Towards Virtual Certification of Gas Turbine Engines With Performance-Portable Simulations

Gihan R. Mudalige*, Istvan Z. Reguly†, Arun Prabhakar*, Dario Amirante‡, Leigh Lapworth§, Stephen A. Jarvis¶
*University of Warwick, UK. {g.mudalige, arun.prabhakar}@warwick.ac.uk
† Pazmany Peter Catholic University, Hungary. reguly.istvan@itk.ppke.hu
‡University of Surrey, UK. d.amirante@surrey.ac.uk
§Rolls-Royce plc., UK. leigh.lapworth@Rolls-Royce.com
¶University of Birmingham, UK. s.a.jarvis@bham.ac.uk

***Abstract*—We present the large-scale, computational fluid dynamics (CFD) simulation of a full gas-turbine engine compressor, demonstrating capability towards overcoming current limitations for virtual certification of aero-engine design. The simulation is carried out through a performance portable code-base on multi-core/many-core HPC clusters with a CFD-to-CFD coupled execution, combining an industrial CFD solver linked using custom coupler software. The application innovates in its design for performance portability through the OP2 domain specific library for the CFD components, allowing the automatic generation of highly optimized platform-specific parallelizations for both multi-core (CPU) and many-core (GPU) clusters from a single high-level source. The code is used for the simulation of a 4.58B node, full-annulus 10-row production-grade test compressor (DLR's Rig250), using a coupled sliding-plane setup on the ARCHER2 and Cirrus supercomputers at EPCC. The OP2 generated multiple parallelizations, together with optimized coupler configurations on heterogeneous/hybrid settings achieve, for the first time, execution of 1 revolution in less than 6 hours on 512 nodes of ARCHER2 (65k cores), with a parallel scaling efficiency of over 80% compared to a 107 node run. Results indicate a speed up of the CFD suite by an order of a magnitude ($\approx 30\times$) relative to current production capability. Benchmarking and performance modelling project a time-to-solution of less than 5 hours on a cluster of $488\times$NVIDIA V100 GPUs, about $3\times$–$4\times$ speedup over CPU clusters. The work demonstrates a step-change towards achieving virtual certification of aircraft engines with the requisite fidelity and tractable time-to-solution that was previously out of reach under production settings.**

***Index Terms*—DSL, Code Coupling, Virtual Certification, Gas Turbine Engines**

I. Introduction

The design of aero engines, and their continued developments for improved efficiency, rely crucially on simulation and modelling enabled by high performance computing. However, next generation engine designs will place demands on computational simulation that cannot be met by incremental changes to current techniques. Challenges not only include engine efficiency demands in terms of performance, but also emissions and pollution goals to meet environmental commitments. Global Net Zero commitments such as [1], require the use of sustainable aviation fuels and electric and hybrid-electric flight. Governmental targets such as the EU ACARE Flightpath 2050 goals [2] demand fundamental changes to engine architectures to meet these challenges, and "virtual certification" of engine designs is recognised as a key technology to deliver the required performance at this level.

A crucial component of virtual certification is the whole engine test to certify operational performance and thrust. The speed, fidelity and accuracy required for such a large, multi-component simulation, for production turbo-machinery are well beyond current simulation and high-performance computing capability. True virtual certification simulations will require new ultra-high resolution physical models and full system simulations that drive us from today's model sizes with 10-100 million mesh elements towards hundreds of billions of elements. Completion of such models in tractable times then critically depends on exploiting emerging and future massively parallel computing platforms at extreme levels of scale. As such, simulation software needs to be performance portable and agile to efficiently utilize novel architectures in a rapidly changing hardware landscape, capable of ultra-large scale executions, while at the same time provide unprecedented levels of trust in the accuracy of the simulations.

A key part of the full engine simulation will be devoted to computing the flow of air through the engine. In a gas turbine engine (see Fig. 1), cold air entering the engine at atmospheric pressure is compressed in the order of 50:1 before it is delivered to the combustion chamber, where it is sprayed with fuel and ignited. The hot air exhaust from the resulting combustion provides the thrust that drives the turbines which, in turn, spins the compressor and fan. While the majority of thrust that propels the aircraft is produced by the fan, the hot air exhaust after combustion also contributes to the total thrust.

Flow simulation tools adopted within the industry rely heavily on the steady RANS (Reynolds-Average Navier-Stokes) models [3] based on the solution to the Navier-Stokes PDEs. Here the flow is assumed to be steady, and circumferential averaging is enforced at the interfaces between the blade rows. RANS models do not consider the unsteady rotor-stator interaction and can be limited to one blade passage for each row. Extension to Unsteady-RANS (URANS) is known to significantly improve performance predictions [4] but raises the mesh requirements by two orders of magnitude for a full annulus model. These refer to a full 360 degree domain for a particular row of blades (such as a rotor or a stator), and is generated from a single blade for the associated row,

based on the number of blades required to achieve a complete 360 degree domain. Use of smaller sectors is limited by the condition that adjacent rotor/stator zones must have the same sector angle, and in most cases, if not all, this requires an alteration of the geometric pitch [5] which obviously introduces an approximation error. To complicate matters, unsteady simulations require methods to model the relative motion between rotating and stationary parts of the mesh. These techniques are, in general, not scalable [6]–[8]. All these conditions render the cost of unsteady multi-row simulations prohibitively high, and represent a key barrier on the path towards virtual certification of engine designs.

This paper provides a contribution to overcome this limit, for a problem size sufficient to meet the requirements for virtual certification, for the first time achieved with a code scaling to give a tractable time-to-solution. More specifically, we present the simulation of a full industrial compressor setup, the Rig250 from Deutsches Zentrum für Luft- und Raumfahrt (DLR) [9] (see Fig. 2). The test case encompasses some of the challenges outlined above. The full compressor consists of a number of blade-rows, which we simulate with Rolls-Royce's Hydra CFD suite [10], coupled to the next blade row using novel coupler software, leading to 10 rotor-stator interfaces. URANS simulations are conducted for a 360 degree model, with the largest mesh size consisting of 4.58 billion elements. Computations have been carried out on the ARCHER2 and Cirrus supercomputers at the Edinburgh Parallel Computing Centre (EPCC), achieving the grand challenge problem of 1 compressor revolution in less than 6 hours, when running on 512 nodes (2×64 cores per node) of ARCHER2. This represents a $30\times$ improvement over current production capability, which is based on software that uses a non-coupled monolithic execution of the Hydra CFD suite and prior-generation hardware (Haswell/Broadwell). Performance on the Cirrus GPU cluster indicates a speedup of approximately $3.4\times$ over the ARCHER2 performance on a *power-equivalent* number of nodes, for problems fitting in GPU memory. Projected speedups on larger GPU clusters indicate a $3-3.5\times$ speedup over the ARCHER2 (CPU) performance for the full compressor achieved within the same power envelope ($\approx$ 122 nodes with $4\times$V100 GPUs per node).

This is a step-change in capability for the prohibitively expensive simulation of a full gas-turbine engine compressor and is a first for this domain. Two key factors lie behind the significant reduction in the simulation costs: (1) OP2 [11], [12], the domain specific library which automatically generates highly optimized and scalable, platforms-specific parallelization for both multi-core (CPU) and many-core (GPU) clusters, and (2) JM76, the coupler software used to couple the flow solutions computed by Hydra in different blade rows. In this work we demonstrate how these applications and configurations are used to overcome performance and scaling challenges to realize this large-scale simulation. The work details how the current prohibitive simulation costs can be significantly reduced, enabling industrial developers to carry out tractable and agile design explorations towards virtual certification, through increasingly cost-effective HPC systems.

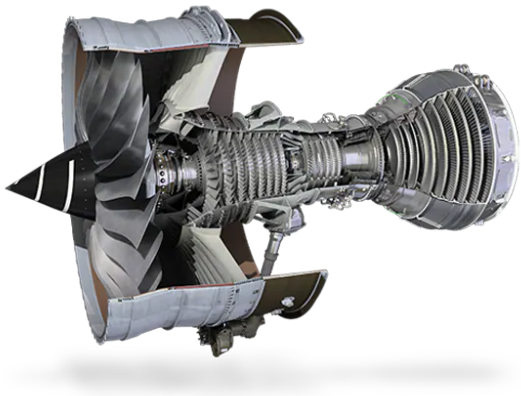


Fig. 1: RR Trent XWB Engine (©Rolls-Royce plc. Reproduced with Permission)

This paper is organized as follows: Section II details the background of the problem domain and the current state-of-the-art HPC solutions in this area. The key contributions of our work is presented in Section III, followed by performance results from the simulations which are found in Section IV. We discuss related work in Section V. Conclusions and future work are presented in Section VI.

II. Background

The standard numerical algorithms used in industrial design with CFD is based on the solution of the Navier-Stokes equations, a broadly accepted mathematical model to describe the motions of turbulent flow. The Reynolds-Averaged Navier-Stokes (RANS) approach is the most commonly used in industry due to its low cost. However, RANS models have some well known limitations [13] and advanced techniques such as Unsteady RANS (URANS), Large Eddy Simulations (LES), Hybrid RANS-LES and Direct Numerical Solutions (DNS) are used for cases where RANS models are inadequate. Of course, simulations based on these techniques becomes prohibitively expensive for industrial design with DNS attempted only for the simplest of flows, usually found in academic studies [9].

