Accelerating the FlowSimulator:

Tracing and Profiling of Python Toolchains for Industry-Grade Simulations

Cristofaro Marco, Wendler Johannes, Reimer Lars, Huismann Immo

German Aerospace Center (DLR) – Dresden & Brunswick

Goal: green aviation

EU FlightPath 2050

- 75 % CO_2 reduction
- 90 % NO_x reduction
- 65 % noise reduction

=> Radically different aircraft designs needed

Aircraft development & certification

Flight envelope analyses requires:

- > 10,000 data points
- > 1,000 flight hours
- > 100 steady-state, high-fidelity simulations

Motivation for simulations

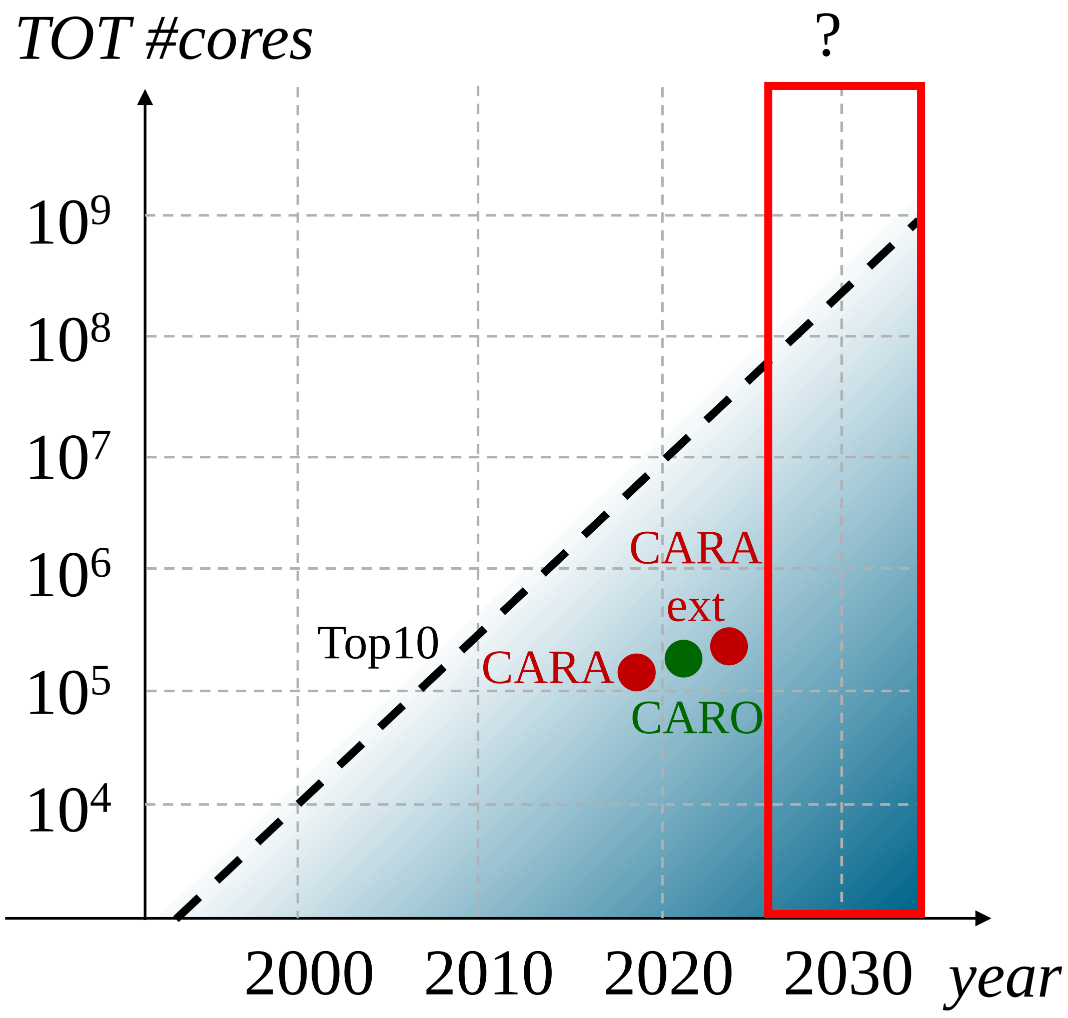
- Simulations with acceptable accuracy **may** replace costly/unfeasible testing

> 5.5 Engineering Analysis. An engineering analysis that includes the effects of the ice accretions as defined in Part II of Appendix C and Appendix O to CS-25 may be used to substantiate the performance and handling characteristics. The

- Aeroelastic problems can be modelled with fluid-structure interaction simulations:
  - CFD solver
  - CSM solver
  - Interpolation
  - Mesh deformation

- **High-performance computing** can be exploited to reach **acceptable time-to-solution** of high-fidelity simulations

Trend in HPC computational resources



Increase in resources

↓

shorter time-to-solution

&

larger meshes

2006	*A380**	*~50 10^6 elements*
2024	*HLPW5***	*~1.2 10^9 elements*

BUT WE NEED

SCALABLE SOFTWARE!

* Schwammborn et al "The DLR TAU-code: recent applications in research and industry," *ECCOMAS 2006*

***https://commonresearchmodel.larc.nasa.gov/*

Measurement platform

CARO

- DLR HPC System
- 174,592 cores
- #135 Top500 (11/2021)
- Göttingen (DE)
- each node:
  - 2x AMD EPYC 7702 (64 cores)
  - RAM: 256 GB DDR4
  - 16 cores per NUMA domain
  - 16 MB L3 cache shared among 4 cores

