ParaSiF_CF: A Partitioned Fluid-Structure Interaction Framework for Exascale

Wendi Liu[1], Alex Skillen, Charles Moulinec

Scientific Computing Department, Science and Technology Facilities Council,

Daresbury Laboratory, Warrington WA4 4AD, UK

ABSTRACT

A new massively parallel partitioned fluid-structure interaction (FSI) simulation framework, ParaSiF_CF, has been built. This FSI framework employs Code_Saturne to solve the computational fluid dynamics (CFD), FEniCS to solve the computational structure mechanics (CSM) and the Multiscale Universal Interface (MUI) for data exchange. The codes adopted in this framework have demonstrated excellent individual performance in highly parallel simulations [1-3]. The partitioned approach to FSI offers several advantages over a monolithic coupling. The primary advantage motivating the present study is the ability to leverage the highly specialised legacy codes that have had decades invested in them (i.e. Code_Saturne for CFD and FEniCS for CSM), and are tailored to the task they are designed to solve (e.g. consider the wealth of turbulence models in Code_Saturne, which are simply unavailable in FEniCS). Both linear elastic and hyper-elastic structure solvers have been implemented in the framework. The parallel performance of the framework has been optimised and its scalability has been assessed aiming for high parallel efficiency on large problem sizes. Around 85% parallel efficiency has been achieved with high node counts in our tests.

Keywords: Fluid-structure interaction; High performance computing; Partitioned approach.

1. Introduction

The phenomenon of fluid-structure interaction (FSI) occurs frequently in many areas of engineering. Here we focus on the field of low-carbon and renewable energy generation. Examples include the aeroelasticity of a wind turbine blade, hydroelasticity of mooring lines for floating offshore energy devices, the flow-induced vibration of slender rods within a nuclear reactor and the seismic fragility of the shield building for a nuclear power plant.

Due to its non-linear, time-dependent and multi-physical nature, numerical simulation of FSI problems requires greater computational resources relative to pure fluid dynamic or structural dynamic problems. Many billions of cells would be required for a scale-resolving FSI simulation of a floating offshore wind turbine farm, or the core of a nuclear power plant, for instance. These are highly ambitious simulations that are only recently becoming affordable. But to achieve simulations of this scale, an efficient and highly scalable FSI simulation tool is needed. The existing state-of-the-art for partitioned FSI frameworks (such as Code-Saturne coupled with Code_Aster) use direct solvers for the structure domain, which typically scale

[1]Corresponding author. E-mail address: wendi.liu@stfc.ac.uk

up to 128 MPI tasks, severely limiting their applicability on machines like ARCHER2 where node RAM is relatively low.
In this project, we established a new massively parallel partitioned fluid-structure interaction simulation framework, ParaSiF_CF, by employing a partitioned approach with existing well-developed and highly performant computational fluid dynamics (CFD) and the computational structure mechanics (CSM) codes for ARCHER2, with a view towards future Exascale resources. Both linear elastic and hyper-elastic structure solvers have been coupled with Code_Saturne. The former will provide the ability to compute massively parallel cases involving "typical" operating conditions in the fields of nuclear, offshore wind, and marine turbines (among others). The latter will provide a highly efficient capability in the simulation of speciality plastic polymer or composite material components that are involved in offshore turbines or nuclear power plants. The parallel performance of the framework has been optimised and its scalability has been assessed aiming for high parallel efficiency on large problem sizes. About 85% of parallel efficiency on the 3-D FSI simulation with linear structure solver is achieved for 15M cells on the CFD domain and 390k DoFs on the CSM domain going from 2 nodes to 9 nodes on ARCHER2. About 75% of parallel efficiency on the 3-D FSI simulation with both linear and nonlinear structure solvers are achieved for 233M cells on the CFD domain and 2.7M DoFs on the CSM domain going from 5 nodes to 34 nodes on ARCHER2.

2. Code selection and preliminary scalability test on each solver (WP1).

ParaSiF_CF comprises three parts: the CFD fluid solver, the CSM structure solver and the interface coupling tool, MUI, between solvers. Codes adopted for ParaSiF_CF are described briefly below:
Code_Saturne V6.0 (https://www.code-saturne.org) is an open-source multi-purpose massively parallel CFD code written in C (~55%), Fortran (~32%), and Python (~13%). Parallelism is handled by a hybrid MPI+OpenMP programming paradigm and some parts of the code support CUDA. The part of the code used herein relies on Finite Volume discretisation to solve the governing equations. Moving meshes required by the FSI framework are handled by the Arbitrary Lagrangian-Eulerian (ALE) algorithm. The code is highly scalable and portable. It has been tested on many different platforms. In addition, the code is part of the PRACE UEABS for CPUs and accelerators.
FEniCS V2019.1.0 (https://fenicsproject.org) is an open-source platform that solves partial differential equations using the Finite Element Method. It comprises a collection of libraries, with the Finite Element Method core being written in C++. The typical user interface is via Python. The framework is widely used by the ARCHER/ARCHER2 community. Structure mechanics solvers have been developed for ParaSiF_CF, with FEniCS as the backend. We implement both the Hyper-elastic Saint-Venant-Kirchhoff model and the Generalized harmonic oscillator method, which allows solving a large variety of problems.
The Multiscale Universal Interface (MUI) library V1.1.3 (https://github.com/MxUI/MUI) and its FSI coupling utility are used by the framework as the coupling interface. The MUI library is a header-only concurrent framework for coupling heterogeneous solvers by domain decomposition. MUI is written in C++ and has wrappers for C, Fortran and Python. It has demonstrated good scalability up to 512 nodes of ARCHER [3] and allows an arbitrary number of codes to communicate with one another over MPI via a cloud of point data. The Radial Basis Function (RBF) spatial sampler of the MUI library is an essential function for FSI simulations to ensure the forces are conserved at the code interface. A third-party utility of the MUI library with the Fixed Relaxation, Aitken's δ2 and Quasi-Newton methods to ensure a tight and stable FSI coupling has been used in the framework.

FEniCS-X was under active development throughout this project. We decided to keep using the latest stable version of FEniCS (v2019.1.0) at M2 (November 2020) of this project. The first alpha version of FEniCS-X components was released in May 2021. Follow-up versions (FEniCS-X v0.2.0 and v0.3.0) were released in August 2021. The new version of FEniCS (FEniCS-X) is a major redevelopment and re-write for the core code of the FEniCS project. Although some features are still awaiting reimplementation, the latest released FEniCS-X demonstrated promising performance and functions, some of which being essential for ParaSiF_CF to enhance its functionality and performance. FEniCS-X employs distributed memory parallel design and implements faster user kernels form in Python, which makes the FEniCS-X faster and more scalable compared to old FEniCS versions. FEniCS-X is much easier to extend and maintenain compared with old FEniCS versions, especially on its Python interface. The new functionality of point-source will make the FSI dynamic condition easier to implement in the structure solver. There is scope to work closely with the FEniCS-X development team to port the existing ParaSiF_CF structure solvers that are based on FEniCS v2019.1.0 to the latest FEniCS-X in the near future.

3. Code coupling between Code_Saturne and FEniCS with MUI for linear structure deformation (WP2).

The CFD code, Code_Saturne (V6.0.6), has been coupled with the FEniCS (V2019.1.0) based linear structural solver, through the MUI (V1.1.3) toolbox, to simulate small deformations of the structure. The two-way coupling between Code_Saturne and FEniCS is relying on accurate and conservative interpolations using the MUI Radial Basis Function spatial sampler and MUI's coupling utility for FSI simulations.