Navier-Stokes equations are essentially PDEs and the majority of industrial production CFD applications use finite volume (FV) methods for their solution. Given the need to model complex physical geometries, highly detailed mesh typologies are essential for achieving the requisite accuracy. Consequently, unstructured-mesh-based solutions are preferred over structured-mesh designs. This is a feature common to the majority of commercial or industrial solvers, such as Ansys Fluent [14], Code_Saturn [15] and Hydra [16], [17], the last of which we use in our current study.

The key computation-communication pattern for the unstructured-mesh motif [18] is indirect accesses, particularly indirect increments that require special handling of data races when parallelizing. For example, a parallel iteration over mesh edges, incrementing some value on nodes connected to each edge, would lead to multiple edges updating the same node simultaneously. Depending on the parallel architecture, a range

of different techniques are required to handle such data races and the orchestration of computation and communication. These techniques are well documented with the best optimizations and the resulting performance for each demonstrated previously [19]–[22].

Typical production codes are designed to target a single parallel architecture type, usually employing low-level languages such as C/C++ or Fortran, implementing one of the data-race resolving techniques appropriate for the parallel platform or parallelization model. Such applications, while performant, have limited portability across different hardware. More importantly they lack *performance portability* - the ability for a single code-base to achieve good performance on systems belonging to a range of architecture types, without significant manual modifications. If multiple architecture support is available, with each providing good performance, it is often through separate code bases. In this case, significant effort is required to maintain code and target new hardware, particularly if the code-base runs to hundreds of thousands or millions of lines of code. In some cases, code will need to be entirely rewritten in a new programming model if it is to best utilize novel hardware. Consequently, such static single code-bases severely limit a production applications' ability to exploit modern massively parallel and emerging heterogeneous systems, which are essential for achieving industrial virtual certification goals. This problem continues to intensify in the drive to deploy exascale HPC systems, with a rapidly changing hardware landscape competing to deliver ExaFlop/s performance.

A. High-Level Abstractions and DSLs

As a solution to the above, the idea of *separation of concerns* for achieving performance portability has been gaining widespread use in the HPC community. The use of domain specific languages (DSLs), C++ template libraries (e.g. Kokkos [23] and Raja [24]) and similar high-level abstractions allows the separation of the science source – what is to be computed, from its parallel implementation – how to program the hardware. For unstructured mesh applications, several frameworks have demonstrated how performance portability can be achieved on a wide range of hardware architectures. One of the earliest of these frameworks is OP2 [11] which we use in this paper. Other DSLs include FeniCS [25], Firedrake [26] and PyFR [27] all of which provide a higher-level notation for declaring problems and employ automatic code generation to produce concrete parallel executables. Other notable DSLs, aimed at different domains include Devito [28] for the solution of finite difference problems on structured meshes, particularly arising in seismic inversion problems, STELLA [29] (and its successor GridTools) and PSyclone [30], [31] for weather/climate modelling and OPS [32] for multi-block structured mesh applications.

B. Code Coupling

While DSLs provide hardware and parallel programming abstractions, separating a monolithic simulation code to its components and *coupling* them (usually interfaced via a separate code) has also gained traction. This strategy is especially advantageous for simulating complex multi-physics phenomena allowing the utilization of optimal methods for modelling the physics of each component. For the domain scientists, this provides the flexibility to select, for instance, the best numerical method and problem scale for each component, or even parallelization/target architecture to execute the components. It also simplifies code maintenance and extension. We view coupling as implementing essentially a *horizontal* separation of concerns [33], where expertise in developing different simulation models for different domains can be leveraged to gain the best results.

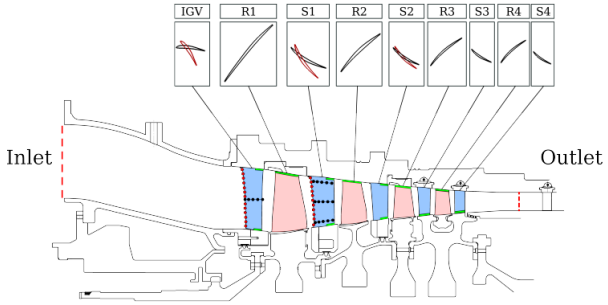


Fig. 2: Rig250 Schematic. Blue – stationary zones, Pink – rotating zones (reproduced with permission [9])

Code coupling is not a new technique and already there exist a number of general frameworks for implementing the information exchange between discrete pieces of simulation codes. Data communicated between codes via a coupler could have a direct correlation between the connected interfaces or indeed would require some interpolation or connectivity search if for example mesh elements are not aligned. General frameworks such as MUI [34] and preCICE [35] act purely as an interface where data can be sent and retrieved. Others such as MCT [36], are more involved, with dedicated classes for data fields and methods for interpolation and other transformations. The coupler framework, named JM76, used in our work operates at a much lower level, with the communication and interpolation specifically coded for the problem and routines involved in the coupling. It most closely resembles the OpenPALM framework, developed in-part by CERFACS [37]. In the current study we utilize JM76 for implementing a sliding planes interface between CFD simulations for modelling the full Rig250 compressor.

C. Rig250 Compressor and Sliding-Planes

Rig250 is a 4.5 stage test rig assembled at DLR [9] (see Fig. 2), modelling the high-pressure compressor in gas turbine engines [38]. It consists of four rotor-stator stages followed by an outlet guide vane at the exit, for a total of 9 distinct fluid zones. To handle the relative motion between stator and rotor passages, the flow equations in each zone are solved in a relative frame of reference. While Rig250 represents an industrial case and has been widely utilized in prior work [39]–[41], it is particularly of interest in the work presented in this

paper as it contains a large number of grid zones in relative motion.

In order to account for the unsteady coupling between rotors and stators, it is necessary to employ appropriate strategies for the corresponding interfaces. Sliding plane techniques are the de-facto method used in turbomachinery unsteady simulations for such interfaces. Here, in a pre-processing stage, the two meshes, forming the interface, are extruded to form an additional layer of halo nodes on both sides of the interface. The two layers of halo nodes form a one-cell overlap with the mesh of the adjacent zone. During the computation, the flow variables computed in one zone are interpolated, after appropriate rotation, to set the flow variables on the halo nodes of the adjacent zone. As the relative position between the zones varies with time, it is necessary to identify at any time step, via a search, the donor element of each target node.

However, sliding planes are difficult to optimize for parallel efficiency [8]. Typical partitioning tools, such as Metis or the Recursive Bisection method, optimise the workload related to the discretization of flow equations, i.e. the construction of Navier-Stokes equations and the subsequent time advancement. The extra cost associated with sliding plane nodes may result in a workload imbalance that is difficult to compensate for. As more processors are used, the ratio between the sliding plane to the discretization workload increases significantly, since the sliding planes nodes remain "trapped" in a limited number of processors. The configuration does not allow for increasing/decreasing the number of processes, as the assignment is done based on the global partitioning. As a result, in large models consisting of several rows, the sliding planes become the main bottle-neck to achieve good scaling performance. Note that this is the configuration adopted by the most popular commercial CFD solvers as well as the current production version of Hydra. Throughout the rest of the paper, this configuration will be referred to as the "monolithic" approach.

In this paper we use a "coupler" approach, as discussed above, where discrete coupler software runs separately, but simultaneously to the CFD simulations of the blade rows, on a set of processors dedicated exclusively to performing the search and interpolation. This is reminiscent of the "rendezvous" strategy discussed by Plimpton et al. [42]. The only additional cost for processes with sliding planes nodes on the side of the CFD code is the gathering, communication, and scattering of sliding plane data, which is on par with the cost of a standard halo exchange. For the Rig250 problem, this approach has two major advantages. First, it guarantees a more even distribution of the workload and control of this distribution. Secondly, some of the operations needed (in particular, the search) can be overlapped with the work done by the processes dedicated to CFD, enabling better performance and scaling.

III. Advances Achieved

The CFD component of the Rig250 simulation consists of a number of RANS solvers representing the rotors and stators of the Rig250 compressor. These were simulated using Hydra [16], [17], Rolls-Royce's in-house production CFD solver. Hydra is a multi-component piece of software, which was designed to simulate various aspects of turbomachinery designs. It is an unstructured finite-volume solver for the compressible Reynolds Averaged Navier-Stokes equations in their steady or unsteady formulation (RANS/URANS). The flow equations are solved in a time marching fashion. In a first step, all the spatial differential operators are discretized to form a residual. In a subsequent phase, the flow variables are updated using a Runge-Kutta method. For steady RANS computations, the flow equations are iterated towards steady state. Unsteady computations (URANS) adopt a Dual Time Stepping approach [16], with the flow iterations nested into a further loop over the physical time steps.

Hydra was first developed over 20 years ago starting from Fortran 77 parallelized using the OPlus library, which provides an abstraction for MPI parallelization. Recent work on Hydra [12], [43], including the work presented in this paper makes use of the OP2 domain specific framework to re-engineer the Hydra code-base to realize a performance portable application, OP2-Hydra, capable of utilizing modern and emerging hardware.