CARO node

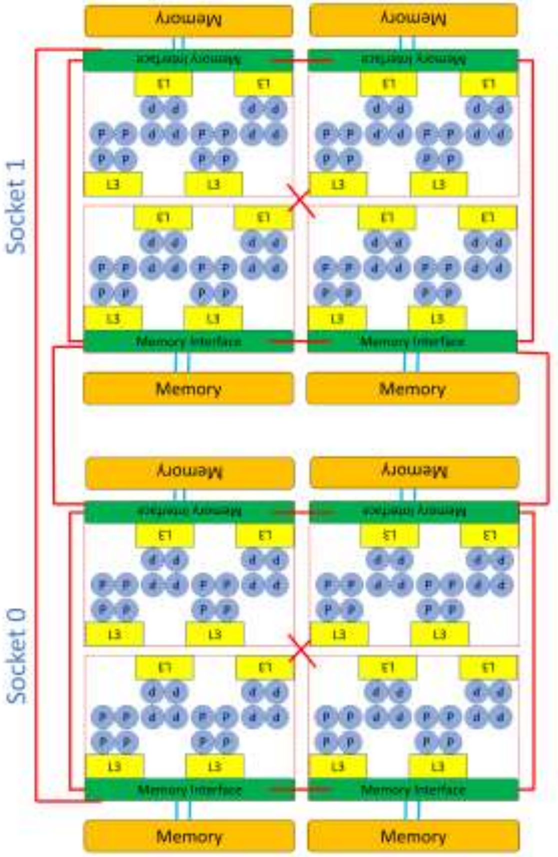


Image by Hirav Patel

FlowSimulator

Simulation frameworks for **multidisciplinary analyses** and optimizations

&

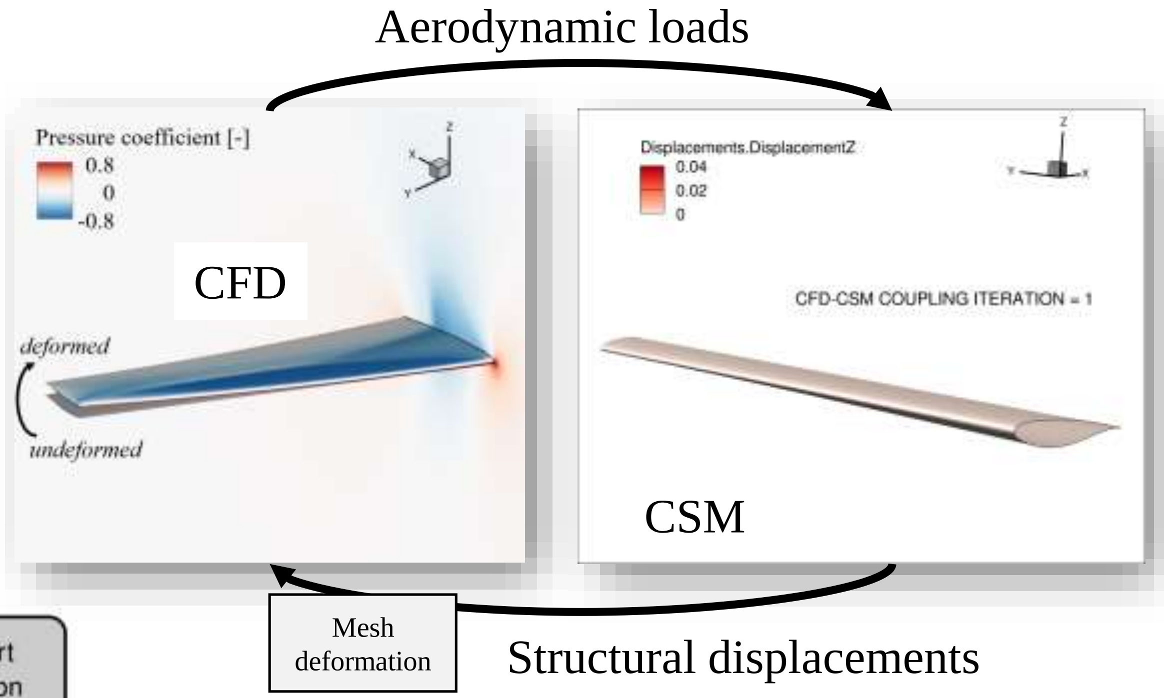
Innovative simulation softwares for high-fidelity predictions on **HPC**

