ACROSS

HPC Big DAta ArtifiCial intelligence cross
Stack PlatfoRm TOwardS ExaScale

PART A: PARALLEL OUTPUT IN OPM
PART B: TOWARDS SIMULATION ON SEISMIC GRIDS

DAMARIS INTEGRATION & DUNE-ALUGRID INTEGRATION

JOSHUA BOWDEN | INRIA

ELYES AHMED | SINTEF DIGITAL

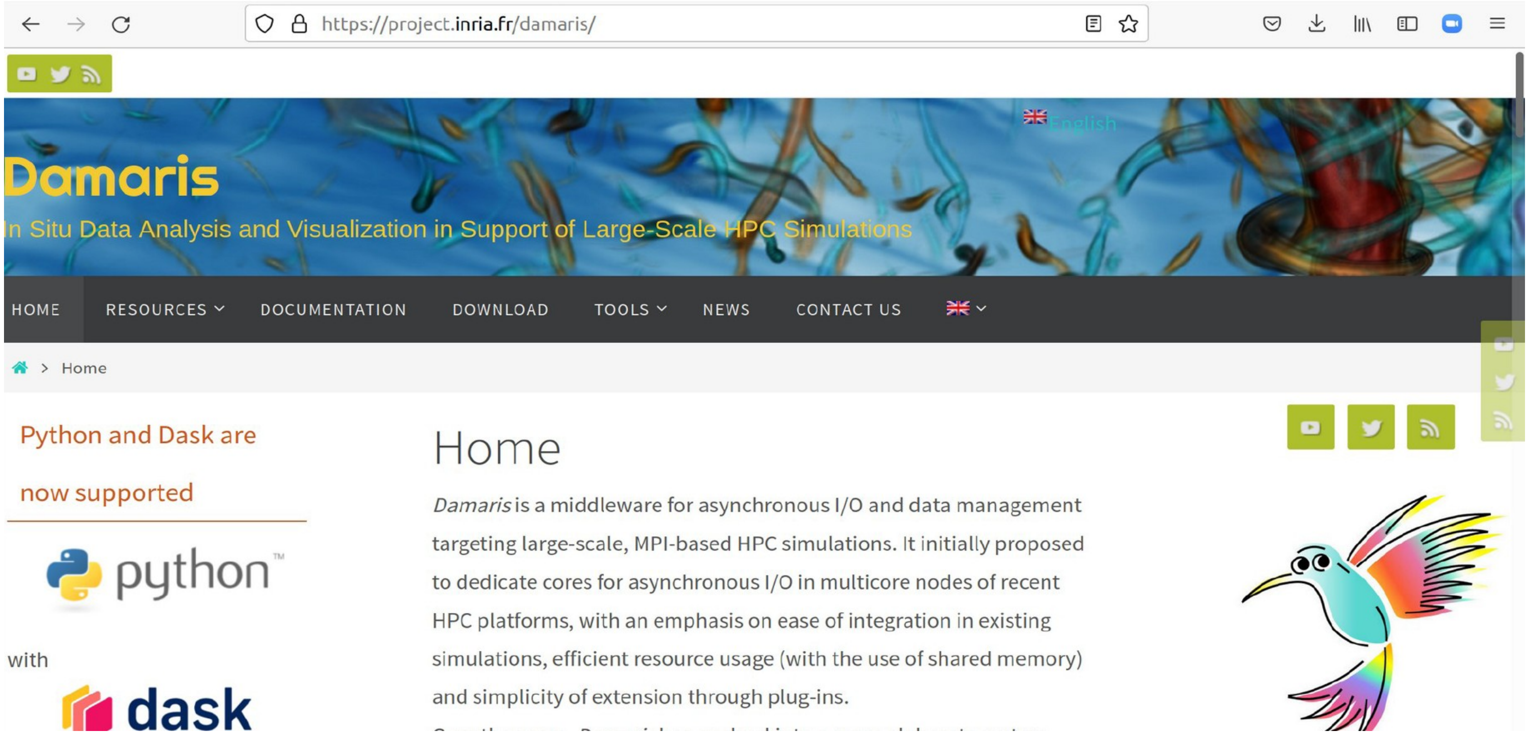
OPM-SUMMIT 2022

Trondheim | 30/08/2022

DAMARIS INTEGRATION

A QUICK WALKTHROUGH

The **Damaris** library is being used in the **EuroHPC** ACROSS project to improve the I/O performance of **OPM FLOW** which is one of the three pilot use cases within the project.



