FIELD HANDLING AND VISUALIZATION WITH SALOME

Anthony Geay (EDF R&D)        Adrien Bruneton (CEA/DEN)

SALOME USER DAY | 26 NOV 2015
What’s the deal with field « manipulation »?

Code A
(e.g. thermics)

Code B
(e.g. CFD)

Physical problem
- heat flux
- pressure field
- …

MED files

Visualization

Paravis

MEDReader

Detached viz.

Field manipulation

MEDCoupling

Media

Detached viz.
AGENDA

1. Introduction
2. MED, MED or MED?
   - A short tour
   - Documentation
3. ALAMOS
   - A concrete illustration of MED capabilities
4. Pickling and multiprocessing with MEDCoupling
   - Or how to better couple my codes and use those 24 cores
5. MEDReader
   - From manipulation to visualization
6. Interactive field manipulation
7. Parallelism: MEDReader and benchmarking
   - Reminder about remote and parallel visualization
   - Performance benchmark of Paravis
8. Conclusion
Introduction

Manipulation and visualization
Why do we need …

- … field manipulation?
  - Extracting a time step
  - Extracting a subpart of the geometrical domain
  - Converting units
  - Find the max of a field over several timesteps
  - More complex operations
    - Integration of a field over a slice through the 3D domain
    - Interpolation/projection from one mesh to the other

- … visualization? (more obvious!)
  - For a mere control of what is being computed
  - As a decision tool for a design process
  - Reporting
  - …

All of this should work in a parallel context!
TYPICAL MEDCOUPLING USE CASES (1)

Remap or interpolate fields (with/without conservation) for code coupling

```python
srcField=... ; trgMesh=...
rem=MEDCouplingRemapper()
rem.prepare(srcField.getMesh(),trgMesh,"P0P0")
trgField=rem.transferField(srcField)
```

Fix, lighten, extend a MED file, convert from/to MED file

```python
mfd=MEDFileData(fileName)
del mfd.getFields()["Pressure"][::2]  # example : suppress 1 time step out of 2
mfd.write(fileName,2)
```
**Post-process a computation result**

- Ex: Extract the integral, max value of field across time on a precise subpart of it

```
res=[]
for flts in fmts:
    f=flts.getFieldAtLevel(ON CELLS,0)
    ids=f.getArray()[:,1].getIdInRange(0.,1e300)
    res.append((f[ids].integral(True), f[ids][:,0].getMaxValue())
```

**Pre-process from mesher to code**

- Ex: Renumbering, Partitioning

```
ren=RenumberFactory("BOOST")
a,b=mesh.computeNeighborsOfCells()
n2o,o2n=ren.renumber(a,b)
meshRenum=mesh[n2o]
```
Example: magnetic field visualization

Credits: C. Bourcier / F. Nunio (CEA)
MED, MED or MED?

A short tour
What do we understand under MED in SALOME?

Core structures
Arrays, meshes, … and algorithms
(libmedcoupling.so)

Projection
Interpolation and field projection
(remapper)

Parallelism
MPI based functionalities
(DEC – Data Exchange Channel)
(libParaMEDMEM.so)

Graphical user interface
Interactive field manipulation
(MED GUI client)

File I/O
I/O and advanced operations on MED files
(libmedloader.so)

MEDCoupling

MED Reader
ParaView plugin to visualize MED files
(libMEDReader.so)
An ongoing effort …

- Documentation has been rationalized and improved in SALOME 7.7
- And will keep on being improved in future releases!
- Notably, introduction of a
  - « Getting started » section
  - Frequently Asked Questions (FAQ) page
  - and inclusion of the tutorial used in the training sessions

The core elements of the API are:

- CommInterface, this is the wrapper around the MPI library, and an instance of this object is contained in each communicator.
- ParaMESH, the parallel instantiation of a MEDCoupling mesh
- ParaFIELD, the parallel instantiation of a MEDCoupling field
- MPIProcessorGroup (which inherits from the abstract ProcessorGroup), a group of processors

In an advanced usage, the topology of the nodes in the computation is accessed through the following:

- BlockTopology, specification of a topology based on the (structured) mesh. The mesh class exposes a method to return the block topology.
- ExplicitTopology (not fully supported yet and only used internally), specification of a user-defined topology.
- ComponentTopology, specification of a topology allowing the split of several field components

