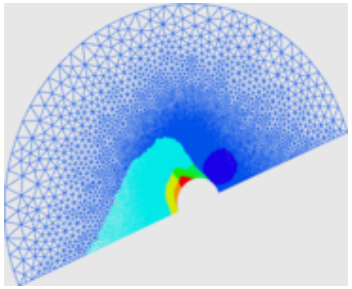
- Complex objects
  - Scalar, tabular, particles, Grid, unstructured meshes
  - Many data series (velocity, pressure, ...)
- Distributions
  - Regular objects : spatial approach: HPF, rectilinear, implicit





  - Irregulars objects:





Data model in EPSN

Symbolic description of data : spatial approach

- Data distribution on processors
- Memory organisation of data on a processor

Three steps

- Describe global domain

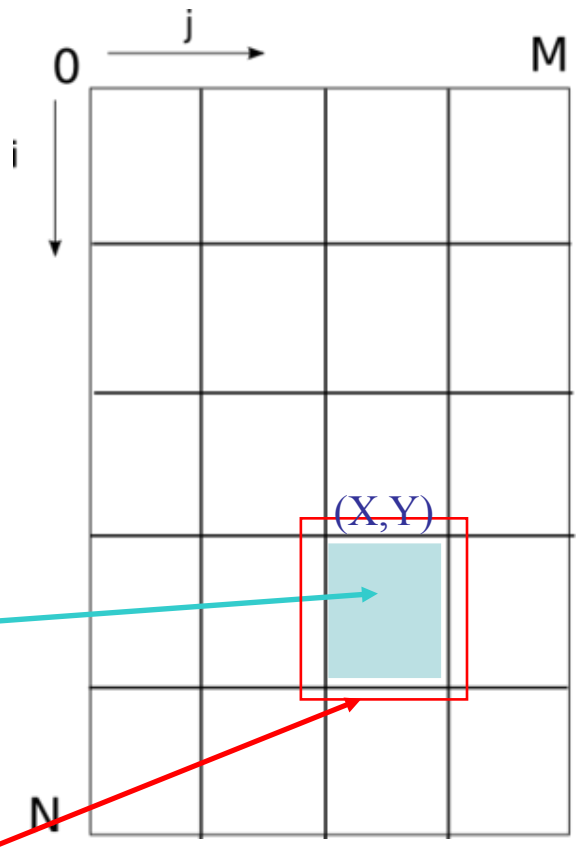
  Block : (0,0) (N,M)

- Describe local domains (block)

  Block : (X,Y) (X+a,Y+b)

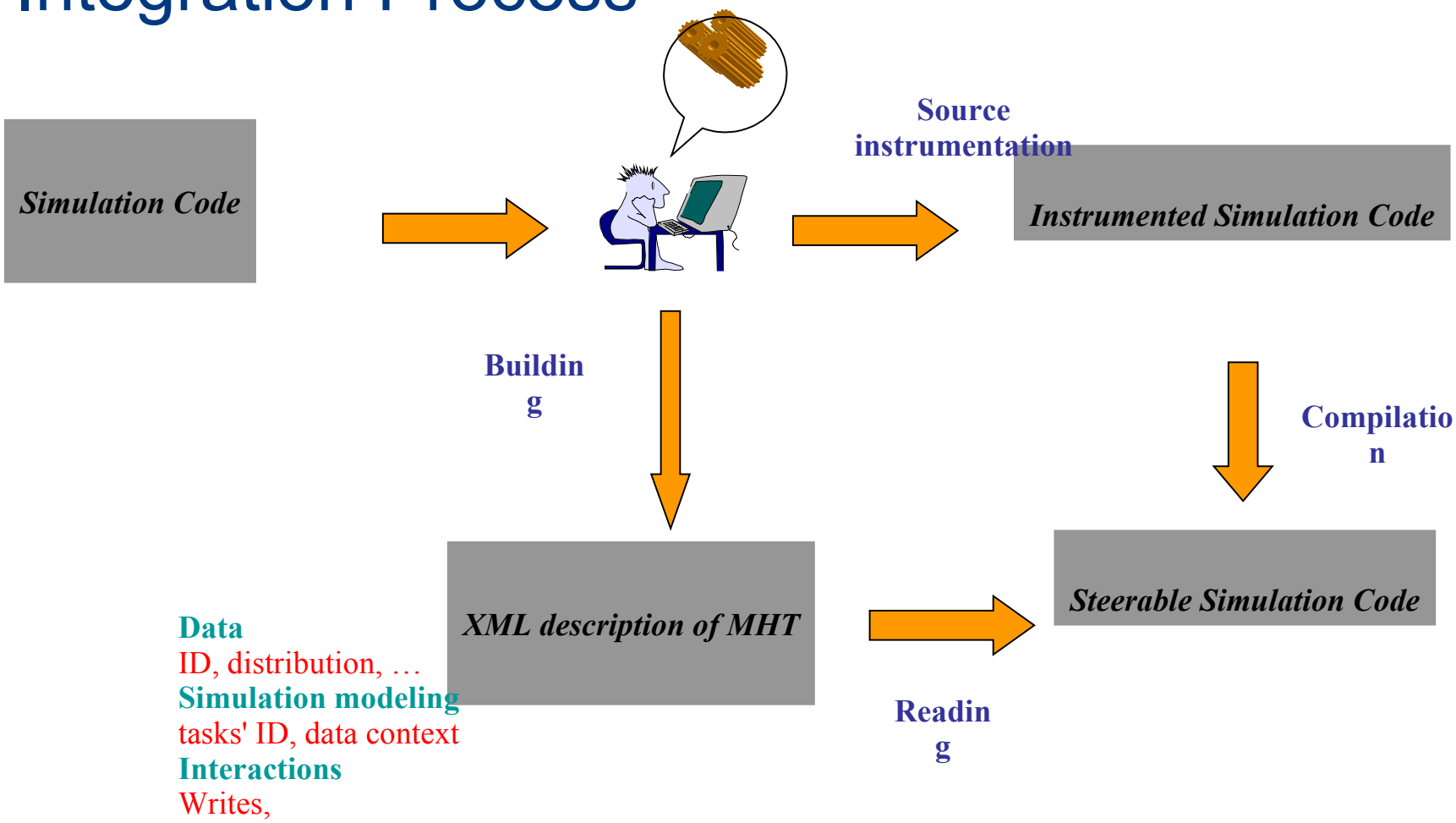
- Describe memory agencement for blocks.

  Block : (X-offset,Y-offset) (X+a+offset,Y+b+offset)



How build a steerable simulation?

