Runtime Data Analysis for CSE Applications

Alvaro L. G. A. Coutinho[1] Marta Mattoso[2] José Camata[3]
Vitor Silva[4] Linda Gesenhues[1] Renan Souza[2,5]

[1]High Performance Computing Center, COPPE/Federal University of Rio de Janeiro
[2]Computer Science, COPPE/Federal University of Rio de Janeiro
[3]Computer Science, Federal University of Juiz de Fora
[4]Dell EMC Brazil R&D
[5]IBM R&D Center Brazil

SIAM CSE
Spokane, WA, Feb 25 – Mar 1, 2019

Outline

Motivation: Simulation of sediment deposition

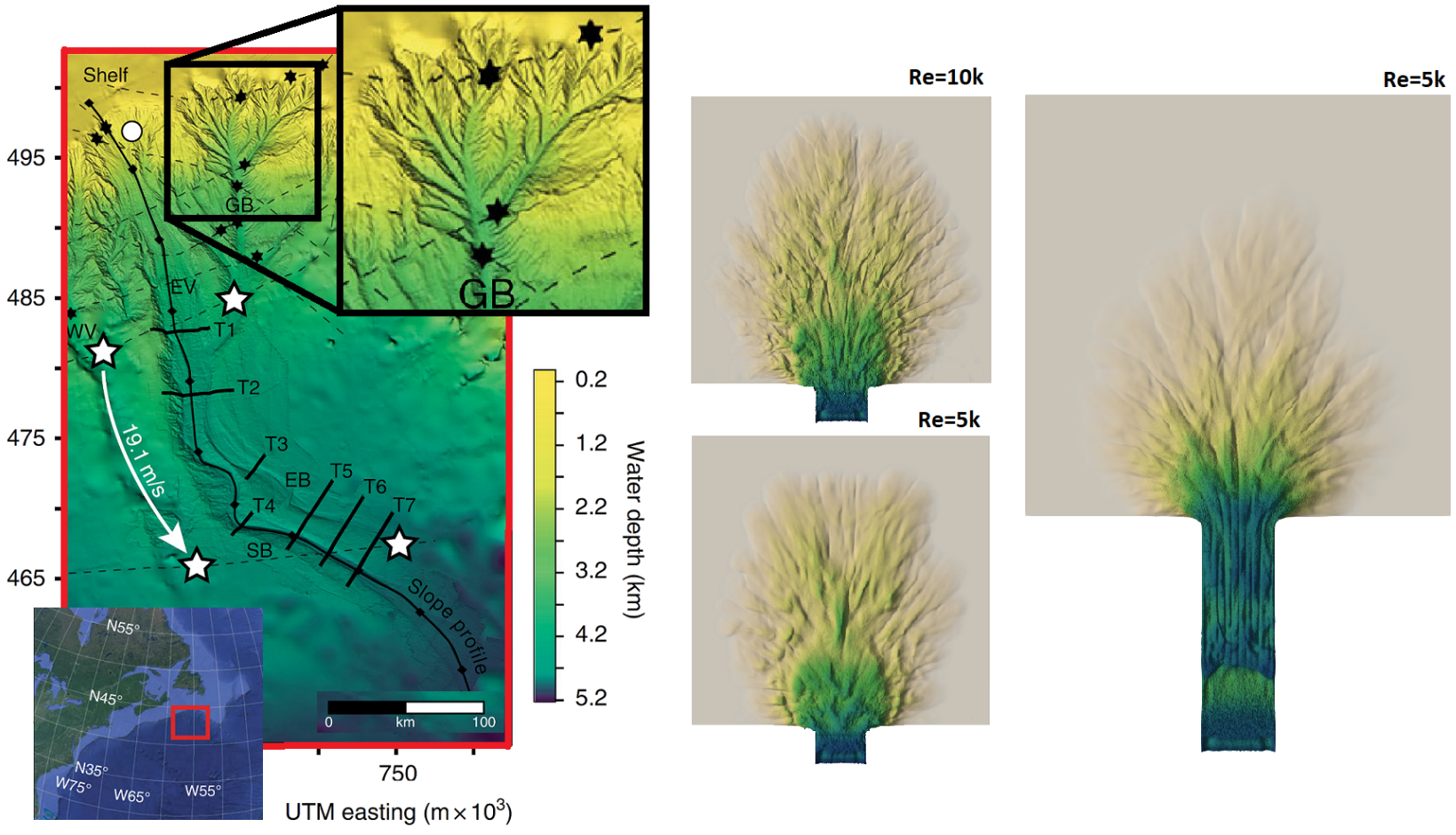


Figure: **Left**: Reconstruction of the 1929 Giant turbidity current deposits; **Top insect**: sediment deposit detail[1]; **Right**: Sediment deposits from a turbidity current simulation at different Re (COPPE, 2017)

[1] C.J. Stevenson, et al, Nature Communications (9): 2616 (2018).

Why such simulations are complex?

Large scale simulations:

1. **Large in size** $\Rightarrow$ Unstructured grids with $10^9 - 10^{12}$ elements;
2. **Large in coupling** $\Rightarrow$ multiphysics/multiscale problems;
3. **Large in physical parameters** $\Rightarrow$ different viscosities, densities, etc.
4. **Large in control parameters** $\Rightarrow$ solver options, tolerances, AMR/C, etc...
5. **Large in complexity** $\Rightarrow$ several softwares and human intervention

Several runs are often necessary:

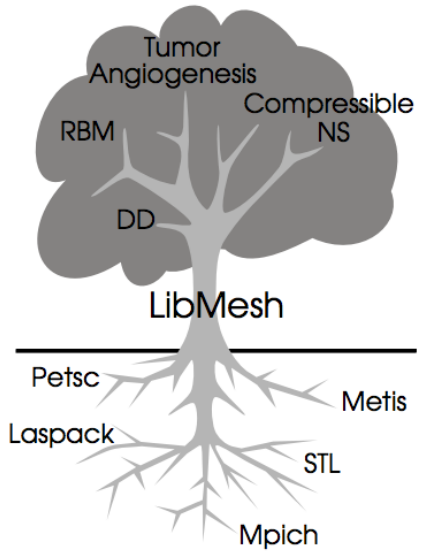
- **Sampling stochastic space** $\Rightarrow$ Uncertainty quantification
- **Parameter sweep** $\Rightarrow$ Many Task Computing
- **Reduced Order Modeling** $\Rightarrow$ Generating snapshots from high fidelity simulations
- **Machine Learning** $\Rightarrow$ Generating training sets

