Predictive Analysis of Large-Scale Coupled CFD Simulations with the CPX Mini-App

A. Powell*, K. Choudry*, A. Prabhakar*, I.Z. Reguly†, D. Amirante‡, S.A. Jarvis§, G.R. Mudalige*
*University of Warwick, UK. {a.powell.3, k.choudry, arun.prabhakar, g.mudalige}@warwick.ac.uk
† Pazmany Peter Catholic University, Hungary. reguly.istvan@itk.ppke.hu
‡University of Surrey, UK. d.amirante@surrey.ac.uk
§University of Birmingham, UK. s.a.jarvis@bham.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 the high pressure compressor of turbofan engines, where instances of rotor/stator CFD simulations are coupled. Configuring couplers and allocating resources correctly can be challenging for such problems due to the sliding interfaces between codes. In this research, we present CPX, a mini-coupler designed to model the performance behaviour of a production coupler framework at Rolls-Royce plc., used for coupling rotor/stator simulations. CPX, the first mini-coupler framework of its kind, is combined with a CFD mini-app to predict the run-time and scaling behaviour of large scale coupled CFD simulations. We demonstrate high qualitative and quantitative predictive accuracy with a less than 17% mean error. A performance model is developed to predict the 'optimum' configuration of resources, and is tested to show the high accuracy of these predictions. The model is also used to project the 'optimum' configuration for a 6 Billion cell test case, a problem size representative of current leading-edge production workloads, on a 100,000 core cluster and a 400 GPU cluster. Further testing reveals that the 'optimum' configuration is unstable if not set up correctly, and therefore a trade-off needs to be made with a marginally less-than-optimal setup to ensure stability. The work illustrates the significant utility of CPX to carry out such rapid design space and run-time setup exploration studies to obtain the best performance from production CFD coupled simulations.**

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

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-field. 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 [1] in HPC, where expertise in developing different simulation models can be leveraged to gain the best results.

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 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 an incompressible flow solver for modelling flow within a combustion chamber with the compressible flow in the outlet vane [2] or using a hybrid RANS/LES modelling approach where the problem domain is decomposed and different turbulence models are applied depending on the flow conditions of each zone [3]–[5]. More complex coupling scenarios have demonstrated the close interaction of CFD models with other numerical simulations, such as FEM and/or CEM for large scale systems, for instance in [6] where a full engine simulation is attempted with coupled multi-physics models.

Within a coupled CFD simulation, solvers interact with each other by transferring some "interface"or "boundary" data to its neighbouring simulation via the coupler(s). The couplers map the values, or fields from one simulation to the other, interpolating data / translating the information as required [7]. Naturally, adequate resources should also be allocated to the coupler components for efficient performance, aiming for a smooth overall simulation without performance bottlenecks due to the information exchange. This comes in addition to the more common challenge of balancing resources for each simulation, where maximum concurrency needs to be achieved without idling components awaiting data. As such, a key factor in developing and carrying out performant coupled simulations is the design and execution space exploration required to obtain the optimum load and resource balance between the multiple simulations and couplers. However, given the scale of production executions including long execution times and the need to use large HPC systems, a direct, brute force tuning of parameters, load balancing and runtime configuration is prohibitively expensive and laborious. In this paper we investigate techniques that aid such a design and runtime parameter/configuration space exploration for a large-scale production CFD workload.

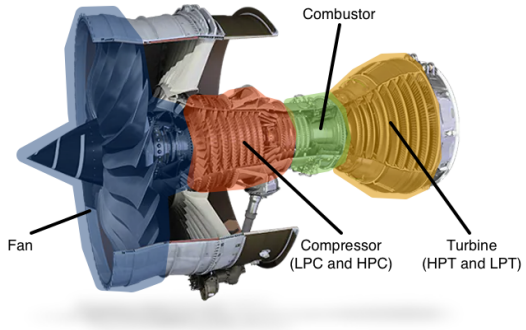

Fig. 1: RR Trent XWB Engine [9] (Reproduced with Permission)

Recent ongoing work re-engineering Rolls-Royce's production CFD suite, aims to modernize their turbomachinery design simulations for utilizing massively parallel architectures, such as the emerging Exascale systems. The underlying goal is to carry out ultra-high fidelity simulations for virtual certification of whole engine designs [8] which require current model sizes of 10-100 million elements to extend to trillions of elements. A key component of such large-scale simulation setups is the use of coupler software, linking together different components of the engine simulation. The coupler software used in this project supports a number of interfaces such as sliding planes (sliding meshes), conjugate heat transfer, mixing planes or overset (Chimera) interfaces depending on the information transferred from one simulation model to the other.

In the present work, we focus on the engine compressor, made with many blade-rows attached to a shaft which spins at high speed to compress air entering the front of the engine. Its function is to increase the air pressure, forcing the high pressure air into the combustion chamber, within which air is sprayed with fuel and ignited. The hot air exhaust from the resulting combustion provides the thrust that drives the turbines which in turn spin the compressor and fan (see Fig. 1). Simulation of the full compressor therefore consists of a coupled simulation with a number of Reynold's Average Navier-Stokes (RANS) or hybrid RANS/Large eddy simulation (LES) models representing simulation of rotor and stator blades, interlinked with sliding plane interfaces (see Fig. 2). Each of the rotors and stators are simulated using instances of the Rolls-Royce's Hydra CFD application [10], an unstructured-mesh application which uses a 5-step Runge-Kutta method for time-marching, accelerated by multigrid and block-Jacobi preconditioning. The Hydra instances (we henceforth call Hydra Units, HUs) are linked together by custom Coupler Units (CUs) as illustrated in the figure above. The custom coupling configuration is then set up such that the HUs operate on distinct meshes that cover adjacent or overlapping zones of the physical space [11]. On a parallel HPC system, distributed memory (MPI) parallelism is used, where each HU is assigned a number of (MPI) processes. A CU is also allocated a number of processes to carry out the inter-linking between the HUs. In order to simplify the interfaces, improve parallelism and in-turn performance, a single CU will only manage one geometric interface, namely the interface (or part of the interface) shared