Integration Process



How do you steer your simulation with EPSN? (1)

Three steps:

1) Build a Fortran 90 module

- Goal: initialization of EPSN and add all the steerable variables

2) Instrumentation of the code with the F90 API describes bellow

- CEPSN call EPSN_F90_BeginHTM( )

3) Makefile modification

How do you steer your simulation with EPSN? (2)

Extension of the module .F or.F90

Two methods

- 1) Initialization of EPSN

2) Describe and map your data in EPSN with the RedSYM API

```
module PGM_EPSN
#ifdef  __USE_EPSN__
   USE moduleEPSN    ! To Access to the EPSN API
   USE …             ! To access to yours data
!
   INTEGER(EPSN_KIND_STATUS)  :: status
   INTEGER                    :: proxyNode, err
   INTEGER(EPSN_KIND_HANDLE)  :: data1,data2,
     ….
 public:: initEPSN, addData, …
Contains
      ….
#else
      INTEGER :: EMPTY
#endif

end module PGM_EPSN
```

How do you steer your simulation with EPSN? (3)

Modification in the Makefile.

Add a target to compile with EPSN

Set compiler flags

Allow the call of the preprocessor

Change the extension of the file .F or .F90

Add a flag (gfortran f77-cpp-input or f90-cpp-input)

Two flags: -D__USE_EPSN__ -D'CEPSN='

Linker flags

EPSN_LIBS= -lepsn_simulation_f90 -lepsn_simulation \

-lredsym -lmetis

-L$(EPSN_TOP)/lib -L$(EPSN_LIBS)

The RedSYM Point Object (1)

Describe a set of points (x,y,z) in space: particles, atoms, ….

Allow to put them together to build

molecule, system galaxy, ….

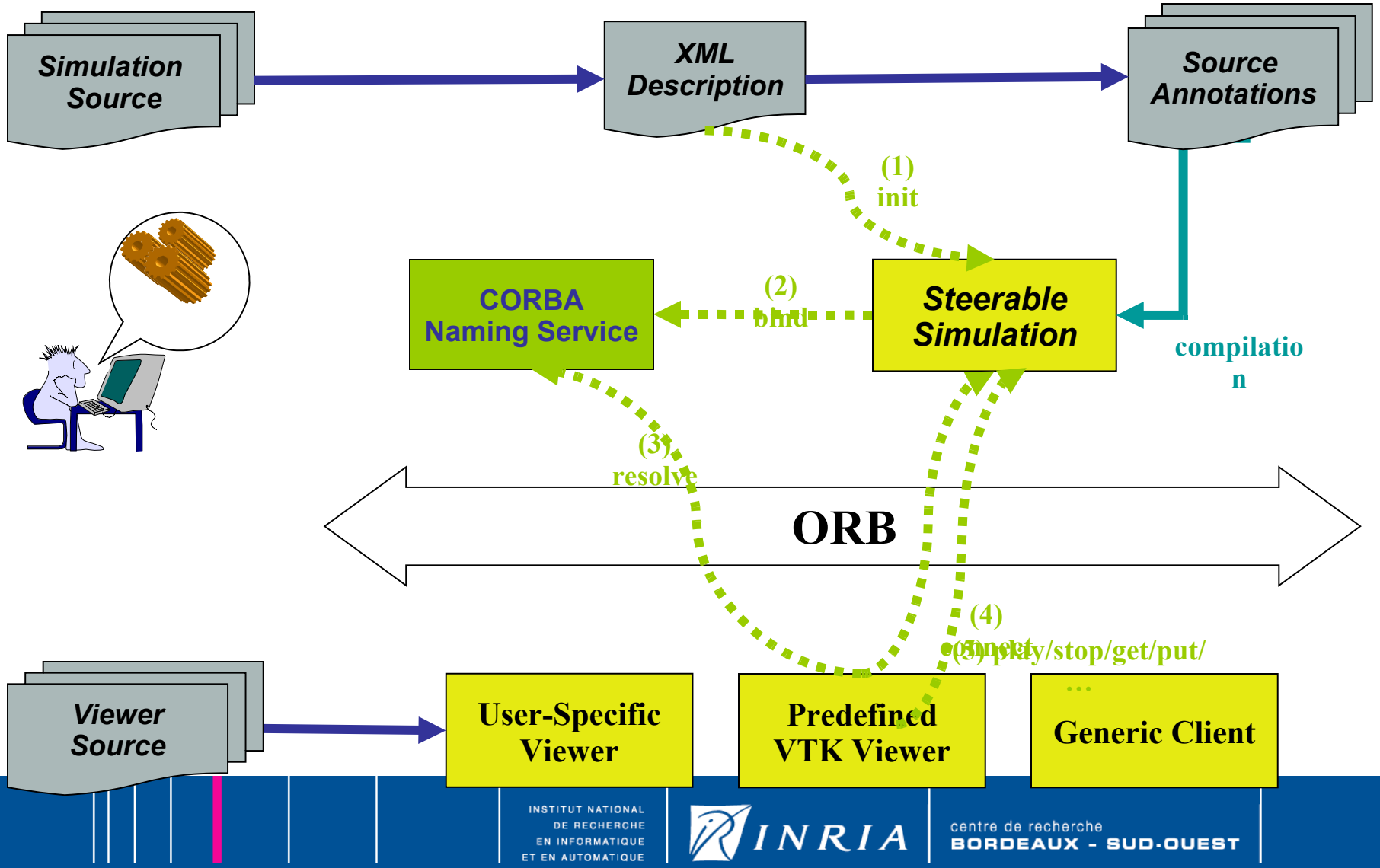
Two steps

1) RedSYM Object description

   Name, region, variables' name

2) Mapping between RedSYM Object and the real data
   - Distribution on the processors and the mapping in memory.
   - Link between the symbolic redsym object and the data

Instrumentation Process + Deployment



What is the EPSN and RedSYM Fortran API

Fortran 90 API

Two fortran types - KIND

- EPSN_KIND_STATUS
- EPSN_KIND_HANDLE

Three kinds of functions

1. Global functions
   - EPSN_F90_Init
   - EPSN_F90_Finalise
   - EPSN_F90_Print
2. Data management functions
   - EPSN_F90_Ready
   - EPSN_F90_AddData
   - EPSN_F90_UpdateData
1. Code structure descriptions
   - EPSN_F90_BeginHTM
   - EPSN_F90_EndHTM
   - EPSN_F90_BeginTask
   - EPSN_F90_EndTask
   - EPSN_F90_BeginLoop
   - EPSN_F90_EndLoop
   - EPSN_F90_Point

Global variable

EPSN_STATUS

- EPSN_SUCCES
- EPSN_FAILURE

Fortran 90 API (1)

EPSN_F90_init(*simulationName, xmlFilename, myrank, totalNumberOfProcs, proxyNode)*

integer	myRank	Local processor ID
integer	totalNumberOfProcs	Total number of Processor in Simulation
Integer	proxyNode	Proxy owner ID
Character	simulationName	Simulation name. This name can be found in naming service.
Character	xmlFilename	XML file name associated to the simulation

Initialize EPSN environment:

- Create EPSN thread and proxy.
- Record simulation in naming service.