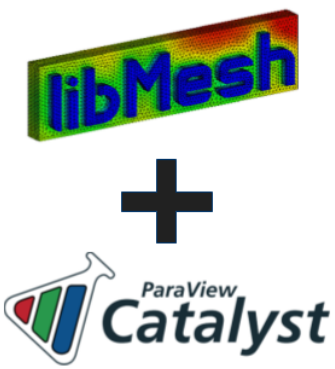
CSE Software

- Translate complex mathematical models into predictive tools
- Usually coded in Fortran, C/C++, Python, Java or a mix
- Often invokes components of CSE frameworks and libraries
- Components are invoked to provide for:
  - support for PDE discretization methods like libMesh, FEniCS, MOOSE, deal.II, GRINS, OpenFOAM, PetIGA;
  - mesh generation, like Gmsh;
  - building blocks for solving numerical problems with parallel computations, like PETSc, LAPACK, SLEPc;
  - visualizations, like ParaView, VisIt;
    - in-situ, like ParaView Catalyst, SENSEI
  - I/O data management, like ADIOS.
- Modification of parameters at runtime allowed in several of these, e.g., PETSc/SAWs
- **Log files have to be manually inspected or relevant data registered and shown in a browser; no query and provenance support at runtime**

libMesh[1]: a framework for finite element analysis

- Open-source library with parallel adaptive mesh refinement and coarsening (AMR/C) support
  - AMR/C is an optimal strategy for large-scale simulations
  - libMesh supports h, p and $h-p$ adaptive strategies
- Integration between **libMesh** and **Paraview Catalyst** is provided by an `adaptor` – implemented in one of the Paraview Catalyst APIs
  - We map mesh, velocity, pressure, sediment appearance, etc, from our solver to VTK's data model
  - Usage: output .png files every X time steps - write data every kX times steps, with $k \gg 1$.





[1] http://libmesh.github.io/index.html

Data Analysis

Data-Analysis can help in:

- Data Monitoring
  - Track parameters and relate huge numbers of raw data files
  - Contributes to computational experiments reproducibility
- Data Steering
  - Queries can help users to steer their simulations
    - Changing parameters in runtime
    - Extracting relevant subsets of raw data associating them to QoIs
  - Runtime query relating raw data from different files, provenance data, and performance execution data is challenging
    - Access to raw data files while they are generated
    - Parse raw data file to find relevant data
- Current solutions are offline: our in–transit solution is a step beyond[1,2]

---

[1] R. Souza et al. Data reduction in scientific workflows using provenance monitoring and user steering. Future Generation Computer Systems (2017).

[2] V. Silva et al. Raw data queries during data-intensive parallel workflow execution. Future Generation Computer Systems 75:402–422 (2017).

Data-Analysis and Data Provenance

Enhancing In–Transit Data Analysis for answering provenance queries involving user steering actions:

- **Who** adapted parameters $P1$, $P2$, $P3 \cdots$ in a data transformation at time T
- **What** were the values for $P1$, $P2$, $P3$ before and after T?
- **When** did a runtime tuning took place?
- **How** are the QoIs when mesh adaptation happened?
- **Why** did the user decide for a parameter tuning?

What is Data Provenance?

- **Data provenance** refers to records of the inputs, entities, systems, and processes that influence data of interest, providing a historical record of the data and its origins.
- Current provenance capture approaches for CSE applications present a high overhead and no runtime query support.

DfAnalyzer: a dataflow tool for online analysis

- In highly complex simulations, data is efficiently managed in memory and stored in thousands of *isolated* files (HDF5, XDMF, viz, graphs)
- These data have to be related to foster data analyses and visualization at runtime and after the simulation.
- Relating data after the simulation is **not** an option.

- **Inserting calls** on source code (like using in-situ viz tools)
- **Monitoring, debugging, dataflow analysis** by providing:
  - Provenance management
  - In-transit data analysis
  - Scientific data extraction
- **Small time overhead** in large-scale parallel executions

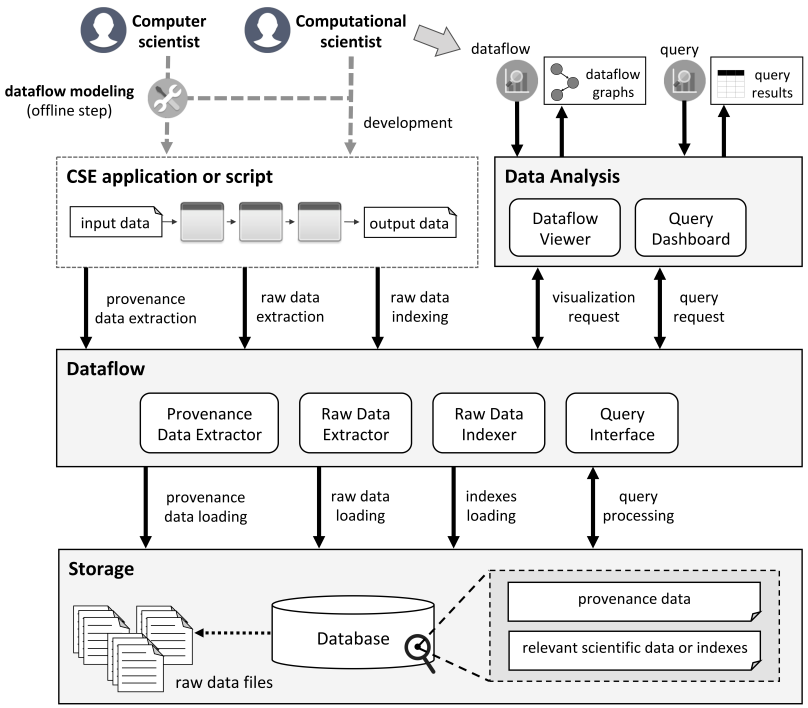


Figure: DfAnalyzer scheme.