Damaris

In Situ Data Analysis and Visualization in Support of Large-Scale HPC Simulations

Python and Dask are now supported

with

Home

Damaris is a middleware for asynchronous I/O and data management targeting large-scale, MPI-based HPC simulations. It initially proposed to dedicate cores for asynchronous I/O in multicore nodes of recent HPC platforms, with an emphasis on ease of integration in existing simulations, efficient resource usage (with the use of shared memory) and simplicity of extension through plug-ins.

https://www.acrossproject.eu

https://project.inria.fr/damaris

- **DAMARIS** (DEDICATED ADAPTABLE MIDDLEWARE FOR APPLICATION RESOURCES INLINE STEERING) is a middleware for asynchronous I/O and data management targeting large-scale, MPI-based HPC simulations

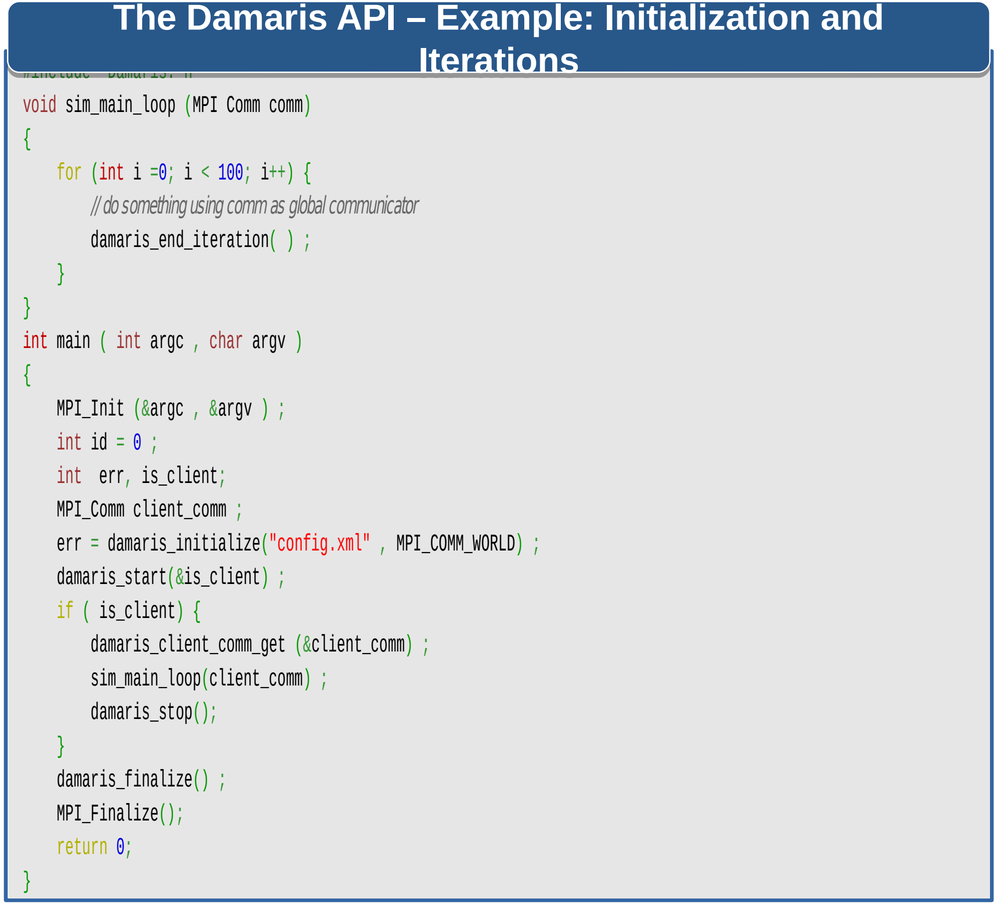
WHY DAMARIS?