Data Exchange Channel - DEC

A Data Exchange Channel (DEC) allows the transfer and/or the interpolation (remapping) of fields from one communicator to another.
ALAMOS – An illustrative use case

Leveraging on MEDCoupling capabilities
ALAMOS, pre-processing for neutron transport codes

Data setting for neutron transport problems:
- Geometry (mostly planar)
  - MEDCoupling meshes
- Associated properties (typically *materials*)
  - MEDCoupling fields

Objectives

- Handling several refinement levels and a hierarchical design
  - E.g. a repeated pattern in a coarser grid
- Provide a simple and intuitive interface for simple cases
  - Quick geometry design
  - Field creation and assignation

Practically

- Uses MEDCoupling + MEDLoader, mostly through the Python API
- Relies heavily on intersection algorithms
Pickling and multiprocessing

New possibilities in MEDCoupling
Pickle, the Python serialization mechanism

DataArrays, Meshes, Fields of MEDCoupling are *pickelizable* since 7.6.0.

Implementation of *pickelization* is based on the same serialization mechanism used to exchange through MPI/CORBA channels

```python
import cPickle
st=cPickle.dumps(mesh,cPickle.HIGHEST_PROTOCOL)
assert(isinstance(st,str))
assert(cPickle.loads(st).isEqual(mesh,0.))
```
YACS usecase

- Exchange between YACS nodes:
  - meshes,
  - arrays
  - and fields

without extra lines of code, and running on different Python interpreters!

Example of mesh exchange:

- Description of mesh:
  - Unstructured mesh with name: "mesh"
  - Time attached to the mesh [unit]: 0 [s]
  - Mesh dimension: 2
  - Space dimension: 2
  - Info attached on space dimension: "X [m]" "Y [m]"
  - Number of nodes: 100
  - Number of cells: 81
  - Cell types present: NORM_QUAD4

Example of field exchange:

- Description of field:
  - FieldDouble with name: "ExampleOffield"
  - FieldDouble space discretization is: PO
  - FieldDouble time discretization is: One time label, time is defined by iteration=-1 order=-1 and time=0.
  - FieldDouble nature of field is: "NonNature"
  - FieldDouble default array has 1 components and 81 tuples

Mesh support information:

- Unstructured mesh with name: "mesh"
  - Time attached to the mesh [unit]: 0 [s]
  - Mesh dimension: 2
  - Space dimension: 2
  - Info attached on space dimension: "X [m]" "Y [m]"
  - Number of nodes: 100
  - Number of cells: 81
  - Cell types present: NORM_QUAD4
Multiprocessing capabilities

- Parallelize costly field operations in Python scripts using `multiprocessing` standard module.
- *Pickleization* of input objects is a (technical) prerequisite of multiprocessing.

```python
def work(meshPart):
    ...
    r=mr.MEDCouplingRemapper()
    def work(meshPart):
        ...
        r=mr.MEDCouplingRemapper()
        r.prepare(m,meshPart,"P0P0")
        ...
```

```python
slices=[GetSlice(slice(0,nbCells,1),nProc) for i in xrange(nProc)]
pool=mp.Pool()
asyncRes=pool.map_async(work,[m2[s] for s in slices])
subMatrices=asyncRes.get()
```
Coming after 7.7.0

- Pickelization of all complex objects in MEDCoupling
  - multi timesteps fields with profiles and joints
  - Etc ...

- Exchange memory to memory between
  - SMESH,
  - MED
  - and PARAVIS
  - modules *without* the file system bottleneck
MEDReader

Leveraging on MED I/O capabilities
**MEDReader** - a Paraview plugin hosted in Paravis

- New MEDReader engine powered by MEDCoupling
  - Harmonization with SALOME’s backbone
  - MEDReader engine is fully testable in python

---

MEDReader engine / MEDReader plugin serverside

<table>
<thead>
<tr>
<th>MED File</th>
<th>MEDCoupling</th>
<th>MED file Data Structure in memory</th>
<th>MEDCoupling</th>
<th>Arrays in VTK format</th>
<th>VtkMEDReader</th>
<th>VTK datasets Pipeline info</th>
</tr>
</thead>
</table>

---

SALOME User Day | 26 NOV 2015 | PAGE 20
Interactive field manipulation

A graphical user interface to MED capabilities
The MED graphical user interface

- Most common field manipulation operations **graphically available**
  - Field support change
  - Time step extraction … etc

- Naturally calls for an **easily accessible control visualization**
  - Being comfortable with ParaView / PARAVIS has an (assumed) cost

- Specifications have been established
  - **Minimal set of requirements**, with the assumed objective not to have this grows
    - One should invest in learning PARAVIS if required
  - Covered:
    - Mesh display
    - Scalar map
    - Isocontours, Slices
    - Vector fields, deflection shape
    - Point sprites.

