WARWICK
THE UNIVERSITY OF WARWICK

**Manuscript version: Author's Accepted Manuscript**
The version presented in WRAP is the author's accepted manuscript and may differ from the published version or Version of Record.

**Persistent WRAP URL:**
http://wrap.warwick.ac.uk/173244

**How to cite:**
Please refer to published version for the most recent bibliographic citation information. If a published version is known of, the repository item page linked to above, will contain details on accessing it.

**Copyright and reuse:**
The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions.

Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

**Publisher's statement:**
Please refer to the repository item page, publisher's statement section, for further information.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk.

**warwick.ac.uk/lib-publications**

# Predictive Analysis of Code Optimisations on Large-Scale Coupled CFD-Combustion Simulations using the CPX Mini-App

A. Powell, G.R. Mudalige
University of Warwick, Coventry, UK
{a.powell.3, g.mudalige}@warwick.ac.uk

***Abstract*—As the complexity of multi-physics simulations increases, there is a need for efficient flow of information between components. Discrete 'coupler' codes can abstract away this process, improving solver interoperability. One such multi-physics problem is modelling a gas turbine aero engine, where instances of rotor/stator CFD and combustion simulations are coupled. Allocating resources correctly and efficiently during production simulations is a significant challenge due to the large HPC resources required and the varying scalability of specific components, a result of differences between solver physics. In this research, we develop a coupled mini-app simulation and an accompanying performance model to help support this process. We integrate an existing Particle-In-Cell mini-app, SIMPIC, as a 'performance proxy' for production combustion codes in industry, into a coupled mini-app CFD simulation using the CPX mini-coupler. The bottlenecks of the workload are examined, and the performance behavior are replicated using the mini-app. A selection of optimizations are examined, allowing to estimate the workload's theoretical performance. The coupling of mini-apps is supported by an empirical performance model which is then used to load balance and predict the speedup of a full-scale compressor-combustor-turbine simulation of 1.2Bn cells, a production representative problem size. The model is validated on 40K-cores of an HPE-Cray EX system, predicting the runtime of the mini-app work-flow with over 75% accuracy. The developed coupled mini-apps and empirical model combination demonstrates how rapid design space and run-time setup exploration studies can be carried out to obtain the best performance from full-scale Combustion-CFD coupled simulations.**

***Index Terms*—Coupling, Mini-App, Performance model, CFD, Combustion**

## I. Introduction

Concurrent execution of multiple physical models as a single simulation has emerged as an important approach for modelling large-scale multi-physics phenomena. This strategy, as opposed to carrying out a single monolithic simulation, allows for decomposing complex systems into a series of smaller, interconnected components, which can utilize the optimal method for modelling the physics of each sub-domain. Not only does this approach simplify code development and maintenance, but also allows domain scientists to select the optimal numerical methods and architectural optimizations for modelling different physical environments.

The challenge with such a modular approach is the efficient flow of information between the multiple models through common interfaces that does not lead to (1) numerical errors that a non-coupled or monolithic simulation would not have caused and (2) performance bottlenecks degrading the time-to-solution or throughput. A *coupler* acts as a discrete piece of code dedicated to implementing this flow of information and its design crucially determines the performance of the full simulation. For the domain of computational fluid dynamics (CFD), coupled simulations provide flexibility to combine specialized flow solvers and/or different turbulence models. Such coupled simulations are currently common practice in industry. Examples include the coupling of incompressible flow solvers for modelling flow within a combustion chamber with the compressible flow in the outlet vane [1] or using a hybrid RANS/LES modelling approach where the problem domain is decomposed and different turbulence models are applied depending on flow conditions of each zone [2], [3].

The most complex coupling scenarios consists of close interaction of CFD models with other numerical simulations, including FEM and combustion, for real-world systems. One such example [4] is modelling a gas turbine engine, consisting of multiple high-pressure compressor and/or turbine stages interacting with the combustion chamber. In this setup (see Figure 1), cold air entering the engine is compressed before it is delivered to the combustion chamber, where it is sprayed with fuel and ignited. The exhaust air from the resulting combustion provides the thrust that drives the turbines which, in turn, spins the compressor and fan. Currently, the components are designed and manufactured separately, which can lead to reduced engine performance due to optimizations made within individual elements being negated as a result of integration with other components. This is because without coupled simulations, there is information loss as unsteady interaction between components cannot be modelled [5]. Such coupled simulations have become furthermore important with the industry ambitions to reach virtual certification of full aero-engine designs [6], [7]. Ultra-high fidelity simulations are required, pushing current model sizes of 10-100 million elements to tens of billions of elements.

In this paper we focus on one such complex scenario – the coupled CFD simulation of the compressor-combustor-turbine components, directed by these industry challenges. In this setup, the compressor and turbine are typically modelled using a *density-based*, explicit CFD solver, with the combustion chamber employing a LES turbulence model with Lagrangian fuel spray [9], and a *pressure-based*, implicit CFD solver. The

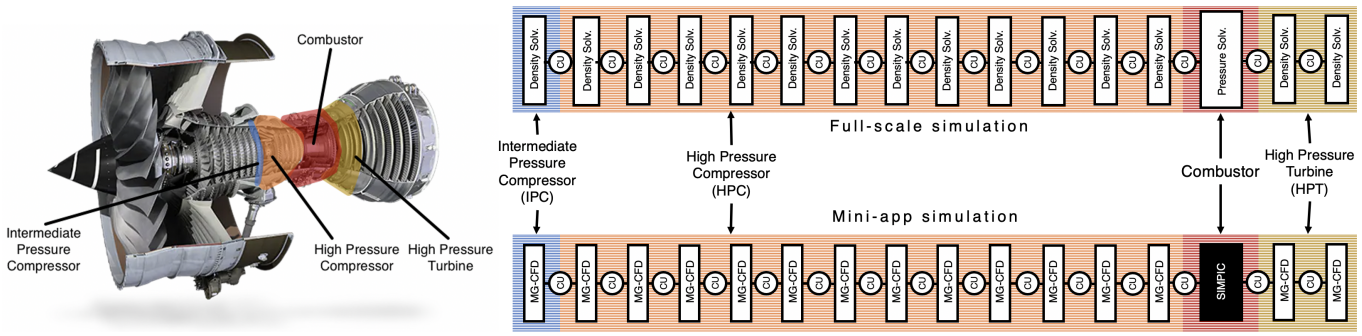



**Fig. 1:** A breakdown of the 1.5Bn Rolls-Royce test case setup for both the mini-app and full-scale simulation (Right). The chambers have been labelled on a representative RR Trent XWB Engine [8] (Left, reproduced with permission).

rotor/stator interaction in the density solver, uses the sliding planes method. A steady-state approach is used to model interaction between the density and pressure solvers [10]. Dedicated coupler(s) software implements these interactions. Our motivating problem is a representative 1.5Bn cell test case from Rolls-Royce plc., consisting of 13 compressor rows, a combustor chamber, and two turbine rows. Work is in progress to run this test case using Rolls-Royce's internal density-based CFD application [11], pressure-based combustion CFD application [9], and an in-house coupling framework [12]. As the content of this work is applicable to any typical coupled gas-turbine engine simulation using equivalent solvers, we will use the term 'density solver' and 'pressure solver' when referring to these applications.

When coupling between component simulations, codes are tasked with communicating boundary information, where the coupler(s) map values/fields from one simulation to the other, interpolating data and transferring them as required [13]. A key challenge to gain efficient execution is load balancing. Resources (e.g. compute processes) need to be carefully allocated for each component simulation, to elicit maximum concurrency. At the same time, couplers themselves need sufficient dedicated resources to execute the transfer of information, without which they will lead to performance bottlenecks, slowing down the overall simulation. However, given the different coupling interfaces and interactions between components, the scale of production executions, including long execution times (in the order of days or weeks) and the need to use large HPC systems, a direct, brute force tuning of runtime parameters and load balancing is prohibitively expensive and unachievable under production settings.