3.1. Code_Saturne with MUI embedded

Code_Saturne is used as the fluid solver in ParaSiF_CF to solve the incompressible Navier-Stokes equations.. The ALE function of Code_Saturne is used to model the structure elasticity in the fluid domain. As most of the source code of Code_Saturne is written in the C programming language, the C wrapper of the MUI library is used and embedded into the source code of Code_Saturne. The static libraries of the MUI C wrapper and the MUI utility C wrapper have to be linked to Code_Saturne. All the essential MUI functions, such as MUI uniface creation, MUI sampler defines, etc. have been collected in *Code_Saturne* with further customisations according to the need of ParaSiF_CF. Since a certain proportion of Code_Saturne source code is written by Fortran, all the MUI related functions in C language are interfaced with Fortran code. Moreover, the subroutine *cs_push_force_MUI_Coupling()* is also included to collect the forces at each boundary face of interest and push them along with the corresponding coordinates of the boundary face centre. The module *cs_user_mui_coupling* is written to collect all the user control parameters for the MUI coupling.
MUI related headers, *mui_c_wrapper_3d.h* and *cs_MUI_Coupling.h*, have to be included to expose MUI functions to Code_Saturne.
The MUI function *mui_mpi_split_by_app()* is used in the *_cs_base_mpi_setup ()* function to create the MPI common world for Code_Saturne. The MUI function *setup_MUI_Coupling()* is included in the end of the *_cs_base_mpi_setup ()* function to declare MUI uniface object as well as spatial and temporal samplers.
The MUI initialisation is achieved just before the time loop. In the MUI initialisation block, the MUI smart sent functions are defined to minimise the MPI communication overhead. The objects of FSI coupling algorithms are also created and initialised. At the end of the MUI

initialisation block, the MUI commit function is used to commit zero step to announce the completion of the solver/MUI initialisation.
The MUI push subroutine *cs_push_force_MUI_Coupling()* and MUI commit subroutine *commit_MUI_Coupling()* are called just before the call of ALE boundary conditions subroutine *usalcl()*. These two subroutines allow forces of all boundary faces of interest to be exposed to the structure solver with the timestamp of the total FSI sub-iterations.
In the ALE boundary conditions subroutine *usalcl()*, the MUI fetch function *cs_FSI_Coupling_Fetch()* is used to obtain displacements of nodes of interest that are solved by the structure solver. Functions of FSI coupling algorithms are used to filter the raw data received from the structure solver before being applied to the fluid domain so that to make the FSI coupling tight and stable.
The FSI sub-iteration function is required by the FSI coupling algorithms. It has been implemented just after the ALE structure displacement and the implied looping test.

3.2. Coupling between Code_Saturne and FEniCS

The data exchange scheme of Code_Saturne with MUI embedded is shown in the left of Figure 1. The integer i represents the number of total sub-iterations. It is a function of the number of time steps and the number of FSI sub-iterations per time step. If the number of FSI sub-iterations per time step is set to 1, the number of total iterations, i, will have the same value as the number of time steps. At each sub-iteration, Code_Saturne push forces at each boundary face of interest and commit them with timestamp i. It fetches displacements from the structure solver with timestamp i (or i-1 if the parallel FSI coupling scheme is used) and update them to nodes of interest with the aid of FSI coupling algorithms. It solves the fluid domain with the updated ALE boundary condition and moves to the next sub-iteration.

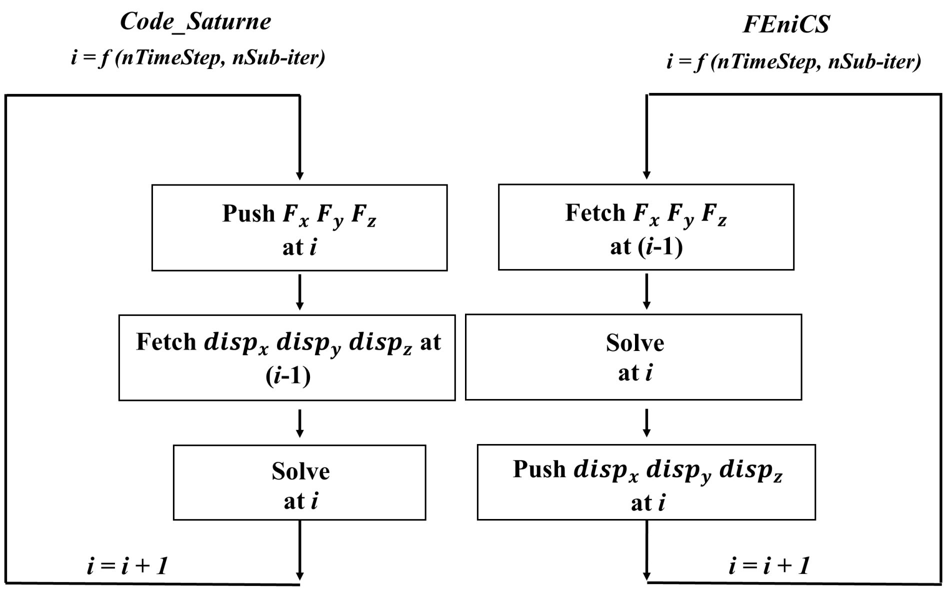

Code_Saturne
i = f (nTimeStep, nSub-iter)
Push F_x F_y F_z at i
Fetch $disp_x$ $disp_y$ $disp_z$ at (i-1)
Solve at i
$i = i + 1$
FEniCS
i = f (nTimeStep, nSub-iter)
Fetch F_x F_y F_z at (i-1)
Solve at i
Push $disp_x$ $disp_y$ $disp_z$ at i
$i = i + 1$

Figure 1 Data exchange scheme through MUI library for Code_Saturne (left) and FEniCS (right)

Code_Saturne solver is coupled with the linear structural solver, which was developed based on the FEniCS library, in ParaSiF_CF. The elastodynamics formulation can be expressed in the form of a Generalized harmonic oscillator (GHO) equation in terms of the body fitted coordinates of the solid structure. The generalised-α method is used to achieve a second-order accuracy for the time stepping [4, 5]. The GHO equation is valid for small deformations

of the structure. MUI library is embedded in the FEniCS linear structural solver through MUI Python wrapper. The data exchange scheme for the FEniCS linear structural solver with MUI embedded is shown in the right of Figure 1, which is similar to that of Code_Saturne with MUI embedded, but in a different order.
The fluid and structural domains of a partitioned FSI approach are coupled by kinematic and dynamic conditions at the interface where the domains meet [6, 7].

3.3. Code optimisation

The initial ParaSiF_CF code is based on a staggered (serial) FSI coupling scheme, as it is simple to implement. Figure 2 shows a flow chart of ParaSiF_CF code with the staggered FSI coupling scheme at an arbitrary sub-iteration. The sidebar of the flow chart indicates the status of both solvers with traffic light colours at a specific machine time. It is clear that only one solver is running at a specific time with the staggered FSI coupling scheme. This will greatly affect the parallel performance of ParaSiF_CF.