Comparison of Existing Provenance Solutions

Features	Provenance Approaches		
	noWorkflow	**PROV-Template**	**DfAnalyzer**
Requires Code Adaptation	No	No	Yes
Deployment	Bindings with mapping code	Bindings with mapping code	Insertion of library calls
Provenance Capture	System-call trace analysis	Instruction-level dynamic instrumentation	Compile-time static instrumentation
Provenance Granularity	Function and file level	Function-level	Function and file
Raw Data Extraction	Not available	Not available	Available
Provenance Data Analysis	Prolog and SQL (w/o file content)	SPARQL	SQL+GUI
HPC Support	No	No	Yes

Table: Comparing Provenance Solutions[1,2,3]

[1] noWorkflow: J.F. Pimentel, et al. noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts. Proceedings of the VLDB Endowment, 10(12):1841–1844, 2017

[2] PROV-Template: L. Moreau, et al. A Templating System to Generate Provenance. IEEE Transactions on Software Engineering, 44(2): 103–121, 2018

[3] DfAnalyzer: V. Silva, et al, Capturing Provenance for Runtime Data Analysis in Computational Science and Engineering Applications, ProvenanceWeek, 9–13 July 2018, London, UK

FEniCS Code for Cahn–Hilliard with DfAnalyzer calls[1]

```
dataflow_tag = "fenics-df"
t1 = Task(1, dataflow_tag, "MeshCreation")
t1.add_dataset(DataSet("iMeshCreation", [Element([96, 96])]))
# Create mesh
mesh = UnitSquareMesh(96, 96)
t1.add_dataset(DataSet("oMeshCreation",
                       [Element([mesh.num_vertices(), mesh.num_cells()])]))
t1.end()

t2 = Task(2, dataflow_tag, "FunctionSpace", dependency=t1)
t2.add_dataset(DataSet("iFunctionSpace", [Element(["Lagrange", 1])]))
# Define function spaces
V = FiniteElement("Lagrange", mesh.ufl_cell(), 1)
ME = FunctionSpace(mesh, V*V)
t2.add_dataset(DataSet("oFunctionSpace", [Element([ME.dim()])]))
t2.end()

# parts of code were omitted
# (...)

t3 = Task(3, dataflow_tag, "NewtonSolver", dependency=t2)
t3.add_dataset(DataSet("iNewtonSolver",
                       [Element(["lu", "incremental", 1e-6])]))
# Define Newton solver
solver = NewtonSolver()
solver.parameters["linear_solver"]       = "gmres"
solver.parameters["convergence_criterion"] = "incremental"
solver.parameters["relative_tolerance"]    = 1e-6
t3.add_dataset(DataSet("oNewtonSolver",
                       [Element(["gmres", "incremental", 1e-6])]))
t3.end()
# continue in next frame
```

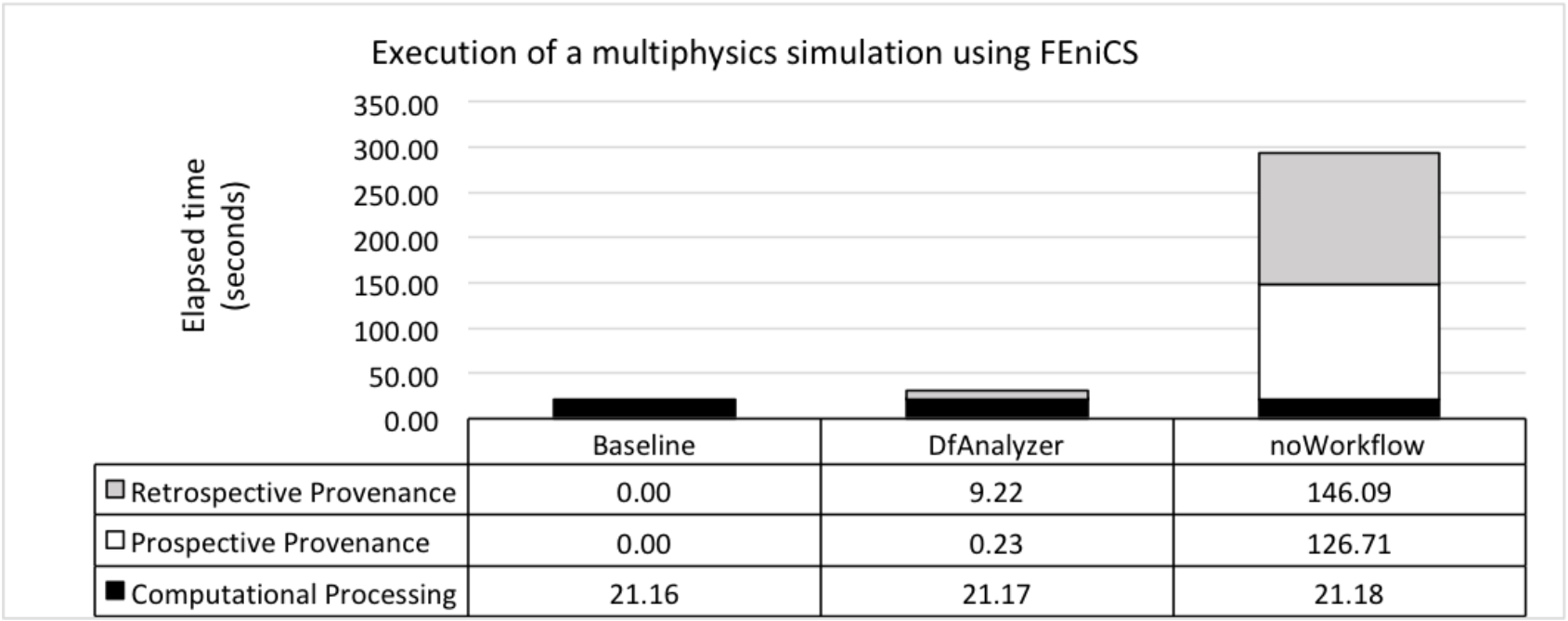
```
# Output file
file = File("output.pvd", "compressed")

# Step in time
t = 0.0; T = 50*dt; i =0
prev = t3
while (t < T):
  t += dt; i += 1
  current = Task(int(t3._id)+i ,dataflow_tag,"TimeStep", dependency=prev)
  current.add_dataset(DataSet("iTimeStep", [Element([t,dt])]))
  # Solver execution
  u0.vector()[:] = u.vector()
  iter_count, converged_flag = solver.solve(problem, u.vector())
  current.add_dataset(DataSet("oTimeStep",
                [Element([converged_flag,iter_count,solver.residual()])]))
  current.end()

  twrite = Task(int(current._id)+1, dataflow_tag, "Visualization"+iter_count,
                dependency=current)
  twrite.add_dataset(DataSet("iVisualization", [Element(["output.pvd"])]))
  # Visualization
  file << (u.split()[0], t)
  # Raw data extraction
  extracted_data = Extractor(ExtractorCartridge.PROGRAM, "output.pvd")
  twrite.add_dataset(DataSet("oVisualization", [Element(extracted_data[i-1])]))
  twrite.end()
```

Labels:
Black → Python native code
Red → FEniCS invocation
Green → DfAnalyzer invocation
Purple → VTK invocation

Figure: Code adapted from FEniCS manual.