As a solution, in this work, we create a simplified version of a full scale Compressor-Combustor-Turbine simulation using scaled-down, but representative applications. The idea follows on from the widely used technique in HPC where simplified versions of large applications, called *mini-apps*, are used to explore co-design, performance and optimum configurations of applications on HPC systems [14], [15]. Building on the coupled multi-row compressor mini-app from [13] with MG-CFD [16] and CPX [13] as the proxy for the density-solver and coupler respectively, a new pressure solver proxy modelled by the SIMPIC [17] Particle-In-Cell (PIC) code is added to create the compressor-combustor-turbine triple-components as a coupled mini-app simulation. A comparison of the full-scale and proxy simulations can be seen in Figure 1, along with a diagram highlighting the related engine components simulated. An extended empirical performance model is developed to reason-about and explore the optimal resource allocation for the coupled mini-apps when executing a large mini-app engine simulation. We then use the analysis and predictions from the mini-app simulation and performance model to predict the optimum allocation of resources for the full-scale simulation of the 1.5B HPC-Combustor-HPT test case. More specifically, we make the following contributions:

- The electrostatic Particle-In-Cell mini-app, SIMPIC [17], is used as a black-box performance proxy to model the behavior of the combustor pressure solver typically found in aero-engine simulations. This base SIMPIC test-case (Base-STC) is tested using a variety of configurations, predicting the pressure-solver run-time with a max error of 22%. The mini-app is used to gain insights into the main bottlenecks of the production pressure solver.
- A selection of particle and multigrid solver optimizations are explored, and their contribution to performance are estimated theoretically, if applied to the production pressure-solve. An optimized SIMPIC test-case (Optimized-STC), is created from SIMPIC which synthetically matches this theoretical performance.
- SIMPIC is then coupled with multiple instances of the MG-CFD mini-app together with the CPX mini-coupler to create a coupled mini-app simulation. An empirical performance model is developed, extended from [13], to include the pressure solver and its coupling. This allows us to load balance and predict the speedup of the complete mini-simulation. Both the Base-STC and Optimized-STC versions are modelled.
- The model predicts the runtime of each individual component of the coupled mini-app simulation, each with less than 25% error. Furthermore, the large coupled mini-app simulation of the HPC-Combustor-HPT test case is executed on up to 40,000 cores on the ARCHER2 (HPE-Cray EX) supercomputer comparing runtimes with predictions from the performance model. Predictions are over 75% accurate for the both Base-STC and Optimized-STC coupled versions.

The primary objective of the work is to predict the best allocation of resources for running coupled CFD/combustion

using mini-apps, as there can be a significant bottleneck on the parallel efficiency of the simulation if resources are not adequately distributed. Using the full production codes will waste HPC resources and due to the scale required to run, the problem quickly becomes intractable when reaching current production problem sizes that are at 1B to 5B cells. The challenge of configuration exploration becomes even greater as the amount of compute resources increases. In addition, we aim to determine the bottlenecks for the workload in the codes themselves and examine optimizations, speculating what the optimized run-time would be. Similarly, without mini-apps, it becomes difficult and time consuming to achieve due to the size and complexities of the production codes. Addressing these bottlenecks represents a step as part of an ongoing push towards virtual certification. While we focus on the behavior of this gas-turbine simulation problem, the use of CPX can be applied when coupling mini-apps in other domains. Additionally, the particle and solver optimizations mentioned can also be applied to other similar problems.

## II. Background and Related Work

Numerical simulations based modelling of aero-engine design, traditionally limit their scope to individual segments/componenets of the engine such as the compressor or combustion chamber. Interactions between different sections are handled using boundary conditions, often taken from the time-averaged solution of the adjacent components. However, information is often lost as a result of this process, as bidirectional transfer of some information, particularly unsteady interactions such as turbulence intensity, are not possible. Therefore, it is desirable to run multiple components together in a single simulation such that interactions can be more accurately captured [5].

A common approach to model components together is to use a Large Eddy Simulation (LES) or Reynolds-averaged Navier-Stokes (RANS) approach in the Combustor and a Unsteady Reynolds-averaged Navier-Stokes (URANS) for the blade rows [5]. While this method has been applied to a variety of coupled gas turbine models [18] [19], accurately coupling together all three components (compressor, combustor, turbine) remains a key challenge. The first published example, by Stanford University's Centre for Turbulence Research, use a LES approach for the Combustor and URANS for the compressor and turbine. However, due to the difficulty in coupling the codes, the work reports inaccuracies with the generated results [20]. The first mathematically accurate example was performed by NUMECA, simulating compressor-combustor-turbine interaction in a KJ66 gas turbine engine [21]. However, this was achieved using a RANS approach throughout the simulation, which is a much low accuracy method compared to a URANS or LES model that is typically needed for production aero-engine design. Furthemore, the engine simulated was two orders of magnitude smaller in length than a typical aero-engine found on commercial aircraft [22].

More recently, a team at CERFACS successfully ran a coupled three component simulation, modelling the fan, compressor, and combustion chamber of a DGEN-380 turbine engine [5]. This simulation used a LES approach throughout, which ensures very high accuracy. However, the solution require an initial simulation of a group of smaller components as stand-alone simulations; the integrated/coupled simulation cannot finish without this. Nevertheless, their work provides a good point of reference as the mesh size used is very similar to the test case we are examining in this research (2.1Bn cells), albeit with fewer components.

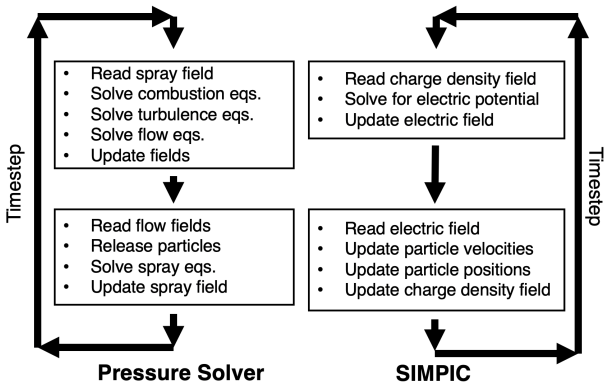



**Fig. 2:** A comparison between the compute-communication pattern of both the pressure-solver and SIMPIC [23] [24] [17]

### A. Interface types

In this work, we focus on a coupled URANS-LES simulation, with URANS used for the density solver (compressor and turbine), and LES used for the pressure solver (combustion chamber). While this is faster than using a LES throughout, URANS-LES interaction does result in approximation of some quantities. Therefore, frequent interaction/transfer between components is required to ensure stability of the overall system. When building the mini-app simulation, we will therefore explore the overhead of using an overlapping approach, where a composite domain is created from a larger portion of the interacting meshes. This is similar to using 'overset' methods for modelling rotor/stator interaction [25].

The coupling between density-solvers is modeled as a sliding-plane interaction, whereas the interaction between the density and pressure solvers is modeled as steady-state. In a sliding-plane, the rows of rotor blades move relative to the stator blades rows every timestep, thus the mapping between each domain must be recomputed every time-step [25] so that data can be transferred to and from each instance. The recalculation, consisting of a search routine among other operations, incurs significant computational overheads [13]. On the other hand, the density-pressure interaction requires a larger interface (5% of the mesh compared to 0.42% in sliding plane interaction), but the instances are not moving relative to one another. Thus, the mapping between the domains only needs to be computed once. As a result, the interaction between density-pressure solver contributes a much smaller overhead to the run-time over the course of the entire simulation than that of the density-density solvers.