A. OP2-Hydra

OP2 [11], [44], is a high-level embedded domain specific language for writing unstructured-mesh algorithms. It has an API embedded in C/C++ and Fortran, and utilizes automatic code generation to translate the specification of a problem written with the high-level API to concrete parallelizations making use of a range of multi-core and many-core programming models. These include SIMD [19], OpenMP, OpenACC, CUDA, SYCL [45] on top of distributed memory parallelization with MPI. The translated code can be executed on most of the currently dominant processor architectures such as traditional multi-core CPUs (from Intel, AMD, IBM, ARM), accelerators such as GPUs (from NVIDIA, AMD and Intel) and their clusters. The OP2 API, code-generation software and associated libraries are maintained as open source software [44]. The proprietary Hydra code was converted to use OP2's Fortran API, using semi-automatic techniques.

1) The OP2 API: Declaration of an unstructured-mesh problem with OP2 separates the algorithm into four distinct parts: (1) types of mesh elements (called *sets*), (2) data defined on sets, (3) connectivity between sets (called *maps*) and (4) operations over sets. Sets can be, for example, mesh nodes, edges, triangular faces, quadrilateral faces or 3-D elements such as tetrahedrons. Data associated with these sets for example can be node coordinates, edge weights, velocities. Mappings between sets define how elements of one set connect with the elements of another set. These allows an explicit connectivity list (a mapping table) to be declared between sets. The same notion of connectivity (defining the neighboring elements) for regular structured-mesh computations is provided by a stencil. Computations over the mesh declared with OP2 are restricted to explicit numerical methods. Operations are declared by a parallel loop construct together with an outlined

```
1  ! Declaring the mesh with OP2
2  ! sets
3  call op_decl_set(nnode,nodes,'nodes')
4  call op_decl_set(nedge,edges,'edges')
5  call op_decl_set(ncell,cells,'cells')
6  ! maps
7  call op_decl_map(edges,nodes,2,edge ,pedge ,'pedge' )
8  call op_decl_map(edges,cells,2,ecell,pecell,'pecell')
9  ! data
10 call op_decl_dat(nodes,2,'real(8)',x,p_x,'p_x')
11 call op_decl_dat(cells,4,'real(8)',q,p_q,'p_q')
12 call op_decl_dat(cells,1,'real(8)',adt,p_adt,'p_adt')
13 call op_decl_dat(cells,4,'real(8)',res,p_res,'p_res')
14
15 ! Elemental kernel
16 subroutine res_calc(x1,x2,q1,q2,adt1,adt2,res1,res2)
17   IMPLICIT NONE
18   REAL(kind=8), DIMENSION(2), INTENT(IN) :: x1
19   REAL(kind=8), DIMENSION(2), INTENT(IN) :: x2
20   ...
21   REAL(kind=8) :: dx,dy,mu,ri,p1,vol1,p2,vol2,f
22   dx = x1(1) - x2(1)
23   dy = x1(2) - x2(2)
24   ...
25   f = 0.5 * (vol1 * q1(1) + vol2 * q2(1)) + &
26   &     mu * (q1(1) - q2(1))
27   res1(1) = res1(1) + f
28   res2(1) = res2(1) - f
29   ...
30 end subroutine
31 ! Calculate flux residual - parallel loop over edges
32 call op_par_loop_8 (res_calc, edges, &
33 & op_arg_dat(x,   1, edge,  2,"real(8)", OP_READ), &
34 & op_arg_dat(x,   2, edge,  2,"real(8)", OP_READ), &
35 & op_arg_dat(q,   1, ecell, 4,"real(8)", OP_READ), &
36 & op_arg_dat(q,   2, ecell, 4,"real(8)", OP_READ), &
37 & op_arg_dat(adt, 1, ecell, 1,"real(8)", OP_READ), &
38 & op_arg_dat(adt, 2, ecell, 1,"real(8)", OP_READ), &
39 & op_arg_dat(res, 1, ecell, 4,"real(8)", OP_INC ), &
40 & op_arg_dat(res, 2, ecell, 4,"real(8)", OP_INC ))
```

Fig. 3: An OP2 loop with associated sets, maps, data and elemental kernel declarations from [46].

elemental kernel defining the per mesh element computation. A parallel loop iterates over one set, accessing data declared on the set directly, or accessing data declared on other sets indirectly via a mapping table. The restriction to explicit methods narrows OP2's domain, but still covers a wide range of numerical solutions, particularly for solving PDEs. The restriction also means that there is no order dependency in which loop iterations are computed, allowing for maximum parallelization possibilities.

Fig. 3 illustrates the OP2 API for declaring a parallel loop, an `op_par_loop`, over mesh edges (see [46] for the full application). The sets, maps and data used in the loop are declared in lines 3-13. Here the `op_par_loop` describes the iteration over mesh `edges`, detailing the per set element computation as an outlined kernel (lines 16-30), while making explicit indication as to how each argument to that kernel is accessed (`OP_READ` - read only, `OP_INC` - increment) and the mappings `edge` and `ecell` used for indirectly accessing the data (`x`, `q`, `adt` and `res`) held on the sets. As can be seen, the loop declaration is expressed in a purely scalar manner without any indication of parallelization. The OP2-Hydra application consists of about 300 such `op_par_loops` with a total LoC of about 50K.

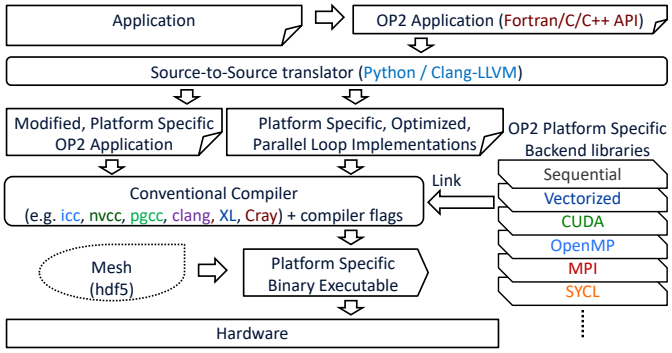


Fig. 4: OP2 Application Development

2) Automatic Code-generation: Once converted to the OP2 API, the workflow for generating concrete parallel implementations would be done as detailed in Fig. 4. The OP2 application is parsed by the code-generation layer which produces modifications to the *op_par_loop* calls in the code, together with concrete parallel code for each loop. Thus, for example the code generator will produce an OpenMP version for multi-threaded CPUs and a CUDA version for NVIDIA GPUs, each making use of one of the data-race handling strategies for implementing indirect increments. The generated code is human readable and can be compiled using a conventional compiler (e.g. `gcc`, `icc`, `nvcc`) and linked against platform specific OP2 back-end libraries to generate the final executable. These back-end libraries simply implement common functions for each parallelization including distributed memory parallelization and HDF5-based I/O. Generated parallel code for the above loop using SIMD, OpenMP and CUDA (among others) can be viewed in [46].

OP2 innovates in its code-generation tool-chain, written using Python and Clang-libtooling, to generate radically different code-paths for different parallelizations targeting different hardware. The data races due to indirect increments can be handled using several strategies, with the OP2 code generator allowing the application developer to automatically generate multiple versions. For a detailed description of these strategies we refer the reader to [11], [21], [45]. For the Rig250 compressor performance benchmarking in this paper, we have used the MPI and MPI+CUDA code generated through OP2. The standard owner compute model with halo exchanges and redundant computation is used at the MPI level, and atomic operations are used on the GPUs to resolve data races. The MPI parallelization automatically implements latency hiding techniques to reduce message passing overheads.

3) Coupling and Optimizations: To integrate an OP2 application to work with external coupler software, we implemented specific interfacing capabilities in OP2 to give the coupler direct access to the OP2 `maps` and `dats`. Such capability was not previously available in OP2 and provides an example of a DSL-based application interfacing with third-party software for a production problem. The performance in Section IV indicates that very good scaling can be achieved without loss in productivity and portability. This was a result of a number of additional optimizations to OP2 that was carried out to

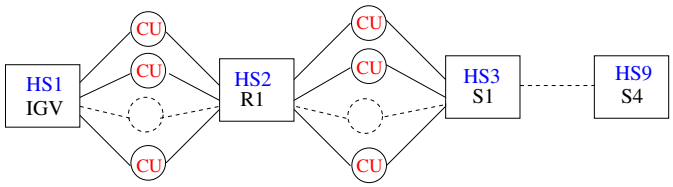


Fig. 5: JM76 scheme for the Rig250. Each row passage corresponds to a Hydra Session (HS). Hydra Sessions are interconnected by Coupler Units (CU).

overcome the scaling challenges of a sliding planes simulation. These include (1) partial halo exchanges to communicate only the boundary elements of a set (2) grouping of MPI messages to reduce the number of messages sent and (3) GPU-side gather of data when communicating with the JM76 coupler. In Section IV-A5 we quantify the gains from these optimizations, focusing on how they aided in reaching the requisite low-overhead scaling for the full compressor simulation.