Fortran 90 API (2)

subroutine EPSN_F90_Finalize()

Close the environment and delete the simulation in the naming service.

subroutine EPSN_F90_Print()

Display information about simulation:

```
EPSN Data
    Simulation name      : loss4d
    XML file   name      : loss4d.xml
    Number of processes  :  4   my rank 0
    Proxy on node        : 0
```

Fortran 90 API: data management

subroutine **EPSN_F90_AddData**(*handle_var*)

INTEGER(**EPSN_KIND_HANDLE**), intent(IN) :: *handle_var*

Add the RedSYM variable descriptor *handle_var* in EPSN.

subroutine **EPSN_F90_UpdateData**(*name*)

Specify to the environment that data structure associated to name ID have changed. Required for dynamic of variable, this implies that the communication matrix reconstruction may be necessary.

subroutine **EPSN_F90_Ready**()

Specify to the environment that data are available and allocated. After this function, it's possible to recover data from a client.

API Fortran 90 : HTM Description (1)

EPSN_F90_BeginHTM() HTM area begins at this point

EPSN_F90_EndHTM() HTM area stops at this point.

Example

```
CALL EPSN_F90_BeginHTM()
 DO J = Jdeb, Jfin
    DO I = 1, NX
        Unp1(I,J) =  …
  ENDDO
 CALL EPSN_F90_BeginHTM()
```

XML description

```
<htm >
…
</htm >
```

API Fortran 90 : MHT Description (2)

EPSN_F90_BeginTask(*name*) Mark the beginning of the task *name*

EPSN_F90_EndTask(*name*) Mark the ending of the task *name*

- *name* is the task ID
- Allow to define an area for the request processing

Example

```
CALL EPSN_F90_BeginTask("update")
 DO J = Jdeb, Jfin
     DO I = 1, NX
                 Unp1(I,J) =  …
  ENDDO
 CALL EPSN_F90_BeginTask("update")
```

XML description

```
<task id=" update " >
…
</task >
```

Fortran 90 API: HTM Description (3)

EPSN_F90_BeginLoop(*name*) Mark the beginning of a loop named *name*

EPSN_F90_EndLoop(*name*) Mark the ending of the loop *name*

- *name is the loop ID*
- Allow to define an iteration counter - useful to manage data release

Moreover in the loop one must have a task.

Example

```
CALL EPSN_F90_BeginLoop( "timeLoop")
DO t = 1, NB_ITERMAX          ! Time loop
CALL EPSN_F90_BeginTask("bodyLoop")
      …..
CALL EPSN_F90_BeginTask(" bodyLoop ")
END DO
CALL EPSN_F90_EndLoop( "timeLoop")
```

XML description

```
<loop id="timeLoop">
      <task id= " bodyLoop " />
</loop>
```

API Fortran 90 : MHT Description (3)

EPSN_F90_Point(*name*) tâche ponctuelle

name refers to task in MHT description

Forcément un surcoût si une requête est traitée ou si une action est activée

Example

```
CALL EPSN_F90_Point( " BeforeTimeLoop ")
DO t = 1, NB_ITERMAX          ! Time loop
…..
END DO
```

XML description

```
<point id="BeforeTimeLoop" />
```

XML file

Describes the EPSN steering Model

- *data*: describes code data
- *interaction* : describes actions that can be done
- *htm*: contains structure code and specify data context

XML structure

```
<?xml version="1.0" encoding="UTF-8" ?>
<simulation id="heat">
  <data>
      …
  </data>
  <interaction>
      …
  </interaction>
  <htm >
      …
  </htm>
</simulation>
```

XML file (data)

Data group represents the RedSYM complex objects

- Types: scalar, grid, points, mesh

  <object class=type id="name" label="text" />

- For each object we can add one or more variables

  variable: one variable of the code

  <variable id="name" label="text" />

Example

```
<data>
    <object class="points"
id="particles">
      <variable id="XYZ"/>
      <variable id="Charge"/>
      <variable id="Mass"/>
      <variable id="Velocity"/>
      <variable id="Acceleration"/>
    </object>
</data>
```

XML file (interactions)

interaction group includes actions authorized in a task

<action id="name" distribution-kind ="(*located*|*replicated*)" label="text" />

parameters

distribution-kind specify how the action is launched. Default is replicated.

located:

replicated: All the processes of the simulation perform the action

Example

```
<interaction>
    <action id="particles">
</interaction >
```

XML file (code structure)

htm group represents the code structure

- Declaration <htm auto-start="*true*|*false*">

auto-start specify if simulation starts or stops on the function EPSN_F90_BeginHTM(). In the later case, an EPSN client must unblock simulation.

- The attributes
  - Code structure

    task, loop, switch, point
  - Context

    data-context, action-context, bench-context

XML file (code structure)

Control attributes *task, loop, switch, point*

<type synchronize=(*no*|*auto*|*block*) id="name" label="text" />

Parameters

type: *task, loop, switch, point*

synchronize: specify if synchronization is necessary entering or leaving block code.
Default value is *no*.

auto : same as the MPI_BARRIER synchronization

block : stops simulation on the task

Example

```
<task id="A" synchronize="no">
…
</task>
```

XML file (code structure)

Context attributes: data-context

<data-context ref="name" [variable-id="name"] context = (*readable*|*writable*|*protected*|*modified*) />

Parameters

1) ref is the id of the data object and it must exist in the data group.
2) variable-id is the id of the variable that you want to precise the context in the object.
3) context specify the access context of the data on the task and all its children
   - ***readable***: the data is readable, so we can get it
   - ***writable****:* the data is writable, that is to say a client can modify the data.
   - ***protected****:* protected the data is protected, this context is use to deny all
   - ***modified****:* variable is unreachable and is modified. This value specify that a new value for the variable has appeared.

Example

```
<task id="advection”>
    <data-context ref="density" context="modified"/>
</task>
```

XML file (code structure)

Context attributes : action-context

<action-context ref="name" context = (*allowed*|*forbidden*)/
>

Parameters

1) ref is the id of the action and it must exist in the *interactions* group.

1) context specify the access context of the data in the task and all its children

allowed *:* action is allowed in the task.

forbidden *:* action is forbidden in the task.

XML file (code structure)

Context code : bench-context

<bench-context frequency = "freq"/>

Parameter

frequency the frequency of data recovering expressed in step.
Default value is 1.

Fortran 90 and F77 RedSYM API

The API depends on the object

1) Points
2) Grid
3) Mesh
4) Scalar

Two steps

1) RedSYM Object description.
2) Mapping between the code variables and the RedSYM object.

RedSYM POINTS
Fortran API

Fortran 90 API: RedSYM object creation (1)

REDSYM_POINTS_CREATE(***key***, *name*, *myrank*, *nbProcs*, *dimension*)

- *key* (INTEGER(EPSN_KIND_HANDLE): the key to access to the RedSYM object
- *name (Character):* name of the object
- MyRank (integer) : the rank of the processor
- nbProcs (integer) : maximum number of processors
- dimension (integer) the dimension of the system.