### B. Coupling

As Figure 1 highlights, the component interactions/transfers are carried out by a series of *coupled* components. For the

domain scientists, this provides the flexibility to select, for instance, the best numerical method and problem scale for each component, or even parallellization/target architecture to execute the sub-components. It also significantly simplifies code maintenance and extension essentially implementing a horizontal *separation of concerns* approach, akin to the "vertical" separation of concerns achieved by DSL's [26] in HPC, where expertise in developing different simulation models can be leveraged to gain the best results.

There are a number of tools available which implement the required information exchange between discrete codes. These frameworks vary in scope and function; frameworks such as MUI [27] and preCISE [28] act purely as an interface where data can be sent and retrieved, whereas others, such as MCT [29], are more involved, with dedicated classes for data fields and methods for interpolation and other transformations. JM76, modeled in this research, operates at a lower level, with the communication and interpolation being hand coded; however, from a design perspective it most closely resembles the OpenPALM framework, developed in-part by CERFACS [30].

Regardless of the chosen framework, the significant challenges of resource allocation in coupled codes is well documented [12]. Depending on the coupling approach, the exchange of information between solvers may contribute significantly to the run-time of the simulation. Since the publication of previous work covering coupled compressor problems [13], significant progress has been made to reduce the coupling overhead to less than 10% of run-time, resulting in 88% parallel efficiency of the simulation at ∼10,000 cores [31]. It makes use of the same Rolls-Royce internal density solver and coupling framework as modelled in this paper. We demonstrates equivalent efficiency with smaller test cases. Thus, we can conclude that similar level of parallel efficiency can be achieved in the compressor and turbine stages of the proposed test case in this research.

Existing research suggests the scalability of the pressure solver application used in the combustor is comparatively worse; in an 84M cell test case, the parallel efficiency dropped below 50% parallel efficiency at only 896 cores [32]. Even when improvements were made to eliminate the overhead of the particle load balancing, parallel efficiency still dropped below 80% at the same 896 cores.

While hardware differences make a direct comparison with current knowledge difficult (with the pressure solver being tested on a machine with 32 Cores per node vs 128 Cores per node for the density solver), current analysis suggests in a coupled gas-turbine engine simulation, the pressure solver will most likely be the bottleneck. This is further exacerbated as the length of the pressure solver time-step is approximately half as long as those of the density solver, resulting in twice the number of iterations being required in a coupled simulation. This is due to the pressure solver using a more accurate LES model compared to the URANS model of the density solver, which typically requires a finer temporal resolution [33], [34].

### C. The pressure solver and mini-apps

The pressure solver operates using a synchronous coupled Lagrangian-Eulerian approach; within a time-step, flow, combustion, and turbulence fields are updated, passed to the particles, and the particle field is then updated [24] [9]. This process is then repeated for the next time-step, in Figure 2.

In previous coupled simulation work, where the compressor stage of a gas turbine engine was modelled [13], a "proxy" simulation was created by developing a new coupling mini-app, CPX, and coupling instances of the density solver mini-app, MG-CFD [16]. In this paper, we extend this mini-simulation by modelling the compressor, combustion chamber and turbine stages. Additionally, previously the size of all of the instances' meshes and their respective interfaces were the same; in our mini-simulation we are using a selection of meshes and coupling interface sizes to more accurately model the simulation.

In contrast to that work, a suitable functional mini-app for the pressure solver is not available. Hence an existing electrostatic mini-app, SIMPIC [35], developed by Sandia National Laboratories, was selected to represent the pressure solver. SIMPIC, in this case was chosen to act as a "performance proxy" - an application in a similar domain, designed to replicate the performance characteristics of an application rather than the functions within. The aim with a performance proxy is to demonstrate how the parallel efficiency of one solver can affect the scalability of a large connected simulation.

SIMPIC is used to model plasma interaction within nuclear fusion simulations, combining a particle-in-cell approach with a Poisson solver. Similar to production pressure solvers, it follows a synchronous Lagrangian-Eulerian approach; each time-step, the program solves an electric field, passing this information to the particles, which then update their field respectively [17]. A comparison between the two applications can be seen in Figure 2. SIMPIC is small (LoC≈ 1500), open source, and follows our motivating compute-communication pattern found in a typical pressure-solve. It is also written in C++, thus simplifying the integration into the CPX mini-coupler in comparison to say a Fortran-based pressure-solver [9]. Additionally, SIMPIC input files are highly customizable, enabling us to generate a selection of test cases which replicate different parallel efficiency characteristics and pressure solver run-times. However, there are limitations in using such an application. For example, SIMPIC operates over a 1-dimensional domain compared to the 3 dimensional domain of a typical cumbustion pressure solver. Nevertheless, the utility of the application in this research is its performance characteristics and scalability, not the underlying solver. It is for this reason we describe it as a "performance proxy", rather than a direct proxy application.

### D. Performance modelling

To assist our mini-app simulation, an existing predictive performance model used to predict run-time in coupled CFD simulations will be adopted [13]. The use of performance modelling is well documented, and have been used for a

variety of single program, multiple domain (SPMD) applications. These include models for particle transport and wavefront applications [36] and nonlinear solvers [37]. Performance models incorporate a number of parameters, such as domain size and frequency of certain loops, using these to produce an approximation for the run-time, or an optimized setup. The model from [13] is extended by adding support to model SIMPIC and its coupled interaction. In this research, we use the predictive model for two purposes: (1) allocating MPI ranks to ensure the lowest possible run-time, and (2) predicting speedups with the Optimized-STC over the Base-STC in the mini-app simulation. A detailed explanation of the model, its use, as well as the extensions model over [13] is presented in Section V.

## III. Performance analysis of the pressure solver and a representative SIMPIC test case

| Pressure solver mesh size | SIMPIC # Cells | SIMPIC Particles per cell | SIMPIC Timesteps |
|---|---|---|---|
| 28M | 512,000 | 100 | 50,000 |
| 84M | 512,000 | 300 | 50,000 |
| 380M | 512,000 | 1800 | 50,000 |

**Fig. 3:** A comparison between the pressure solver test cases and the equivalent SIMPIC test cases which replicate the performance behavior.

Figure 1 shows the combustion chamber in the context of the engine. In our initial performance analysis of the pressure solver, we will examine two test cases: (1) a single sector swirl combustion case, with 28M cells and 7M particles, and (2) a triple sector swirl combustion case with 84M cells and 21M particles [38]. These were run for 10 timesteps, measuring the run-time excluding the pre- and post-processing costs (which are fixed). The hardware used was the 740k core HPE-Cray EX supercomputer, ARCHER2 [39], which uses nodes of 2×64C AMD EPYC 7742 2.25Ghz microprocessors, with 256GB memory per node. All codes were compiled using CRAY 10.0.4 Fortran/C++ compilers and MPICH 8.0.16 MPI.

As seen in Figure 4b, parallel efficiency of the pressure solver drops below 50% at 3000 cores. Despite the increased size of the 84M test case compared to the 28M, the drop in performance is similar. This is significant as the full-scale test case for the pressure solver is large at ∼400M cells (comparable to that of the CERFACS integrated model [5]), indicating parallel efficiency will drop off at a similar point.

To predict the 28M and 84M pressure solver run-time with SIMPIC, we hand picked the parameters seen in Figure 3. Figure 4a and 4b shows the SIMPIC speedup and parallel efficiency compared to the pressure solver. SIMPIC is able to predict the run-time for the pressure solver across both test cases with a mean error of less than 9% and a worst case error of 22%. This error is a result of SIMPIC's parallel efficiency drop-off being marginally less steep than the pressure solver.

We can generate an indication of the pressure solver speedup in the full-scale 380M cell test case from the parameters in Figure 3. Figure 4c shows the results of this test, from 1,000 to 10,000 cores (8 to 80 ARCHER2 nodes). Parallel efficiency approaches 50% at only 10,000 cores, suggesting a maximum speedup of about 6x in this test case.