[1] https://www.dropbox.com/s/ibf9ud311afjqvx/prototype_multi-physics.py?dl=0

Execution Times for FEniCS Code with Provenance



	Baseline	DfAnalyzer	noWorkflow
Retrospective Provenance	0.00	9.22	146.09
Prospective Provenance	0.00	0.23	126.71
Computational Processing	21.16	21.17	21.18

Figure: Execution in a single node in serial mode; Prospective Provenance acts before execution; Retrospective Provenance acts during execution (at every time step)

Turbidity Current Simulation with `libMesh`

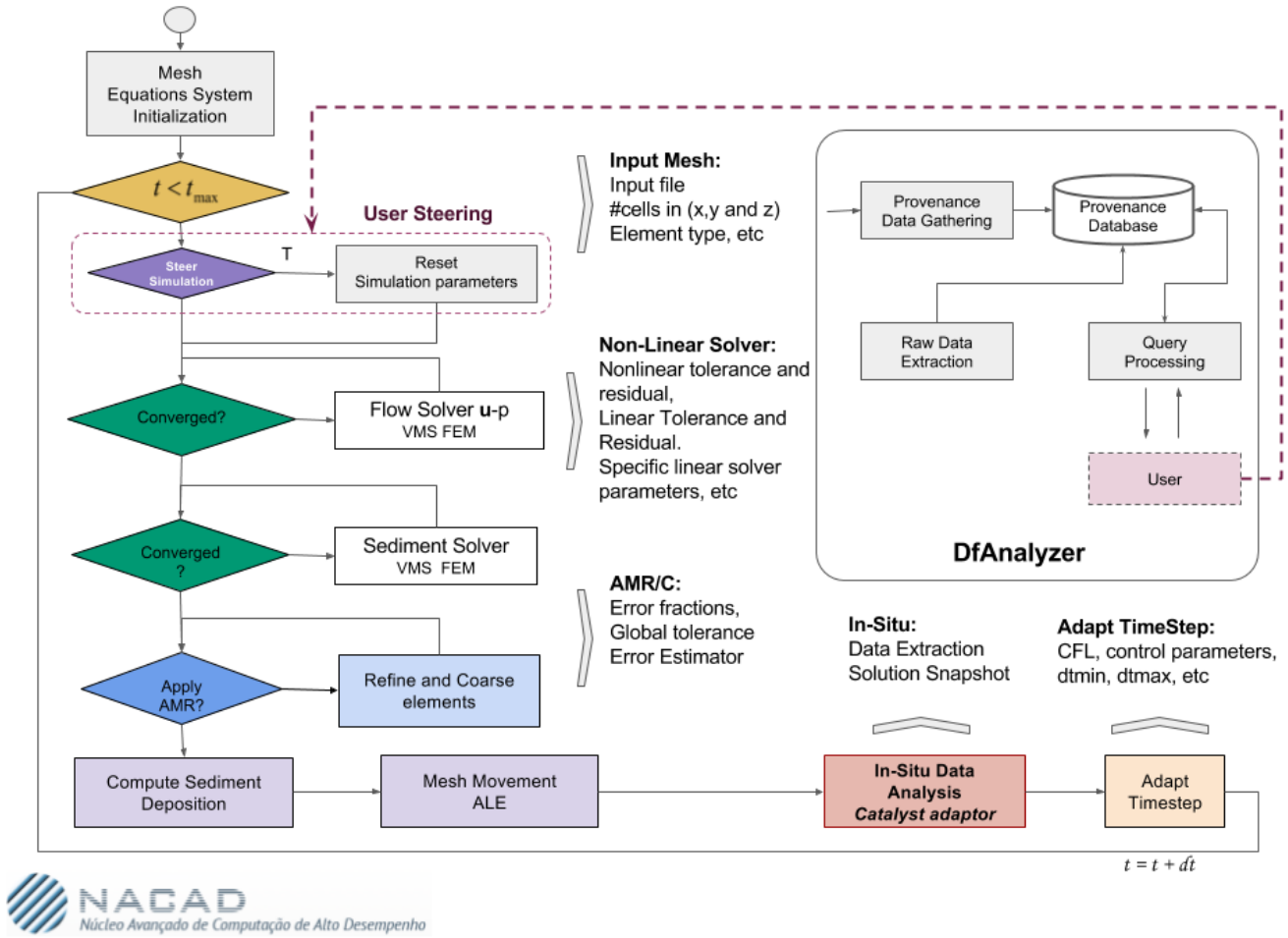


Figure: `libMesh` with In–Situ Visualization and In-Transit Data Analysis

[1] J. J. Camata, et al, In situ visualization and data analysis for turbidity currents simulation, Comput. Geosci., 110:23–31, 2018