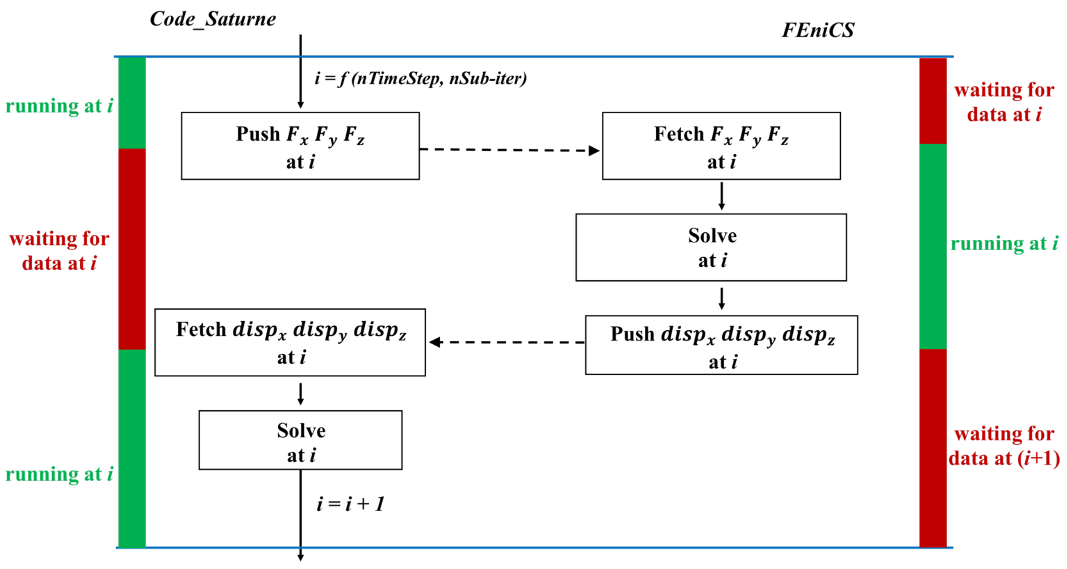
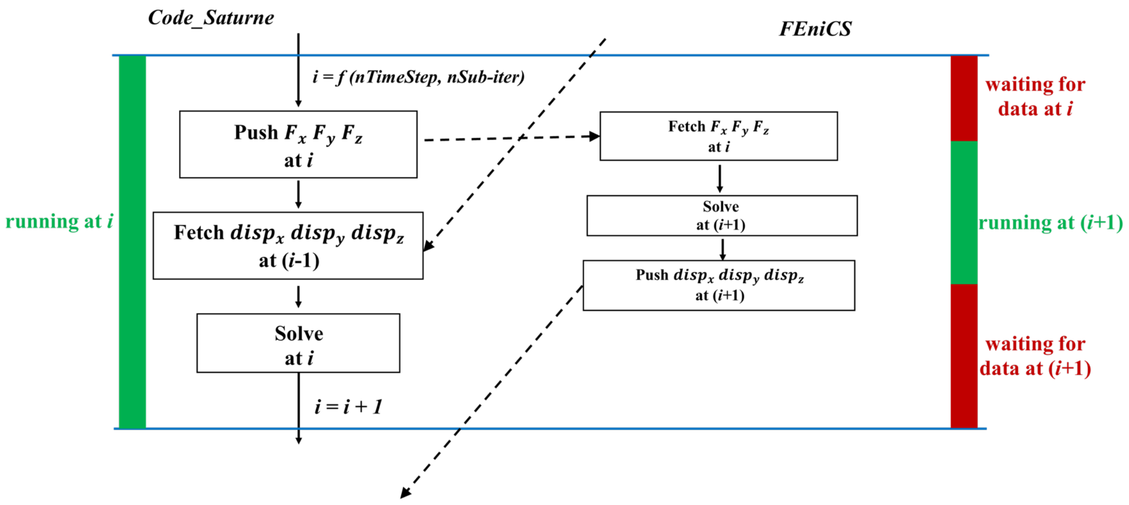

Code_Saturne
FEniCS
$i = f$ (nTimeStep, nSub-iter)
running at i
waiting for data at i
running at i
waiting for data at i
running at i
waiting for data at (i+1)
Push F_x F_y F_z at i
Fetch F_x F_y F_z at i
Solve at i
Push $disp_x$ $disp_y$ $disp_z$ at i
Fetch $disp_x$ $disp_y$ $disp_z$ at i
Solve at i
$i = i + 1$

Figure 2 Flow chart of ParaSiF_CF with staggered (serial) FSI coupling scheme.

In order to achieve a better parallel performance, the parallel FSI coupling scheme has been implemented for ParaSiF_CF. As shown in Figure 3, the parallel FSI coupling scheme allows both solvers to run at the same time, which is able to achieve optimal load balancing and parallelism. The parallel FSI coupling scheme introduces a 1st order temporal distretisation error at the interface, however, this can be effectively mitigated through subiterations at each time-step to ensure global convergence of the overall problem over both subdomains. Also note that this subiteration is generally necssary in FSI anyway for stability in cases where added mass effects are significant. Generally, there are three scenarios for the code running with the parallel FSI coupling scheme depending on the running time of each solver. The first scenario is that the fluid solver and the structure solver have the same running time per sub-iteration. This can be achieved by using the MPI barrier function to synchronize the two solvers. In the second scenario, the run time per sub-iteration of the structure solver is less than that of the fluid solver. After several time steps, it will eventually be stabilised as shown in the flow chart of Figure 3. In this scenario, the fluid solver will keep running without any waiting, since all the data required by the fluid solver will be available when needed. The structure solver will spend some time waiting for the fluid solver to send data. In this scenario, reducing the running time per sub-iteration from the structure side will not affect the total running time of the FSI framework. The only way to accelerate the framework is to reduce the running time per sub-iteration of the fluid solver. Using the MPIbarrier function to synchronise the solvers does not help getting good performance of the framework. In the third scenario, the run time per sub-iteration of the fluid solver is less than that of the structure solver. It is quite close to that shown in Figure 3, but the structure solver will keep running and the fluid solver will spend some time waiting for the structure solver to send the data over. Test cases run by ParaSiF_CF in this project with the parallel FSI coupling scheme belong to the second scenario.



Code_Saturne
FEniCS
$i = f$ *(nTimeStep, nSub-iter)*
Push F_x F_y F_z at i
Fetch $disp_x$ $disp_y$ $disp_z$ at (i-1)
Solve at i
$i = i + 1$
running at i
Fetch F_x F_y F_z at i
Solve at (i+1)
Push $disp_x$ $disp_y$ $disp_z$ at (i+1)
waiting for data at i
running at (i+1)
waiting for data at (i+1)

Figure 3 Flow chart of ParaSiF_CF with parallel FSI coupling scheme.

Apart from the implementation of the parallel FSI coupling scheme, some other optimisations and bug fixes have been done within this project for ParaSiF_CF.
We have identified a major bottleneck in the FEniCS linear solver, which is about the function to convert point-source forces into tractions for high order elements. The initial implementation is to fetch these point-source forces of each first-order element, convert the forces into tractions according to the equivalent area of each first-order element and project the traction field from the first-order elements into the high order elements. The projection between different order of elements is the source of the bottleneck. We have re-written this part with the new workaround to fetch point-source forces of each high order element and convert the forces into tractions according to the equivalent area of each high order element directly. It saves quite a lot of time on the simulation.
We have also identified a bug in the MUI library that affects the parallel performance of ParaSiF_CF. In MUI v1.1.1, the MPI_Isend will check whether the pushed data have been received by the peer solver for each communication (sub-iteration) and will block the process for 10s or more for the large bulk of data check. We provided a fix to keep this check at the last step only so that to achieve a non-block MPI_Isend.
Apart from the FSI coupling algorithms provided by the MUI utility, we further implemented a built-in version of the Fixed Relaxation FSI coupling algorithm in Code_Saturne aiming to further reduce the overhead from the MPI communication. The users can freely choose from both the built-in version and the MUI utility version according to needs.