B. JM76 Coupler

JM76 [47], [48] is a coupling framework designed for aero-thermal simulations of complex configurations. It can be used to couple an arbitrary number of models, providing specialized treatment for fluid-solid and fluid-fluid interfaces. JM76 implements a decentralized model as that of Larson et al. [36] together with a client server scheme introduced in [8]. Fig. 5 illustrates the architecture of the Rig250 coupled solver setup. The setup decomposes the mesh to be solved into a number of Hydra Sessions (HS) communicating through individual Coupler Units (CU). The HSs are distinct CFD simulations over separate physical meshes. For the Rig250 a HS was allocated to simulate each of the rows consisting of either a rotor or a stator of the compressor. One or more CUs sit in between two HSs, carrying out specific "transfer" procedures between the HSs. In our problem the transfer procedures come from the aforementioned requirements of the sliding-planes operation where the mesh interfaces between two HSs move relative to each other during the simulation. Thus, the mapping linking mesh elements in one interface to the other must be recomputed every time the mesh moves. For Rig250, this calculation represents the most performance critical aspect of a CU's operation. Each mesh element in an interface needs to compare itself with every other cell in the other interface, and repeated for all interfaces, both moving and static [33]. Once the "linking" has been found the CUs can transfer the required data from one HS interface to the other usually involving interpolation of data and communication.

The distributed-memory parallel setup of a simulation with JM76 makes use of MPI and consists of a number of MPI processes allocated to each of the HSs (each HS having their own MPI sub-communicator with multiple processes) and one or more MPI processes also allocated to each CU. Given the intensity of the interface search routines a further decomposition of work can be implemented by partitioning the interface mesh and allocating separate segments to CUs, thus allowing multiple CUs to work on separate parts of a single interface. This then significantly reduces the search time as discussed in [33]. The JM76 coupler used in this paper makes use of a new parallel binary tree-search algorithm, replacing the previous "brute-force" sequential search from [48]. The alternating digital tree (ADT) method [49] is used where the data are sorted according to coordinates. This improvement helped reduce coupler overheads and leads to a 35% performance improvement for 30 to 40 CUs, the main configuration we use in this paper. This, and the specific quantitative improvements from the binary search algorithm, is explored in Section IV-A5. The performance results in this present work is also the first time the coupler was used to scale simulations on HPC systems at over petascale machine sizes, not to mention for a production CFD turbomachinery component such as the Rig250 compressor.

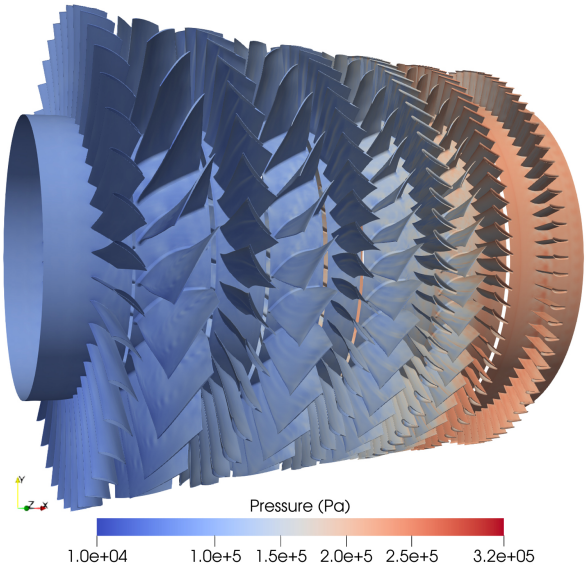


Fig. 6: Rig250 Mesh $1-10_{4.58B}$ - Pressure (Pa) contours at 1000 time-steps.

IV. Results

A. Specifications

1) Problem Sizes: Two models of the Rig250 are considered in our simulations. The first, includes the "swan neck" row that is used to orient the flow into the inlet stage of the compressor. This variant has 430M nodes and represents a coarser grid over the Rig250 geometry. The Hydra sessions simulate the flow through the 1 to 9 rows of its rotors and stators (full annulus), plus the outlet guide vane at the exit. This gives rise to a 1-10 row geometry. We call this mesh $1-10_{430M}$. The second model uses a finer grid, consisting of 4.58B nodes, but omits the swan neck (see Fig. 6). For this version, the boundary conditions at the inlet of the first stage of the compressor are replicated using flow variables from the outflow of the swan neck stage. In our study we simulate the flow for its 1 - 2 rows (which gives a 653M mesh, which we call $1-2_{653M}$) as well as the full 1-10 rows making up the 4.58B nodes grand challenge production problem (we call this $1-10_{4.58B}$). These meshes were selected such that

TABLE I: Systems specifications.

System	ARCHER2 HPE Cray EX [52]	Cirrus SGI/HPE 8600 GPU Cluster [53]
Processor	AMD EPYC 7742 @ 2.25 GHz	Intel Xeon Gold 6248 (Cascade Lake) @ 2.5 GHz + NVIDIA Tesla V100-SXM2-16GB GPU
(procs×cores) /node	2×64	2×20 + 4×GPUs
Memory/node	256 GB	384 GB + 40GB/GPU
Interconnect	HPE Cray Slingshot 2×100 Gb/s bi-directional/node	Infiniband FDR, 54.5 Gb/s
OS	HPE Cray LE (based on SLES 15)	Linux CentOS 7
Compilers	GNU 10.2.0	nvfortran (nvhpc 21.2)
Compiler Flags	`-O2 -eF -fPIC`	CUDA 11.6 and sm_70 -O2 -Kieee
Power/node	660W	≈ 900W

the $1-10_{4.58B}$ is roughly $10\times$ larger than the $1-10_{430M}$ mesh and the $1-2_{653M}$ is the first two rows of the full $1-10_{4.58B}$ mesh. The sizes allow us to investigate scaling performance on the relatively smaller GPU cluster we have access to for benchmarking and enable us to easily make performance projections for larger GPU clusters that could hold the full 10 row mesh. GPU global memory limits the size of the total mesh that can be simulated.

2) Numerical Setup: All the variants of the meshes were solved using the URANS equations in Hydra. The rotor speed is set at 13000rpm for the $1-10_{430M}$ mesh and 11000rpm for the $1-10_{4.58B}$ mesh. These correspond to different points on the compressor's operating map. The speed of 13000rpm is near the design point where boundary layers and corner separations are well behaved. The speed of 11000rpm is near the stall boundary where the higher resolution mesh is needed to capture the larger scale separations that occur.

As previously noted, Hydra uses a dual time-stepping approach with an outer time accurate step, and a number of inner iterations where acceleration schemes, such as pseudo time-stepping and preconditioning, can be used. The choice of outer time step is related to the physical phenomena being modelled and ensuring consistency in the spatial and temporal discretization. An explicit Runge-Kutta scheme is used for the inner iterations and fixed time-steps of $2.725\times10^{-6}s$ and $1.929\times10^{-6}s$ for the outer steps for the $1-10_{4.58B}$ and $1-10_{430M}$ meshes, respectively. All the simulations have been conducted using the Spalart-Allmaras turbulence model [50], enforcing subsonic pressure conditions [51] at the inlet/outlet of the domain.

3) Systems: TABLE I briefly details the distributed memory cluster systems used, namely the HPE-Cray EX system ARCHER2 and SGI/HPE 8600 GPU cluster Cirrus both located at EPCC UK. Each ARCHER2 node consists of two AMD EPYC 7742 processors each with 64 cores (128 total cores) arranged in a 8 NUMA regions per node (16 cores per NUMA region) configuration [52]. Each node is equipped with 256 GB memory. The nodes are interconnected by a HPE Cray Slingshot, 2×100 Gb/s bi-directional per node network. The full machine consists of 5,860 nodes (750,080 cores), but in our benchmarking we only scale as far as 65536 cores (512 nodes) for the largest problem size. How far we scale to was determined by achieving a parallel efficiency of over 75%, the physically available machine size, or the availability of a limited compute budget (e.g 512 ARCHER2 nodes), which ever is larger. The GNU compiler collection version 10.2.0 was used on ARCHER2 with compiler flags noted in the table. The Cirrus GPU cluster consists of 4×V100 GPUs per node configuration, each node also consisting of 2×Intel Xeon Gold 6248 (Cascade Lake) processors, each with 20 cores (40 total cores). Each node has 384GB main memory and each GPU has 16GB global memory. The cluster has 36 nodes in total limiting the problem size solved on the GPUs to $36\times4\times16=2304$ GB. However the $1-10_{4.58B}$ mesh require a minimum of 7800GB (i.e needing a minimum of 122 Cirrus-type nodes) and thus we were not able to run this mesh on the GPU cluster.

On ARCHER2, both HSs and CUs run on the available CPU cores in a node, whilst on Cirrus the HSs used the GPUs (and one CPU core per GPU), using OP2-Hydra's CUDA parallelization compiled with the NVIDIA HPC Toolkit's `nvfortran` and CUs were exclusively allocated to the Intel Cascade Lake processor cores. Thus the Cirrus executions were a heterogeneous processor execution.

4) Node Power Consumption: ARCHER2 node power consumption was obtained by querying the `slurm` scheduler after the execution of jobs on the nodes giving on average 660W per node. This was computed by dividing the total power consumption (kWh) of the job used, by time and number of nodes to get a fixed-point sample of the power per node. Power consumption of longer runs increased linearly with time and we did not see any indications of it varying with time. However, we had no tools available to measure variance across nodes.