Monodisperse current

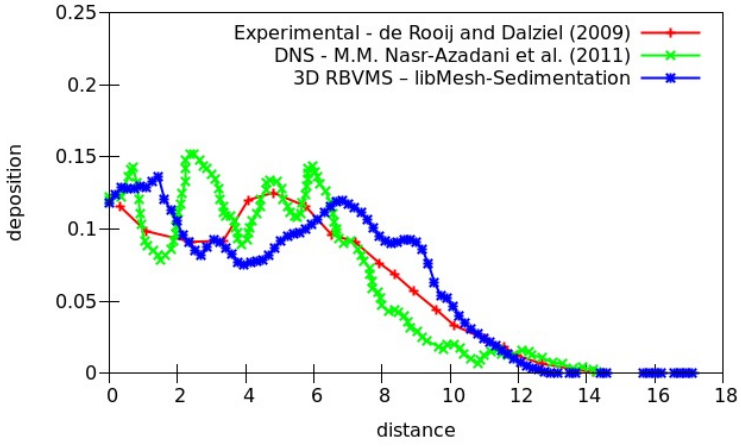


Figure: In-Situ data extraction - deposition plotted over line filter

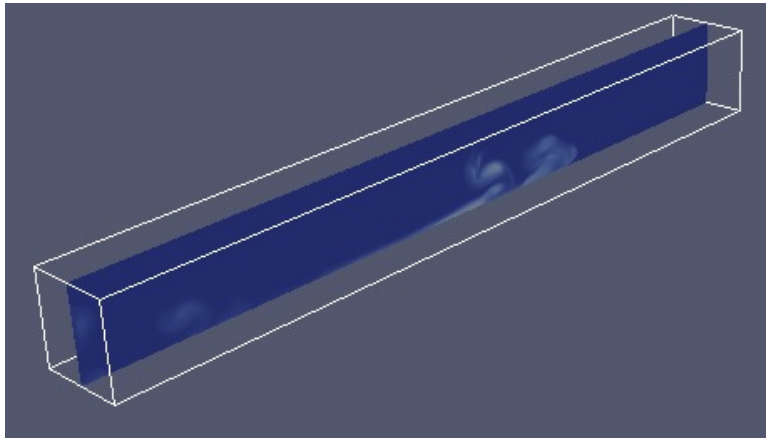


Figure: Sediment concentration profiles at $t = 20$

- Structured mesh with a 0.1 grid spacing.
- Two uniform refinements are applied initially: 4.6M HEX8.
- Kelly's error estimator for $\mathbf{u}$ and c
- Parallel by Block-Jacobi + GMRES(35) with ILU(0); tol 10^{-6}.
- Nonlinear tolerance 10^{-3} and $\Delta t = 0.005$.
- XDMF/HDF5 raw data files are written every 50 time steps.

Performance Analysis

Table: Elapsed time for different simulation stages; execution on 480 MPI processes

Time Contribution	**CPU Time (s)**	**Cost/Call**	**%Cost**
Flow Solver	16,203.71	0.87	32.67%
AMR/C	10,268.32	17.11	20.70%
Sediment Solver	2,797.36	0.15	5.64 %
XDMF/HDF5 Writer	453.96	4.93	0.92%
In Situ Viz + Data Extraction	3,137.24	33.73	6.33%
Provenance (DfAnalyzer)	38.47	0.01	0.08%
Others (libMesh)	16,598.00	–	33.67%
Total	62,171.00		

Query: Tracking sediment deposition

- DfAnalyzer registers deposition along time at predefined locations and pointers to viz files.
- We can query online with a negligible time (< 500 ms).

time	amount of deposit	visualization
12.500	0.000124443	visualization_12.500.png
13.000	0.100216556	visualization_13.000.png
13.500	1.460732820	visualization_13.500.png
14.000	3.586770000	visualization_14.000.png
14.500	6.152664400	visualization_14.500.png
...	...	...

(a)

time	amount of deposit	visualization
12.500	9.77670000E-44	visualization_12.500.png
13.000	2.30743685E-39	visualization_13.000.png
13.500	2.55823726E-37	visualization_13.500.png
14.000	2.80856500E-36	visualization_14.000.png
14.500	9.45688945E-34	visualization_14.500.png
...	...	...

(b)

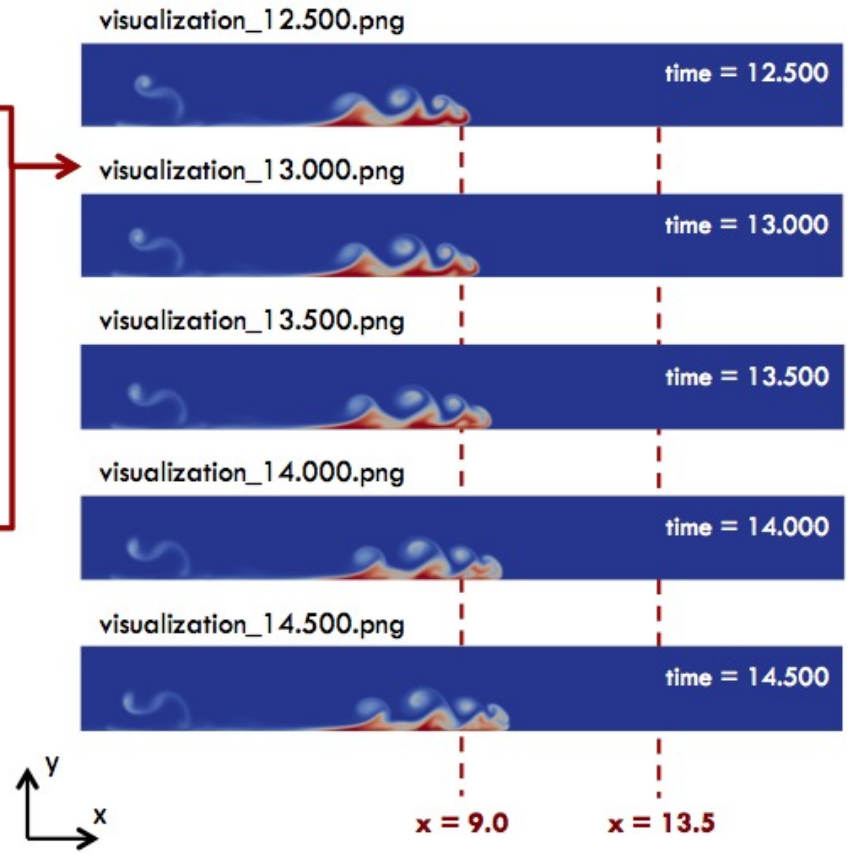


Figure: Sediment deposition monitoring at five time instants at $x = 9.0$ (a) and $x = 13.5$ (b) combining data with in-situ visual information