3.4. The test case, parallel performance and profile of the FSI framework.

A 3-D FSI benchmark case has been run by using ParaSiF_CF. The test case comprises a channel flow with a parabolic velocity inlet profile passing over an elastic beam with its root fixed to the lower channel wall. Simulation parameters can be found in (6). The incoming peak velocity is 0.2 m/s and the flexible beam has a Young's modulus (E) of $1.4 \times 10^6 Pa$.

The Reynolds number based on the beam height is 40 (i.e. this case is in laminar regime). A total of 15M hexahedral cells for the fluid domain and 7,803 DOFs for the structure domain are employed. Both of these two grids were tested to ensure grid independence. The x-axis displacement (D_x) at a selected monitoring point and fluid force (f_x) acting on the beam are taken, as detailed in Tukovic *et al.* [8] and listed in Table 1, where they are compared with results from Richter [9] and Tukovic *et al.* [8]. As can be seen that the results simulated by ParaSiF_CF match well with published results. Streamlines (for the fluid domain) and displacement magnitude contour (for the structure domain) for the case is shown in Figure 4.

Table 1 Displacement of the elastic beam case compared with published results.

	D_x [m]	f_x [N]
Present Simulation	5.94×10^{-5}	1.33
Tukovic *et al.* [8]	5.93×10^{-5}	1.31
Richter [9]	5.95×10^{-5}	1.33

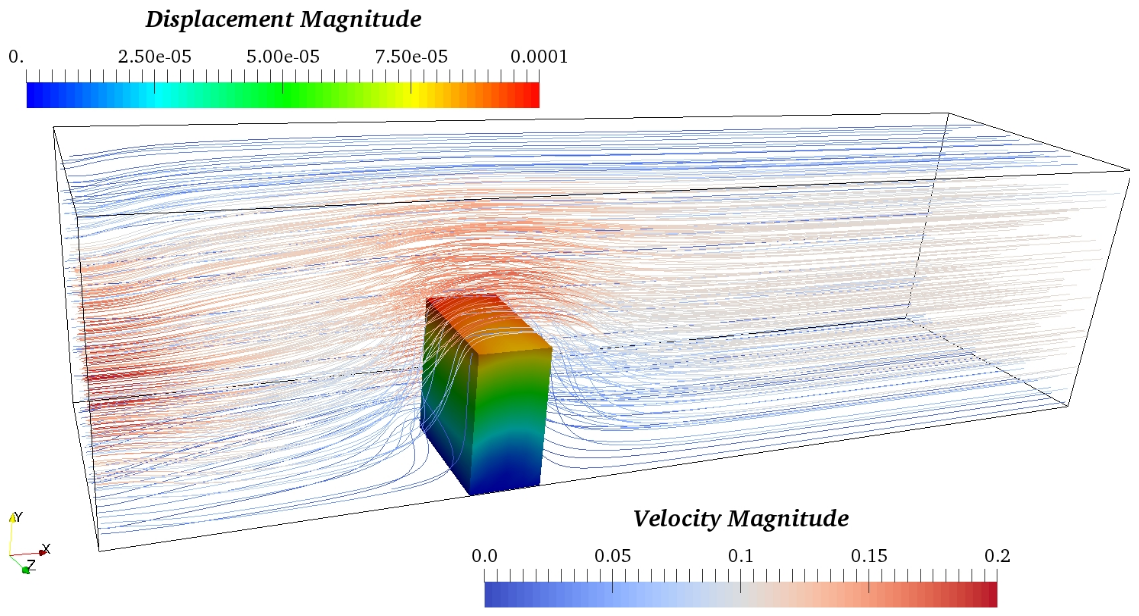


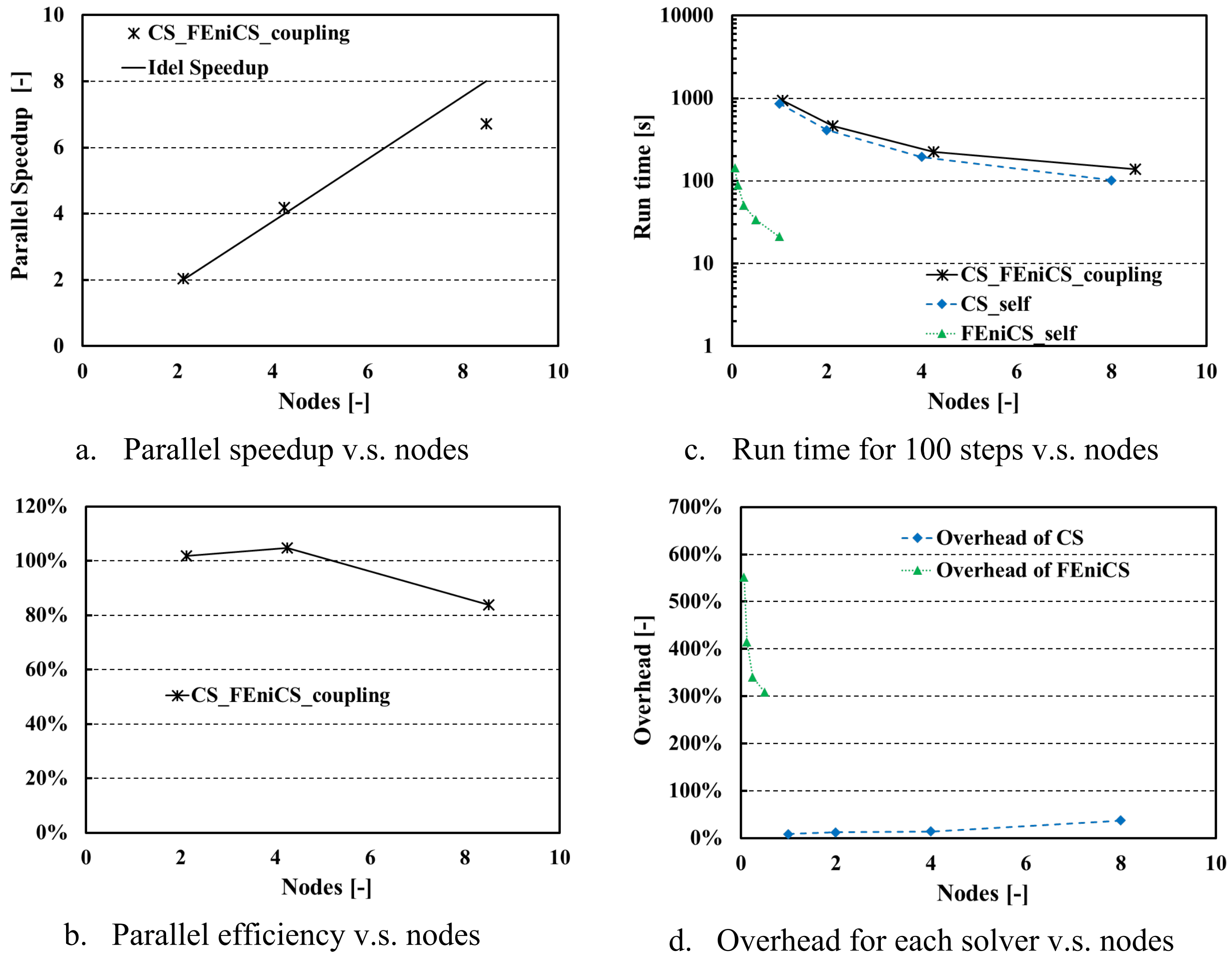
Figure 4 Streamlines (for the fluid domain) and displacement magnitude contour (for the structure domain) for the 3-D flow past an elastic beam using the linear structural solver.