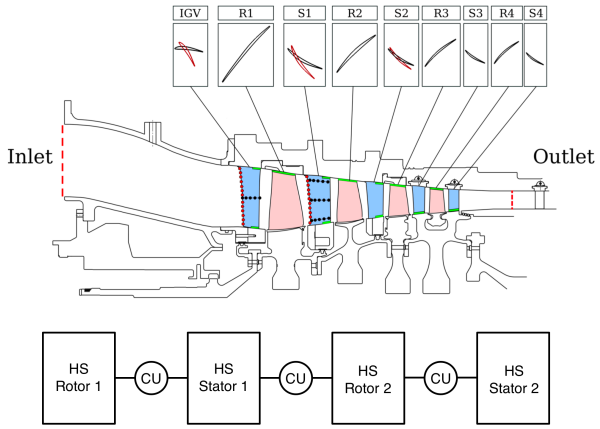

Fig. 2: An annotated model of the DLR Rig250 compressor test case (above) and setup of a Rotor/Stator test case with Hydra Units (HUs) and Coupler Units (CUs) (below) [12] (Reproduced with Permission)

by two HUs. This means that the interface shared by the two HUs can be allocated multiple CUs, each handling a segment of the interface. Consequently this leads to a complex configuration of HUs-to-CUs connectivity and MPI processes to HUs and CUs, that for a given overall simulation, require careful orchestration of distributed memory resources to obtain optimal performance. As each rotor moves through every time step, sliding planes interfaces must be used, where the mapping between the stator and rotor interfaces must also be recomputed for each time step. This process, performed by the CUs, can be a significant bottleneck on the parallel efficiency of the simulation if resources are not adequately distributed. The problem quickly becomes intractable when reaching current production problem sizes that are at 1B to 5B cells for a typical compressor.

In this paper, we attempt to develop techniques to solve the above design space and configuration exploration problem. We develop a stripped-down, "proxy" coupler framework called CPX (derived from **C**ou**p**ler E**x**tensions), producing a tractable, yet representative application to aid the investigation. 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 [13], [14]. To build a proxy coupled CFD simulation, two mini-apps are required - a coupler mini-app and CFD mini-app. Rolls-Royce already uses an open-source mini-app called MG-CFD [15] to assist the development of the Hydra CFD application. As such, our focus in this work is the equivalent coupler mini-app.

By combining instances of MG-CFD together using CPX, we investigate, model and predict the optimum HU-to-CU connectivity/configuration, aiming to understand how to best allocate resources for full-scale production simulations. More specifically we make the following contributions:

- We present CPX, a representative mini-app designed to match the performance behaviour of Rolls-Royce's production coupler framework. CPX is combined with the MG-CFD mini-app to create proxy configurations to capture the

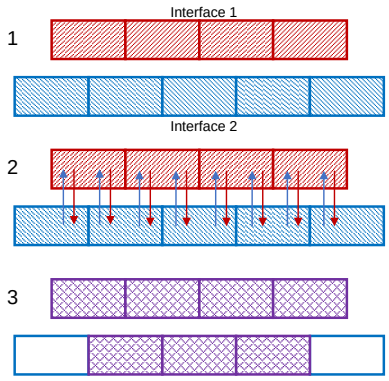

Fig. 3: Interpolation between coupled interfaces. The colours represent the different values in the interfaces, which after data exchange and interpolation then match.

operation of the production simulations. The run-time and scaling behaviour of MG-CFD with CPX is shown to have the same quantitative and qualitative behaviour as Rolls-Royce's production applications.

- An analytical comparison is used to demonstrate that for a certain allocation of coupling resources, CPX and RR Coupler will have similar scalability behaviour. CPX + MG-CFD is then tested using a variety of configurations, and compared to RR Coupler + Hydra, showing that proxy apps can predict the run-time with a less than 17% mean error.
- Using the insights from CPX and MG-CFD, an analytical performance model is created to predict the best resource allocation for a given coupled simulation of the production applications. The model is used to predict optimum configuration settings on a cluster, and the prediction is then tested against other configurations on the 740,000 core ARCHER2 supercomputer [16] to show that the predicted configuration is optimal. Finally, the model is used to predict the optimum configuration for a 6Bn cell test case using 100,000 cores, as well as the same run on a 400 GPU system.

While the main use case of CPX is for capturing the behaviour of a specific production framework, it can equally be used as a representation of a general purpose coupled CFD simulation. Additionally, the design principles seen in CPX can be applied when coupling proxy applications in other domains.

II. Background and Related Work

The process of coupling codes usually involves four core stages. Firstly, interfaces which sit between simulations are defined, and are used as the domain in which data is transferred. These interfaces are made of slices of the original domains, and by ensuring that the values across the interfaces match, the system remains stable. This can be done as a preprocessing step or at run-time. Next, a mapping is created which links the cells in the interfaces, so when data is transferred (e.g for every iteration of a solver) the cells of the interface mesh know where to send and receive data. These first two stages are usually only performed once.

Once the solvers are running, they transfer data between each other during each iteration. When data is received, it is

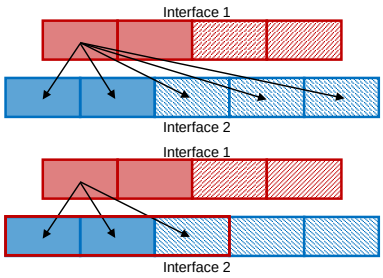

Fig. 4: The interface search routines have been split across 4 MPI ranks, shown by the different colours in each interfaces.

interpolated to ensure the values remain consistent across the interfaces. This interpolation is necessary as the interfaces may not align, thus the values for a particular interface cell will be a combination of multiple cells from the other interface, as well as its prior value. This can be seen in Figure 3. The 'communicate and interpolate' process is repeated for every iteration of the solver, until the simulation concludes.