## IV. Performance analysis and Optimizations

The pressure solver models motion of fluid, as a result of the combustion of fuel, by solving the Navier-Stokes equations using pressure correction methods. It employs a LES approach, including a k-$\epsilon$ model for mdelling turbulence, a common turbulence model used in industry. A Finite-Volume approach is used for the geometry. Fuel spray droplets are handled using a Lagrangian approach, where particles are moved every timestep once the other fields have finished updating, as noted in Figure 2. Combustion-turbulence interaction is implemented with a variety of standard CFD-Combustion models (Eddy break-up, PDFs) [9]. Modelling combustion and turbulence is computationally expensive, but many assumptions can be made to simplify transport equations using scalar and velocity components [40]. However, solving the pressure equation requires many iterations and remains an expensive operation [41].

To understand the parallel efficiency behavior of the pressure solve, the code was profiled using ARM MAP [42] using the small 28M mesh on 2048 cores or 16 ARCHER2 nodes. Above 2048 cores the parallel efficiency drops below 50%, as Figure 4c highlights, hence this configuration was specifically profiled. Figure 5a presents the results from the profiling where we plot the run-time of each of the main functions in the production code relative to the total run-time. These are further split into compute and communication time. 46% of the run-time is spent updating the pressure field, with 21% of it being spent in MPI communications and 25% in compute. The pressure field routines use a Conjugate Gradient solver with Aggregate Algebraic Multigrid (AMG), thus it is likely that the bulk of communications time is being spent in near-neighbour data exchange in SpGEMM and SpMV operations. Further profiling has shown the bulk of compute time is spent iterating through multigrid cycles and in the setup phase, including calculating the Galerkin coarse grid operator.

The particle spray routine is the next most time-consuming routine, spending 96% of its run-time in communications. Other work profiling the pressure solver has shown that this is the result of poor distribution of the particles across cores [32].

As the velocity fields and scalar calculations scale well, we do not focus our efforts on these methods when attempting to improve performance. Instead we focus exclusively on the particle component and the pressure fields. Some of the optimizations we will discuss can also be applied in other methods, specifically those that accelerate SpMV operations, however when estimating the performance of a theoretical optimized pressure solver, we will use the parallel efficiency of these other components from the base code.

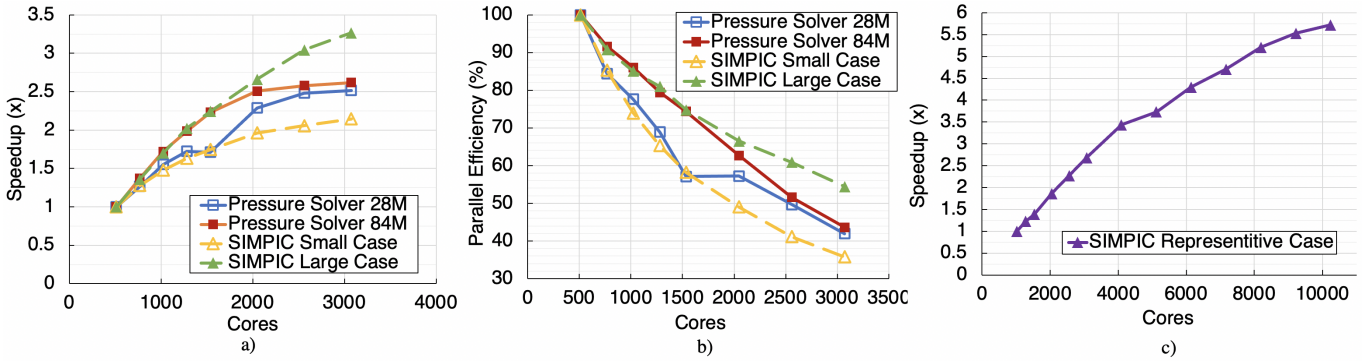



**Fig. 4:** (a) Speedup and (b) parallel efficiency of the pressure solver and SIMPIC on ARCHER2. (c) speed-up of SIMPIC with the representative large base test case on ARCHER2.

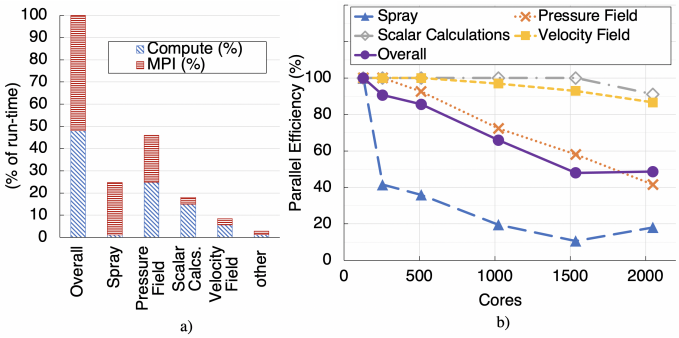



**Fig. 5:** Pressure solver performance (28M cells) - breakdown of the most time-consuming functions: (a) runtime as a proportion of total run-time at 2048 ARCHER2 cores, (b) parallel efficiency of each function up to 2048 cores.

### A. *Spray methods*

Figure 5b gives a further breakdown, detailing the parallel efficiency of each of the methods from 128 to 2048 cores, as well as the overall parallel efficiency curve. The same 28M mesh is used. It is clear that the particle component, modelling the fuel spray, is the biggest bottleneck in the pressure solver application, dropping below 50% parallel efficiency at just 2 ARCHER2 nodes, or 256 cores.

High communication overhead is common with codes that contain a particle-in-cell component, due to challenges handling load balancing and information exchange with the solver efficiently. One approach, spacial partitioning, partitions the solver space, with each MPI rank handling the particles that belong in its partition [43]. This approach, used by the pressure solver, struggles when particles are not evenly distributed across the partitions. An alternate approach, particle sharing, distributes the particles across ranks regardless of their location [44]. However, this approach requires collective operations which can significantly degrade performance at high core counts. As a result, spatial partitioning is most commonly used, often with hybrid MPI+OpenMP to take advantage of shared memory space and improve cache reuse [45].

Recently, an asynchronous task based approach was tested on the pressure solver application, dividing the MPI space into distinct spray and solver communicators [24]. With this method, the two components run independently, synchronising using one-sided MPI shared memory communicators introduced in MPI-3 [46]. This approach was demonstrated to work at scale [32], hence we include it as one of the optimizations to apply and analyze in the pressure solver.

### B. *Pressure field*

Figure 5b shows that the pressure field component also suffers a drop in parallel efficiency, albeit not to the same extent as the spray routines. Even with perfect scaling of the particles, the code will still drop to ~60% parallel efficiency at only 2048 ranks, as shown in Figure 6a. Thus, optimizing the spray alone will likely not be enough to ensure the pressure solver not being a bottleneck in the full-scale test case.

Due to the nature of the pressure equation being solved in the pressure field method, the solver requires many more iterations to converge than other calculations such as momentum or scalar transport equations [41]. Thus, a multigrid method must be used to improve the convergence rate. Multigrid techniques work by gradually lowering the resolution of the grid, solving the set of equations, and then interpolating the grid back up to its original resolution [47]. As the pressure solver being examined is unstructured [9], it uses Algebraic Multigrid (AMG), which applies this technique to the system of equations themselves, rather than the geometry [48]. Further profiling of the pressure field methods indicate the majority of time is spent in this process, thus we focus on accelerating the AMG, with an emphasis on accelerating SpGEMM and SpMV operations as these are also used in other methods such as those solving transport equations [49]. There is a vast body of research optimizing these routines [48] [50]. Here we focus on the most significant optimizations:

*1) AMG setup:*

- For the smoother, Hybrid Gauss-Seidel should be used, employing Gauss-Seidel within a task but Jacobi methods across parallel tasks. This leads to better convergence within each multigrid cycle provided the problem size is sufficiently large [51].
- Within each MG cycle, the grid is restricted, solved, and interpolated until the finest mesh is created. We propose to use extended+i methods in the interpolation, which considers not only neighbors of a gridpoint but also its neighbors' neighbors. While this is more computationally expensive, it also accelerates convergence [52].
- In some AMG solvers, the MG cycle (V-Cycle) convergence time can be reduced using Krylov subspace