Experimental Channel

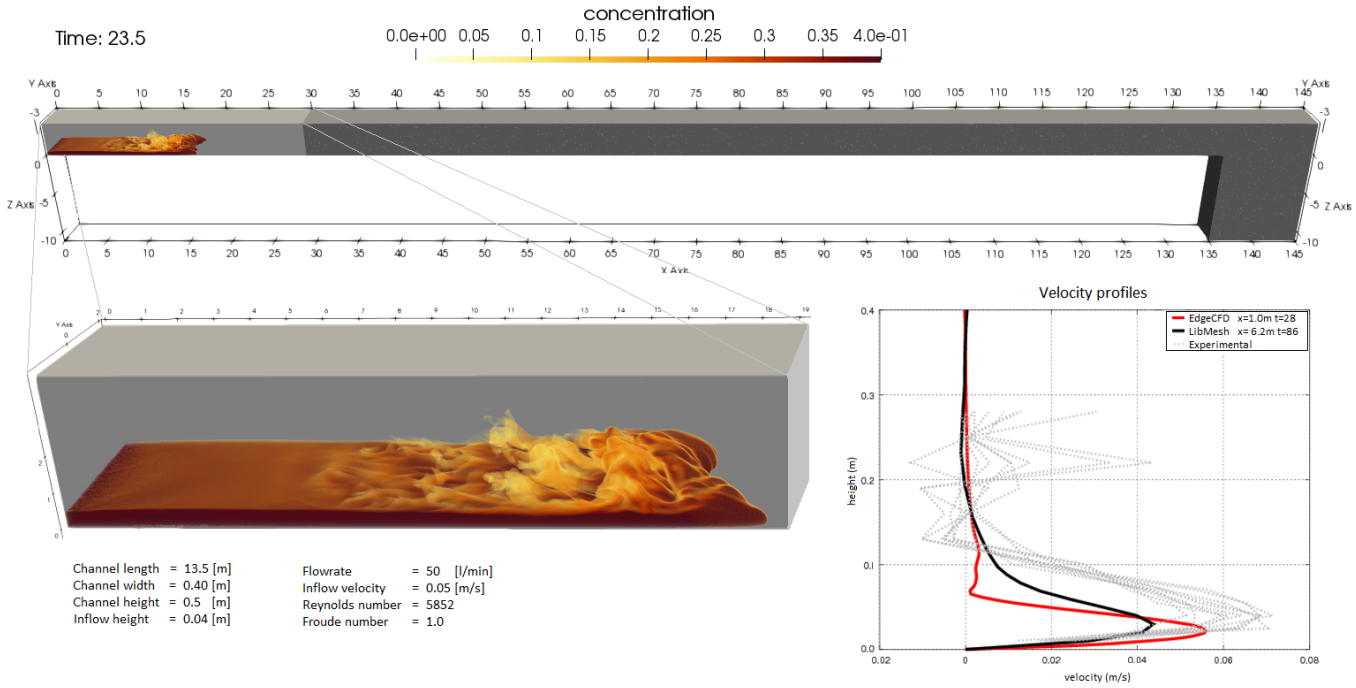


Figure: **Top**: Channel domain; **Bottom Left**: Sediment concentration and physical data; **Bottom Right**: Measurements, EdgeCFD and `libMesh` results

User Steering in libMesh

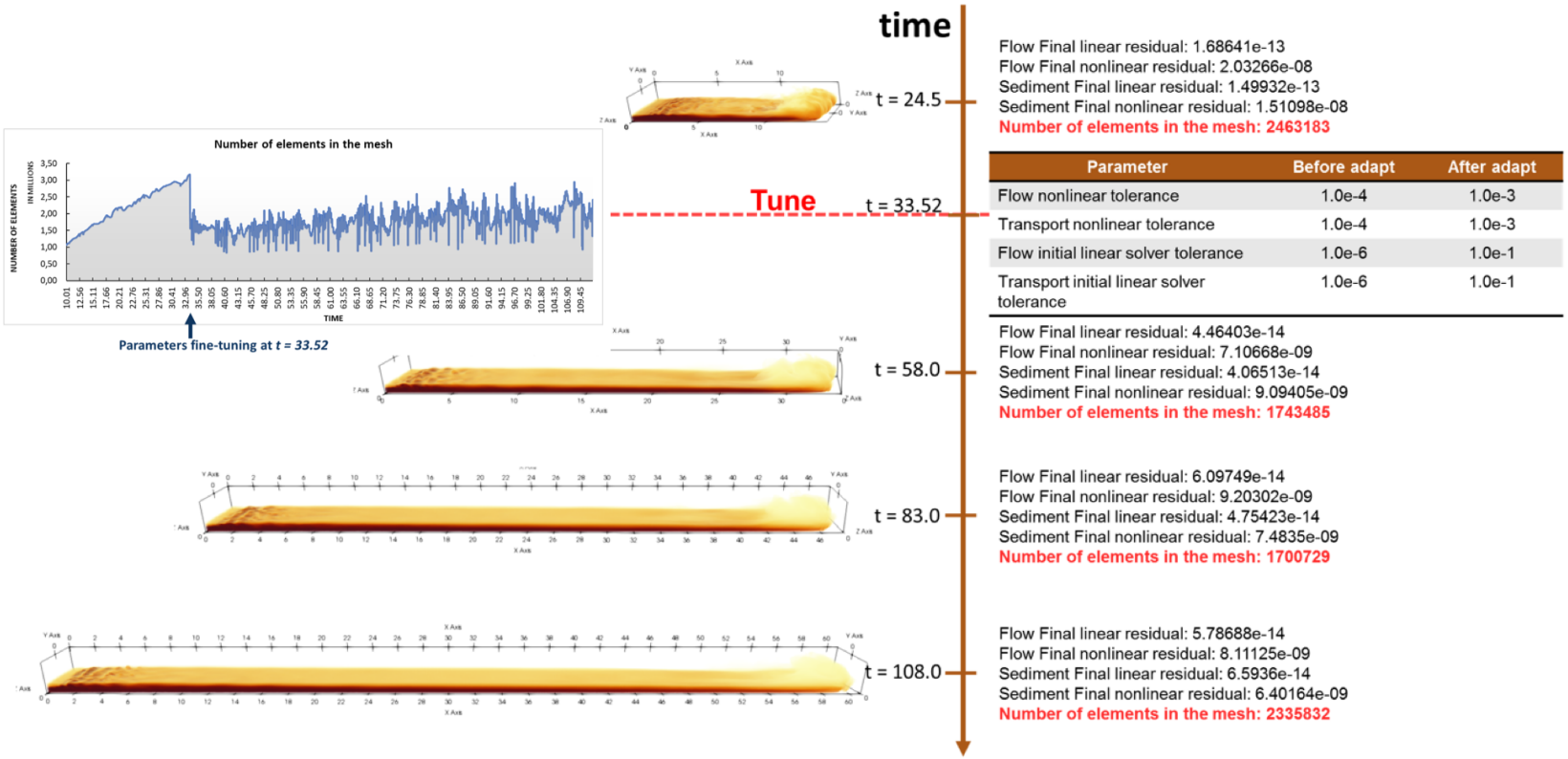


Figure: Parameter-tuning reduced the execution time by 10 days (37%), overhead $< 1\%$ and allowed job to finish successfully[1]

[1] R. Souza et al, Keeping Track of User Steering Actions in Dynamic Workflows, submitted, 2018.