Traditional code coupling is a well-studied field, and a variety of frameworks exist to aid the implementation of these stages. These frameworks vary in scope and function; frameworks such as MUI [17] and preCISE [18] act purely as an interface where data can be sent and retrieved, whereas others, such as MCT [7], are more involved, with dedicated classes for data fields and methods for interpolation and other transformations. The RR Coupler framework 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 [19]. By using these coupling frameworks, developers can simplify the coupling process and speed up development time.

However, when coupling sliding planes, the focus of this research, the process is more involved. With a sliding plane, one of the interfaces moves relative to the other interface, which means that the mapping which links the cells in one interface to the other must be recomputed every time the interface moves. In the context of a series of coupled Rotor/Stator chambers, such as those shown in Figure 2, this recomputing needs to be performed every time the Rotor blade(s) rotate. The process is intensive, as each cell in an interface needs to compare itself with every other cell in the other interface, and must be repeated for all interfaces in a coupled simulation, both moving and static. As a result of this 'brute force search', if the average interface size doubles, the search run-time increases by a factor of 4. This run-time increase becomes a problem as the geometric complexity of simulations grows and more detailed meshes are used.

To ensure that the search routines do not dominate run-time, it is important to split the work across MPI ranks. However, splitting this process across ranks will only decrease the search run-time by a linear factor, since the nodes in an interface will still have to search every node in the other interface. This is because it is not known which cells correlate to each other and so there is no way to restrict the search area.

Instead, the interfaces can be split so that the domain

over which each node has to search is restricted, highlighted in Figure 4. This requires knowledge as to how far the sliding interface will move, as all required cells from the other interface must be included in the restricted domain. By reducing the size of the domain, and splitting the work across MPI ranks, the run-time can be decreased by the number of MPI ranks squared in the best case. Consider a future test case involving geometrically complex meshes, 10 times finer than the existing production case. In a sliding plane example, this will increase the search routine run-time by a factor of 100 (assuming the interface size has increased by the same factor as the mesh). If manual partitioning is used, and partitions are sub-divided by a factor of 10, increasing coupling resources (MPI ranks) by a factor of 10 should result in little-to-no increase in coupling run-time compared to the existing mesh. If no partitioning is used, then the number of MPI ranks has to be increased by a factor of 100 to achieve the same effect.

It is therefore advantageous to partition the mesh to ensure maximum parallel efficiency in sliding plane problems. However, since the boundaries in the Rotor/Stator test cases are radial, they cannot be split during run-time and must be done manually. It makes the process time consuming, particularly when determining how to best split the MPI ranks between the main applications and the coupling, as this process must be done via trial and error. Most significantly, no existing coupling library has tools to help with this problem.

In the RR Coupler, the first stage of coupling (defining the interfaces) is performed before the code is run. The other three stages (communication, search, and interpolation) are performed by the CUs. Each Hydra Session runs for a set number of time-steps, in which the Rotor rotates every time-step, and between each time step the solver runs for a certain number of iterations. At every time-step, the CU(s) run a search routine to recompute the mapping between the interface cells. During solver iterations, the data is transferred from the HSs to the CU(s), where the CU(s) interpolate the data such that the values are consistent across interfaces.

The RR Coupler has 3 methods of increasing parallel efficiency within the coupling process. The first is to assign multiple MPI ranks to a single CU, which splits the search workload between ranks but does not partition the opposite interface, similar to the first example in Figure 4. The second is manually partitioning the mesh, assigning each part to a CU, and having multiple CUs between each HS. A single MPI rank can then be assigned per CU. This is an implementation of the second example in Figure 4. Finally, a hybrid approach can be used, assigning multiple CUs and multiple MPI ranks per CU. All 3 of these approaches will speed up the interpolation equally; however from a performance perspective, the second approach is the best, as the reduction in search time is the greatest when using this method.

The issue with the second option is that unlike with rank assignment, which can be done by simply changing the run-time command, partitioning the interfaces must be done manually. This is time consuming because the partitions must be a similar size to one another to ensure load balancing. In addition, as the number of partitions (and CUs) increase, the likelihood of one interface having no mapping to cells in the opposite interface also increases, which causes the RR Coupler to crash. As a result, having insight on an optimized CU and HU configuration can significantly reduce overall setup time. This research aims to achieve this by using a 'proxy' mini-app simulation and a performance model; a mini-coupler can be used as a test bed for different CU configurations and for verifying performance, and the performance model can be used to determine the optimum CU configurations.

The use of performance models is well documented, and have been used for a variety of SPMD applications. These include models for particle transport and wave-front applications [20] and nonlinear solvers [21] [22]. 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. No such performance model currently exists for a coupling framework, but the design process will be similar to that of traditional models.

III. Building the proxy simulation

To build a 'proxy' coupled CFD simulation, two proxy or mini-apps are required: a mini-coupler, and a standalone proxy CFD application. The mini-coupler component is the subject of this research, whereas for the proxy CFD application we chose an existing mini-app, MG-CFD. MG-CFD has previously been demonstrated to be an accurate proxy for predicting run-times and scaling behaviour of Hydra [15].

A. MG-CFD - A proxy for Hydra

The MG-CFD is a three-dimensional finite-volume discretization of the Euler equations for inviscid, compressible flow over an unstructured grid, and is based on the CFD application by Corrigan et al, included in the Rodinia benchmark suite [23]. The version used in this work is written using the OP2 DSL [24], which enables the automatic generation of parallelizations targeting clusters of multi-core CPUs and GPUs. This means that MPI, OpenMP and SYCL versions of MG-CFD can easily be built, as well as CUDA and HIP binaries that can run on Nvidia and AMD GPUs. This research focuses primarily on the MPI version, since it is the fastest CPU version of the code on any given platform [25]; however, we also consider the CUDA version of the code when developing and evaluating the performance model.