YM_POINTS_ADDVARIABLE(key,name,type,nbComponent,**numVar**)

y (INTEGER(EPSN_KIND_HANDLE): the key to access to the RedSYM object

me (Character): name of the variable

e (integer): the type of the variable (*REDSYM_DOUBLE, ...*)

Component (integer): number of dimension of the variable

m (integer): the variable number in the object (key)

Fortran 90 API: RedSYM object creation (2)

REDSYM_POINTS_SETCOORDINATEVARIABLE (key,numVar)

- *key* (INTEGER(EPSN_KIND_HANDLE): the key to access to the RedSYM object
- numVar (integer): the number of the variable which corresponds to the coordinate of the set of points. The coordinate are in one array of size *dimension*.

REAL(8) XYZ(3,*N)*

x1,y1,z1,x2,y2,z2, ... , xN,yN,zN

REDSYM_POINTS_SETCOORDINATEVARIABLE**S** (key,numVar)

- *key* (INTEGER(EPSN_KIND_HANDLE): the key to access to the RedSYM object
- numVar (integer): array of size *dimension*. The coordinate are in *dimension* arrays of size one.

REAL(8) X(N*), Y(N), Z(N)*
REAl(8) XYZ(N,3)

x1,x2, ..., xN	y1,y2, ..., yN	z1,z2, ..., zN

API Fortran 90 : Mapping (1)

EPSN_POINTS_ADDREGION(*key*, ***idRegion***, *maxPoints*, *nbPoints*, *tag*)

- *key* (integer(epsn_kind_handle)): the key to access to the RedSYM object
- idRegion (integer): the identifier of the region.
- maxPoints(integer): maximum number of points in the region
- nbPoints(integer): number of points in the region
- tag(integer): a tag for the region

A region is a logical set of points in which all information (position, velocity, …) are stored continuously in memory.

API Fortran 90 : Mapping (2)

REDSYM_POINTS_WRAP(*key*, numVar, ***idRegion***, var)

- *key* (integer(epsn_kind_handle)): the key to access to the RedSYM object
- idRegion (integer): the identifier of the region. Region is a set of
- numVar (integer): the variable number
- var: the Fortran variable

Wrap the data and the variable in the RedSYM object.

The data var is stored continously in memory.

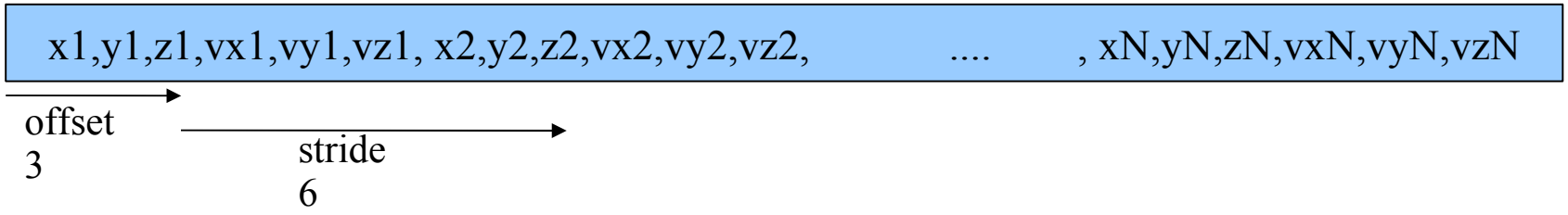
API Fortran 90 : Mapping (3)

DSYM_POINTS_WRAPSTRIDED(*key*, numVar, ***idRegion***, var, stride, offset)

- *key* (integer(epsn_kind_handle)): the key to access to the RedSYM object
- *idRegion* (integer): the identifier of the region. Region is a set of
- *numVar* (integer): the variable number
- *var* is the Fortran variable
- *stride* (integer): is the distance between two elements
- *offset* (integer): is the jump to do to obtain the first element

Wrap the data and the variable in the RedSYM object.

The data var is not stored continuously in memory.



RedSYM MESH
Fortran API

Fortran 90 API: RedSYM object creation (1)

REDSYM_MESH_CREATE(***key***, *name*, *myrank*, *nbProcs*, *dimension*)

- *key* (INTEGER(EPSN_KIND_HANDLE): the key to access to the RedSYM object
- *name (Character):* name of the object
- MyRank (integer): the rank of the processor
- nbProcs (integer): maximum number of processors
- dimension (integer): the dimension of the system.

EDSYM_MESH_ADDVARIABLE(key,name,type,nbComponent,**numVar**)

- *key* (INTEGER(EPSN_KIND_HANDLE): the key to access to the RedSYM object
- *name (Character):* name of the variable
- *type* (integer): the type of the variable (*REDSYM_DOUBLE, ...*)
- *nbComponent* (integer): number of dimension of the variable
- num (integer): the variable number in the object (key)

Fortran 90 API: RedSYM object creation (2)

REDSYM_POINTS_SETCOORDINATEVARIABLE (key,numVar)

- *key* (INTEGER(EPSN_KIND_HANDLE): the key to access to the RedSYM object
- numVar (integer): the number of the variable which corresponds to the coordinate of the set of points. The coordinate are in one array of size *dimension*.

REAL(8) XYZ(3,*N)*

x1,y1,z1,x2,y2,z2, ... , xN,yN,zN

REDSYM_POINTS_SETCOORDINATEVARIABLE**S** (key,numVar)

- *key* (INTEGER(EPSN_KIND_HANDLE): the key to access to the RedSYM object
- numVar (integer): array of size *dimension*. The coordinate are in *dimension* arrays of size one.

REAL(8) X(N*), Y(N), Z(N)*
REAl(8) XYZ(N,3)

x1,x2, ..., xN	y1,y2, ..., yN	z1,z2, ..., zN

API Fortran 90 : Mapping (1)

EPSN_MESH_ADDREGION(*key*, ***idRegion***, *maxNodes*, *nbNodes*, *maxCells*,*nbCells*, *tag*)

- *key* (integer(epsn_kind_handle)): the key to access to the RedSYM object
- idRegion (integer): the identifier of the region.
- maxPoints(integer): maximum number of nodes in the region
- nbPoints(integer): number of nodes in the region
- maxCells(integer): maximum number of cells in the region
- nbCells(integer): number of cells in the region
- tag(integer): a tag for the region

…

RedSYM Parameter Fortran API

Fortran 90 API: RedSYM object creation (1)

EDSYM_PARAMETER_CREATE(***key***, *name*, *myrank*, *nbProcs*, *distribution*)

- *key* (INTEGER(*EPSN_KIND_HANDLE*): the key to access to the RedSYM object
- *name (Character):* name of the object
- myRank (integer) : the rank of the processor
- nbProcs (integer) : maximum number of processors
- *distribution* (integer) : two values *RedSYM_REPLICATED* or *RedSYM_LOCATED*.