In order to test the scalability of the ParaSiF_CF, we have chosen the 3-D flow past elastic beam case, but modified the incoming flow as 2m/s uniform flow and employed Large Eddy Simulation (LES) to tackle the turbulence effects. There are 15M hexahedral cells on the fluid domain. Tetrahedral elements with 390k DoFs for the structure domain are employed. The fluid and structure grids used in this scalability test ensure an approximately equal number between the cells and DoFs at the FSI coupling interface, thereby reducing any potential load imbalance during two-way communication. The first 100 time steps are used for the scalability test. One FSI sub-iteration is set per time step. Synthetic values of forces and displacements are used for the data exchange so as to ensure the results comparable with the simulation by using each individual solvers only (i.e. CS_self and FEniCS_self) and avoid the simulation divergence.
The results of this scalability test are summarised in Table 2 and Figure 5. It can be seen that the parallel efficiency is about 85% going from 2 nodes to 9 nodes in ARCHER2.
The overhead of Code_Saturne (or FEniCS) is calculated by $(t - t_{ref})/t_{ref}$, where t is the total simulation time by ParaSiF_CF and t_{ref} is the total simulation time by using

Code_Saturne (or FEniCS) by itself with the input of the same synthetic value of displacements (or forces). There are many ways to calculate the overhead, but they should draw a similar conclusion. It is noted that the overhead of FEniCS is quite large. It is due to a large bulk of time of the FEniCS solve spent on waiting as shown in Figure 3 and discussed in Section 3.3.
With all the optimisations and bug fixes, we found an execution time reduction of 57% and parallel efficiency increase from 68% to about 85% going from 2 nodes to 9 nodes as a direct result of swapping the CFD solver from OpenFOAM (https://github.com/parMupSiF) to Code_Saturne on the coupled FSI framework.

Table 2 Performance of ParaSiF_CF with linear structure solver for 15M cells on CFD domain and 390k DoFs on CSM domain.

Nodes total [-]	Nodes Code_Saturne [-]	Nodes FEniCS [-]	Averaged time per time step [s]	Parallel speedup [-]	Parallel efficiency [%]	Ideal speedup [-]	Overhead of Code_Saturne [%]	Overhead of FEniCS [%]
1.0625	1	0.0625	9.37	-	-	-	8.36	551.30
2.125	2	0.125	4.60	2.03	101.69	2	11.95	414.99
4.25	4	0.25	2.23	4.19	104.81	4	14.22	339.98
8.5	8	0.5	1.40	6.71	83.87	8	37.17	308.51



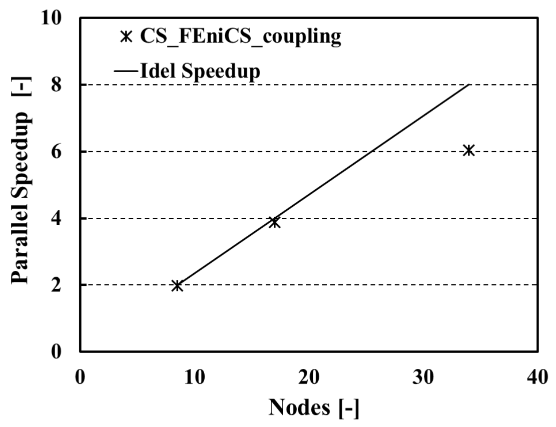
a. Parallel speedup v.s. nodes
c. Run time for 100 steps v.s. nodes
b. Parallel efficiency v.s. nodes
d. Overhead for each solver v.s. nodes

Figure 5 Parallel speedup (a), parallel efficiency (b), total run time for 100 time steps (c) and overhead for each solver (d) on ParaSiF_CF using a linear structural solver for 15M cells on CFD domain and 390k DoFs on CSM domain.

We have simulated the same LES FSI case with larger grids for both domains. There are 233M hexahedral cells on the fluid domain and 2.7M DoFs on the structure domain. The fluid and structure grids used in this test also ensure an approximately equal number between the cells and DoFs at the FSI coupling interface for load balancing. The results of this scalability test are summarised in Table 3 and Figure 6. The parallel efficiency is about 75% going from 5 nodes to 34 nodes in ARCHER2. Since the present test uses the default decomposition method in both Code_Saturne and FEniCS, improved parallel performance should be expected by carefully decomposing both domains.

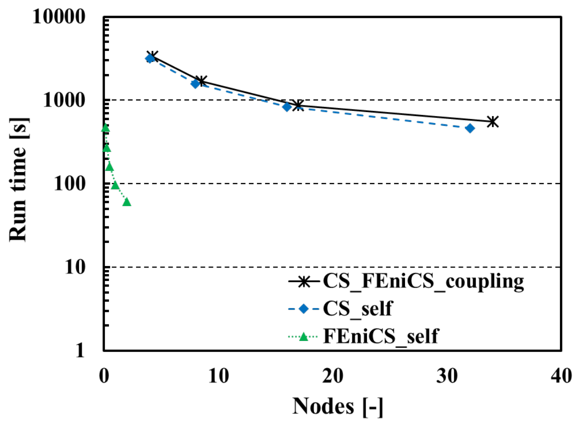
Table 3 Performance of ParaSiF_CF with linear structure solver for 233M cells for the CFD domain and 2.7M DoFs for the CSM domain.

Nodes total [-]	Nodes Code_Saturne [-]	Nodes FEniCS [-]	Averaged Time per time step [s]	Parallel speedup [-]	Parallel efficiency [%]	Ideal Speedup [-]	Overhead of Code_Saturne [%]	Overhead of FEniCS [%]
4.25	4	0.25	33.40	-	-	-	5.48	601.02
8.5	8	0.5	16.91	1.98	98.79	2	7.21	522.09
17	16	1	8.62	3.87	96.83	4	3.24	432.08
34	32	2	5.54	6.03	75.43	8	19.64	465.85

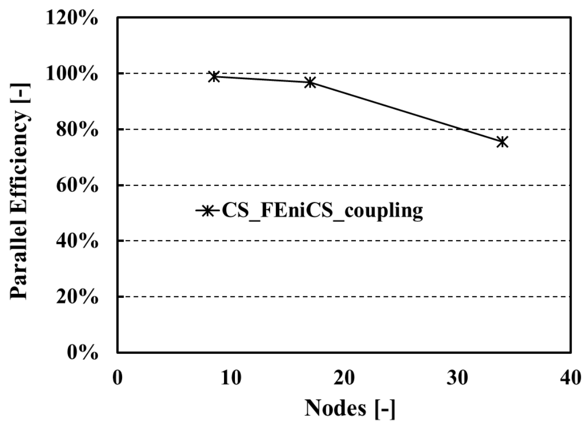


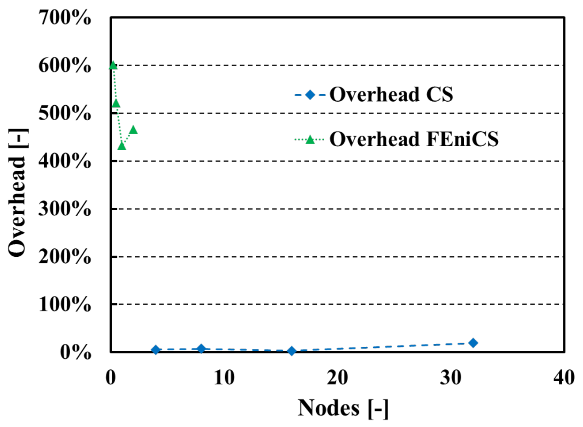
a. Parallel speedup v.s. nodes

b. Parallel efficiency v.s. nodes



c. Run time for 100 steps v.s. nodes





d. Overhead for each solver v.s. nodes

Figure 6 Parallel speedup (a), parallel efficiency (b), the total run time for 100 time steps (c) and overhead for each solver (d) on ParaSiF_CF with a linear structural solver for 233M cells for the CFD domain and 2.7M DoFs for the CSM domain.