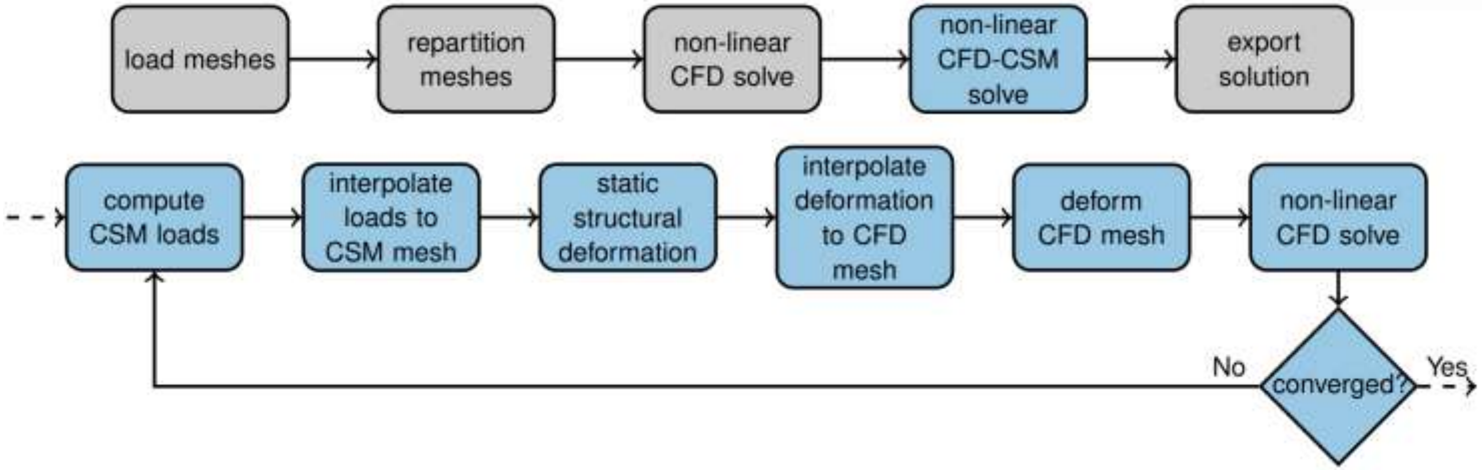
Test case

Steady aeroelastic equilibrium of LANN wing*

semispan=1 m, chord=0.361÷0.144 m

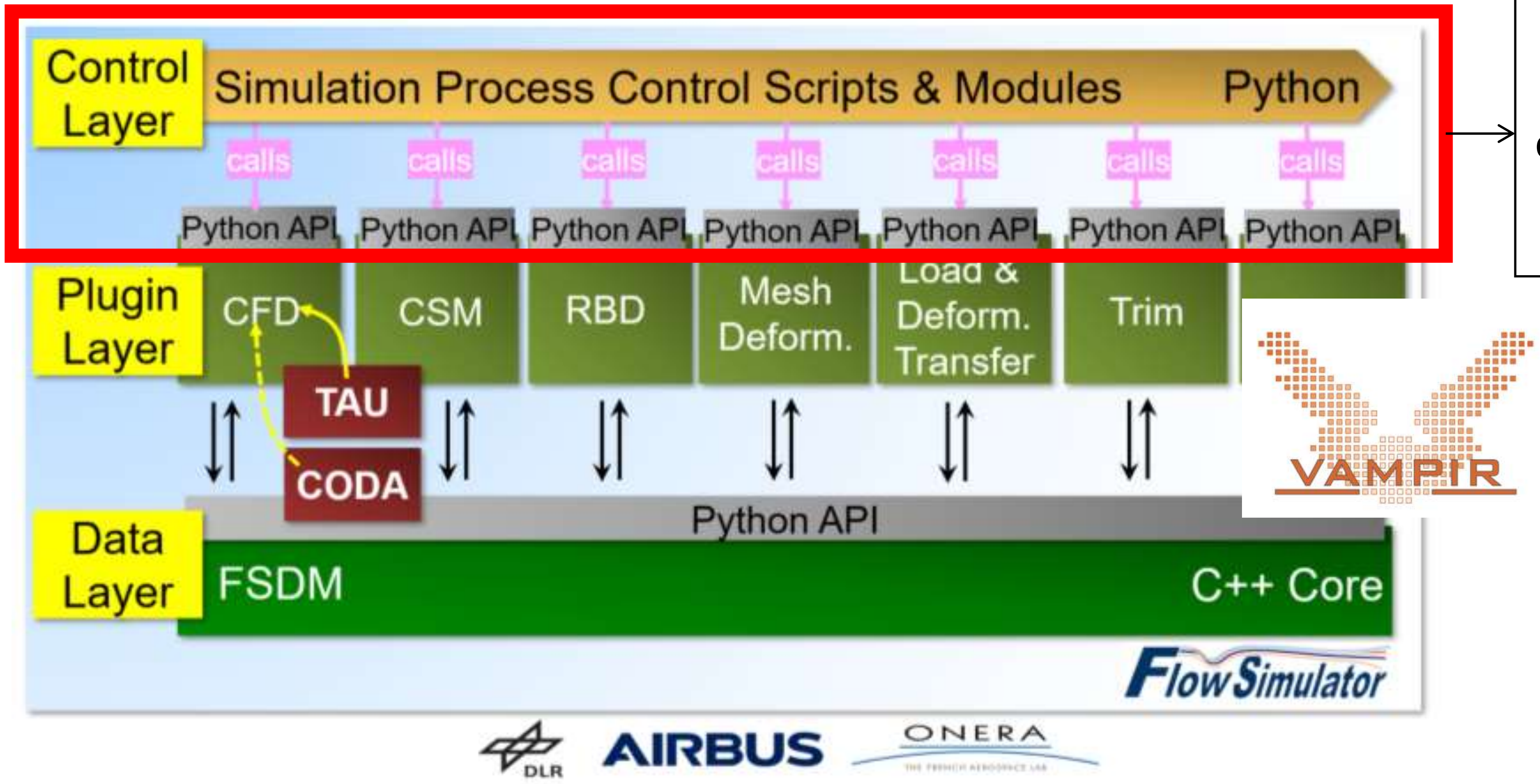
- **CFD mesh:** 10^6 elements, 1.2 10^6 nodes
- **CSM mesh:** 860 elements, 1260 nodes





*Firth, George C. "LANN wing design." *NASA. Langley Research Center Cryogenic Wind Tunnel Models* (1983).
**Cristofaro, et al. "Accelerating the FlowSimulator: Improvements in FSI simulations for the HPC exploitation at industrial level," *Coupled 2023*

FlowSimulator code structure and Score-P Python binding



py-scorep-binding
can profile and trace
complex simulation toolchains:

```
python -m scorep script.py
```

Load trace with **Vampir**

Runtime overview with almost no effort

*Reimer et al "Virtual Aircraft Technology Integration Platform: Ingredients for Multidisciplinary Simulation and Virtual Flight Testing." *AIAA 2021*

py-scorep-binding compiler problem

On CARO:

```
> spack load py-scorep-binding
> module load flowsim
> python -m scorep script.py    : command ['/usr/bin/cc', '-c', '…/scorep_init.c', …] failed
```

Problem:

py-scorep-binding wants to use `/usr/bin/cc`, but

```
> which cc      : /usr/bin/cc
> which gcc     : /sw/rev/23.05/linux-rocky8-zen/gcc-8.5.0/gcc-10.4.0-xozig6/bin/gcc
```

Workaround:

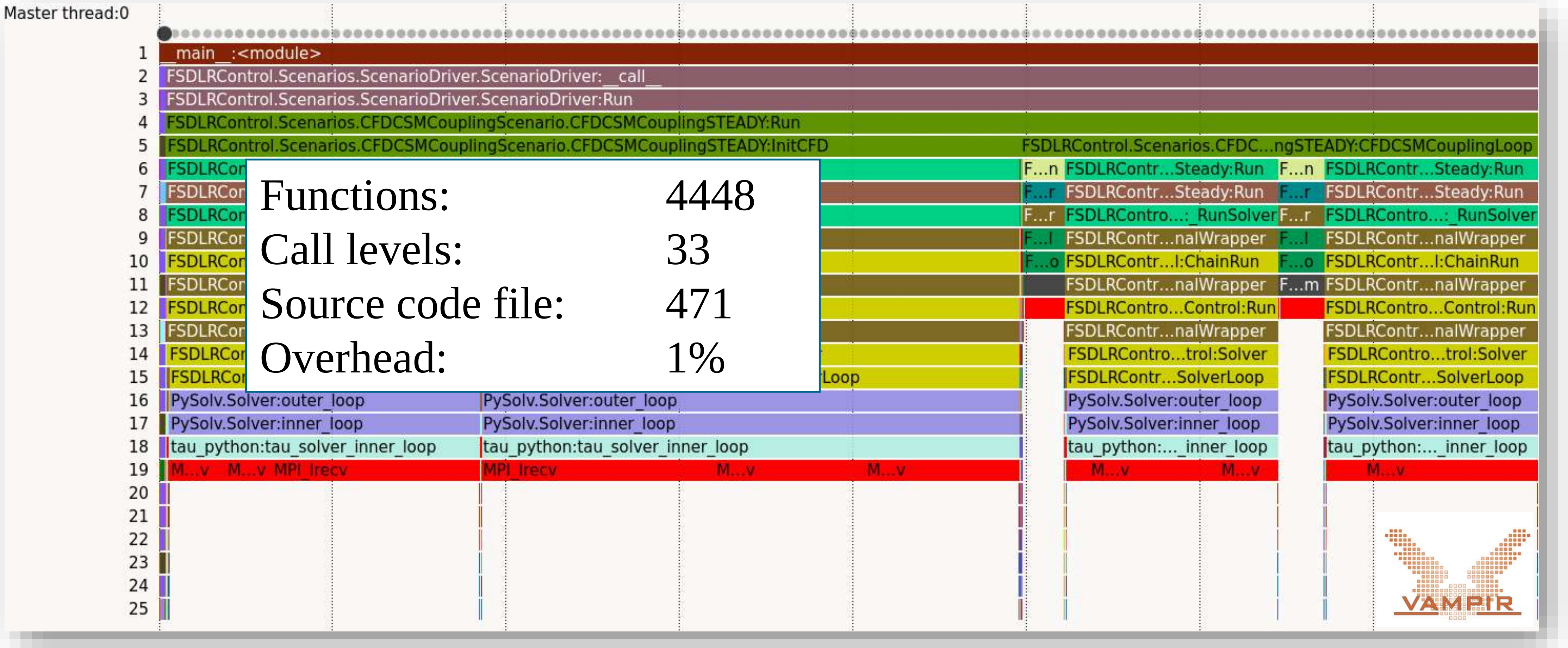
In subsystem.py under py-scorep-binding installation folder add:

```
cc.set_executable("compiler", "gcc")
cc.set_executable("compiler_so", "gcc")
```