For Cirrus, during the execution of the job, we used `nvidia-smi` to poll instantaneous power consumption every second, then averaged to arrive at 182W per GPU. The CPUs are used to control GPUs (4 processes/node), and 4-12 processes per node for the coupler (out of 40 cores). However, 80-90% of the time (depending on the coupler overhead), coupler processes are idle. Our direct power measurements on a Xeon Gold 6226R (vs. Gold 6248 in Cirrus) on a similar workload showed 154W power consumption of the system (excluding GPUs). Accounting for the higher TDP, we estimate the power consumption of a Cirrus node excluding GPUs to be 172W, and therefore a total of $\approx$ 900W($\lceil4\times182+172\rceil$) per Cirrus node. These estimates indicate approximately 1.36× more power consumption by a GPU node compared to an ARCHER2 node. We make use of this power equivalence when directly comparing scaling performance on the two systems in Section IV-B.

5) Configurations: As noted before, JM76 distinguishes between Hydra and Coupler processes. The choice of how to distribute the resources is up to the user, and has to be carefully

TABLE II: Brute Force vs Binary Tree search for the JM76 coupler: $1-10_{430M}$ mesh on ARCHER2 (runtime in seconds)

Nodes	Brute Force					Binary Tree	
	10CUs	20CUs	30CUs	40CUs	60CUs	10CUs	20CUs
10	957.37	644.29	555.95	623.13		358.90	382.37
27	739.45	396.66		236.56	206.56	140.76	137.62

TABLE III: OP2 communications optimizations (runtime in seconds). PH - Partial Halos exchanges, GH - Grouped Halos

	ARCHER2			
	$1-10_{430M}$		$1-10_{4.58B}$	
	10 nodes	27 nodes	107 nodes	283 nodes
Default	41.62	16.55	41.24	18.19
+PH	39.87	15.64	38.36	16.88

	Cirrus		
	$1-10_{430M}$		$1-2_{653M}$
	15 nodes	20 nodes	17 nodes
Default	19.07	13.58	23.79
+GG +PH +GH	5.09	4.23	6.74

evaluated. More CUs can be used to reduce the search time by partitioning a single interface into smaller segments resulting in a smaller search space per CU. On CPU clusters this will require allocating more MPI processes for the increasing CUs, and at the same time reducing the number of processes allocatable to HSs. On GPU clusters one can increase CUs up to the number of available CPU cores on a node, however there are diminishing returns and increasing overheads from having too many CUs, due to the extra communications. TABLE II presents this trade-off for the $1-10_{430M}$ problem on ARCHER2 using 10 and 27 nodes. The first 5 columns use JM76's initial brute-force search routine (BF) with increasing number of CUs. Improving the search to a binary tree search algorithm dramatically reduces the overheads as can be seen from the final two columns of TABLE II. This allows us to allocate a lower number of CUs, and thus more processes to HSs to gain significantly better scaling. From experimenting with the balance of CUs/HSs and MPI processes allocated to them, we found that good performance can be gained with 30 CUs per interface, on CPU systems and 40 CUs per interface on GPU systems for the Rig250 problem. Each CU was solved with 1 MPI process running on 1 CPU core, using the Binary Tree search routine.

Further scaling overhead reductions were gained through optimizations to OP2. OP2's default distributed memory parallelization back-end [54] required all the halo elements of a set, held by an MPI process, to be updated / exchanged if a parallel loop indirectly reads (`READ`) or read-writes (`RW`) from data held on this set that has been updated by a previous loop. However, sets representing the boundary of the mesh, such as boundary nodes and boundary edges only have connectivity with a few internal mesh elements. This provides an opportunity for optimization where only a few elements in the halo of an internal set require updating via an MPI halo-exchange. The performance gains from this *partial halo exchange* (PH) can be seen from TABLE III. We see modest gains of about 5% to 7% for the lower node counts

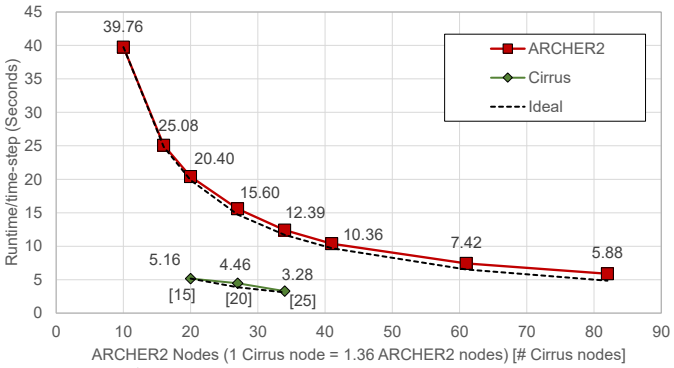


Fig. 7: Rig250 $1-10_{430M}$ Mesh Runtime.

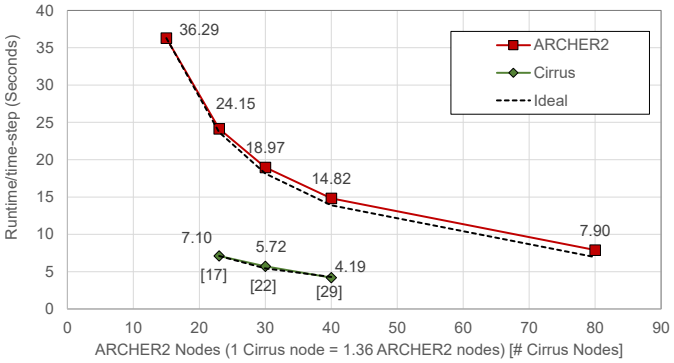


Fig. 8: Rig250 $1-2_{653M}$ Mesh Runtime

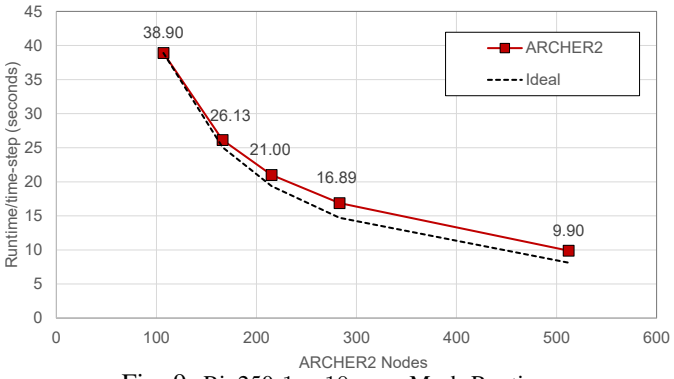


Fig. 9: Rig250 $1-10_{4.58B}$ Mesh Runtime

on ARCHER2. An additional two optimizations, as discussed before provided significant runtime reductions on the GPU cluster. These include the GPU-side gather (GG) when OP2 hands over data to JM76 for communication between CUs, and grouping halos sent per `op_dat` within an `op_par_loop` to a single large MPI message (i.e. grouping of Halos, GH) thus reducing the number of smaller messages sent, that require separate copies over PCIe. Together these resulted in a 60 – 70 % reduction in runtime. However, grouping of halos to a single message, marginally increased the runtime on ARCHER2 and thus it is not shown in this table. This was due to the time required to combine messages (i.e. packing/unpacking) outweighing the reduction in latency on the CPU clusters. As such this optimization was not used for the full scale runs on ARCHER2.

B. Scaling Performance

When designing the benchmark setup for scaling, we considered the differences in size of the various meshes, as well as differences in power consumption between Cirrus and ARCHER2 nodes. Node counts used on ARCHER2 for running the $1-10_{430M}$ problem were multiplied by $10.65\times$ for running the $1-10_{4.58B}$ problem. For the $1-2_{653M}$ problem, we used rounded node counts assigned to the first two rows from the $1-10_{4.58B}$ problem. Cirrus node counts were determined by dividing ARCHER2 node counts by 1.36 (the ratio of power consumption) and rounding them to the nearest integer. This setup allows us to directly compare the performance on different problems with runtime/time-step values - see the first four data points on Figures 7 - 9.

1) $1-10_{430M}$ *Mesh Performance:* When considering the smaller $1-10_{430M}$ problem, we observe a 94% parallel efficiency scaling from 10 nodes to 34 nodes on ARCHER2, on average 5-10% (at 10 and 34 nodes respectively) of time is spent waiting for the coupler. These overheads of coupling are partly due to the cost of interpolation on the interfaces, as well as the load imbalance between HSs, which manifest as waiting times in the coupler due to the implicit synchronization. Scaling this problem out to 82 nodes, we still observe an 82.4% parallel efficiency, with coupling overhead increasing to 20%. At this extreme, there are only 41K mesh nodes per process. For Cirrus, we could only collect 3 data points, running on 60, 80, and 100 GPUs (15, 20, 25 Cirrus nodes, respectively, noted in square brackets on the graph, which is plotted in the figures as equivalent to 20, 27 and 34 ARCHER2 nodes), at which point parallel efficiency is 94%, with the wait time for the coupler between 15-20%. When matching the two systems on the basis of power consumption (1 Cirrus node = 1.36 ARCHER2 node), we see that Cirrus is $3.75-3.95\times$ faster than ARCHER2. In contrast the effect on ARCHER2 is reduced due to the relatively slower HSs and the fact that CUs can only be increased at the cost of reducing HS processes. When comparing node-to-node performance, Cirrus was $5.1-5.37\times$ faster than ARCHER2.