DSYM_PARAMETER_ADDVARIABLE(key,name,type,nbComponent,**numVar**)

- *key* (INTEGER(*EPSN_KIND_HANDLE*): the key to access to the RedSYM object
- *name (Character):* name of the variable
- type (integer): the type of the variable (*REDSYM_DOUBLE, ...*)
- nbComponent (integer): number of dimension of the variable
- num (integer): the variable number in the object (key)

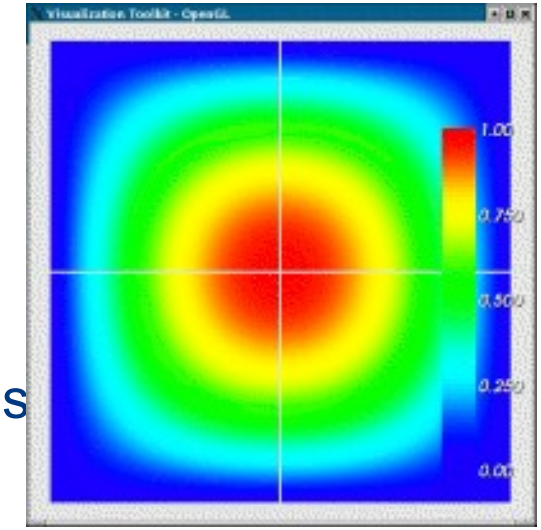
API Fortran 90 : Mapping (2)

REDSYM_PARAMETER_WRAP(*key*, numVar, buffer)

- *key* (integer(epsn_kind_handle)): the key to access to the RedSYM object
- idRegion (integer): the identifier of the region. Region is a set of
- numVar (integer): the variable number
- buffer: the Fortran variable

An example

Academic example



System

- Finite differencefor time discretisation on s
- Euler scheme

Two steps in the algorithm

- Update

$$U_{i,j}^{n+1} = f(U_{i-1,j}^{n}, U_{i+1,j}^{n}, U_{i,j}^{n}, U_{i,j-1}^{n}, U_{i,j+1}^{n})$$

- Swap

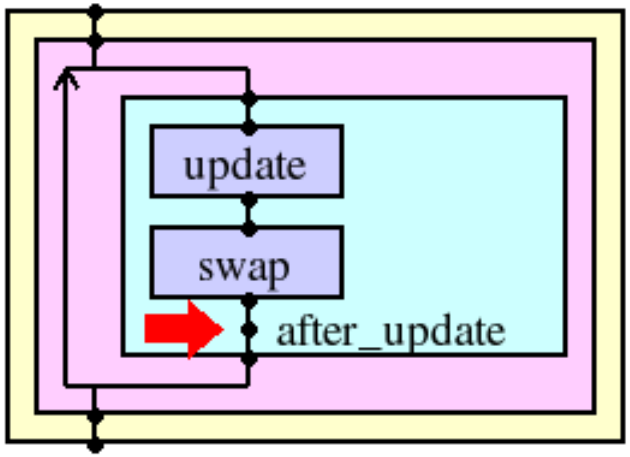
$$U^{n} := U^{n+1}$$

Parallelism

- MPI, master-slaves,
  - Esclaves SPMD,
  - 1D data distribution

Exemple – annotation

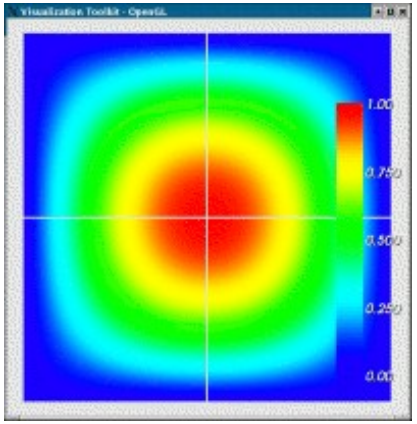
```
CALL EPSN_F90_BeginLoop( "timeLoop")
DO t = 1, NB_ITERMAX   ! Boucle en temps
      CALL EPSN_F90_BeginTask("bodyLoop")
       DO J = Jdeb, Jfin
          DO I = 1, NX
                    Unp1(I,J) =  …
      ENDDO
       CALL EPSN_F90_BeginTask("swap")
       Un(:,:) = unp1(:,:)          ! Swap
       CALL EPSN_F90_EndTask("swap")
       CALL EPSN_F90_Point("after_update")
       CALL EPSN_F90_EndTask( "bodyLoop")
END DO
CALL EPSN_F90_EndLoop( "timeLoop")
```



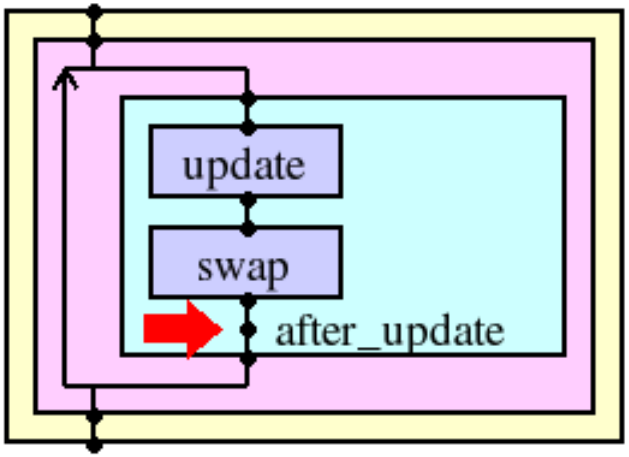
Example – data description

```
USE redsym
INTEGER(EPSN_KIND_HANDLE)  :: GlobalDomain, LocalDomain, AllocatedLocalDomain
INTEGER(EPSN_KIND_HANDLE)  :: Distribution1D,blocks(1),grid

lower(1) = 0 ; lower(2) = 0 ; upper(1) = Nx-1 ;   upper(2) = Ny-1
call RedSYM_BlockInt_create(GlobalDomain,nb_dims,lower,upper)
NBB = Ny/ NB_PROCS ; lower(2) = NBB*rank ; upper(2) = lower(2)+NBB
call RedSYM_BlockInt_create(blocks(1),nb_dims,lower,upper)
call RedSYM_DistributionInt_create(Distribution1D,GlobalDomain, nb_blocks,blocks)

call RedSYM_Grid_create( grid, "heat",my_rank, nb_procs, nb_dims )
call RedSYM_Grid_addVariable(grid,"temperature", REDSYM_DOUBLE ,1)
call RedSYM_Grid_setDistribution(grid, Distribution1D)
  !
Lower(2) = Lower(2) –offset ; upper(2) =upper(2) + offset    ! offset = 1
call RedSYM_BlockInt_create(AllocatedLocalDomain,nb_dims,lower,upper)
call RedSYM_Grid_wrapAllocatedBlock(grid,0,0,U(0,0),AllocatedLocalDomain)
```

Example – XML file



```
<?xml version="1.0" encoding="UTF-8" ?>
<simulation>
  <data>
    <object class="grid" id="heat" label="champ des temperatures">
      <variable  id="temperature" />
    </object>
  </data>
```




```
<htm auto-start="false" >
    <data-context ref="heat" context="readable"/>
    <loop id="timeLoop">
      <task id="bodyLoop">
        <task id="update" />
        <task id="swap">
          <data-context ref="heat" context="modified"/>
        </task>
        <point id="after_update"  />
      </task>
    </loop>
  </htm>
</simulation>
```