Only Python wrapping (no instrumentation)

Ready to use sbatch:

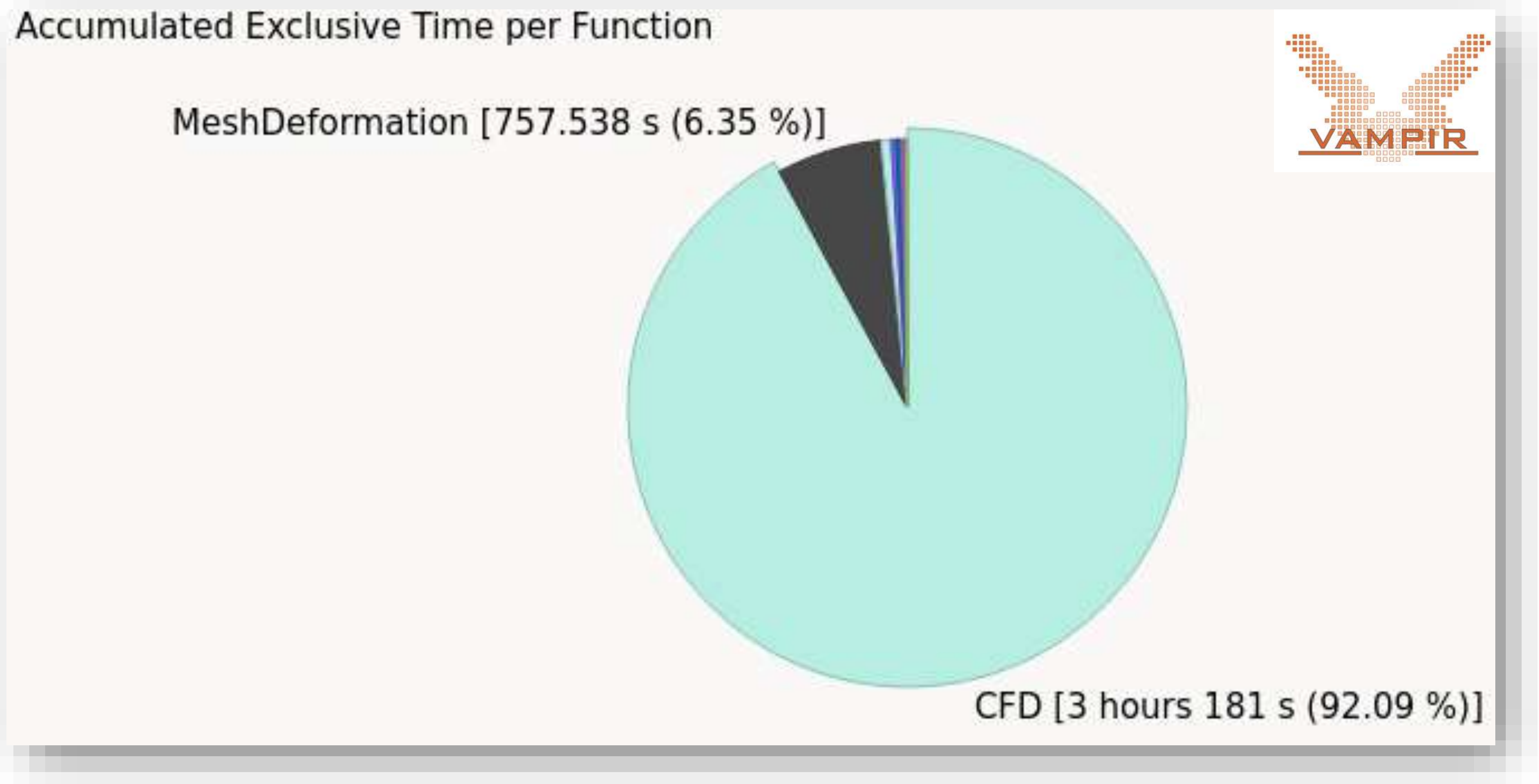
```
spack load py-scorep-binding
module load flowsim
srun python -m scorep --mpp=mpi --thread=omp script.py
```



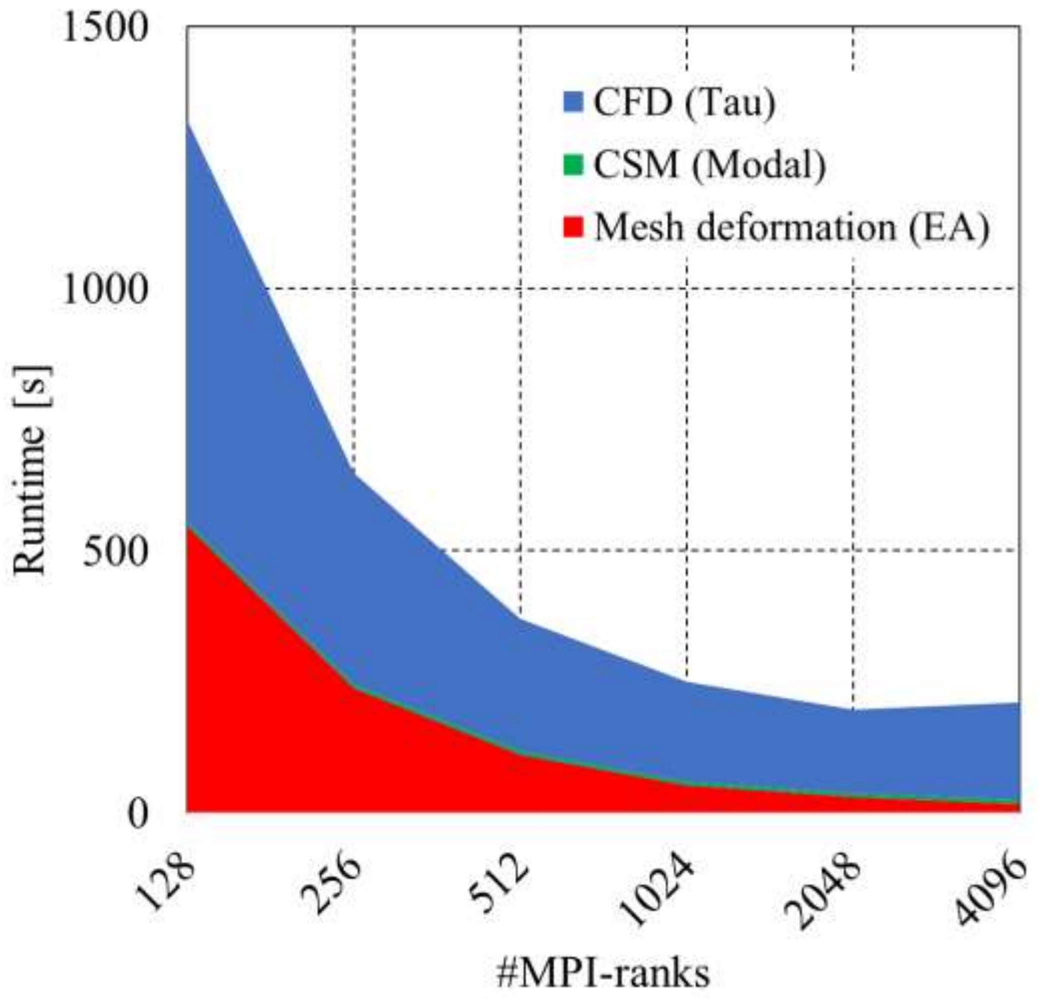
Functions:	4448
Call levels:	33
Source code file:	471
Overhead:	1%