To understand the drop in the parallel efficiency at the highest node counts, profiling of the code is performed.
We use the Cray Performance Analysis Tool (CrayPAT) to profile the CFD (Code_Saturne) part of ParaSiF_CF, as it is able to provide detailed information on the timing and performance of individual application procedures and is easy to use in ARCHER2. Table 4 summarises the results of the profiling for about 57K cells per MPI task. It shows that the MUI sampler is only ~8%. While the communication cost through MPI is 12%.

Table 4 Profiling of the CFD (Code_Saturne) part of ParaSiF_CF using CrayPat

Function	% Samples
MPI	12.0
CFD Solver work	80.2
MUI sampler	7.8

We further use the Tuning and Analysis Utilities (TAU) to profile the CSM (FEniCS) part of ParaSiF_CF, as it is able to handle Python-based solvers. Since we are interested in the overhead brought by MUI and relevant functions instead of the waiting/barrier shown in Figure 3, we use a synthetic solver as the CFD side of ParaSiF_CF to minimise the influence of a MPI_Barrier. Figure 7 shows the results of the profiling for 2.7M DoFs over 2 nodes. The iterative based linear solver with Generalized minimal residual method has the highest cost. MUI related works, such as the function of fetch_many and mpi_split_by_app, have low costs (~10%) overall.

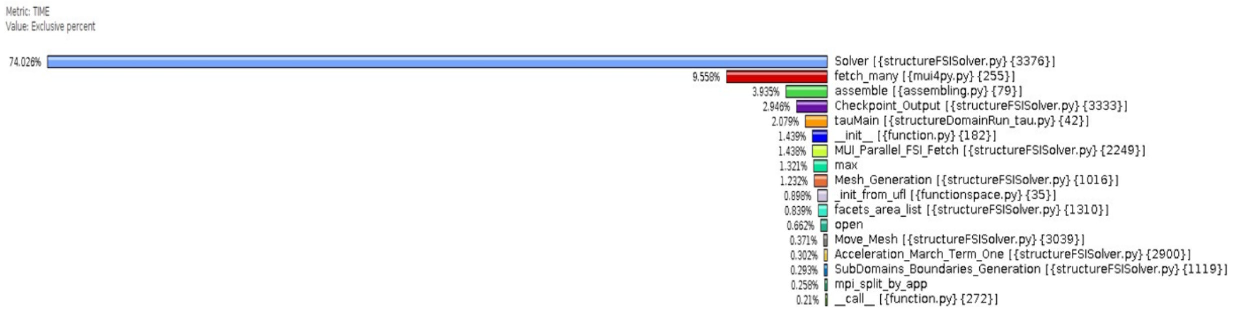


Figure 7 Profiling of the CSM (FEniCS) part of ParaSiF_CF with linear structure solver by TAU

4. Code coupling between Code_Saturne and FEniCS with MUI for nonlinear structure deformation (WP3).

We further coupled the CFD code, Code_Saturne (V6.0.6), with the FEniCS (V2019.1.0) based non-linear structural solver, through the MUI (V1.1.3) toolbox, to simulate large deformations of the structure. The non-linear structural solver is governed by the elastodynamics formulation with the force balance of linear momentum. The stress tensor is expressed by both the hyper-elastic St. Vernant-Kirchhoff model as well as Hooke's law [6]. Hooke's law is limited to small deformations of the structure, while the St. Vernant-Kirchhoff model can handle large deformations but remains valid for small strains [10, 11]. The one-step Θ scheme is implemented for time-stepping of both the Hooke's law and the Saint Vernant-Kirchhoff model. The CFD part of the ParaSiF_CF as well as the two-way coupling scheme between the Code_Saturne and FEniCS remains the same as discussed in Section 3. The non-linear structural solver also inherited part of the optimised codes from Section 3.3.

4.1. Presentation of the test case, parallel performance and profiling of the FSI framework

To demonstrate the performance of the ParaSiF_CF with the non-linear structural solver on solving large structure deformation cases, the 3-D benchmark case simulated in Section 3.4 has been modified by employing a softer beam, which Young's Modulus equal to $1 \times 10^5 Pa$, with a peak flow velocity of $2.0\ m/s$. All other conditions are kept the same. A total of 15M hexahedral cells on the fluid domain and 7,803 DOFs for the structural domain are employed. Figure 8 shows the streamlines of the fluid domain and displacement magnitude contour on the structure domain for the case, which the x-axis displacement (D_x) at the selected monitoring point is 0.015m (0.15 times of its width). It proved that the coupling was successful.

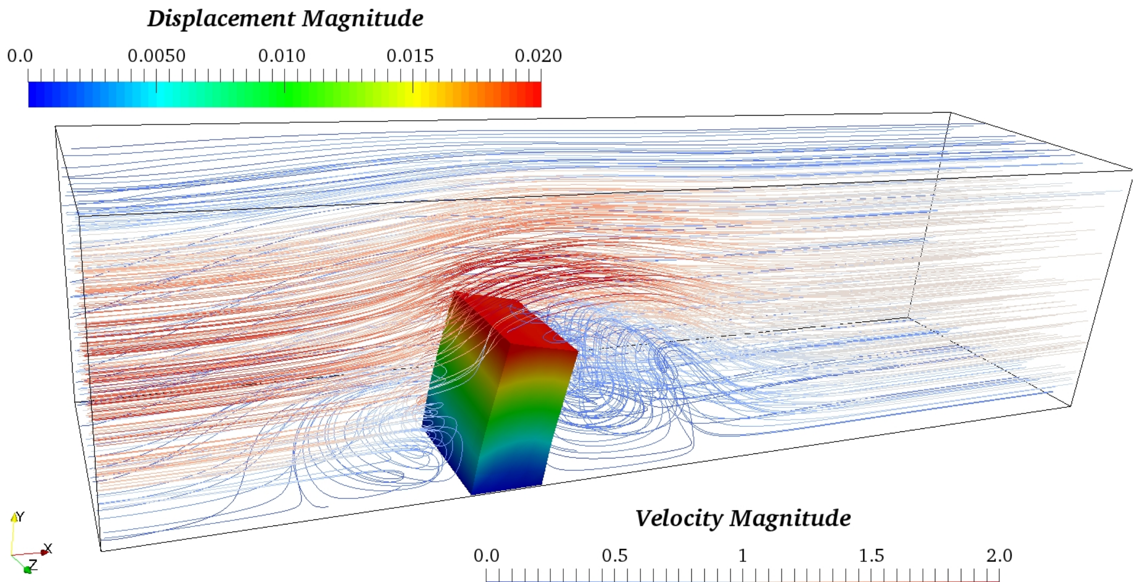
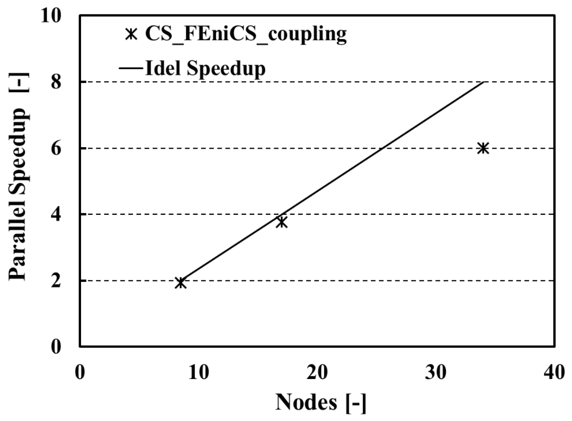


Figure 8 Streamlines (for the fluid domain) and displacement magnitude contour (for the structure domain) for the 3-D flow past elastic beam case with non-linear structure solver.