Concluding Remarks and Discussion

- We have shown how to incorporate online data analysis in complex CSE simulations
- Tools work by inserting calls in the source code (like Catalyst `adaptor` calls)
- We provide provenance, in–transit data analysis and data extraction
- Solution is lightweight, no harm to performance
- In-situ visualization and in–transit data analysis essential for increasing robustness of complex simulations, enabling user-steering
- Provenance management contributes to increase reproducibility
- Annotated data can also feed training sets for ML
- User steering solution can be used in other contexts, e.g., ML, where tools like Google's TensorFlow needs tuning of several parameters

Acknowledgements

- **Data Sciences Team**:
  - Patrick Valduriez (INRIA), Daniel Oliveira, Jonas Dias, Luciano Leite
- **Computational Mechanics Team**
  - Fernando Rochinha, Renato Elias, Andre Rossa, Gabriel Guerra, Henrique Costa, Soulemaine Zio, Adriano Cortes, Gabriel Barros
- **Geology**: Marco Moraes and Paulo Paraizo, both from Petrobras R&D Center
- **Funding**: Petrobras, EU-BR H2020, CNPq, FAPERJ
- **Computational Resources**: @Lobo Carneiro, COPPE/UFRJ, @TACC, UT Austin
- **Special thanks** to Bill Barth (TACC) and Andy Bauer (formerly at Kitware)

Thanks for your attention

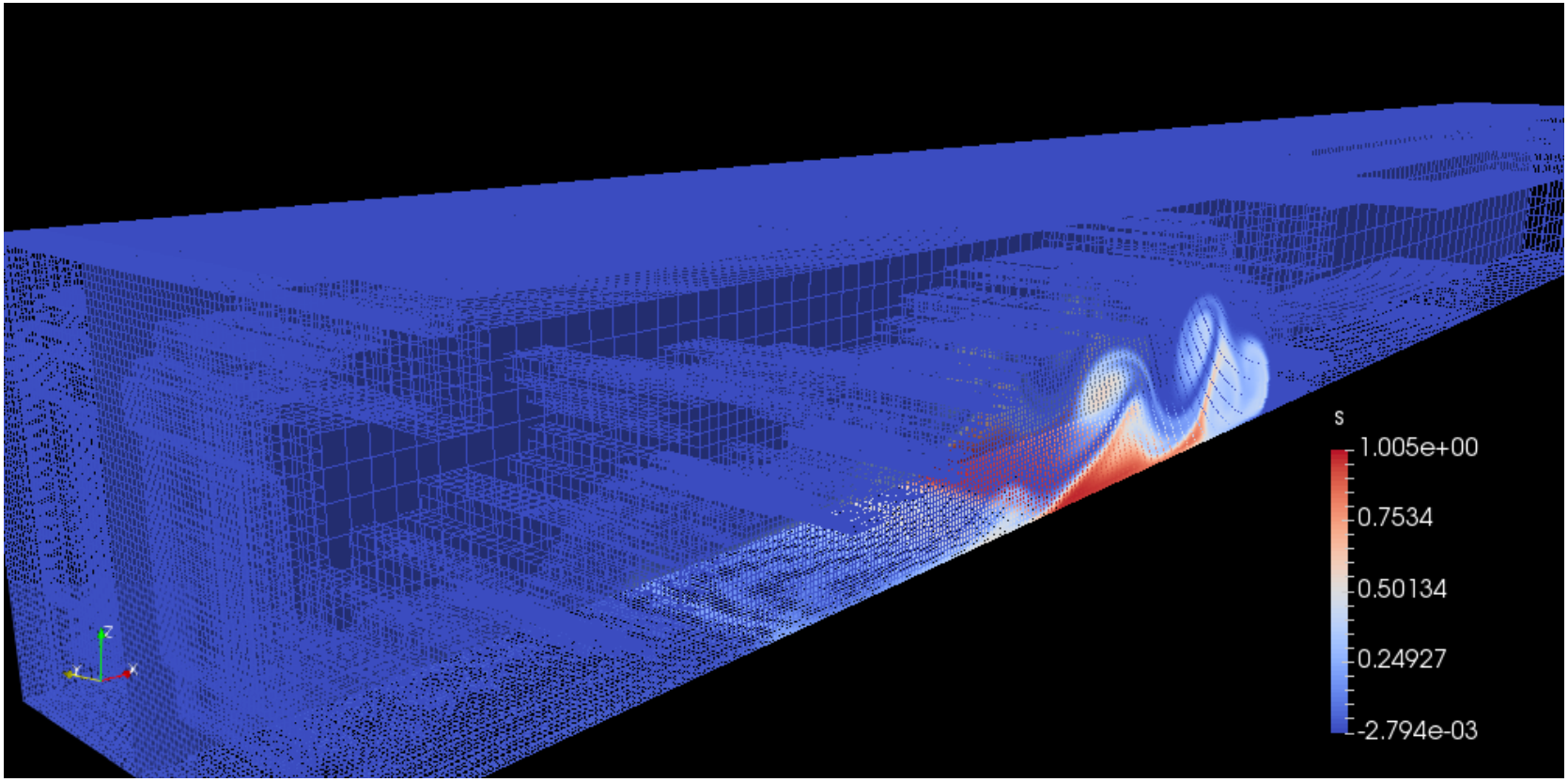


Figure: Monodisperse current: plot generated by Paraview Catalyst

A Simplified CSE Workflow

Pre-processing and mesh generation: select parameters (Δt_{max}, tolerances, solvers, AMR/C fractions, etc.)
Time stepping:
$t \leftarrow 0$
while $t < t_{max}$ **do**
a. Solve nonlinear equations
b. Adapt mesh/time step (Δt)
c. Update solution
d. Save data on disk when required
e. $t \leftarrow t + \Delta t$
end while
Post-processing, visualizing the data and extracting relevant QoIs

- **Step a**: computer time
- **Step b**: accuracy
- **Step d**: I/O in persistent storage
- **Optimal computational complexity**[1]: $\mathcal{O}(n_{eq}^{\frac{4}{3}})$

[1] Burstedde, et al, Extreme–scale AMR, in Proceedings of SC10, ACM/IEEE, 2010.