2) $1-10_{4.58B}$ *Mesh Performance:* Moving to the full $1-10_{4.58B}$ problem, we perform benchmarks on ARCHER2 on $10.65\times$ more nodes compared to the scaled down problem to match the number of elements per node. Fig. 9 shows closely matching performance compared to the smaller problem, with an 82% parallel efficiency when going from 107 nodes to 512 nodes - the coupling overheads increase slightly to 8–15%.

3) $1-2_{653M}$ *Mesh Performance:* Considering that we could not run the full $1-10_{4.58B}$ problem on Cirrus due to a lack of GPU global memory, we also evaluate performance on just the first two rows, as shown in Fig. 8. Performance on ARCHER2 is as expected, with a parallel efficiency of 88% going from 15 to 80 nodes, with a 2-8% coupling overhead. This overhead is noticeably smaller compared to the full 10 rows both on the scaled down and full meshes, because the load between the two HSs is easier to balance. On Cirrus, the scaling efficiency from 17 to 29 nodes is 98%, with a 10-12% coupling overhead. Comparing the two systems (again by node counts matching power on each system), we observe that Cirrus is $3.3-3.4\times$ faster. When comparing node-to-node performance, Cirrus was $4.5-4.6\times$ faster than ARCHER2.

4) Projecting Performance: Projecting to the full $1-10_{4.58B}$ problem given enough GPUs (122 nodes, 488 GPUs) on Cirrus, we believe, would still achieve over $3\times$ speedup over the power equivalent ARCHER2 setup (166 nodes). Based on the above results, TABLE IV summarizes and notes the full simulation time for 1 revolution of the Rig250 mesh. As stated before, the aim was to select machine sizes for demonstrating capability of scaling at over 75% parallel efficiency or as large as the machine would allow (in the case of the GPU cluster), within a limited benchmarking budget on these systems. For the full $1-10_{4.58B}$ problem, the grand challenge was to achieve 1 revolution in less than 24 hours (a minimum requirement for tractable design explorations). The selection of node counts for this, on both machines, again aimed to push for runs as large as possible within the above constraints.

Note here that we have indicated actual (A) and projected (*P*) runtimes. The predictions were done by running a smaller number of time-steps and projecting the total runtime for 2000 time-steps, which completes a full revolution. The only exception to this projection was the Cirrus runtime for $1-10_{4.58B}$ on 122 Cirrus nodes. In this case we considered the $1-2_{653M}$ problem and looked at the time per time-step for Cirrus on 17 nodes (equivalent to 23 ARCHER2 nodes), which is 7.1 seconds. Scaling up the problem to 10 rows will need 122 nodes to fit on GPU memory. ARCHER2 coupling overhead, which is 4% for 2 rows and 11% for 10 rows. On Cirrus the coupling overhead is 10% at 2 rows. We therefore speculate that at 10 rows the coupling overhead on Cirrus should be between 20-30%. This results in 7.8 - 8.5 seconds per time-step on Cirrus. Taking the upper limit (8.5 seconds) for the overhead would give $(8.5\times 2000/3600)$ 4.7 hours for a full revolution on Cirrus for $1-10_{4.58B}$. The power-equivalent number of nodes comparable to 122 Cirrus nodes on ARCHER2 is 166 nodes, which takes 14.5 hours for a full revolution. To obtain a runtime closer to Cirrus's 4.7 hour time to solution, we will need over $3\times$ more ARCHER2 nodes (>512 nodes).

5) Monolithic (non-coupled) application performance: TABLE IV also includes predicted runtimes from the current production setup based on a monolithic execution of the problem, enabling us to compare it with the coupled version. Again, in this case the predictions were done by actually running a smaller number of time-steps, then projecting for 2000 time-steps, without carrying out a full revolution to save time on the clusters. Due to mesh generation issues the $1-10_{4.58B}$ problem could not be assembled into a monolithic mesh for our tests at this time. However, current production simulations of this problem using the monolithic execution setup, has been reported to demonstrate poor scaling, leading to even the best runtime taking over a week for one revolution on internal production clusters [55]. Production runs on a 8000 core Intel Haswell cluster gave a 2000 second time per

TABLE IV: Achieved (A) and *Projected (P)* times to solution (hours) : Rig250, 1 revolution

Rig250 Problem	ARCHER2		Cirrus	
	Runime	#nodes	Runtime	#nodes
$1-10_{430M}$ - Monolithic	*93.0 (P)*	*8*		
$1-10_{430M}$ - Coupled	*85.0 (P)*	*8*	*2.9 (P)*	*15*
$1-10_{430M}$ - Coupled	*3.3 (P)*	*80*	*1.8 (P)*	*25*
$1-2_{653M}$ - Monolithic	*110.0 (P)*	*8*		
$1-2_{653M}$ - Coupled	*40.0 (P)*	*8*	*3.9 (P)*	*17*
$1-2_{653M}$ - Coupled	*8.2 (P)*	*40*	*3.2 (P)*	*22*
$1-10_{4.58B}$ - Coupled	*14.5 (A)*	*166*	*4.7 (P)*	*122*
$1-10_{4.58B}$ - Coupled	*9.4 (A)*	*256*		
$1-10_{4.58B}$ - Coupled	*5.5 (A)*	*512*		

time-step indicating 46 days for 1 revolution. On EPCC's previous ARCHER1 system [56], a Cray XC30 with 2× 2.7 GHz, 12-core Intel E5-2697 v2 (Ivy Bridge) processors, on 100K cores, time for 1 revolution was estimated to be at 9 days. As such, the overall speedups with the coupled simulation suite from this work, together with the use of modern heterogeneous multi-core/many-core hardware, points to an order of magnitude (≈30×) improvement over current production capability. We estimate that 2× to 3× of this is due to next generation hardware for CPUs.

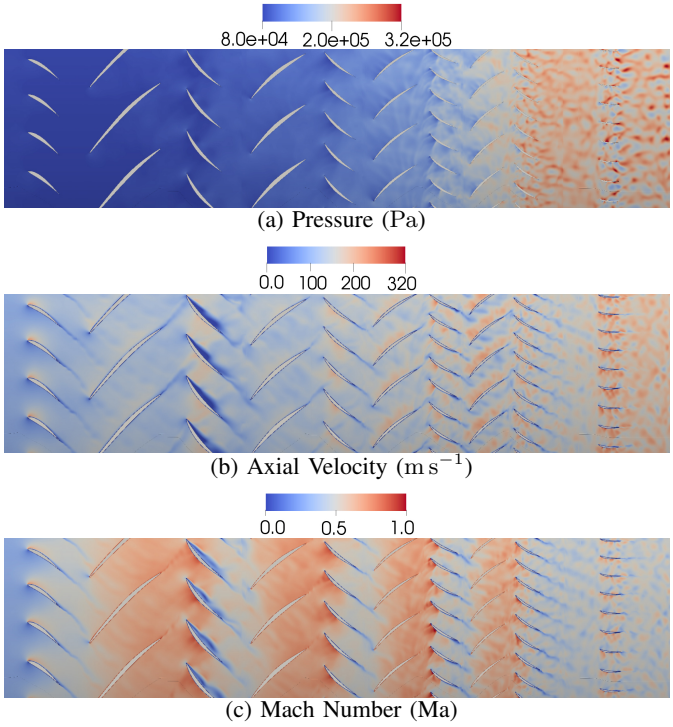
C. Flow Field

Snapshots of the flow field computed from our simulation are presented in Fig. 10. The figure shows contour plots of various fluid variables on a cylindrical surface cutting the rotors and stators at approximately mid-radius, after one disc revolution. It should be noted the continuity of the solution and the absence of wiggles throughout the interfaces, indicating the validity of the sliding plane treatment between the zones.

The flow enters at ambient conditions and the fluid pressure becomes roughly 3.8 times larger as the flow moves through the compressor stages. This visualization was extracted from the solution of the $1-10_{4.58B}$ mesh, which corresponds to off-design operating conditions. For this compressor, the IGV and the first two stators are variable and their angles of attack are adjusted to maintain stability at off-design conditions. The regions of dark blue in Fig. 10(b) show regions where the stator 1 boundary layers are separated. This is consistent with the off-design operating conditions and indicative of an incipient stall mechanism. Strong unsteadiness is visible in the large axial gaps downstream of stator 4 and the outlet guide vane. These are due to the relative motion of the rotor and stator wakes creating strong unsteady mixing. Experimental evidence from engine tests shows that there is high unsteadiness at the compressor exits and is one of the industrial drivers for these very high resolution unsteady simulations.