- More about this in future versions!
Parallelism

MEDReader capabilities
Paravis is designed for *remote* visualization

- Take advantage of **memory space** on cluster (main RAM + graphical)
- Take advantage of parallelism to **reduce time to restitution**
PERFORMANCE OF PARAVIS

Sequential local v.s. remote perfs in 7.7.0

- Bench = time to:
  - Read + render tri3-mesh
  - Render node field in a generated MED file with corresponding size

Local config = Intel Xeon Ivy Bridge, 16 GB, GPU Nvidia K2000 (2 GB)
Remote config = Intel Xeon Haswell, 64 GB, GPU Nvidia K5200 (8 GB)

<table>
<thead>
<tr>
<th># cells (size on GPU)</th>
<th>Local</th>
<th>Remote (x1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 M (0.15 GB)</td>
<td>7 s</td>
<td>5 s</td>
</tr>
<tr>
<td>8 M (0.6 GB)</td>
<td>25 s</td>
<td>12 s</td>
</tr>
<tr>
<td>16 M (1.2 GB)</td>
<td>48 s</td>
<td>23 s</td>
</tr>
<tr>
<td>20 M (1.5 GB)</td>
<td>60 s</td>
<td>25 s</td>
</tr>
<tr>
<td>32 M (2.3 GB)</td>
<td>Crash</td>
<td>46 s</td>
</tr>
</tbody>
</table>

Green = good 3D interactivity, Red = bad 3D interactivity
The parallel mode of Paravis

An overview of the data flow:
Bench #1 - 256 Mcells, 102 Mnodes

- Multi geometric types (tetra4, pyra5, hexa8)
- Numbering OK per geometric type for computation
- MED file of size 34 GB (8GB mesh+26GB fields)
Bench #2 - 194Mcells, 102 Mnodes

- Single geometric type
- Numbering OK for computation
- MED file of size 8 GB (5.9GB mesh+2.1GB field)

Strong scaling on bench #2
Bench #3 - 100Mcells, 100 Mnodes

- Single geometric type
- **Badly numbered** for computation
- MED file of size 7GB (6.3GB mesh+0.7GB field)

**Consequence**

- // visualization performance is very sensitive to mesh numbering!
- Renumber and partitioner in MED module can be used for mesh renumbering
COMING SOON IN PARAVIS (AFTER 7.7.0)

Customizable GUI for connection to a graphic cluster

- Possibility to share Paravis graphic session
- Connection client to server
  - contrary to the current solution (server to client)
CONCLUSION

Field handling

- Enhance set of mesh operations offered by MEDCoupling (SMESH, mesh refinement, mobile mesh)
- In MEDCoupling still, enhance possible operations on mesh/field in an MPI context

Visualization

- PARAVIS is able to deal with big fields.
- Keep on improving the ease for users, fully taking advantage of remote visualization capabilities
- Improve 3D rendering delay
- Development of bricks based on MEDCoupling for in-situ and co-processing visualization
Thanks!
Focus on the reference mechanism
Multiprocessing capabilities

- Parallelize costly field operations in Python scripts using `multiprocessing` standard module.
- *Pickleization* of input objects is a (technical) prerequisite of `multiprocessing`.

```python
import multiprocessing as mp
import MEDCoupling as mc
m=MEDCouplingUMesh... ; m2=MEDCouplingUMesh... ; nbCells=...

def work(meshPart):
    ... # example of a costly field handling operation
    r=mr.MEDCouplingRemapper()
    r.prepare(m,meshPart,"POP0")
    ...

slices=[GetSlice(slice(0,nbCells,1),nProc) for i in xrange(nProc)]
pool=mp.Pool()
asyncRes=pool.map_async(work,[m2[s] for s in slices])
subMatrices=asyncRes.get()
```
EXAMPLE OF VISUALIZATION ON CLUSTER
Thanks!