DL_POLY instrumentation

Integration in dl_poly 2.18

Very few and very localized File modification

1. Makefile – add a target to compile with EPSN (SEQ and PAR)

1. Add EPSNTools.F file – Fortran module

2. Add Instrumentation in dl_poly.f

Data management

- We define two RedSYM complex object
-

1. Atoms contains information on the set of atoms of the whole system. Here we consider the positions, the charge and the name of the atoms

1. Energy contains scalar information on temperature and energy.

```
<data>
  <object class="points" id="atoms">
    <variable id="xxx"/>
    <variable id="yyy"/>
    <variable id="zzz"/>
    <variable id="chge"/>
    <variable id="name"/>
  </object>
  <object class="scalar" id="Energy">
    <variable id="Total"/>
    <variable id="Temperature"/>
    <variable id="Potential"/>
  </object>
</data>
```

Xml data description

Data management - object description

Some text

```
    call redsym_points_create(points,"atoms"//CHAR(0),idnode, mxnode, dim)

    call RedSYM_Points_addVariable(points,'xxx'//CHAR(0),REDSYM_DOUBLE,1,var(1))
    call RedSYM_Points_addVariable(points,'yyy'//CHAR(0),REDSYM_DOUBLE,1,var(2))
    call RedSYM_Points_addVariable(points,'zzz'//CHAR(0),REDSYM_DOUBLE,1,var(3))
    call RedSYM_Points_addVariable(points,'chge'//CHAR(0), REDSYM_DOUBLE,1,var(4))
    call RedSYM_Points_addVariable(points,'name'//CHAR(0),REDSYM_CHAR,1,var(5))

    call RedSYM_Points_setcoordinatevariables(points,var(1))
!
    call EPSN_F90_addData(points)
```

Object description code

Data management - the mapping

Data are replicated on all the *p* processors, so we divide each 1d array of size N in bloc of size N/p.

ideb is the index of the first element on the current processor.

```
     call EPSN_Points_addRegion(points, idRegion, size_bloc, size_bloc, idnode)
!
    call RedSYM_Points_wrap(points, var(1), idRegion, xxx(ideb))
    call RedSYM_Points_wrap(points, var(2), idRegion, yyy(ideb))
    call RedSYM_Points_wrap(points, var(3), idRegion, zzz(ideb))
    call RedSYM_Points_wrap(points, var(4), idRegion, chge(ideb))

    call RedSYM_Points_wrapstrided(points,var(5),idRegion, atmnam(ideb),8,0)
```

Object description code

Name is an array of character of length 8. We consider here only the first character of the name, so we must use the RedSYM_Points_wrapstrided method with an offset of 0 an a stride of 8.

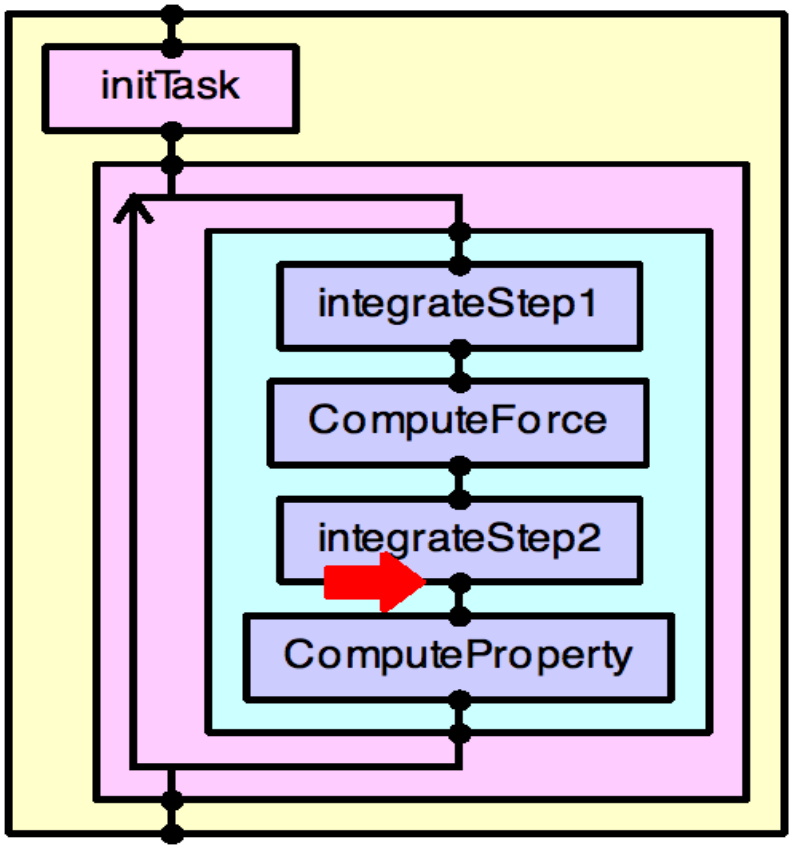
Data management - the scalar

- The scalar are defined on all the processors so we consider the key word REDSYM_REPLICATED.

```
    call redsym_parameter_create(energy,"Energy"//CHAR(0),idnode, mxnode, REDSYM_REPLICATED)

    call redsym_parameter_addvariable(energy,"Total"//CHAR(0), REDSYM_DOUBLE,1, var_sca(1))
    call redsym_parameter_addvariable(energy,"Temperature"//CHAR(0),REDSYM_DOUBLE,1,
                                                        var_sca(2))
    call redsym_parameter_addvariable(energy,"Potential"//CHAR(0), REDSYM_DOUBLE,1, var_sca(3))

    call redsym_parameter_wrap(energy, var_sca(1), stpval(1))
    call redsym_parameter_wrap(energy, var_sca(2), stpval(2))
    call redsym_parameter_wrap(energy, var_sca(3), stpval(3))
!
    call EPSN_F90_addData(energy)
```