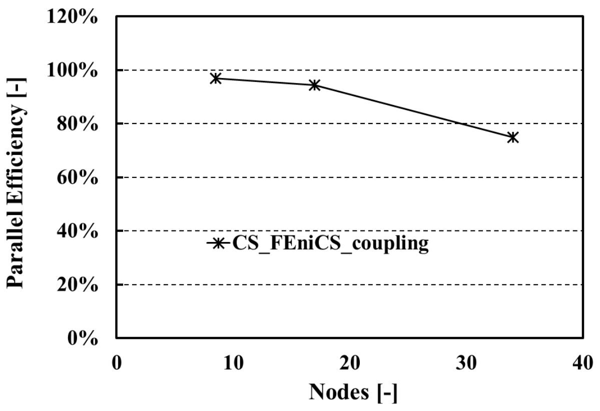
We have simulated this case with larger grids for both domains. There are 233M hexahedral cells on the fluid domain and 2.7M DoFs on the structure domain. The results of this scalability test are summarised in Table 5 and Figure 9. The parallel efficiency is about 75% going from 5 nodes to 34 nodes in ARCHER2. As discussed in Section 3.4, a better result on the parallel performance could be expected by carefully decomposing both domains in the near future, and using different mesh partitioners.

Table 5 Performance of ParaSiF_CF using a non-linear structural solver for 233M cells for the CFD domain and 2.7M DoFs for the CSM domain.

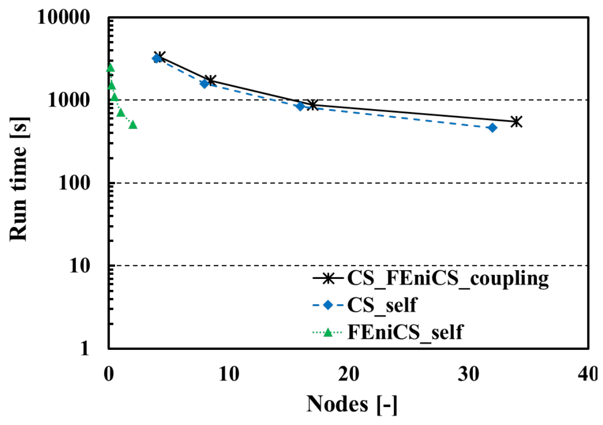
Nodes total [-]	Nodes Code_Saturne [-]	Nodes FEniCS [-]	Averaged Time per time step [s]	Parallel speedup [-]	Parallel efficiency [%]	Ideal Speedup [-]	Overhead of Code_Saturne [%]	Overhead of FEniCS [%]
4.25	4	0.25	33.27	-	-	-	5.07	118.10
8.5	8	0.5	17.15	1.94	97.03	2	8.73	55.88
17	16	1	8.81	3.77	94.36	4	5.53	23.47
34	32	2	5.54	6.00	75.02	8	19.81	8.57

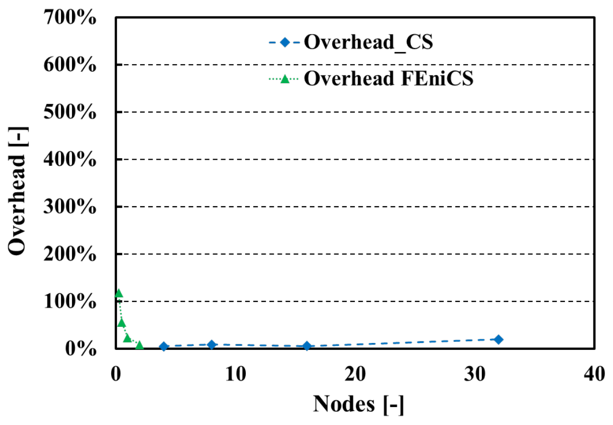


a. Parallel speedup v.s. nodes



b. Parallel efficiency v.s. nodes

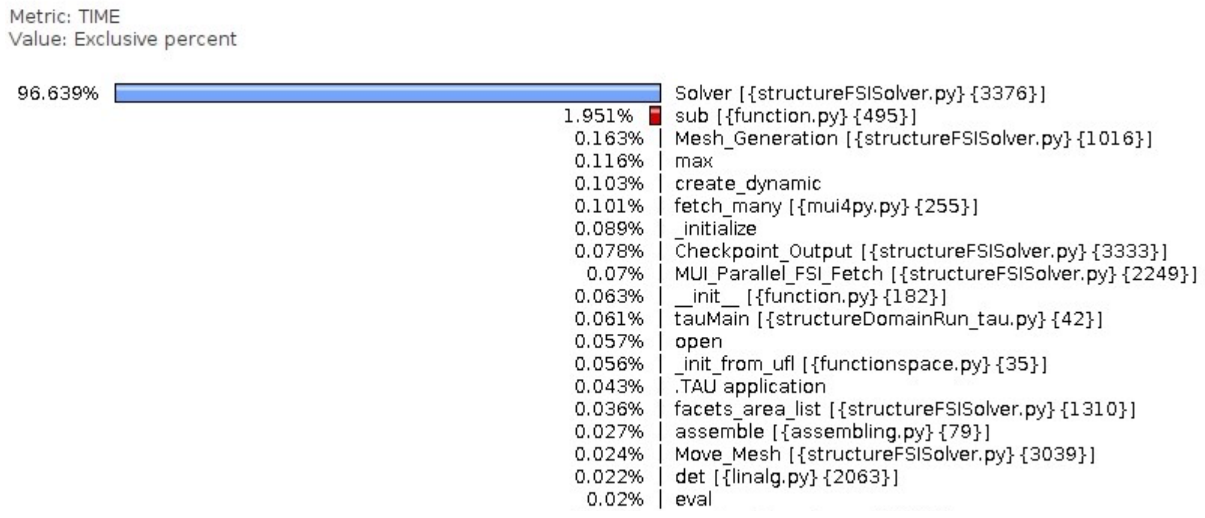




c. Run time for 100 steps v.s. nodes d. Overhead for each solver v.s. nodes

Figure 9 Parallel speedup (a), parallel efficiency (b), total run time for 100 time steps (c) and overhead for each solver (d) on ParaSiF_CF using the non-linear structural solver for 233M cells for the CFD domain and 2.7M DoFs for the CSM domain.