- “In situ” data analysis and visualization by some dedicated cores/nodes of the simulation platform, in parallel with the simulation
- It is easy to integrate it with existing simulators
- It helps the simulator having a predictable run time and an improved scalability

INSTRUMENTING A SIMULATION IN OPM

- **Damaris** requires the simulation to be based on MPI.
- **XML** is a flexible and descriptive syntax to describe a problem and name data
- The **simulator** should use this **client_comm** and not **MPI_COMM_WORLD**, as global communicator
- Many parts of **OPM** Flow assumed that the **MPI_COM_WORLD** communicator (that includes all ranks) should be used.

  ==> Refactoring OPM-FLOW parallel communicators: a smaller communicator can then be dedicated to the simulation



The Damaris API – Example: Initialization and Iterations

```
#include "Damaris.h"
void sim_main_loop (MPI Comm comm)
{
    for (int i =0; i < 100; i++) {
        // do something using comm as global communicator
        damaris_end_iteration( ) ;
    }
}
int main ( int argc , char argv )
{
    MPI_Init (&argc , &argv ) ;
    int id = 0 ;
    int  err, is_client;
    MPI_Comm client_comm ;
    err = damaris_initialize("config.xml" , MPI_COMM_WORLD) ;
    damaris_start(&is_client) ;
    if ( is_client) {
        damaris_client_comm_get (&client_comm) ;
        sim_main_loop(client_comm) ;
        damaris_stop();
    }
    damaris_finalize() ;
    MPI_Finalize();
    return 0;
}
```

DAMARIS XML

The Damaris XML – Data management

```
<?xml version="1.0"?>
<simulation name="opm-flow" language="c"
        xmlns="http://damaris.gforge.inria.fr/damaris/model">
        <architecture>
                <domains count="1"/>
                <dedicated cores="1" nodes="0"/>
                <buffer name="buffer" size="67108864" />
                <placement />
                <queue  name="queue" size="300" />
        </architecture>
```

```
<data>
.
.
.
<variable name="PRESSURE" layout="zonal_layout_usmesh"
  type="scalar"  visualizable="false" unit="Pa"
centering="zonal"  store="_MYSTORE_OR_EMPTY_REGEX_" />
</data>
```

DAMARIS XML

The Damaris XML – Other fields

User/client specified <actions> and in-built post iteration I/O

```
 <actions>
   <event name="my_event" action="my_function" library="libsomething.so" scope="core"/>
</actions>
<storage>
    <store name="MyStore" type="HDF5">
        <option key="FileMode">FilePerCore</option>  (or Collective)
        <option key="XDMFMode">NoIteration</option>
        <option key="FilesPath"></option>
    </store>
</storage>
<visit>
    <path>/path/to/visit2.13.0/src</path>
    <options> -debug 5 </options>
</visit>
<paraview>
    <script>"/path/to/the/first/script1.py"</script>
    <script>"/path/to/the/second/script2.py"</script>
</paraview>
<log FileName="log/2dmesh" RotationSize="5" LogFormat="[%TimeStamp%]: %Message%"  LogLevel="info" Flush="True" />
```

STATUS

- Current branch: https://github.com/ElyesAhmed/opm-simulators/tree/damaris_integ_v3
- The flow code has the inbuilt XML generation for Damaris in: opm/simulators/flow/Main.hpp
- **Collective I/O** mode => all simulation results are written into one single file for each iteration using **Parallel HDF5**.
- With files named *simulation_ItXX.h5*.
- **File-Per-Dedicated-Core** mode => all the simulation results in each node are aggregated by dedicated cores and stored asynchronously at the end of each iteration
- With files named *simulation_ItXX_PrYY.h5*.

```
void writeOutput(bool isSubStep)
{
.
.
#ifdef HAVE_DAMARIS
.
.
.
if (! isSubStep) {
        data::Solution localCellData = {};
        this->eclOutputModule_->assignToSolution(localCellData);
        // now to find the field data
        if (this->eclOutputModule_->getPRESSURE_ptr() != nullptr) {
          damaris_write("PRESSURE",  (void *) this->eclOutputModule_->getPRESSURE_ptr());
        }
        damaris_end_iteration( ) ;
    }
#endif
.
.
.
}
```

ON-GOING

- Integrate Damaris int the build-system
- Add a comman Line Option like –damaris-output=true
- Merge it to master

FUTURE

- How to make this useful?
- Which variables should be present?
- How do you want users to specify the number of Damaris cores or nodes (new keyword, command line,...)?
- How do you want to specify the size of the shared memory buffer?