Runtime distribution (2 MPI x 4 OpenMP)

- `MPI*` filtered out
- Most time consuming simulation blocks:
  - *CFD*
  - *MeshDeformation*
- Negligible simulation blocks:
  - *MeshImport*
  - *CSM*
  - *Interpolation*
  - *MeshExport*



Strong scaling of steady aeroelastic simulation



"Manual" measurements need:

- Predict relevant simulation blocks
- Print runtimes
- Extract for plotting

Cristofaro, et al. "Accelerating the FlowSimulator: Improvements in FSI simulations for the HPC exploitation at industrial level," *Coupled 2023*

Strong scaling with only Python wrapping

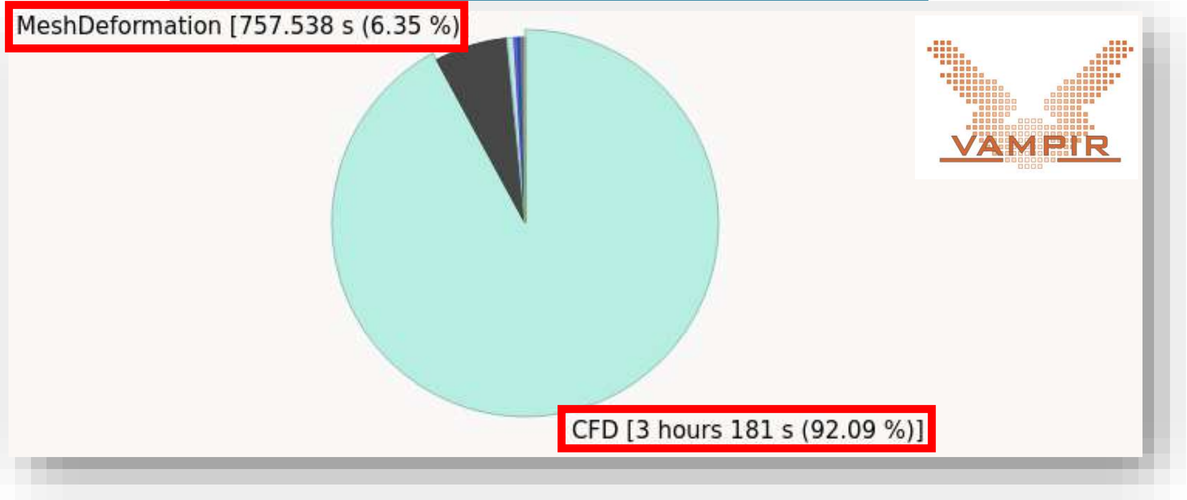
From 2 MPIs to 512 MPIs

- main blocks runtime decrease:
  - *CFD*
  - *MeshDeformation*
- new blocks appears:
  - *importlib* (one time)
  - *MeshOps* for reading meshes (one time)
  - *MeshSelection* for interpolation (mostly init phase)