Object description code

HTM structure

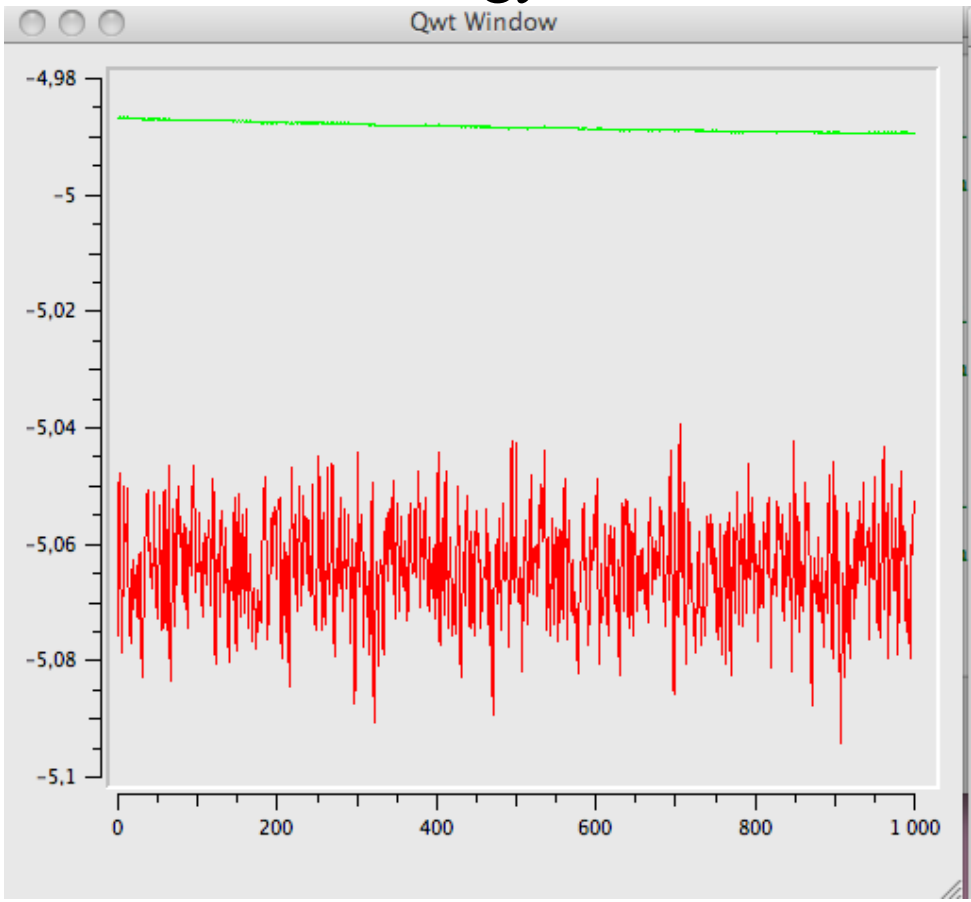



```
<htm auto-start="false">
    <data-context ref="atoms"  context="readable"/>
    <data-context ref="Energy" context="readable"/>
    <task id="initTask" />

    <loop id="timeLoop"  synchronize="no" >
      <task id="bodyLoop">

      <task id="integrateStep1">

                <data-context ref="atoms" context="protected"/>
      </task>
      <task id="ComputeForce">

      </task>
      <task id="integrateStep2">

                <data-context ref="atoms" context="modified"/>
      </task>
      <task id="ComputeProperty">

                <data-context ref="Energy" context="modified"/>
      </task>
      </task>
    </loop>

</htm>
```