DUNE-ALUGRID

A QUICK WALKTHROUGH

GOAL: ADAPTIVE SIMULATION ON SEISMIC GRIDS

- Integration of Dune-ALUGrid module into OPM-FLOW
- ALUGrid=”Adaptive Load balanced Unstructered Grid”
- Dune-ALUGrid has been used with thousand of MPI ranks
- For dynamic load balancing, Space Filling Curve (SFC) approaches are used
- The Dune-ALUGrid module depends only on Dune-Grid
- Parallel grids

- { **eclalugridvanguard.hh, alucartesianindexMapper.hh**} added to opm-simulators/ebos.
- Alugrid => **Dune::ALUGrid< dimgrid, dimworld, eltype, refinetype, communicator >**
- **GRID** being the **Dune::ALUGrid** while **EquilGrid** is the **Dune::CpGrid**.
- **ordering_** is the vector mapping Grid Idx To EquilGrid Idx
- if SFC_ORDERING=1, **ordering_** is different from **globalCell** numbering.

```
void createGrids_()
{
// As for CpGrid we use separate grid objects: one for the calculation of the initial condition
 // via EQUIL and one for the actual simulation via Grid.
.
.
this->equilGrid_ = std::make_unique<Dune::CpGrid>(EclGenericVanguard::comm());
.
.
cartesianCellId_ = this->equilGrid_->globalCell();

for (unsigned i = 0; i < dimension; ++i)
      cartesianDimension_[i] = this->equilGrid_->logicalCartesianSize()[i];

equilCartesianIndexMapper_ = std::make_unique<EquilCartesianIndexMapper>(*equilGrid_);

/////
 // create the simulation grid
/////
factory_ = std::make_unique<Dune::FromToGridFactory<Grid>>();
 grid_ = factory_->convert(*equilGrid_, cartesianCellId_, ordering_);
.
.
.
cartesianIndexMapper_ = std::make_unique<CartesianIndexMapper>(*grid_, cartesianDimension_, cartesianCellId_);
.
.
// update GridvVew, CellProperties…..
.
.
}
```

- Mots work is how to use Dune grid interface directly or the vanguard instead of the helpers.
- Deal with instantiations
- In many places OPM assumes implicitly a certain order...
- Or by assuming EQUILGRID=GRID

```
template class EclGenericProblem<Dune::GridView<Dune::ALU3dLeafGridViewTraits<ALUGrid3CN,
Dune::PartitionIteratorType(4)>>,
                                 BlackOilFluidSystem<double,BlackOilDefaultIndexTraits>,
                                 double>;
```

```
template<class Grid, class EquilGrid, class GridView, class ElementMapper, class Scalar>
data::Solution EclGenericWriter<Grid,EquilGrid,GridView,ElementMapper,Scalar>::

computeTrans_(const std::unordered_map<int,int>& cartesianToActive, const std::vector<int>& map)
{
```

STATUS & ON-GOING

- See **PR#3972** for the merged work (work initiated by Tor-Harald Sandve).
- Tested so far only for equally distant grids (fine for seismic and not fully unstructured grids)
- Next step is to work on the cell orders (SFC)
- Should we allow for different cell orders in OPM?
- Use adaptive features of ALUGRID.

Images sources: Shutterstock.com, Pexels.com, Unsplash.com

ACROSS

HPC Big DAta ArtifiCial intelligence cross
Stack PlatfoRm TOwardS ExaScale

CONTACT

ELYES AHMED

ELYES.AHMED@SINTEF.NO

SINTEF DIGITAL-OSLO

Phone +47 94 726 120

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955648. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Italy, France, Czech Republic, United Kingdom, Greece, Netherlands, Germany, Norway.