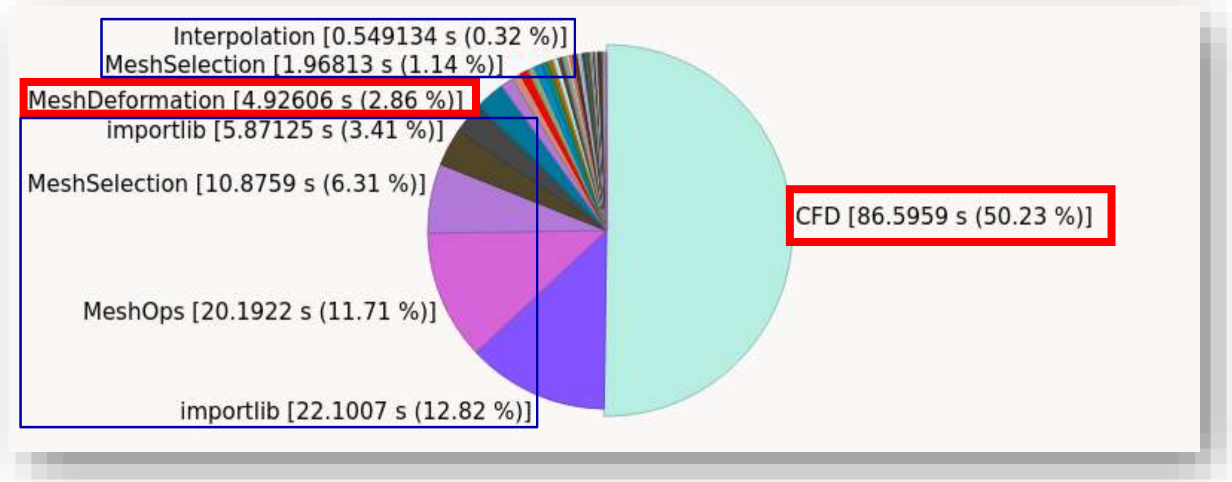
↓

Not analysed with "manual" scaling measurements

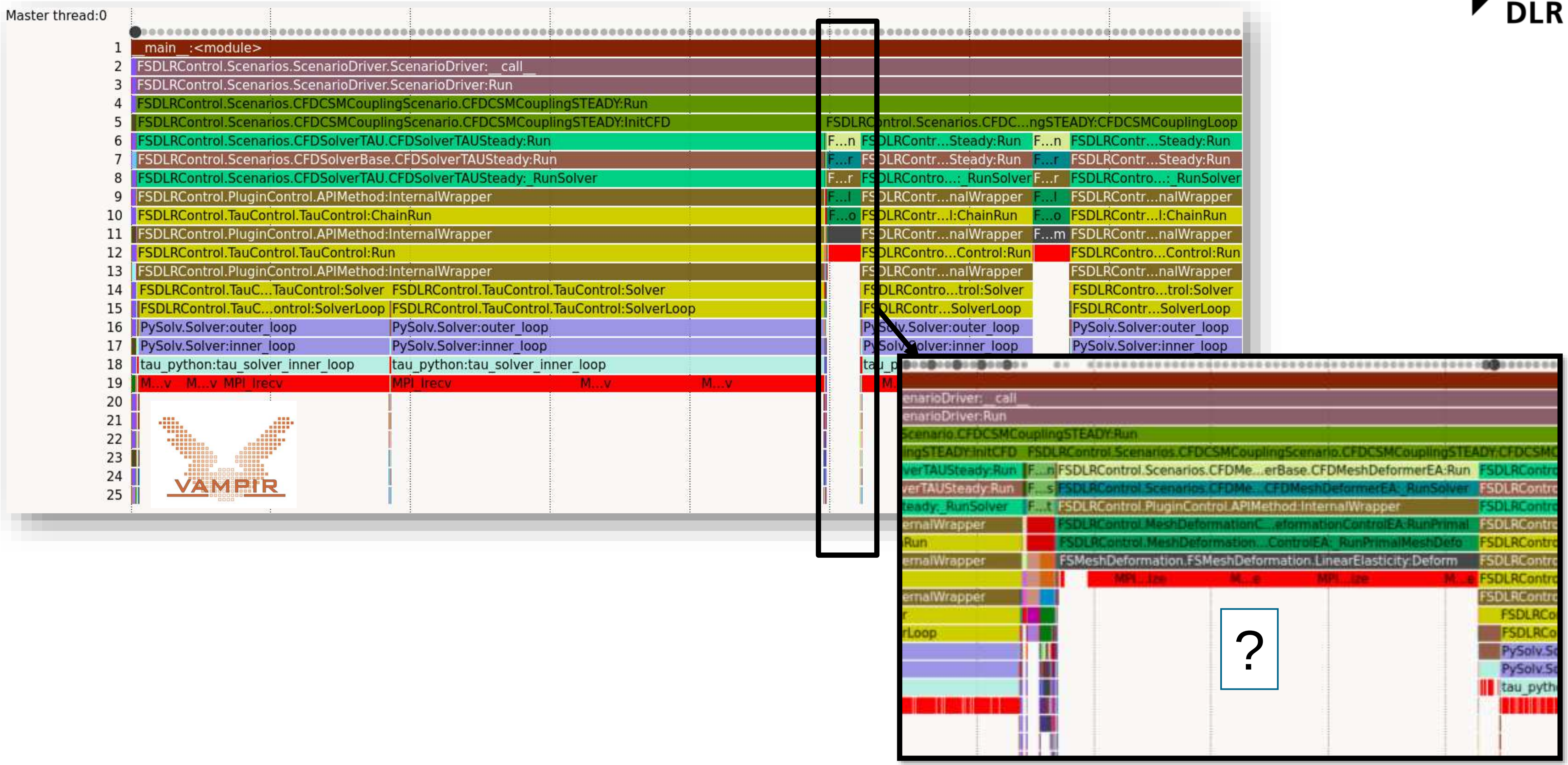
1 node x 2 MPI x 4 OpenMP



16 node x 32 MPI x 4 OpenMP



Missing info in Cpp plugins



Cpp instrumentation for Score-P

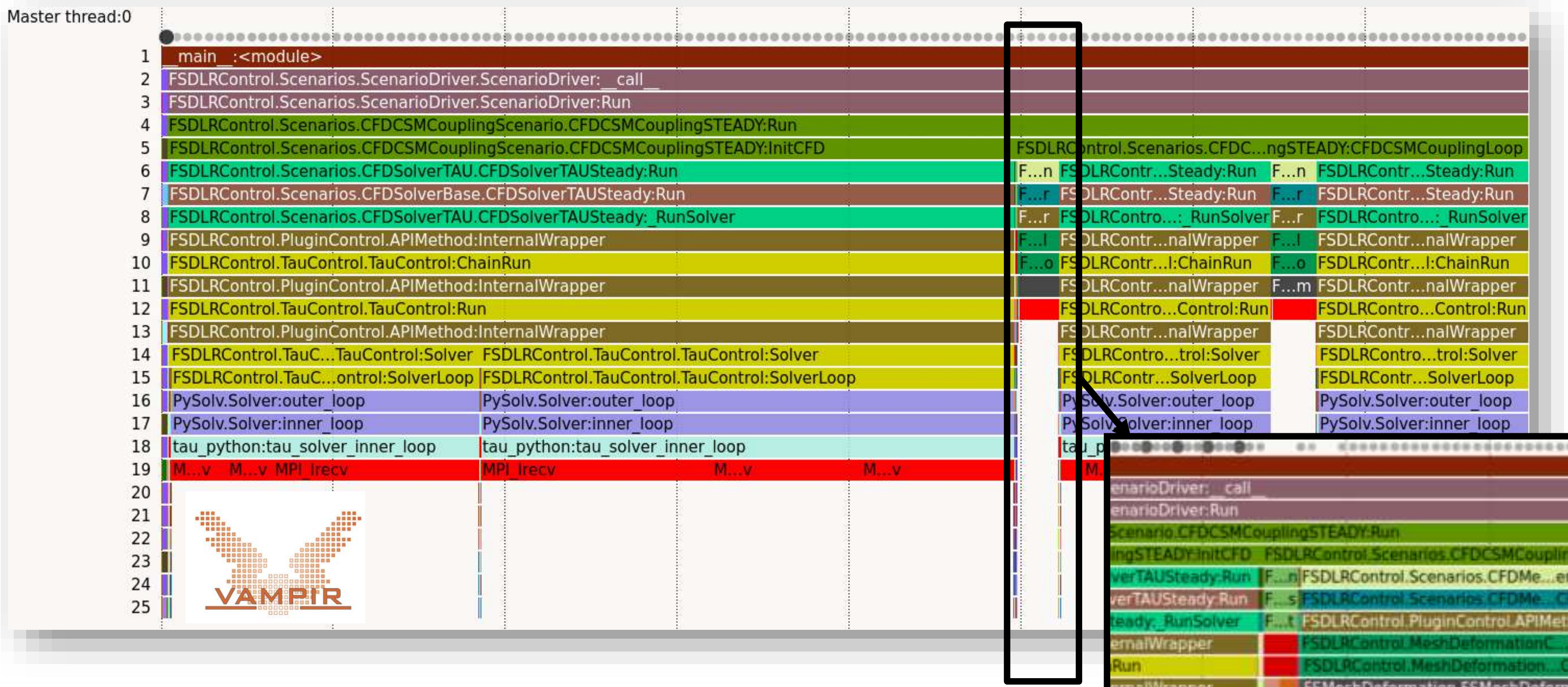
Cpp plugins tracing requires:

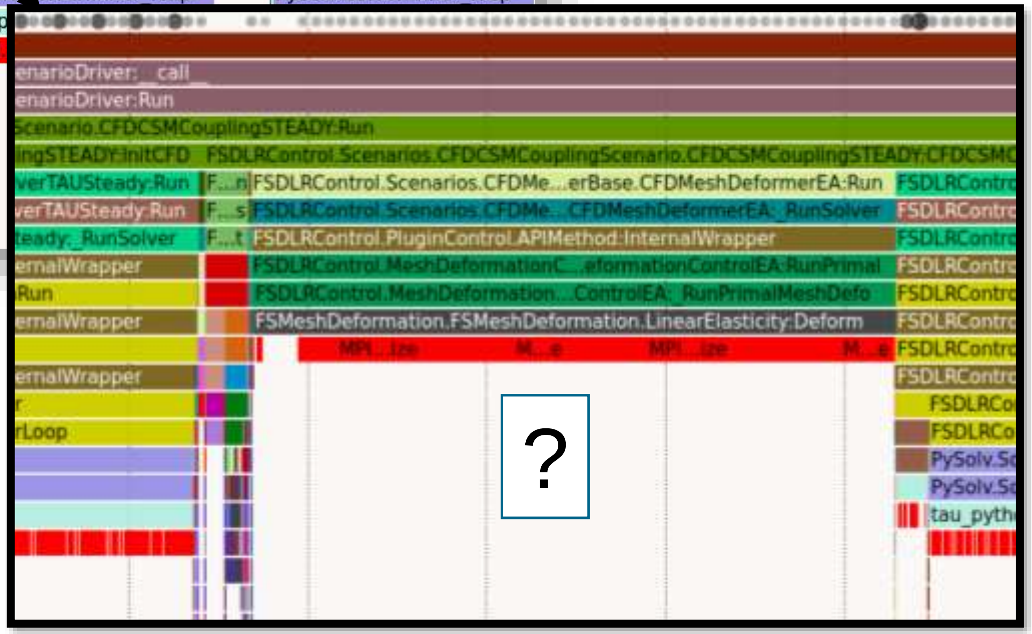
- Score-P wrapper should work with all plugins build systems
  (implementation effort)

- Re-build everything with Score-P wrapper
  (a few hours)

- Define Score-P filters to limit trace size
  (automatic with `scorep-score -g` or manual)