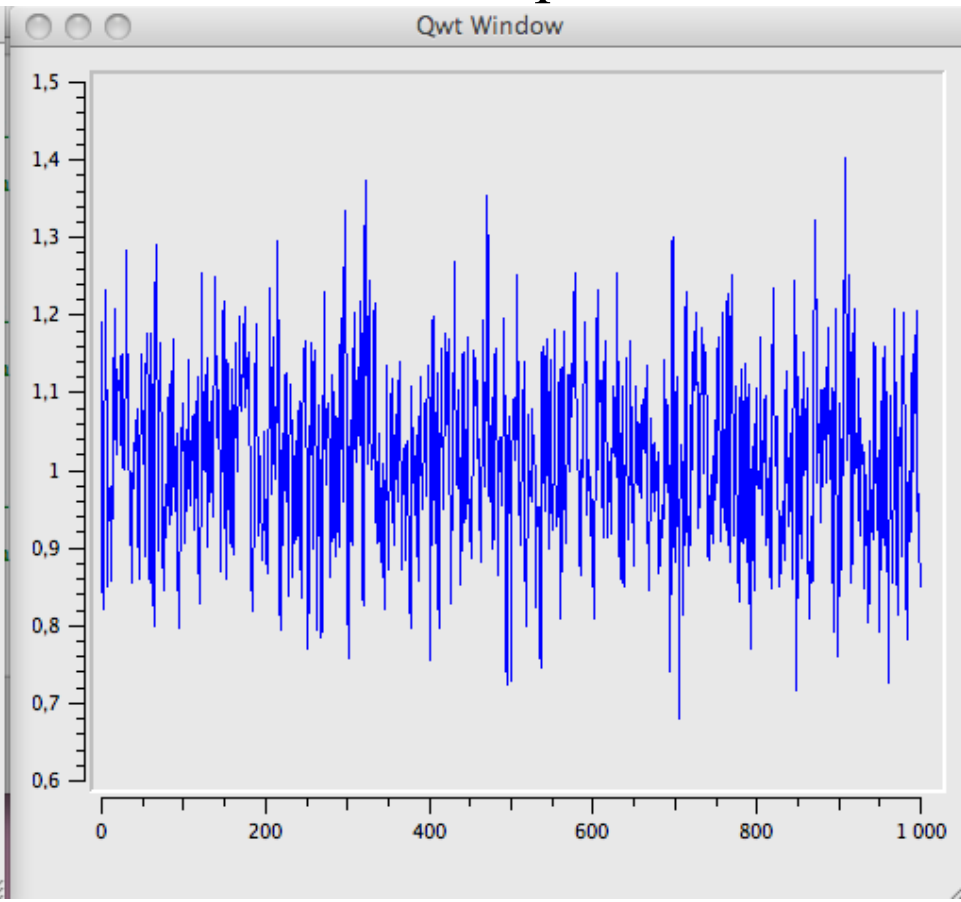
XML htm description

Scalar plot in Simone

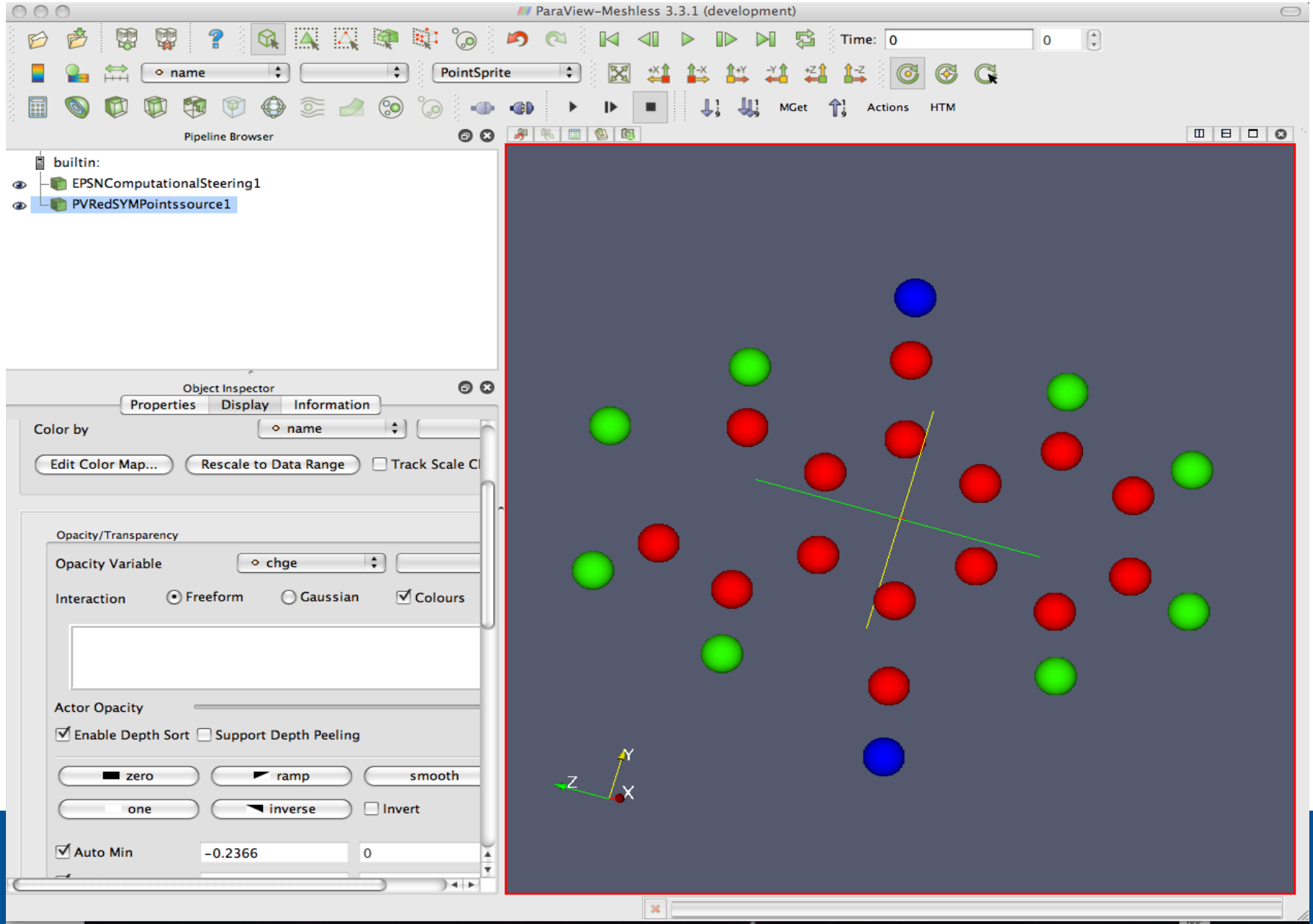
Energy

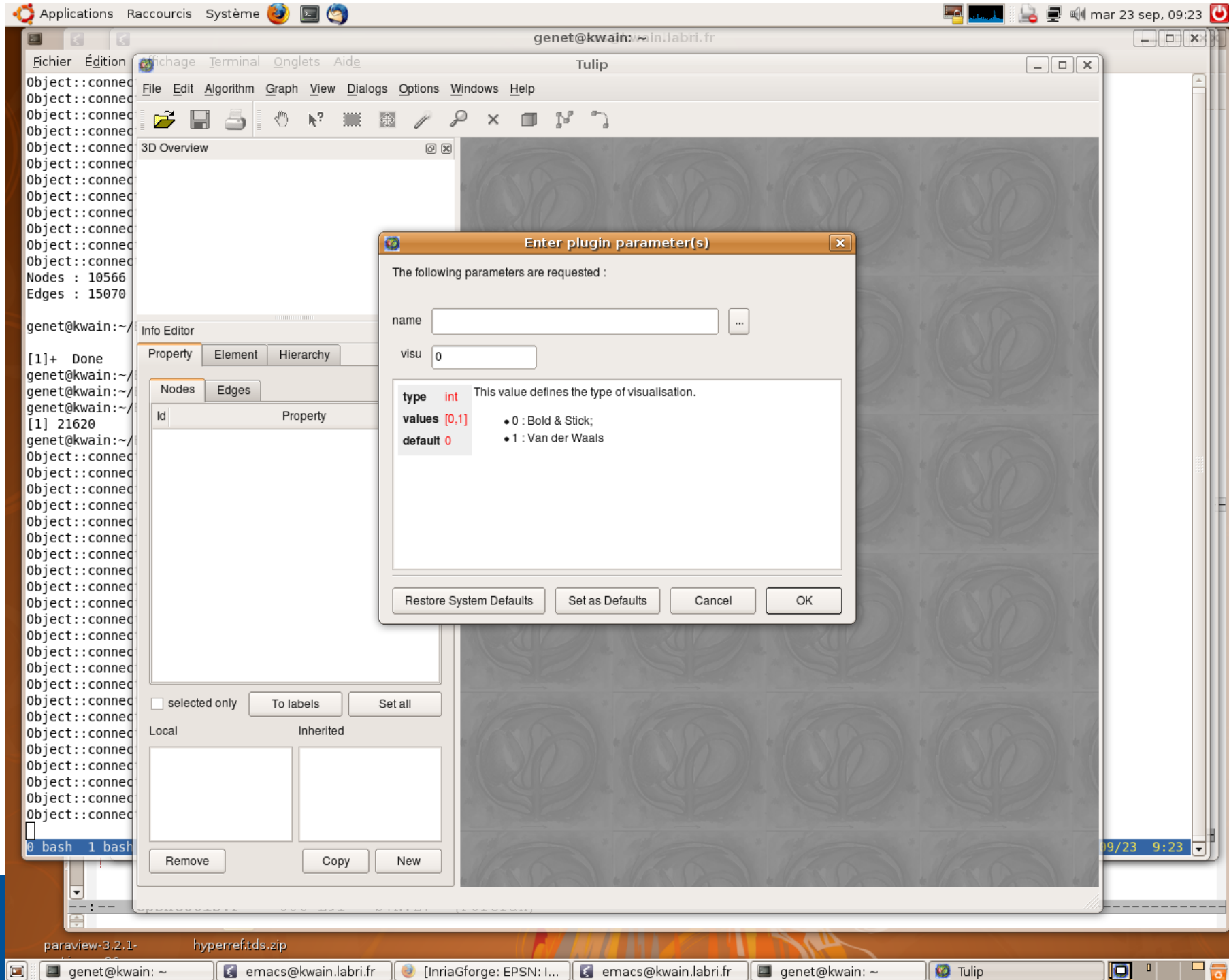


Temperature



Atoms in ParaView 3 (meshless)




Applications Raccourcis Système
mar 23 sep, 09:23
genet@kwain: ~
Fichier Édition Affichage Terminal Onglets Aide
Tulip
File Edit Algorithm Graph View Dialogs Options Windows Help
3D Overview
Info Editor
Property Element Hierarchy
Nodes Edges
Id Property
selected only
To labels
Set all
Local
Inherited
Remove
Copy
New
Enter plugin parameter(s)
The following parameters are requested :
name
visu
0
type int
values [0,1]
default 0
This value defines the type of visualisation.
• 0 : Bold & Stick;
• 1 : Van der Waals
Restore System Defaults
Set as Defaults
Cancel
OK
Nodes : 10566
Edges : 15070
paraview-3.2.1-
hyperref.tds.zip
genet@kwain: ~
emacs@kwain.labri.fr
[InriaGforge: EPSN: I...
emacs@kwain.labri.fr
genet@kwain: ~
Tulip