This proxy application is designed to be a representation of Hydra's *iflux* and *vflux* loops, which are part of Hydra's nonlinear solver and make up nearly half of the run-time. These handle inviscid and viscid flow, with the remainder of time spent in gradient sensitivity and calculating the source term of other equations. MG-CFD predict run-time of the *iflux* loop with less than 16% error when not using SIMD extensions [15]. The *vflux* loop is more computationally intense than *iflux*, but it has a similar memory access pattern, thus the number of cycles can be increased to model *vflux*. Consequently, MG-CFD fulfils our requirement of a proxy application for Hydra, particularly its most time consuming loops *iflux* and *vflux* which determines total run-time.

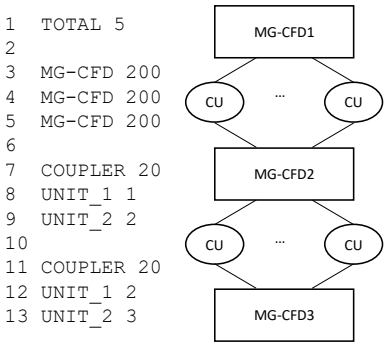

Fig. 5: A typical CPX input configuration file (left), specifying three MG-CFD Sessions, each with 200 MPI ranks, and two Coupler Units, each with 20 MPI ranks. The generated configuration (right), with Multi-Unit Mode (MUM) enabled.

B. Initial design considerations of a proxy coupler

When developing a proxy application, it is important to focus on what should be included and what is not required in order to adequately capture the properties of the larger application. The central aim of any proxy or mini-application is to have similar performance and scalability characteristics to the larger code, even if the methods used to achieve this are different [26]. It is important that the application should be limited in scope, to retain the properties that make proxy applications beneficial, but still close enough to the full application to remain useful. As a result, a number of simplifications were made to the design of the proxy framework in comparison to the full framework:

- The proxy code should use identical, static meshes compared to the different mesh sizes of the full simulation. The meshes should be approximately the size of the average mesh size in the full simulation
- The interfaces should also have a fixed size, whereas this is not the case with the full framework setup
- The coupling routines (communication, interpolation and searching) should represent/match the computational intensity of the full simulation. There is no requirement to carry out a valid calculation, and data should simply be discarded once it has been sent back to each MG-CFD Session.
- The framework should support either the RR Coupler's 'single Coupler Unit with multiple ranks' or the RR Coupler's 'multiple Coupler Units' from a performance perspective, switchable with a flag. Since increasing Coupler Units is always preferable, there is no need to have a mixture of Coupler Unit + MPI rank behaviour.

C. Components of RR Coupler and Design of CPX

1) Input files: The basic design of the proxy coupling framework, CPX, is very similar to the full scale RR Coupler framework [11], using MPI for distributed-memory parallelization. The input is read in from an input configuration file which specifies the number of MPI ranks assigned to each Coupler Unit, the number of MPI ranks assigned to MG-CFD Sessions and which Coupler Units are connected to which MG-CFD Sessions.

A flag can be set in the separate CPX global configuration file which switches between running a Coupler Unit in a '1 Unit per rank' mode and a 'Single Unit with many ranks' mode. This is known as Multi-Unit Mode (MUM). When enabled, the interface is split across ranks during the CPX brute force search routine. When disabled, each rank must perform a brute force search over the entire interface. As a result, the scaling behaviour in these two scenarios can be tested quickly without having to manually divide up the mesh, since the CPX meshes are static and the interface is pre-defined. Due to its rotating domain, doing this in the RR Coupler would be significantly more time consuming. Note that CPX can only ever have 1 physical Coupler Unit per 2 MG-CFD Sessions (i.e only 1 communicator), but enabling MUM, configures the single Coupler Unit to scale in a similar manner to that of using multiple RR Coupler CUs. Figure 5 shows an example CPX input configuration file, along with the resultant CPX configuration with MUM enabled. The input of the mesh remains the same as in the regular MG-CFD, being a parameter that is passed in at run-time.

2) Computation within RR Coupler and CPX: During the RR Coupler sliding-plane execution, there are three main operations: (1) Communication - the Hydra Sessions send data to their Coupler Unit(s), which then sends data back, (2) Search - the Coupler Units must do a brute force search on each of the interfaces to see which nodes overlap between them and (3) Interpolation - the data must be interpolated before being sent back to the Hydra Sessions.

Algorithm 1: The RR Coupler sequence of operations for a Coupler Unit (CU)

```
while true do                                  ▷ For every time step
    Open communications to receive data
    Search algorithm
    for i = 1, niter do
        if i ≠ 1 then open communications to receive data
        Wait until receive is completed
        Interpolate data on target nodes
        Send interpolated data
    end
end
```

Algorithm 2: The CPX sequence of operations for a Coupler Unit (CU)

```
for i = 1, number of MG cycles do
    Open communications to receive data
    Wait until receive is completed
    if i % searchfreq then
        Search algorithm
    end
    Interpolate data on target nodes
    Send interpolated data
end
```

Fig. 6: The CPX Coupler Unit (CU) algorithm in comparison to the RR Coupler Coupler Unit (CU) algorithm. CPX uses linear interpolation, a simplified version of the interpolation scheme used in the RR Coupler [11].

Out of the above, the search routine is the most expensive, since it is a brute force search running in $\mathcal{O}(n^2)$, where n

is the size of the interface mesh. However, it only needs to be done every time step and not every iteration, since the interface only changes every time step. The interpolate stage, which averages the data from the interfaces, must be run every time step. It runs in linear time, but significantly more number of times that the brute force search. In a typical RR Coupler + Hydra run, there is at the very least 10 iterations per time step, and often up to 50. The communication stage must also take place every time step, but it also scales linearly with a smaller contribution to run-time than the other operations.

Figure 6 shows the differences between the RR Coupler and CPX operations within each Coupler Unit. CPX follows a similar model to the RR Coupler, with search, interpolation and send routines, but with a few changes. Firstly, due to the static interfaces MG-CFD deals only with multigrid cycles, not time steps and iterations. As a result, the frequency at which the search routine operates is specified in the CPX global input file. Additionally, CPX uses blocking communication in the transfer between each MG-CFD Session and the respective Coupler Unit(s), so this must be completed before any computation can be achieved.