V. Related Work

This work is part of an ongoing push towards virtual certification by Rolls-Royce. Altitude strain gauge testing has already been replaced by aero-mechanical analysis; and the certification requirement to withstand the impact of a 4kg bird on take-off or landing has been replaced with an ALE simulation. Virtual certification replaces a physical test and



(a) Pressure (Pa)

(b) Axial Velocity ($\mathrm{m\,s^{-1}}$)

(c) Mach Number (Ma)

Fig. 10: Contours through rotors and stators at 1 rev.

hence the analysis methods must initially be higher than those used in design. This, therefore, requires much greater computing. Virtual certification is achieved by working closely with the certification authorities and detailed validation of the method. This often requires specific details around the way calculations are run, meshes used and modelling parameters.

A number of related work on targeting virtual certification objectives have been published. A 2016 study by Laskowski et al. [57] provides estimates for the URANS simulation of a full engine compressor as 38 days on a 100K CPU cores (Figure 7), predicting the prohibitive cost of such simulations. Work by Saito et al. [58] and Haug et al. [59] also show the high cost of unsteady full annulus simulations with their use limited to research. The 2018 DLR study [38] of the Rig250 mesh geometry with a mesh size of 1B nodes, indicated 10 days to simulate 1 revolution on 1200 CPU cores. However, this simulation was done using a block-structured mesh setup. The 2021 work on LES of a full engine by Arroyo et al. at CERFACS [60], [61] demonstrated a full annulus large-eddy simulation with over 2.1B cells of the DGEN-380 demonstrator engine enclosing a fully integrated fan, compressor and annular combustion chamber. It utilizes coupling techniques similar to our work here, but simulates a smaller number of compressor rows as part of the full engine, using a code only executable on CPU clusters.

Our work in this paper shows a calculation of a 4.8B element mesh for a single compressor. It is currently believed to be what is needed to meet the requirements for virtual certification. However, the mesh is 50-100 times larger than what current design tools use. The main limits on virtual certification are three fold: firstly, the computing power is beyond

what industry could reasonably afford to make available to multiple engineers. Secondly, given that models of this size have only just become possible, there is, as yet, no validation database on which to build. Thirdly, validation can only begin when there is a code that scales on modern HPC platforms. The work in this paper focuses and overcomes the third of those limits. It took 10 years working with the FAA to certify the 4kg bird simulations. We are just at the start of this road, but we expect the methodology and the scaling we have achieved to be part of the process towards virtual certification.

VI. Conclusions

In this work, we have performed time accurate simulations for the full annulus model of the Rig250 test compressor achieving a breakthrough time to solution of less than 6 hours on the ARCHER2 HPE-Cray EX system at EPCC. The simulation code innovates in (1) its design for performance portability by using the OP2 DSL for the CFD components and (2) use of custom built discrete coupler software for decomposing the sliding-planes problem leading to better load-balance and heterogeneous execution. The current work further demonstrates how key techniques consisting of reduction in overheads through distributed memory communication avoidance, including the reduction of host-device overheads on GPU clusters, and faster coupler interface search algorithms play an essential role in overcoming the challenges in scaling for such problems.

The CFD component's use of OP2 enabled us to automatically generate highly optimized platform-specific parallelizations for both multi-core (CPU) and many-core (GPU) clusters via a single high-level source. Performance of the GPU version of the code on reduced and partial meshes of the same problem on a GPU cluster with Nvidia V100 GPUs indicate a $3.4\times$ speedup over ARCHER2 on a power-equivalent number of nodes. If a node-to-node comparison is considered, then these results indicate that the GPU nodes in Cirrus outperform the CPU nodes in ARCHER2 over $4.5\times$. Such speedups point to an order of a magnitude improvement compared to current production capability. These innovations demonstrate how currently prohibitive simulation costs for such problems can be dramatically reduced, enabling industrial designs to be carried out in tractable times. As recognized in one of the grand challenges in the CFD Vision 2030 study by NASA [13], reaching such a capability, is crucial for achieving the virtual certification goals of the industry. The work detailed in this paper demonstrate a step-change towards achieving these objectives with production applications.

Acknowledgment

This research is supported by Rolls-Royce plc., and by the UK EPSRC (EP/S005072/1 – Strategic Partnership in Computational Science for Advanced Simulation and Modelling of Engineering Systems – ASiMoV). Gihan Mudalige was supported by the Royal Society Industry Fellowship Scheme (INF/R1/1800 12). This work used the ARCHER2 UK National Supercomputing Service (https://www.archer2.ac.uk). This work used the Cirrus UK National Tier-2 HPC Service at EPCC (http://www.cirrus.ac.uk) funded by the University of Edinburgh and EPSRC (EP/P020267/1). We are thankful to EPCC for the support provided in using the ARCHER2 and Cirrus clusters. We would also like to thank Christopher Goddard, Paolo Adami and the Hydra developer team at Rolls-Royce plc., for useful technical discussions.

References

[1] *Net Zero Strategy: Build Back Greener*. HM Government, Crown Copyright, 2021. UK Department for Business Energy and Industrial Strategy.

[2] *Flightpath 2050 : Europe's vision for aviation : maintaining global leadership and serving society's needs*. Publications Office, 2011. European Commission and Directorate-General for Mobility and Transport and Directorate-General for Research and Innovation.

[3] F. Wang, M. Carnevale, L. di Mare, and S. Gallimore, "Simulation of Multistage Compressor at Off-Design Conditions," *Journal of Turbomachinery*, vol. 140, 12 2017. 021011.

[4] L. Cozzi, F. Rubechini, M. Giovannini, M. Marconcini, A. Arnone, A. Schneider, and P. Astrua, "Capturing Radial Mixing in Axial Compressors With Computational Fluid Dynamics," *Journal of Turbomachinery*, vol. 141, 01 2019. 031012.

[5] S. Kim, K. Kim, and C. Son, "Three-Dimensional Unsteady Simulation of a Multistage Axial Compressor With Labyrinth Seals and its Effects On Overall Performance and Flow Characteristics," *Aerospace Science and Technology*, vol. 86, pp. 683–693, 2019.

[6] N. Hills, "Achieving high parallel performance for an unstructured unsteady turbomachinery cfd code," *The Aeronautical Journal (1968)*, vol. 111, no. 1117, p. 185–193, 2007.

[7] E. L. Blades and D. L. Marcum, "A sliding interface method for unsteady unstructured flow simulations," *International Journal for Numerical Methods in Fluids*, vol. 53, no. 3, pp. 507–529, 2007.

[8] V. Ganine, D. Amirante, and N. Hills, "Enhancing performance and scalability of data transfer across sliding grid interfaces for time-accurate unsteady simulations of multistage turbomachinery flows," *Computers and Fluids*, vol. 115, pp. 140 – 153, 2015.

[9] V. Marciniak, A. Weber, and E. Kügeler, "Modelling transition for the design of modern axial turbomachines," in *Proceedings of the 6th European Conference on Computational Fluid Dynamics, Barcelona, Spain*, pp. 20–25, 2014.

[10] L. Lapworth, "Hydra-CFD: a framework for collaborative CFD development," in *International conference on scientific and engineering computation (IC-SEC)*, vol. 30, 2004.

[11] G. R. Mudalige, M. B. Giles, I. Reguly, C. Bertolli, and P. H. J. Kelly, "OP2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures," *2012 Innovative Parallel Computing, InPar 2012*, 2012.

[12] I. Z. Reguly, G. R. Mudalige, C. Bertolli, M. B. Giles, A. Betts, P. H. J. Kelly, and D. Radford, "Acceleration of a Full-Scale Industrial CFD Application with OP2," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1265–1278, 2016.

[13] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis, *CFD Vision 2030 Study : A Path to Revolutionary Computational Aerosciences*. NASA-CR ; 2014-218178, National Aeronautics and Space Administration, Langley Research Center, 2014.

[14] A. Inc, "Ansys fluent user's guide," vol. Release 17.2, 2016.

[15] "Code_Saturne," 2022 (Online). https://www.code-saturne.org/cms/web/.

[16] P. Moinier, J.-D. Muller, and M. B. Giles, "Edge-based multigrid and preconditioning for hybrid grids," *AIAA Journal*, vol. 40, no. 10, pp. 1954–1960, 2002.

[17] M. B. Giles, M. C. Duta, J.-D. Muller, and N. A. Pierce, "Algorithm developments for discrete adjoint methods," *AIAA Journal*, vol. 41, no. 2, pp. 198–205, 2003.

[18] P. Colella, "Defining Software Requirements for Scientific Computing.," 2004. Presentation.

[19] G. R. Mudalige, I. Z. Reguly, and M. B. Giles, "Auto-vectorizing a large-scale production unstructured-mesh cfd application," in *Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing*, WPMVP '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[20] I. Z. Reguly, E. László, G. R. Mudalige, and M. B. Giles, "Vectorizing unstructured mesh computations for many-core architectures," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 2, pp. 557–577, 2016.

[21] A. A. Sulyok, G. D. Balogh, I. Z. Reguly, and G. R. Mudalige, "Locality optimized unstructured mesh algorithms on gpus," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 50–64, 2019.

[22] G. R. Mudalige, M. B. Giles, C. Bertolli, and P. H. Kelly, "Predictive modeling and analysis of op2 on distributed memory gpu clusters," PMBS '11, (New York, NY, USA), p. 3–4, Association for Computing Machinery, 2011.