Using PrIMe to deploy DfA-prov

In this work, we follow PrIMe methodology[1] for enabling provenance support on fluid flow applications;
PrIMe has the phases:

- Phase 1: **Provenance Question Capture and Analysis**
  - Identification of provenance questions, data items, data transformations, and data dependencies
  - As output, it creates an UML class diagram with dataflow specification
- Phase 2: **Application Structure Analysis**
  - Identification of which parts of the application code we should add function calls
- Phase 3: **Adaptations on the Application**
  - Binding of variables from application code to values
  - Addition of provenance registering invocations on the parts of the source code identified in Phase 2

[1] S. Miles, P. Groth, S. Munroe, L. Moreau: PrIMe: A methodology for developing provenance-aware applications. ACM Transactions on Software Engineering and Methodology. 20:1–42 (2011)

Using PrIMe to deploy DfA-prov

User Phases with DfA-prov[1]

- Phase 4: **Configuration and submission of CSE application coupled to DfA-prov components**
  - Specification of input parameter values
  - Definition of the HPC environment
- Phase 5: **Submission of monitoring queries to our provenance database** which returns query results
- Phase 6: **Fine tunings on the CSE application** to adapt parameters at runtime

[1] V. Silva, D. de Oliveira, P. Valduriez, M. Mattoso. DfAnalyzer: Runtime Dataflow Analysis of Scientific Applications using Provenance. In: PVLDB. Rio de Janeiro, Brazil (2018)

DfA-prov: a PrIMe-based approach for developing provenance-aware applications

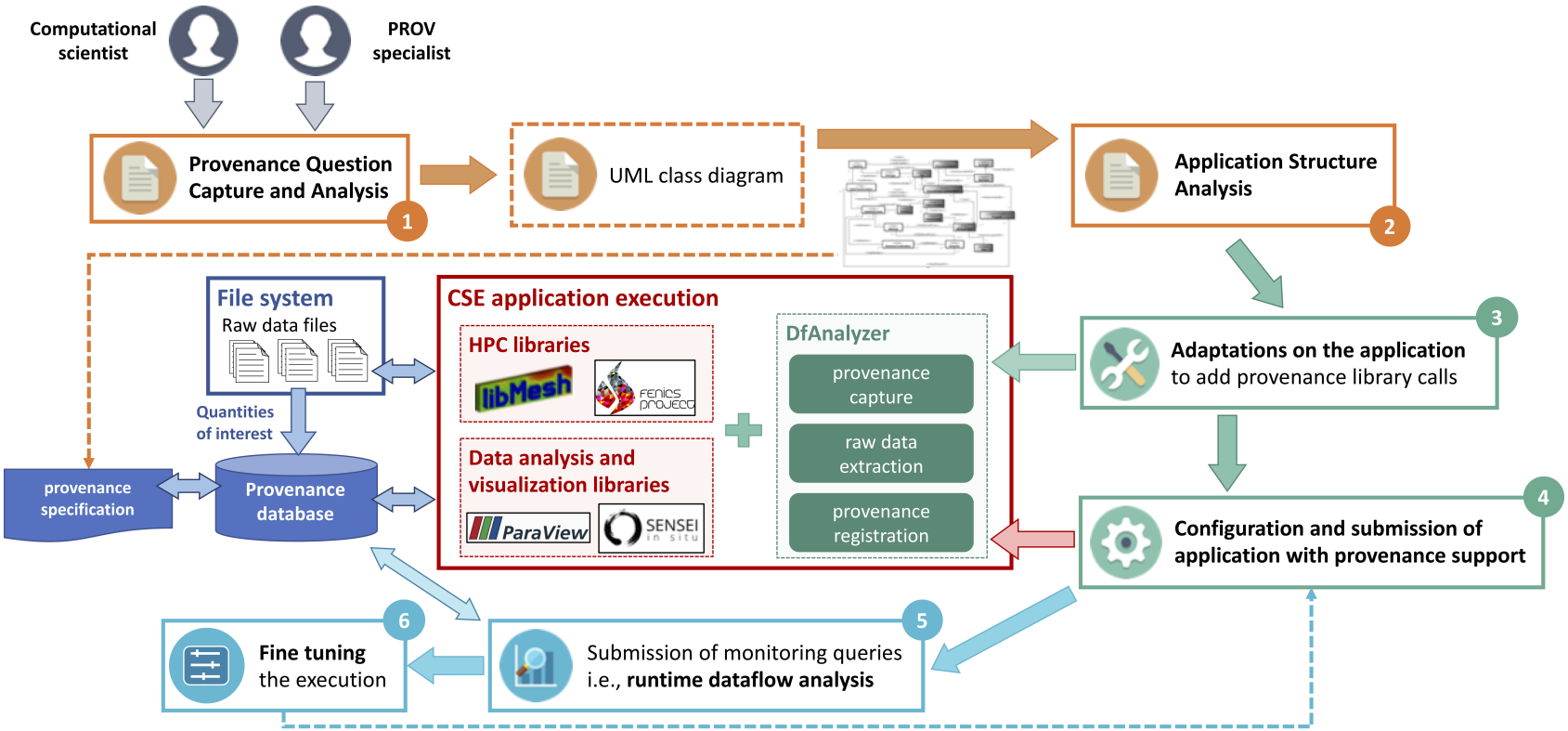


Figure: Our approach for developing provenance-aware CSE applications based on PrIMe methodology. The color codes represent the 6 PrIMe phases

User Steering in libMesh

```
while(t < t_max) {
      (...)
      if (is_file_exist("reset.run"))  {
            printf("RESETTING INPUTS");
            GetPot reset(input);
            dt          = reset("time/deltat", dt);
            tmax        = reset("time/tmax", tmax);
            (...)
            nonlinear_tolerance = reset("nonlinear_tolerance", nonlinear_tolerance);
            max_linear_iter     = reset("max_linear_iterations", max_linear_iter);
            linear_solver_tol   = reset("linear_solver_tolerance", linear_solver_tol);
            (...)
            ref_interval  = reset("amr/r_interval", ref_interval);
            r_fraction    = reset("amr/r_fraction", r_fraction);
            c_fraction    = reset("amr/c_fraction", c_fraction);
            max_h_level   = reset("amr/max_h_level", max_h_level);
            (...)
      }
      (...)
}
```

Figure: libMesh source code for supporting user steering – parameters amenable to be changed in runtime: time step (Δt), maximum simulation time (t_{max}), nonlinear and linear solver tolerances, AMR/C error fractions (r_{frac}, c_{frac}), and others not shown.