The search routine in CPX is a brute force search of the interface mesh, which is simply a small slice of the original MG-CFD mesh. This is completed by inserting the data from each node at the beginning of a C++ `std::vector`, and repeating the process for the size of the interface mesh. This gives the $\mathcal{O}(n^2)$ scaling with mesh size to match RR Coupler's search algorithm. The interpolation is an average of the interface nodes, which are accessed in constant time. All nodes in the interface are averaged, so the interpolation also matches RR Coupler's $\mathcal{O}(n)$ scalability. Finally, the process of communication between the MG-CFD Sessions and Coupler Unit(s) is also linear in interface size.

With Multi-Unit Mode enabled, the interface is split between the number of Coupler Unit ranks during the search routine. As shown in Section IV-A, if the mesh is split by a factor of m, then the run-time of the search algorithm decreases by $\mathcal{O}(m^2)$. This is because the interfaces (of both MG-CFD Sessions) will decrease in size by m, and the search routine involves a brute force search (of time $\mathcal{O}(m)$) which must be done m times. With Multi-Unit Mode disabled, the search routine is repeated m times, to replicate how the run-time of the RR Coupler search algorithm only decreases at most $\mathcal{O}(m)$, where m is the number of ranks, when a single Coupler Unit with multiple ranks is used.

IV. CPX vs the RR Coupler - Comparison

A. Analytical comparison

To ensure that CPX scaling is in line with that of the RR Coupler, we need to ensure that the CPX interpolation and search routines scale in an equivalent manner. We can compute the complexity of the algorithms (big-$\mathcal{O}$) in these routines as a function of the interface sizes and MPI ranks for both RR Coupler and CPX, to demonstrate equivalence.

1) RR Coupler run-time: Consider two Hydra Sessions, connected together with one or multiple Coupler Unit(s). Then let R_C be the number of Coupler Units/ranks, N_C^L be the interface size of the 'Left' Hydra Session and N_C^R be the interface size of the 'Right' Hydra Session. Then for 1 CU with multiple ranks:

$$\mathcal{O}(search) = \mathcal{O}((\frac{N_C^L}{R_C} \cdot N_C^R) + (\frac{N_C^R}{R_C} \cdot N_C^L)) = \mathcal{O}(\frac{N_C^L \cdot N_C^R}{R_C}) \quad (1)$$

$$\mathcal{O}(interpolate) = \mathcal{O}(\frac{N_C^L}{R_C} + \frac{N_C^R}{R_C}) = \mathcal{O}(\frac{N_C^L + N_C^R}{R_C}) \quad (2)$$

For multiple CUs:

$$\mathcal{O}(search) = \mathcal{O}(2 \cdot \frac{N_C^L}{R_C} \cdot \frac{N_C^R}{R_C}) = \mathcal{O}(\frac{N_C^L \cdot N_C^R}{{R_C}^2}) \quad (3)$$

$$\mathcal{O}(interpolate) = \mathcal{O}(\frac{N_C^L}{R_C} + \frac{N_C^R}{R_C}) = \mathcal{O}(\frac{N_C^L + N_C^R}{R_C}) \quad (4)$$

2) CPX run-time: Consider two MG-CFD Sessions, connected together with one Coupler Unit. Let R_C be the number of Coupler ranks and N_C be the interface size of each MG-CFD Session (since the interfaces are all the same size). Then for 1 CU with multiple ranks (MUM=OFF):

$$\mathcal{O}(search) = \mathcal{O}(2 \cdot (\frac{N_C}{2 \cdot R_C} \cdot \frac{N_C}{R_C} \cdot R_C)) = \mathcal{O}(\frac{{N_C}^2}{R_C}) \quad (5)$$

$$\mathcal{O}(interpolate) = \mathcal{O}(2 \cdot (\frac{N_C}{R_C \cdot R_C} \cdot R_C)) = \mathcal{O}(\frac{N_C}{R_C}) \quad (6)$$

For multiple CUs (MUM=ON):

$$\mathcal{O}(search) = \mathcal{O}(2 \cdot (\frac{N_C}{2 \cdot {R_C}^2} \cdot \frac{N_C}{R_C} \cdot R_C)) = \mathcal{O}(\frac{{N_C}^2}{{R_C}^2}) \quad (7)$$

$$\mathcal{O}(interpolate) = \mathcal{O}(2 \cdot (\frac{N_C}{R_C \cdot R_C} \cdot R_C)) = \mathcal{O}(\frac{N_C}{R_C}) \quad (8)$$

3) Comparison: The result of the analytical comparison shows that the scaling behaviour for the RR Coupler and CPX should be similar, other than the difference of RR Coupler's left and right Hydra interface sizes. If the CPX interface sizes are equal in size to the average of the RR Coupler interface sizes, then the scaling should be close. Of course, scaling is affected by a variety of factors, not just the number of ranks and the size of the domain. In addition, since the CPX search routine is artificial and simplified compared to the RR Coupler, even if the search routine's parallel efficiency is consistent with RR Coupler, its contribution to overall run-time may be different. This can be determined via testing however, and thus the comparison shown highlights how CPX scaling should approximately mirror RR Coupler, even if the overall run-time may need to be adjusted.