We further use TAU to profile the non-linear CSM (FEniCS) solver of ParaSiF_CF. We use a synthetic solver as the CFD side of ParaSiF_CF to minimise the MPI_Barrier. Figure 10 shows the results of the profiling for 2.7M DoFs over 2 nodes by the non-linear structural solver. The iterative-based nonlinear solver with the Scalable Nonlinear Equations Solvers (SNES) approach shows the highest cost (~97%). MUI related works have very low costs (~0.5%).



```
Metric: TIME
Value: Exclusive percent

96.639%  Solver [{structureFSISolver.py} {3376}]
 1.951%  sub [{function.py} {495}]
 0.163%  Mesh_Generation [{structureFSISolver.py} {1016}]
 0.116%  max
 0.103%  create_dynamic
 0.101%  fetch_many [{mui4py.py} {255}]
 0.089%  _initialize
 0.078%  Checkpoint_Output [{structureFSISolver.py} {3333}]
  0.07%  MUI_Parallel_FSI_Fetch [{structureFSISolver.py} {2249}]
 0.063%  __init__ [{function.py} {182}]
 0.061%  tauMain [{structureDomainRun_tau.py} {42}]
 0.057%  open
 0.056%  _init_from_ufl [{functionspace.py} {35}]
 0.043%  .TAU application
 0.036%  facets_area_list [{structureFSISolver.py} {1310}]
 0.027%  assemble [{assembling.py} {79}]
 0.024%  Move_Mesh [{structureFSISolver.py} {3039}]
 0.022%  det [{linalg.py} {2063}]
  0.02%  eval
```

Figure 10 Profiling of the CSM (FEniCS) part of ParaSiF_CF using the non-linear structural solver, using TAU.

5. Availability, sustainability and usability of code (WP4).

The ParaSiF_CF code is accessible to all ARCHER2 users and the entire community through the GitHub project (https://github.com/ParaSiF-CF), incorporating both linear (small deformation) and nonlinear (large deformation) structural solvers with their tutorial cases and supporting documentation. Due to the partitioned nature of this framework, several parts of the code (MUI library, FSI coupling utility, structural solver part and fluid solver part) have been released under different repositories but collected into one GitHub project. Each part of the code (repository) has its own license. The MUI library and its RBF spatial sampler are already available on GitHub (https://github.com/MxUI/MUI). The details of each ParaSiF_CF component on their license, availability and maintenance plan are as follows.

-Code_Saturne V6.0 (V6.0.6) is open-source, licensed by the GNU General Public License (GPL) version 2, developed and maintained by EDF R&D. The developed interface between the Code_Saturne and MUI, in which the Code_Saturne core code is the dependency of the interface, has been included in the public repository on GitHub with GPLv2. The Code_Saturne-MUI interface developed by the project team will be integrated into the official release of Code_Saturne with MUI incorporated as a third-party library and maintained by EDF R&D in the future, as agreed between the ParaSiF_CF team and the Code_Saturne's main developer.
-FEniCS V2019.1.0 is licensed by the GNU Lesser General Public License (LGPL) version 3. It is developed and maintained by the FEniCS Steering Council and is overseen by the FEniCS Advisory Board. Both linear and nonlinear structure solvers with MUI embedded and use FEniCS V2019.1.0 as the dependency have been included in the public repository on GitHub under LGPLv3. The developed structure solvers will be maintained by the project team. Future upgrades of the MUI embedded structure solvers (such as port the developed solvers to the latest FEniCS-X) will be handled by the project team and community through the project repository on GitHub.
-MUI is maintained by STFC and is dual-licensed under both the GNU General Purpose License (GPL) version 3 and Apache License, version 2.0. Bug fixes and modifications of the MUI library during this eCSE project have been merged to the MUI official release (MUI V1.1.3) and maintained by the STFC MUI team after the project.

6. Conclusions

We have developed a new massively parallel partitioned FSI simulation framework, ParaSiF_CF, in order to prepare for Exascale FSI simulations. This FSI framework employs Code_Saturne v6.0.6 to solve the fluid field, FEniCS v2019.1.0 to solve the structure field and MUI v1.1.3 library for data exchange. ParaSiF_CF takes advantage of leveraging the highly specialised legacy codes that have had decades invested in and are tailored to the task they are designed to solve. Both linear elastic and hyper-elastic structural solvers have been implemented in the framework on the simulation of both small and large structure deformations. After some optimisations and bug fixes, the simulation time of ParaSiF_CF is reduced by about 57% in terms of execution time compared with that of the existing FSI simulation framework (coupling between OpenFOAM and FEniCS through MUI). About 85% of parallel efficiency on the 3-D FSI simulation using a linear structural solver is achieved for 15M cells for the CFD domain and 390k DoFs for the CSM domain going from 2 nodes to 9 nodes on ARCHER2. About 75% of parallel efficiency on the 3-D FSI simulation with both linear and nonlinear structural solvers are achieved for 233M cells for the CFD domain and 2.7M DoFs for the CSM domain going from 5 nodes to 34 nodes on ARCHER2. Future works on porting the existing ParaSiF_CF structure solvers to the latest FEniCS-X release have been identified and will be targeted in the near future.

Acknowledgement

This work was funded under the embedded CSE programme of the ARCHER2 UK National Super-computing Service (https://www.archer2.ac.uk/ecse/).

References

1. Fournier Y, Bonelle J, Moulinec C, Shang Z, Sunderland A, Uribe J. Optimizing Code_Saturne computations on Petascale systems. Computers & Fluids. 2011;45(1):103-8.
2. Hoffman J, Jansson J, Jansson N, editors. FEniCS-HPC: Automated predictive high-performance finite element computing with applications in aerodynamics. International Conference on Parallel Processing and Applied Mathematics; 2015: Springer.
3. Skillen A, Longshaw S, Cartland-Glover G, Moulinec C, Emerson D, editors. Profiling and application of the multi-scale universal interface (MUI). 6th European Conference on Computational Mechanics (ECCM 6); 2018.
4. Bleyer J. Numerical tours of computational mechanics with FEniCS. Zenodo. 2018.
5. Erlicher S, Bonaventura L, Bursi OS. The analysis of the generalized-α method for non-linear dynamic problems. Computational Mechanics. 2002;28(2):83-104.
6. Liu W, Wang W, Skillen A, Longshaw S, Moulinec C, Emerson D, editors. A Parallel Partitioned Approach on Fluid-Structure Interaction Simulation Using the Multiscale Universal Interface Coupling Library. 14th WCCM-ECCOMAS Congress 2020; 2021.
7. Liu W, Longshaw SM, Skillen A, Emerson DR, Valente C, Gambioli F. A High-Performance Open-Source Solution for Multiphase Fluid-Structure Interaction. International Journal of Offshore and Polar Engineering. (in press).
8. Tuković Ž, Karač A, Cardiff P, Jasak H, Ivanković A. OpenFOAM finite volume solver for fluid-solid interaction. Transactions of FAMENA. 2018;42(3):1-31.
9. Richter T. Goal-oriented error estimation for fluid–structure interaction problems. Computer Methods in Applied Mechanics and Engineering. 2012;223:28-42.
10. Slyngstad AS. Verification and Validation of a Monolithic Fluid-Structure Interaction Solver in FEniCS. A comparison of mesh lifting operators 2017.
11. Razzaq M, Turek S, Hron J, Acker J, Weichert F, Grunwald I, et al. Numerical simulation and benchmarking of fluid-structure interaction with application to hemodynamics. Fundamental Trends in Fluid-Structure Interaction: World Scientific; 2010. p. 171-99.