Traces with no code instrumentation

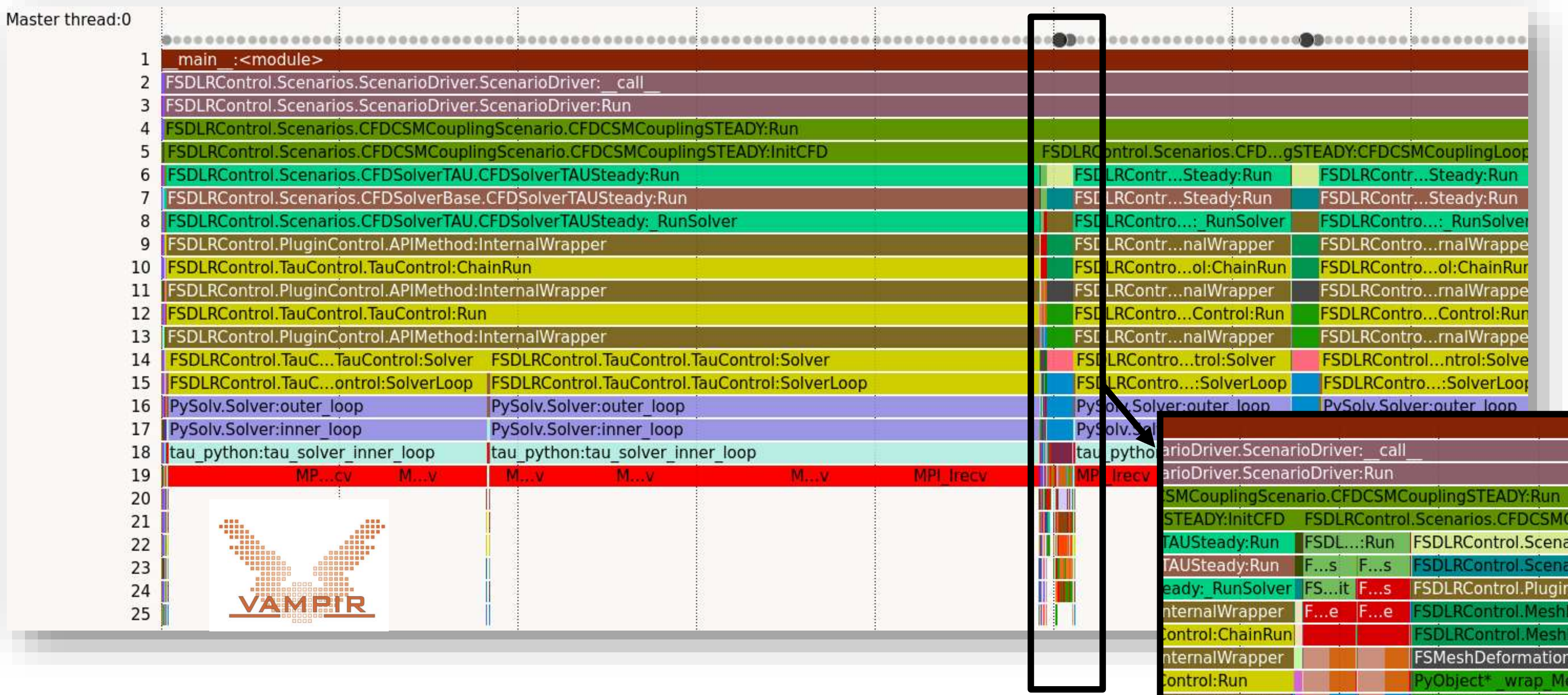


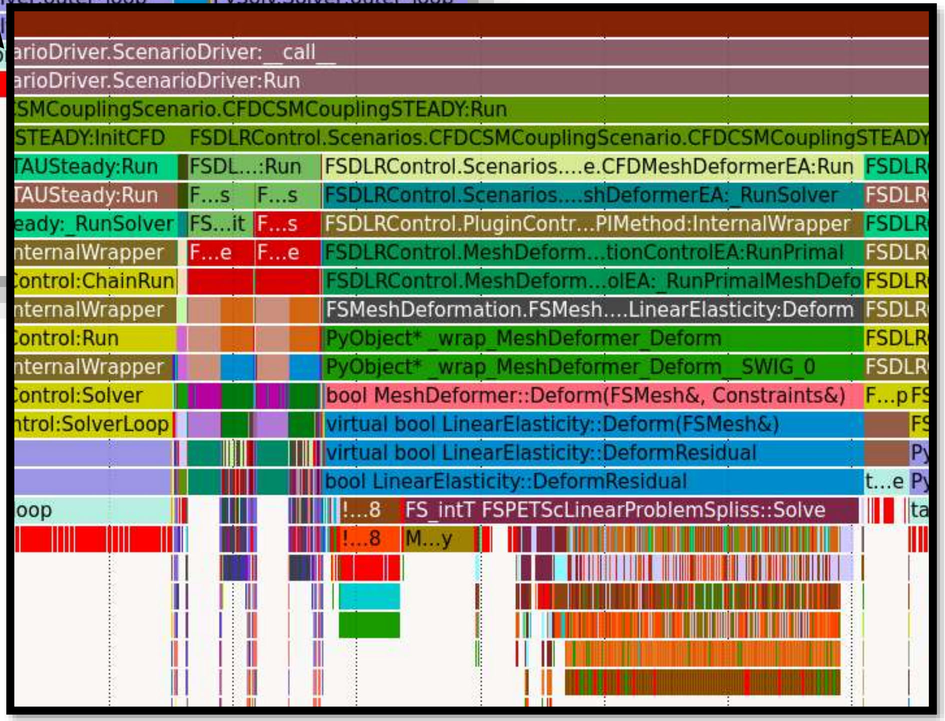


Trace size: 138 MB

Overhead: 1%

Traces with Cpp instrumentation





Trace size:	5 GB
Overhead:	6%
	depends on filter

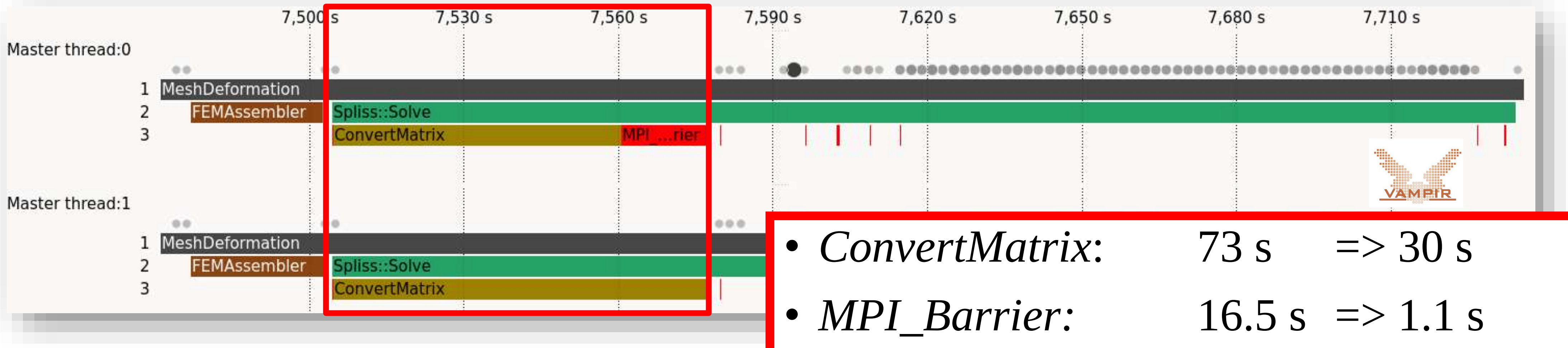
Mesh deformation (2 MPI x 4 OpenMP)

Focusing on *MeshDeformation*

- *ConvertMatrix* 73 s
  - 27% of *MeshDeformation* runtime
  - More than *FEMAssembler*
  - 16.5 s: *MPI_Barrier*

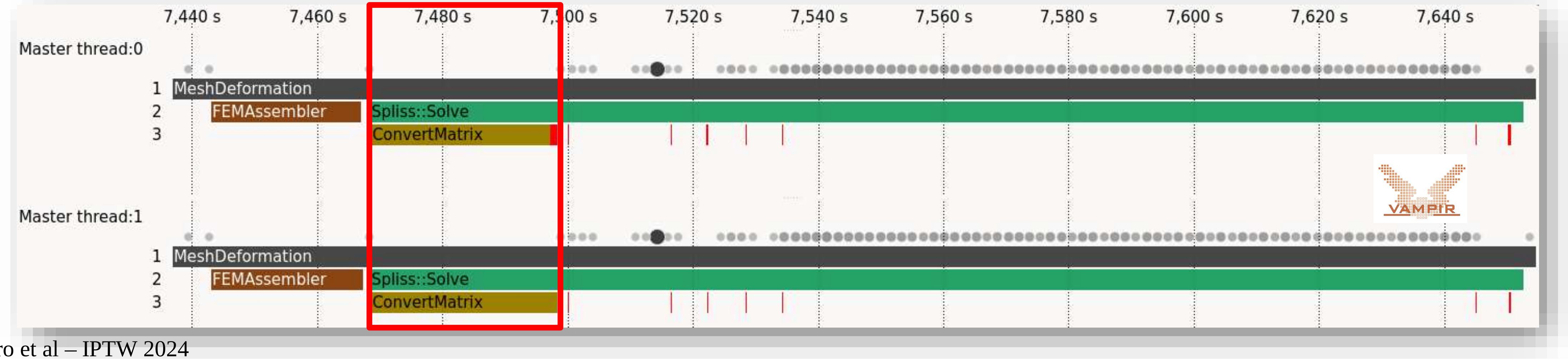
Matrix conversion algorithm improvement (2 MPI x 4 OpenMP)

Original algorithm



- *ConvertMatrix*: 73 s => 30 s
- *MPI_Barrier:* 16.5 s => 1.1 s
- *MeshDeformation*: -18%

Improved algorithm



Conclusion

- Tracing of steady aeroelastic simulations
  - industrial-grade toolchain
  - aircraft design and certification with HPC
  - *FlowSimulator* (>50 plugins)

- Only Python binding
  - Readily available with pre-installed software
  - Good overview of main simulation blocks runtime
  - No info about Cpp implementations

- With instrumentation
  - Time consuming:
    - work to fit all build systems
    - re-build everything
    - define filters
  - Large traces

  => beneficial to specific code analyses

Acknowledgments

The authors gratefully acknowledge the scientific support and HPC resources provided by the German Aerospace Center (DLR). The HPC system CARO is partially funded by "Ministry of Science and Culture of Lower Saxony" and "Federal Ministry for Economic Affairs and Climate Action"