B. Numerical testing

1) 10M Cell Test 1: To test the run-time and scaling is close to the production coupling setup, two comparisons are performed. The first comparison involves a simple comparison of ~10M cell test cases, with the production codes using internal ~11M average cell test cases and the mini-apps using instances of the publicly available NASA Rotor37 8M mesh. The test is performed using 2 Hydra Sessions, with either 100, 200 or 400 MPI ranks each. For the RR Coupler, three Coupler Unit configurations are tested: A single CU with 1 MPI rank, a single CU with 4 MPI ranks and 4 CUs with a single MPI

TABLE I: RR Coupler ~10M Cell Tests (Test Case 1)

Hydra ranks	CUs	Ranks per CU	Time (s)
100	1	1	195.4
100	1	4	108.1
100	4	1	77.5
200	1	1	176.8
200	1	4	89.0
200	4	1	59.4
400	1	1	158.8
400	1	4	70.5
400	4	1	42.2

TABLE II: CPX ~10M Cell Tests (Test Case 1)

MG-CFD ranks	CUs	Ranks/CU	MUM?	Time (s)
100	1	1	OFF	192.4
100	1	4	OFF	81.8
100	1	4	ON	72.0
200	1	1	OFF	178.6
200	1	4	OFF	64.0
200	1	4	ON	53.7
400	1	1	OFF	167.0
400	1	4	OFF	55.5
400	1	4	ON	42.6

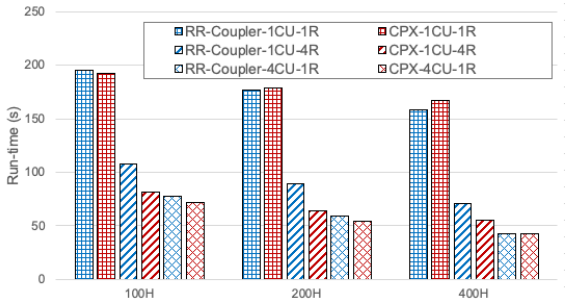

Fig. 7: Rolls-Royce Coupler vs. CPX. 1st~10M cell test case with 1 Coupler Unit, 4 Coupler Units and 1 Coupler Unit with 4 MPI ranks. (H - number of Hydra/MG-CFD ranks)

rank. The RR Coupler test case runs for 5 time steps, with 10 iterations per time step.

The equivalent setup in CPX with MG-CFD Sessions is then tested, with 5 search routines, and 250 MG cycles. Prior testing has shown MG cycles being 5x the number of total iterations provides a close comparison. The size of the boundary mesh is set to be the same across both test cases (~28,000), so the proxy and main coupler performance can be directly compared. However, the overall MG-CFD mesh is smaller than the Hydra mesh, and as such there will be small differences in the run-time. The tests are ran on the 740,000 core ARCHER2 supercomputer [16], which uses nodes of 2x64C AMD EPYC 7742 2.25Ghz microprocessors, with 256GB memory per node. All codes are compiled using the CRAY 10.0.4 compiler and MPICH 8.0.16 MPI.

Tables I and II show the performance comparison between RR Coupler and CPX. The result is displayed in Figure 7. The results show that the CPX produces a mean error of 11% when predicting the production code run-time, with a worse case prediction error of 28%. One noticeable characteristic of the CPX results is that the MUM=OFF results are much closer to the MUM=ON results than for the RR Coupler; this is due to the search routines being simpler on the mini-app than the production code. While they theoretically scale similarly, the search time durations is longer for the production code. This difference accounts for the majority of the prediction error, and since it is always ideal to assign MPI to one CU per rank over assigning multiple MPI ranks to a single CU, this error is largely insignificant. The error when comparing the RR Coupler multiple CU results to CPX (MUM=ON) results is small by comparison. The 11% average prediction error for CPX shows that the CPX mini-app behaviour is close to RR Coupler production code in this first test.

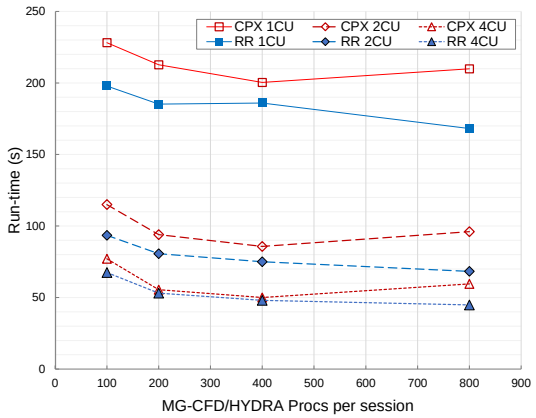

Fig. 8: Rolls-Royce Coupler vs. CPX. 2nd~10M cell test case with 1, 2, and 4 Coupler Units.

2) 10M Cell Test 2: The second comparison involves a different ~10M cell test case, this time with two rows of the DLR Rig250 test case [12]. These second production meshes have an average size of ~10M (slightly smaller than the first test case), but have interfaces approximately 20% larger in size than the first case. The mini-apps use instances of the same NASA Rotor37 8M test case, but the interface sizes have been increased by 30%. In this test, two Hydra sessions are connected together using 1, 2 and 4 CUs, and the number of MPI ranks is increased incrementally from 100 to 800. The Hydra + RR Coupler test case has 5 time steps and 10 iterations per time step, so the CPX + MG-CFD case is run with 250 MG cycles and 5 search routines. The hardware and software configurations remain the same as the first ~10M cell test case.

Figure 8 shows the performance scaling of both CPX and RR Coupler with this test case. The CPX mini-app produces a mean error of 16% when predicting production code run-time, with the maximum error being 41%. This maximum error is due to multiple MG-CFD sessions becoming out of sync with one another and having to wait for the slowest session whenever data is sent to the coupler. To match solver run-time, we run 5 iterations of the MG-CFD solver for every Hydra iteration, as a result of MG-CFD's simplified solver. Due to the similarities in computation and memory access, the two solvers should have similar variance per iteration, but as a result of the increased number of iterations of MG-CFD, it is likely overall variance is higher, resulting synchronization issues and increased run-time at high numbers of MPI ranks per MG-CFD session. This must be considered when using CPX + MG-CFD