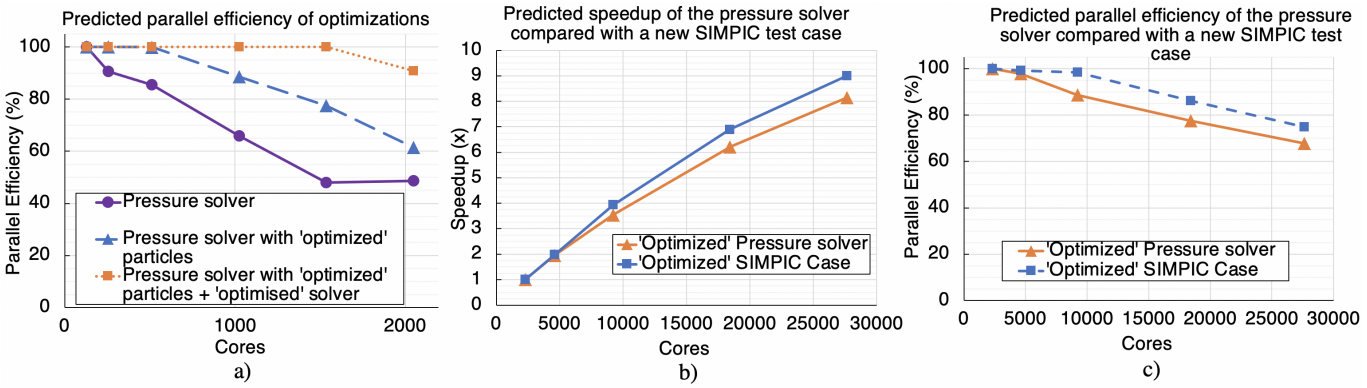



**Fig. 6:** Predicted pressure solver parallel efficiency before and after particle and solver optimizations (a), along side the predicted speedup of an optimized pressure solver and the actual speedup of the Optimized-STC on ARCHER2 (b,c).

acceleration, known as a K-cycle. However, this method can result in poor scalability of the solver with large numbers of cores [50]. We therefore recommend using V-Cycles with smoothed parallel pattern matching [53].

*2) Matrix and vector multiplication:*

- In traditional SpGEMM operations, the input matrices must be read twice: once to determine the size of the output matrix, and again during the multiplication [54]. Instead, each thread can be allocated a large chunk of memory and the disjoint results can be copied to a contiguous memory space [48].
- To reduce the amount of branching in SpGEMM operations, a sparse accumulator (SPA) can be constructed which allows access of any matrix element in constant time [48] [55].
- During interpolation and restriction, which uses SpMV, values at the same points are are mapped directly to the mesh above or below. As a result, the matrix can be rearranged such that the first rows are an identity matrix, which reduces computation and saves memory bandwidth [48].
- When AMG is used in a distributed system, matrix rows are spread across cores, in a compressed sparse row format. During a SpGEMM operation, a thread must renumber its column mapping array as it is likely it has received new values in the halo exchange. This is an expensive process as efficient parallel reordering is difficult to achieve. Instead, each thread can build a hash map, which is then merged into a global array using a parallel merge sort. Using a reverse mapping, the values can be distributed back to the relevant threads [48].

### C. Extrapolating improvements

When applying the particle optimizations to our previous results in Figure 5, we set the parallel efficiency of the spray routines to 100%, as the research indicates there is little difference in scaling between the pressure solver with the optimized spray and the same code without any spray routines [32]. For the pressure field, the AMG setup changes are required such that the matrix and vector multiplication optimizations work as effectively as possible, so these do not directly affect run-time. With matrix and scalar vector multiplication changes, the research [48] indicates a speedup of ~3.5x over the base solvers at 128 nodes, limited by communication in halo exchanges. Since our cluster has ~9x the cores per node as in the research, we expect performance to be more in line with the parts of the AMG solver which are less affected by communication, which give an average speedup of 5x. We therefore apply a 5x speedup to the pressure field to approximate the benefits of the improvements with this optimization. The results from the the estimated performance gains of these optimizations are shown in Figure 6a. While this is only an approximation, it is clear from the above related work/research that there is potential for significantly improved scalability compared to the base application.

A SIMPIC configuration can be generated to match the parallel efficiency of the above estimated optimizations for the pressure solver. This consists of 1.18M cells, 60,000 particles per cell, and running for 450 time-steps. The configuration predicts the estimated optimized pressure solver run-time with an error of less than 7%. The speedup and parallel efficiency of both can be seen in Figure 6b and 6c.

## V. An Empirical Performance model for Resource Allocation and Performance Analysis

We have shown how each mini-app (MG-CFD, SIMPIC and CPX) can be configured to match the performance behavior of the full applications (density-solve, pressure-solve, coupler). The individual execution times of each mini-app can then be used to reason about how best to allocate resources to each full application to obtain best performance. We build an empirical performance model to achieve this. A similar model was created previously for the coupled production density solver [13] to generate an optimized configuration of a compressor test case. We follow the same development process to tackle the larger problem at hand, that of modelling the HPC-Combustor-HPT coupled execution.

As discussed before, a coupled simulation progresses by exchanging information at some regular frequency between the coupled applications through coupler units (see Figure 1). The frequency of exchanges is determined by each application; for example the density solver, after each iteration would communicate with the pressure solver, awaits for the pressure solver to complete some number of iterations, and return results before progressing to the next iteration. This cascading dependency down a sequence of application instances results in the overall simulation progressing at the speed of the slowest

component. Therefore, a key criteria for improving run-time of the full simulation is to allocate compute resources to the slowest component to balance the load. The model aims to solve this resource distribution problem, allocating MPI ranks such that the combined run-time of mini-apps and coupler units is the lowest possible. This is challenging as not only do the mesh sizes differ between instances, so too does frequency of the iterations; for each density solver time-step, there are 2 pressure solver time-steps. The size of the coupler unit interfaces also vary, as well as their frequency; instances of coupled density solvers must exchange data every iteration of the solver, whereas the coupling between pressure and density solvers only exchange every 20 iterations.

**Algorithm 1:** Distribute ranks to coupled mini-apps

**Input:** $T[apps]$, $Size[apps]$, $Size[CUs]$, $Iter[apps, CUs]$
**Result:** Optimized core distribution and predicted run-time of coupled simulation
**for** $i$ = 1, *Num. of mini-app instances* **do**
  $T[app_i] = T[app_i] * \frac{Size[app_i]}{Base\_size} * \frac{Iter[app_i]}{Base\_iter}$
**end**
**for** $i$ = 1, *Num. of coupler units* **do**
  $T[CU_i] = T[CU_i] * \frac{Size[CU_i]}{Base\_size} * \frac{Iter[CU_i]}{Base\_iter}$
**end**
$ranks \leftarrow$ # cores
**while** $ranks > 0$
  $App_{max} \leftarrow MAX(T[app_1], T[app_2], ..., T[app_n])$
  $CU_{max} \leftarrow MAX(T[CU_1], T[CU_2], ..., T[CU_n])$
  $App_{diff} \leftarrow T(cores[App_{max}]) - T(cores[App_{max}] + 1)$
  $CU_{diff} \leftarrow T(cores[CU_{max}]) - T(cores[CU_{max}] + 1)$
  **if** $CU_{diff} > App_{diff}$ **then**
    $cores[CU_{max}] \leftarrow cores[CU_{max}] + 1$
  **else**
    $cores[App_{max}] \leftarrow cores[App_{max}] + 1$
  **end**
  $ranks = ranks - 1$
**end**
$output \leftarrow MAX(app_1, ..., app_n) + MAX(CU_1, ..., CU_n)$
$output \leftarrow cores[apps], cores[CUs]$

The model, Alg. 1, implemented as an iterative algorithm, loops to allocate the total core budget, $N$ to each of the coupled components. For each iteration, the algorithm compares the run-times of each mini-app instance, selecting the instance with the longest run-time as well as the slowest coupler unit. The program then compares the reduction in run-time of allocating a core to both, choosing the option where the reduction in run-time is the greatest. As the run-time of the simulation is the sum of the slowest mini-app instance and the slowest coupler unit, this results in the greatest reduction of run-time for every core allocated.