[23] H. Carter Edwards, C. R. Trott, and D. Sunderland, "Kokkos," *J. Parallel Distrib. Comput.*, vol. 74, pp. 3202–3216, Dec 2014.

[24] R. D. Hornung and J. A. Keasler, "The RAJA portability layer: Overview and status," tech. rep., Lawrence Livermore National Lab. (LLNL), 9 2014.

[25] K. B. Ølgaard, A. Logg, and G. N. Wells, "Automated Code Generation for Discontinuous Galerkin Methods," *CoRR*, vol. abs/1104.0628, 2011.

[26] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, "Firedrake: Automating the Finite Element Method by Composing Abstractions," *ACM Transactions on Mathematical Software*, 2017.

[27] P. Vincent, F. Witherden, B. Vermeire, J. S. Park, and A. Iyer, "Towards green aviation with python at petascale," in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, Nov 2016.

[28] M. Lange, N. Kukreja, M. Louboutin, F. Luporini, F. Vieira, V. Pandolfo, P. Velesko, P. Kazakas, and G. Gorman, "Devito: Towards a generic finite difference dsl using symbolic python," pp. 67–75, IEEE, 2016.

[29] T. Gysi, C. Osuna, O. Fuhrer, M. Bianco, and T. C. Schulthess, "Stella: A domain-specific tool for structured grid methods in weather and climate models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, (New York, NY, USA), pp. 41:1–41:12, ACM, 2015.

[30] "PSyclone Project - GitHub Repository," 2022 (Online). https://github.com/stfc/PSyclone.

[31] S. Adams, R. Ford, M. Hambley, J. Hobson, I. Kavčič, C. Maynard, T. Melvin, E. Müller, S. Mullerworth, A. Porter, M. Rezny, B. Shipway, and R. Wong, "Lfric: Meeting the challenges of scalability and performance portability in weather and climate models," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 383–396, 2019.

[32] I. Z. Reguly, G. R. Mudalige, and M. B. Giles, "Loop tiling in large-scale stencil codes at run-time with ops," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, pp. 873–886, April 2018.

[33] A. Powell, K. Choudry, A. Prabhakar, I. Reguly, D. Amirante, S. Jarvis, and G. Mudalige, "Predictive analysis of large-scale coupled cfd simulations with the cpx mini-app," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 141–151, 2021.

[34] Y.-H. Tang, S. Kudo, X. Bian, Z. Li, and G. E. Karniadakis, "Multiscale universal interface: A concurrent framework for coupling heterogeneous solvers," *Journal of Computational Physics*, vol. 297, pp. 13–31, 2015.

[35] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, and B. Uekermann, "precice–a fully parallel library for multi-physics surface coupling," *Computers & Fluids*, vol. 141, pp. 250–258, 2016.

[36] J. Larson, R. Jacob, and E. Ong, "The model coupling toolkit: A new fortran90 toolkit for building multiphysics parallel coupled models," *The International Journal of High Performance Computing Applications*, vol. 19, no. 3, pp. 277–292, 2005.

[37] F. Duchaine, S. Jauré, D. Poitou, E. Quémerais, G. Staffelbach, T. Morel, and L. Gicquel, "Analysis of high performance conjugate heat transfer with the openpalm coupler," *Computational Science & Discovery*, vol. 8, no. 1, p. 015003, 2015.

[38] O. K. Reutter, G. A. Nicke Eberhard, and E. Kuegeler, "Comparison of Experiments, Full-Annulus- Calculations and Harmonic-Balance-Calculations of a Multi-Stage Compressor," Zenodo, May 2018. GPPS Montreal 2018 (GPPS-NA-2018), Montreal Canada.

[39] T. Röber, E. Kügeler, and A. Weber, "Investigation of Unsteady Flow Effects in an Axial Compressor Based on Whole Annulus Computations," vol. Volume 7: Turbomachinery, Parts A, B, and C of *Turbo Expo: Power for Land, Sea, and Air*, pp. 2643–2655, 06 2010.

[40] A. Schmitz, M. Aulich, D. Schönweitz, and E. Nicke, "Novel Performance Prediction of a Transonic 4.5 Stage Compressor," vol. Volume 8: Turbomachinery, Parts A, B, and C of *Turbo Expo: Power for Land, Sea, and Air*, pp. 2123–2134, 06 2012.

[41] C. Reiber and V. A. Chenaux, "Compressor Mild Surge Simulation With Variable Nozzle Models: Influence of Throttle Area On Surge Behavior and Aeroelastic Stability at Reverse Flow Conditions," *13th European Conference on Turbomachinery Fluid Dynamics and Thermodynamics*, 2019.

[42] S. J. Plimpton, B. Hendrickson, and J. R. Stewart, "A parallel rendezvous algorithm for interpolation between multiple grids," *Journal of Parallel and Distributed Computing*, vol. 64, no. 2, pp. 266–276, 2004.

[43] I. Z. Reguly and G. R. Mudalige, "Modernising an industrial cfd application," in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 191–196, 2020.

[44] "OP2 for Many-Core Platforms," 2022 (Online). https://github.com/OP-DSL/OP2-Common.

[45] I. Z. Reguly, A. M. B. Owenson, A. Powell, S. A. Jarvis, and G. R. Mudalige, "Under the hood of sycl – an initial performance analysis with an unstructured-mesh cfd application," in *High Performance Computing* (B. Chamberlain, A.-L. Varbanescu, H. Ltaief, and P. Luszczek, eds.), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 391–410, Springer, Jun 2021.

[46] "OP2 Example Parallel Applications," 2022 (Online). https://github.com/OP-DSL/OP2-APPS.

[47] D. Amirante, P. Adami, and N. J. Hills, "A multifidelity aero-thermal design approach for secondary air systems," *Journal of Engineering for Gas Turbines and Power*, vol. 143, no. 3, 2021.

[48] D. Amirante, V. Ganine, N. J. Hills, and P. Adami, "A coupling framework for multi-domain modelling and multi-physics simulations," *Entropy*, vol. 23, no. 6, 2021.

[49] J. Bonet and J. Peraire, "An alternating digital tree (adt) algorithm for 3d geometric searching and intersection problems," *International Journal for Numerical Methods in Engineering*, vol. 31, no. 1, pp. 1–17, 1991.

[50] P. Spalart and S. Allmaras, *A one-equation turbulence model for aerodynamic flows*. AIAA 1992-439. 30th Aerospace Sciences Meeting and Exhibit, January 1992.

[51] D. L. Rodriguez, M. J. Aftosmis, and M. Nemec, *Correction: Formulation and Implementation of Inflow/Outflow Boundary Conditions to Simulate Propulsive Effects*.

[52] "ARCHER2," Accessed Jan 2022. https://www.archer2.ac.uk.

[53] "Cirrus," Accessed Jan 2021. https://www.cirrus.ac.uk/.

[54] G. R. Mudalige, M. B. Giles, C. Bertolli, and P. H. Kelly, "Predictive modeling and analysis of op2 on distributed memory gpu clusters," PMBS '11, (New York, NY, USA), p. 3–4, Association for Computing Machinery, 2011.

[55] "From discussions with Rolls-Royce Hydra developer team," Aug 2021 - Jan 2022.

[56] "ARCHER," Accessed Jan 2022. https://www.archer.ac.uk/.

[57] G. M. Laskowski, J. Kopriva, V. Michelassi, S. Shankaran, U. Paliath, R. Bhaskaran, Q. Wang, C. Talnikar, Z. J. Wang, and F. Jia, *Future Directions of High Fidelity CFD for Aerothermal Turbomachinery Analysis and Design*. June 2016.

[58] S. Saito, K. Yamada, M. Furukawa, K. Watanabe, A. Matsuoka, and N. Niwa, "Flow Structure and Unsteady Behavior of Hub-Corner Separation in a Stator Cascade of a Multi-Stage Transonic Axial Compressor," vol. Volume 2A: Turbomachinery of *Turbo Expo: Power for Land, Sea, and Air*, 06 2018. V02AT39A030.

[59] J. P. Haug and R. Niehuis, "Full annulus simulations of a transonic axial compressor stage with distorted inflow at transonic and subsonic blade tip speed," *International Journal of Turbomachinery, Propulsion and Power*, vol. 3, no. 1, 2018.

[60] C. Pérez Arroyo, J. Dombard, F. Duchaine, L. Gicquel, B. Martin, N. Odier, and G. Staffelbach, "Towards the large-eddy simulation of a full engine: Integration of a 360 azimuthal degrees fan, compressor and combustion chamber. part i: Methodology and initialisation," *Journal of the Global Power and Propulsion Society*, no. May, pp. 1–16, 2021.

[61] C. Pérez Arroyo, J. Dombard, F. Duchaine, L. Gicquel, B. Martin, N. Odier, and G. Staffelbach, "Towards the large-eddy simulation of a full engine: Integration of a 360 azimuthal degrees fan, compressor and combustion chamber. part ii: Comparison against stand-alone simulations," *Journal of the Global Power and Propulsion Society*, no. May, pp. 1–16, 2021.