TABLE III: Performance model

$RT_{sf} = (\frac{S_{newmesh}}{8}) * (\frac{B_{newfrac}}{0.03})$	E1
$MG_{newcycl} = N_{TS} * I_{TS} * 5$	E2
$BT_{newmain} = BT_{main} * RT_{sf} * (\frac{MG_{cyclnew}}{MG_{cycl}}) * SF_{mesh} * \log_{S_{mesh}}(RT_{sf} * S_{mesh})$	E3
$BT_{newsr} = \frac{N_{TS}}{FRQ_{sr}} * BT_{sr} * {RT_{sf}}^{1.8}$	E4
$BT_{newint} = BT_{int} * RT_{sf} * (\frac{MG_{cyclnew}}{MG_{cycl}})$	E5
$SF_{main} = 2 - 0.01 * ((\frac{R_{main}}{128}) * 0.4)/(\frac{\log_e(RT_{sf} * 8 + e)}{10}))$	E6
$SF_{cpl} = 2 - 0.01 * ((\frac{R_{main}}{8}) * 0.4)/(\frac{\log_e(RT_{sf} * 8 + e)}{10}))$	E7
$T_{newx} = T_x * (1 + (SF_x - 1) * \frac{(\frac{R_x - 1}{R_x})}{SF_x})$	E8

TABLE IV: Performance model parameters

Hyd+Cpl Parameters	Mini-Apps Parameters	Symbol
Size of mesh	Size of mesh	S_{mesh}
No. of time steps	$\frac{\text{MG cycles}}{\text{Its. per time step}}$	N_{TS}
Iterations per time step	$\frac{\text{MG cycles}}{\text{No. of time steps}}$	I_{TS}
No. of time steps (alt)	MG-CFD search frequency	FRQ_{sr}
Iterations per time step × time steps	MG cycles	MG_{cycl}
Boundary mesh fraction	Boundary mesh fraction	B_{frac}
No. of Hydra Sessions	No. of MG-CFD Sessions	$Main_s$
Hydra scaling factor	MG-CFD scaling factor	SF_{main}
RR Coupler scaling factor	CPX scaling factor	SF_{cpl}
Mesh scaling factor	Mesh scaling factor	SF_{mesh}
Base Hydra time	Base MG-CFD time	BT_{main}
Base RR Coupler interpolation time	Base CPX interpolation time	BT_{int}
Base RR Coupler search time	Base CPX search time	BT_{sr}
Ranks per HS	Ranks per MG Session	R_{main}
Ranks per CU	Ranks per CU	R_{cpl}
Total/target ranks	Total/target ranks	R_{total}

as a prediction of RR Coupler + Hydra run-time. However, when considering that the underlying MG-CFD mini-app has an mean prediction error of 16% [15], the average error from the mini-coupler is small, even with the high core count results, and can be used as a representation of the RR Coupler framework. Furthermore, the overall scalability trend is very similar to that of the RR Coupler, so its qualitative accuracy is high. Thus, if the performance model produces an 'optimum' configuration for the RR Coupler, then that configuration should also be optimal for CPX and vice versa.

V. Building and applying the performance model

To develop a performance model, the parameters of both the full-scale code and the mini-app must be considered. These include domain size, run-time parameters, scaling parameters, and base case performance numbers. Table IV highlights the RR Coupler parameters, the equivalent CPX parameters, and their symbols.

S_{mesh} is a parameter representing the size of the mesh in millions of cells. N_{TS} and I_{TS} are the number of time steps and iterations per time steps in the Hydra run, and MG_{cycl} is the MG-CFD equivalent. Because this model uses a hybrid of mini-simulation and full simulation data, conversions between the shared values are shown in table. For example, the number of Hydra time steps, N_{TS}, is equal to the frequency of the CPX search routine, FRQ_{sr}, and there is conversion between MG-CFD cycles and Hydra total iterations ($MG_{cycl} = N_{TS} * I_{TS}$).

The objective of the model is to calculate the values of R_{main} and R_{cpl} such that the overall run-time, $T_{overall}$, is as small as possible. $T_{overall}$ is the sum of $T_{main} + T_{search} + T_{interpolate}$, which are the final run-times of the main solver (either MG-CFD or Hydra), the run-time of the coupler search routine, and the run-time of coupler interpolation and communication.

B_{frac} is the average interface size as a proportion of the average total mesh size. For example, if meshes have 10M cells and the interfaces are 30,000 cells in size, then $B_{frac} = 0.003$. $Main_s$ is the number of Hydra or MG-CFD sessions, and SF_{main}, SF_{cpl} and SF_{mesh} are either functions or constants which determine how much speedup is gained when doubling the number of MPI ranks or CUs. BT_{main}, BT_{int} and BT_{sr} are the base run-times for the main Hydra/MG-CFD application, and for CPX search and interpolate operations, and are used as a starting point in which the values are manipulated by the other variables. Finally, R_{main}, R_{cpl}, and R_{total} are the number of ranks or CUs assigned for the main application, for CPX/RR Coupler and in total.

Initial values are set based on an MG-CFD/CPX test case of $S_{mesh} = 8M$ and $B_{frac} = 0.004$, with $MG_{cycl} = 500$, equivalent to $N_{TS} = 10$, $I_{TS} = 10$. There are 10 search routines, thus $FRQ_{search} = 10$. These values are collected on a run of $R_{main} = 100$ and $R_{couple} = 1$, with $BT_{main} = 44s$, $BT_{search} = 68s$ and $BT_{interpolate} = 117s$.

First, the initial run-times must be scaled based on the mesh size, which gives an adjusted run-time factor. This is shown in Equation E1. Next, we modify the base MG-CFD time and adjust it based on this factor, as well as I_{TS} and N_{TS} values, seen in Equation E2. It also needs to be scaled by SF_{mesh}, which is how much the MG-CFD run-time increases above linearly if the mesh size is doubled. From weak scaling testing we know the value is approximately $SF_{mesh} = 1$. This is shown in Equation E3.