The initial mini-app and coupler run-times are calculated by comparing the number of timesteps and the problem size of each instance to a base case, and scaling the initial run-time up or down for the required core count. For example, with MG-CFD, the base case is run on an 8M mesh for 25 timesteps. If a simulation contains an instance of MG-CFD, with a mesh of 24M cells and a simulation length of 250 timesteps, the initial run-time will be 30x the base case. To estimate the run-time for each mini-app and coupler unit, we benchmark each

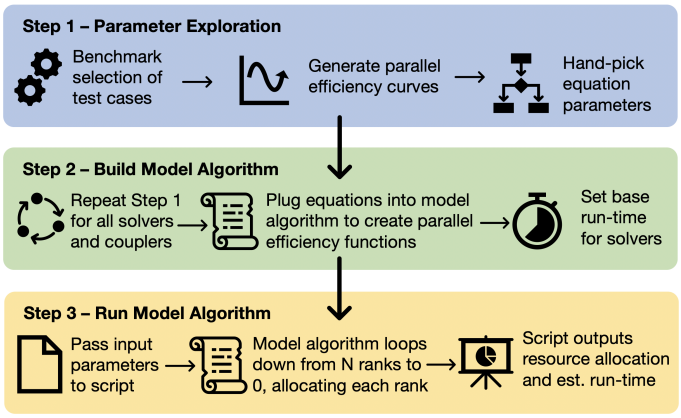


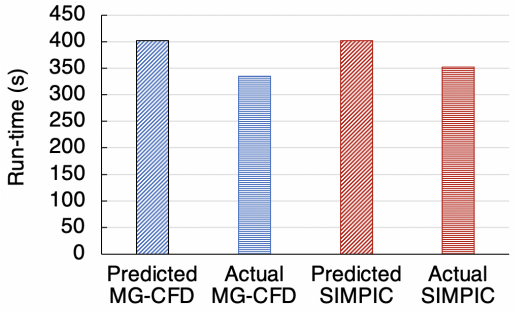

**Fig. 7:** A breakdown of how the empirical model is used to generate the resource allocation and predict the run-time of a coupled simulation.

mini-app standalone across problem sizes and core counts, generate a parallel efficiency graph for each problem size, and then fit a curve to the graph. From this, the script contains functions take in the number of cores and the mesh size, and output the parallel efficiency of the mini-app at that core count / configuration. Figure 7 provides an overview of this benchmarking and modelling process to estimate run-time in a coupled simulation leading to their use in Alg. 1.

This work presents several significant improvements over prior work. In the original implementation [13], all instances of the solvers must have the same mesh and interface size. We have extended the model so each mini-app instance can have a different mesh size and interface size. As a result, run-time comparisons and core allocation is done on a per-instance basis, rather than crudely allocating resources to either all solver instances or all coupler instances. Additionally, the model now supports instances of both pressure-solver and density solver mini-apps, whereas before it could only generate resource allocation and run-time predictions of coupled density solver mini-simulations. The extended model, therefore enables rapid resource distribution decision making for a full HPC-Combustor-HPT simulations, facilitating load-balancing for more complete workloads and real-world scenarios.

### *A. Model Validation: Small Test - 150M/28M*

Figure 8a compares a run-time prediction in the extended predictive model with the mini-app runs in a typical medium-size High-Pressure Turbine test case. This scenario consists of 1 SIMPIC unit and 2 MG-CFD units, with 5,000 cores allocated to the total simulation. The MG-CFD meshes are instances solving the NASA Rotor37 [56] meshes (150M nodes), which represent the geometry of a transonic axial compressor rotor, widely used for validation in CFD. The SIMPIC test case represents a pressure solver over 28M cells. 331 ranks are allocated per MG-CFD unit, with 4,253 allocated to SIMPIC. For this validation, SIMPIC is configured in its Base-STC setup. 63 CU are used between the MG-CFD units, with 22 CU being allocated between the SIMPIC and MG-CFD units. As can be seen from Figure 8a, the performance




a)

| Instance # | Application | Mesh size (millions) | MPI Ranks (Base-STC) |
|---|---|---|---|
| 1 | SIMPIC | 28 | 4,253 |
| 2,3 | MG-CFD | 150 | 331 |

b)

**Fig. 8:** Left (a) presents a comparison between predicted and actual run-times of MG-CFD and SIMPIC in a small test case, and (b) details the mesh size of each component in the small mini-app simulation.

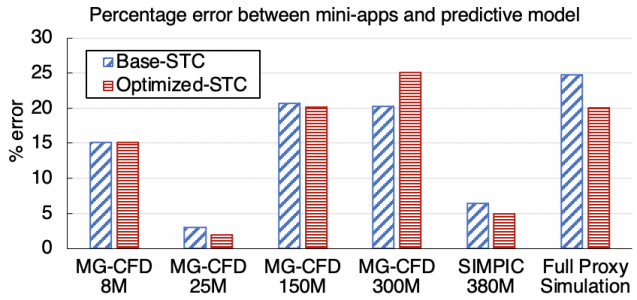
model is able to load balance the applications and predict the run-time with a maximum error of 18%.

### *B. Model Validation : Large full-scale equivalent test*

The second validation case attempts to replicate the HPC-Combustor-HPT setup from Figure 1. The mesh sizes for each component simulation is detailed in Figure 8b. Note that the listed SIMPIC mesh size is the equivalent cell size in the full application; the actual input size that represent performance of a 1.2M cells problem. However, we choose to quote 380M since the size of the interfaces passed to the coupler units are made to match the full-scale simulation. In total, the effective size of this mini-app simulation is 1.25Bn cells, equivalent to our main motivating problem from industry.

Research suggests that beyond ~30,000 cores, the setup cost of the AMG in the pressure solver becomes significant [50]. As a result, we enter these parameters into the model with a core allocation budget of 40,000 cores; ~30,000 cores for SIMPIC and ~10,000 cores to the rest of the simulation. We ran the simulation for the equivalent of 20 time-steps of the pressure solver, comparing the model's predicted run-time of the simulation with the standalone run-time of each mini-app instance. The number of cores allocated to each instance by the model is shown in Figure 9b. This was done such that in the full, longer mini-app simulation, we have confidence that the resource allocation algorithm will work correctly. Figure 9a shows the percentage error of each instance size for mini-app simulations using the Base-STC and the Optimized-STC. It demonstrates the model can predict the run-time of all instances used in both mini-apps and by extension full-scale simulations with a worst case error of 25% and a mean error of 12%.

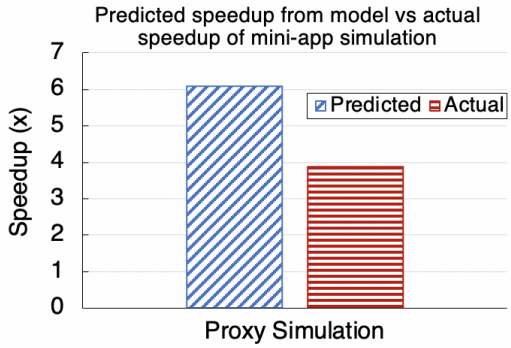
The same setup was then ran for the equivalent of 1,000 time-steps of the density solver, which is the time taken for half a revolution of a gas-turbine engine. The length was




a)

| Instance # | Application | Mesh size (millions) | MPI Ranks (Base-STC) | MPI Ranks (Optimized-STC) |
|---|---|---|---|---|
| 1 | MG-CFD | 8 | 100 | 100 |
| 2-12 | MG-CFD | 24 | 100 | 163 |
| 13 | MG-CFD | 150 | 167 | 1218 |
| 14 | SIMPIC | 380 | 13428 | 32201 |
| 15 | MG-CFD | 150 | 167 | 1218 |
| 16 | MG-CFD | 300 | 338 | 3357 |

b)




c)

**Fig. 9:** (a) percentage error between the mini-apps and predictive model for individual mini-app simulations using the Base-STC and the Optimized-STC. (b) rank allocation to each instance of mini-app, (c) predicted and actual speedups for 1 revolution of the mini-app simulation using the Optimized-STC compared to a mini-app simulation using the Base-STC.

chosen as it is large enough to give representative data without wasting HPC resources. With the large mini-app simulation using the Base-STC example, the bottleneck was SIMPIC, which reached 50% parallel efficiency at 13,428 ranks, shown in Figure 9b. The difference between the overall predicted run-time and the SIMPIC component was 5%.

The model starts rank allocation at 100 for this problem size as mini-app base cases run in a sufficiently short time, yet have a higher parallel efficiency at 100 ranks. As a consequence, the MG-CFD 8M instance gets allocated more ranks than necessary to match the run-time of the overall simulation. However, in this base test case, the only place to re-allocate additional ranks to improve run-time would be to SIMPIC, and as the parallel efficiency of the SIMPIC instance is already at 50%, the impact on overall run-time would be negligible. Thus, in the full-scale simulation, we predict the pressure solver application would be the bottleneck in the simulation.

In the mini-app simulation using the Optimized-STC, the model predicts that both the density and pressure solvers will scale well at 40,000 cores. The model predicts that with 32,201 cores allocated, or ∼262 ARCHER2 nodes, parallel efficiency of the pressure solver will be 87%, this being the worst out of all the application instances.

The model predicts coupling overhead to be <0.5% of overall run-time. This is in contrast to the significant bottleneck predicted by the model in [13]. We attribute the reduction in overhead due to the improved search algorithm, with a tree-based search routine and by pre-fetching the cells required for the next iteration, for sliding-planes interaction which was subsequently implemented into the industrial coupler [31].

Comparing the two test cases, Figure 9c presents the predicted speedup in a 1 revolution simulation if an optimized pressure solver was used in place the original. The model predicts that by optimizing the pressure solver with the improvements discussed in Section III, the simulation can be sped up significantly by a factor of 6x.

### C. Final Results and Analysis

Figure 9c compares the predicted speedup with the measured speedup on ARCHER2 for both the Base-STC and Optimized-STC mini-app setups. To save time on the HPC system, the measured run-time is taken from a 0.5 revolution run and then doubled to match the 1 revolution prediction. The model is able to predict the run-time of both scenarios with a maximum error of less than 25%, and demonstrates a speed-up of approximately ∼4x when comparing the Optimized-STC simulation to the coupled mini-app simulation using the Base-STC. The difference between the predicted and actual mini-app simulation run-time appears mostly a result of SIMPIC's base test case running faster than expected. The input parameters required to match the base pressure solver scaling, which are not typical of a real world electrostatic problem, cause run-time per time-step to improve when the number of time-steps are significantly increased. This behavior is not seen in the production pressure solver, and highlights a limitation with using a mini-app that was not derived from the production application. Despite this, the predictive model and large mini-app run suggest that a speedup of between 4x-6x is possible.

It is important to be aware of the other limitations with the 'performance proxy' approach presented here. The optimizations presented may deliver different parallel efficiency improvements, and the performance characteristics may not resemble those that were predicted. In the ideal case, where the speedup matches the best cases from the quoted research, we predict overall speedup to be ∼7.5x over the base case. In a worst case scenario, where the particle optimizations are applied, pressure field run-time is reduced only by 30% and pressure field parallel efficiency doesn't improve from the base case, we predict speedup to be only 2.3x. The results supporting these optimization improvements are available only in the mini-app domain, as work running the full-scale simulation is still in progress. However, it is clear from profiling the pressure solver and existing research on the application itself that there is clear scope for efficiency gains.

## VI. Conclusions

In this paper, we have demonstrated the use of mini-apps, the CPX mini-coupler, and a performance model to investigate, model and predict the run-time for a gas-turbine engine simulation. Extending previous work on coupled CFD instances [13], we profiled a production combustion application and reasoned about the theoretical efficiency gains using a variety of existing solver and particle optimizations. Combining this with an improved performance model and a representative 'performance proxy', we were able to optimally load balance and accurately predict the run-time of a large mini-app simulation using 40,000 cores. Finally, the bottlenecks of this simulation were examined, highlighting that due to increased accuracy and smaller time-step size of a LES, even an optimized version of the pressure solver code would likely remain the bottleneck in the simulation. The techniques presented can be applied to a variety of mini-applications and kernels, allowing researchers to optimize load balancing and investigate performance bottlenecks of a coupled system in a relatively short space of time. In addition, work is ongoing to include FEM solvers for thermal coupling of the engine casing, allowing us to run coupled CFD, Combustion and Structural simulations.

## Acknowledgements

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). We would also like to thank Chris Goddard at Rolls-Royce for their guidance for this work.

## References

[1] K. Kannan and G. Page, "Coupling of compressible turbomachinery and incompressible combustor flow solvers for aerothermal applications," in *Turbo Expo: Power for Land, Sea, and Air*, vol. 45608, p. V02AT40A004, American Society of Mechanical Engineers, 2014.

[2] J. Fröhlich and D. Von Terzi, "Hybrid les/rans methods for the simulation of turbulent flows," *Progress in Aerospace Sciences*, vol. 44, no. 5, pp. 349–377, 2008.

[3] J. U. Schlüter, X. Wu, S. Kim, S. Shankaran, J. Alonso, and H. Pitsch, "A framework for coupling reynolds-averaged with large-eddy simulations for gas turbine applications," 2005.

[4] "Joliot-curie supercomputer used to build first full, high-fidelity aircraft engine simulation," Accessed May 2022. https://cerfacs.fr/en/actualite/first-360-degrees-large-eddy-simulation-of-a-full-engine/.

[5] C. P. 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*, vol. 2021, no. May, pp. 1–16, 2021.

[6] "Strategic partnership in computational science for advanced simulation and modelling of engineering systems - asimov," Accessed May 2022. https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/S005072/1.

[7] *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.

[8] "World's most efficient large aero-engine - trent xwb," Accessed May 2022. https://www.rolls-royce.com/products-and-services/civil-aerospace/airlines/trent-xwb.aspx#section-technology.

[9] M. Anand, R. Eggels, M. Staufer, M. Zedda, and J. Zhu, "An advanced unstructured-grid finite-volume design system for gas turbine combustion analysis," in *Gas Turbine India Conference*, vol. 35161, p. V001T03A003, American Society of Mechanical Engineers, 2013.

[10] 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, p. 758, 2021.

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

[12] 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, p. 758, 2021.

[13] A. Powell, K. Choudry, A. Prabhakar, I. Reguly, D. Amirante, S. Jarvis, and G. R. 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, IEEE, 2021.

[14] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy, and S. A. Jarvis, "Exploring simd for molecular dynamics, using intel® xeon® processors and intel® xeon phi coprocessors," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 1085–1097, IEEE, 2013.

[15] I. Z. Reguly, G. R. Mudalige, and M. B. Giles, "Design and development of domain specific active libraries with proxy applications," in *2015 IEEE International Conference on Cluster Computing*, pp. 738–745, IEEE, 2015.

[16] A. Owenson, S. A. Wright, R. A. Bunt, Y. Ho, M. J. Street, and S. A. Jarvis, "An unstructured cfd mini-application for the performance prediction of a production cfd code," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 10, p. e5443, 2020.

[17] "Simpic - simple 1d pic prototype," Accessed May 2022. https://lecad-peg.bitbucket.io/simpic/simpic.html.

[18] S. Jacobi, C. Mazzoni, B. Rosic, and K. Chana, "Investigation of unsteady flow phenomena in first vane caused by combustor flow with swirl," *Journal of Turbomachinery*, vol. 139, no. 4, p. 041006, 2017.

[19] S. Salvadori, G. Riccio, M. Insinna, and F. Martelli, "Analysis of combustor/vane interaction with decoupled and loosely coupled approaches," in *Turbo Expo: Power for Land, Sea, and Air*, vol. 44748, pp. 2641–2652, American Society of Mechanical Engineers, 2012.

[20] J. Schlüter, S. Apte, G. Kalitzin, E. vd Weide, J. Alonso, and H. Pitsch, "Large-scale integrated les-rans simulations of a gas turbine engine," *Annual Research Briefs*, pp. 111–120, 2005.

[21] L. Romagnosi, Y. Li, M. Mezine, M. Teixeira, S. Vilmin, J. E. Anker, K. Claramunt, Y. Baux, and C. Hirsch, "A methodology for steady and unsteady full-engine simulations," in *Turbo Expo: Power for Land, Sea, and Air*, vol. 58578, p. V02CT41A002, American Society of Mechanical Engineers, 2019.

[22] E. Saab Gripen, "Rolls-royce trent xwb awarded easa type certification,"

[23] G. Houzeaux, M. Garcia, J. C. Cajas, A. Artigues, E. Olivares, J. Labarta, and M. Vázquez, "Dynamic load balance applied to particle transport in fluids," *International Journal of Computational Fluid Dynamics*, vol. 30, no. 6, pp. 408–418, 2016.

[24] A. Thari, N. C. Treleaven, M. Staufer, and G. J. Page, "Parallel load-balancing for combustion with spray for large-scale simulation," *Journal of Computational Physics*, vol. 434, p. 110187, 2021.

[25] 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 & Fluids*, vol. 115, pp. 140–153, 2015.

[26] T. Cleenewerck and I. Kurtev, "Separation of concerns in translational semantics for dsls in model engineering," in *Proceedings of the 2007 ACM symposium on applied computing*, pp. 985–992, 2007.

[27] A. Skillen, S. Longshaw, G. Cartland-Glover, C. Moulinec, and D. Emerson, "Profiling and application of the multi-scale universal interface (mui),"

[28] 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.

[29] 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.

[30] 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.

[31] G. R. Mudalige, I. Z. Reguly, A. Prabhakar, D. Amirante, L. Lapworth, and S. A. Jarvis, "Towards virtual certification of gas turbine engines with performance-portable simulations," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 206–217, 2022.

[32] A. Thari, M. Staufer, and G. Page, "Asynchronous task based eulerian-lagrangian parallel solver for combustion applications," *Journal of Computational Physics*, vol. 458, p. 111103, 2022.

[33] M. Young and A. Ooi, "Comparative assessment of les and urans for flow over a cylinder at a reynolds number of 3900," 2007.

[34] A. Sadiki, A. Maltsev, B. Wegner, F. Flemming, A. Kempf, and J. Janicka, "Unsteady methods (urans and les) for simulation of combustion systems," *International Journal of Thermal Sciences*, vol. 45, no. 8, pp. 760–773, 2006.

[35] B. T. Yee and M. M. Hopkins, "Particle methods for revealing kinetic plasma behavior.," tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2020.

[36] G. R. Mudalige, M. K. Vernon, and S. A. Jarvis, "A plug-and-play model for evaluating wavefront computations on parallel architectures," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–14, IEEE, 2008.

[37] R. A. Bunt, S. A. Wright, S. A. Jarvis, Y. Ho, and M. J. Street, "Predictive evaluation of partitioning algorithms through runtime modelling," in *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, pp. 351–361, IEEE, 2016.

[38] J. Sidey, A. Giusti, P. Benie, and E. Mastorakos, "The swirl flames data repository," 2017.

[39] "Archer2," Accessed May 2022. https://www.archer2.ac.uk.

[40] H. K. Versteeg and W. Malalasekera, *An introduction to computational fluid dynamics: the finite volume method*. Pearson education, 2007.

[41] W. Shyy, M.-H. Chen, and C.-S. Sun, "Pressure-based multigrid algorithm for flow at all speeds," *AIAA journal*, vol. 30, no. 11, pp. 2660–2669, 1992.

[42] "Arm map - arm forge," Accessed May 2022. https://www.arm.com/products/development-tools/server-and-hpc/forge/map.

[43] R. Pankajakshan, B. J. Mitchell, and L. K. Taylor, "Simulation of unsteady two-phase flows using a parallel eulerian–lagrangian approach," *Computers & fluids*, vol. 41, no. 1, pp. 20–26, 2011.

[44] Y. Shigeto and M. Sakai, "Parallel computing of discrete element method on multi-core processors," *Particuology*, vol. 9, no. 4, pp. 398–405, 2011.

[45] M. T. Bettencourt, D. Brown, K. L. Cartwright, E. C. Cyr, C. A. Glusa, P. T. Lin, S. G. Moore, D. McGregor, R. P. Pawlowski, E. G. Phillips, *et al.*, "Empire-pic: a performance portable unstructured particle-in-cell code," *Communications in Computational Physics*, vol. 30, no. SAND-2021-2806J, 2021.

[46] W. Gropp, "Mpi 3 and beyond: why mpi is successful and what challenges it faces," in *European MPI Users' Group Meeting*, pp. 1–9, Springer, 2012.

[47] S. Thakur, J. Wright, W. Shyy, J. Liu, H. Ouyang, and T. Vu, "Development of pressure-based composite multigrid methods for complex fluid flows," *Progress in Aerospace Sciences*, vol. 32, no. 4, pp. 313–375, 1996.

[48] J. Park, M. Smelyanskiy, U. M. Yang, D. Mudigere, and P. Dubey, "High-performance algebraic multigrid solver optimized for multi-core based distributed parallel systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2015.

[49] F. Trias, X. Alvarez-Farré, A. Alsalti-Baldellou, A. Gorobets, and A. Oliva, "Dns/les using a minimal set of algebraic kernels: Challenges and opportunities,"

[50] P. D'Ambra, F. Durastante, and S. Filippone, "Amg preconditioners for linear solvers towards extreme scale," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. S679–S703, 2021.

[51] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, "Multigrid smoothers for ultraparallel computing," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2864–2887, 2011.

[52] H. De Sterck, R. D. Falgout, J. W. Nolting, and U. M. Yang, "Distance-two interpolation for parallel algebraic multigrid," *Numerical Linear Algebra with Applications*, vol. 15, no. 2-3, pp. 115–139, 2008.

[53] P. D'Ambra and P. S. Vassilevski, "Adaptive amg with coarsening based on compatible weighted matching," *Computing and Visualization in Science*, vol. 16, no. 2, pp. 59–76, 2013.
[54] M. Patwary, M. Ali, N. R. Satish, N. Sundaram, J. Park, M. J. Anderson, S. G. Vadlamudi, D. Das, S. G. Pudov, V. O. Pirogov, *et al.*, "Parallel efficient sparse matrix-matrix multiplication on multicore platforms," in *International Conference on High Performance Computing*, pp. 48–57, Springer, 2015.
[55] F. G. Gustavson, "Two fast algorithms for sparse matrices: Multiplication and permuted transposition," *ACM Transactions on Mathematical Software (TOMS)*, vol. 4, no. 3, pp. 250–269, 1978.
[56] J. D. Denton, "Lessons from rotor 37," *Journal of Thermal Science*, vol. 6, pp. 1–13, Mar. 1997.