The same process is repeated for the coupler interpolation and search times, as these also need to be scaled based on the mesh size. The interpolate time scales with RT_{sf}, the size of the boundary mesh, and the increase or decrease of MG cycles compared to the base case. The search time scales with the

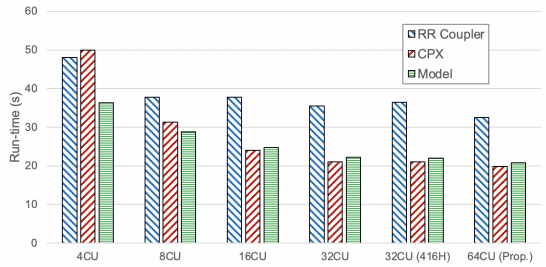

Fig. 9: Rolls-Royce Coupler vs. CPX vs. Model ~10M Cell Test 2, testing different CU numbers with 400 Hydra/MG-CFD MPI ranks.

square of the size of the boundary mesh, along with the search frequency. These updated times can be seen in Equations E4 and E5. Finally, the Hydra/MG-CFD scaling factor and RR Coupler/CPX scaling factor needs to be defined. Modelling their scaling requires a function, with size of the mesh and number of ranks as parameters, as having a constant scaling factor would not be enough detail to accurately model the setup. These can be seen in Equations E6 and E7, which models the scaling with different mesh sizes on dual socket AMD EPYC 7742 2.25Ghz clusters, up to 64 nodes.

The model takes in the adjusted base run-times for the main application (MG-CFD or Hydra), the coupling search time and the coupling interpolation time. These are $BT_{newmain}$, $BT_{newsearch}$ and $BT_{newinterpolate}$. These run-times are for the setup described earlier, with $R_{main} = 100$ and $R_{couple} = 1$. The ranks per HS and number of CUs is then passed into Equation E8, which computes the time that both the main application and the coupling will take to complete if the MPI ranks are increased by 1. Whichever time provides a greater run-time decrease is assigned the extra MPI rank, and the process is repeated until the sum of R_{main} and R_{search} are equal to the R_{total} parameter, which is user defined.

A. Applying the performance model

To test the performance model, the input parameters were set to $S_{mesh} = 10$, $N_{TS} = 5$, $I_{TS} = 10$, $B_{frac} = 0.0033$, $Main_s = 2$ and $R_{total} \approx 850$. This is the setup used for the '10M Cell Test 1' test case. These parameters are designed to see how many CUs the model suggests is suitable when 400 ranks are used for each HS. Using this setup, the model suggests that for 400 ranks for each Hydra session, 64 is the optimum number of CUs, for a total of 864 total ranks.

Figure 9 shows the proposed 'optimal' run-time compared to the same setup but with fewer CUs, testing the setup with both the mini-coupler and RR Coupler. For both, the reduction in run-time as a result of increasing CUs decreases the more CUs are added. For this reason, reallocating Hydra ranks beyond 64 CUs would likely not result in a decreased run-time. When looking at the proposed 'optimal' configuration, taking coupler resources and allocating them to Hydra/MG-CFD sessions also does not result in a decreased run-time. This suggests that the proposed solution is optimal for $R_{total} \approx 850$, and shows that the performance model is working as intended. As the coupling run-time is minimal here, the performance discrepancy between CPX and the RR coupler run-time is due to the MG-CFD mini-app running faster than Hydra.

Using the performance model we can project the optimum configurations for production problems of interest, with CPUs and also GPUs, by converting the GPU run-time into a BT_{main} equivalent. One such problem is the 10-passage, full annulus Rig250 problem, tested on both CPUs and GPUs. The total problem size of this simulation is 6 Billion cells, where the average mesh size per Hydra instance is approximately 600M cells. It is worth noting that the model does not take into account any additional overhead for CPU-to-GPU PCIe communication, which may reduce coupling efficiency when running coupled Hydra on GPUs. As a result, while optimal configurations should be valid, the GPU run-times should be taken with a degree of caution.

Prior testing of standalone Hydra has shown that a node with 4 Nvidia V100 16GB GPUs will produce similar run-time to 550 AMD EPYC 7742 CPU cores. However, given the Thermal Design Power (TDP) of an Nvidia V100 GPU (300W) and an AMD EPYC 7742 CPU (225W), when factoring in the power usage of further equipment that's required for node functionality (such as networking equipment), we estimate that a node containing 4 Nvidia V100 GPUs will consume around 2.5x more power than a node containing 2 AMD EPYC 7742 CPUs. The large instance test case requires approximately 400 GPUs to fit the mesh into GPU memory, which is equivalent (in terms of performance) to 220,000 CPU cores. As a result, the model parameters for the GPUs are set at $S_{mesh} = 600$, $N_{TS} = 10$, $I_{TS} = 10$, $B_{frac} = 0.004$, $Main_s = 10$ and $R_{total} = 220000$, producing an optimal configuration of 40 GPUs per Hydra instance, and 1280 CUs per pair. Furthermore, since we estimate that the power consumption of 100 GPU nodes is similar to the power consumption of 250 CPU nodes, using the performance model, we can predict the performance on 250 CPU nodes using 32,000 MPI ranks.

It is worth noting that the CUs must run on the CPU, even when Hydra is ran on GPUs, which results in additional power consumption. When this is taken into account, 100 GPU nodes combined with the CUs is equal to a pure CPU setup of 42,000 MPI ranks. The parameters of the power-equivalent CPU run were therefore set at $S_{mesh} = 600$, $N_{TS} = 10$, $I_{TS} = 10$, $B_{frac} = 0.004$, $Main_s = 10$ and $R_{total} = 42000$. With this setup, the model suggests approximately 3180 ranks per HS and 1130 CUs per pair of HSs. The run-time predictions for execution on both 100 GPU nodes and 250 CPU nodes can be seen in Figure 10, with the results demonstrating the GPU nodes' estimated power efficiency benefits over the CPU node configuration for a variety of CU counts.

Similarly, the performance model can be used to predict the optimal configuration as we scale up the number of CPUs or GPUs used for execution. Figure 11 shows the run-time predictions for the optimal configurations as the number of computing resources are increased. The model predicts that there is a hard scaling limit (around 20,000 Hydra ranks per session and ~1300 CUs), at which point allocating any more resources does not improve run-time. Internal testing has shown that Hydra's parallel efficiency is only 67% at just 8,200 ranks per session in this test case, so while the predictive

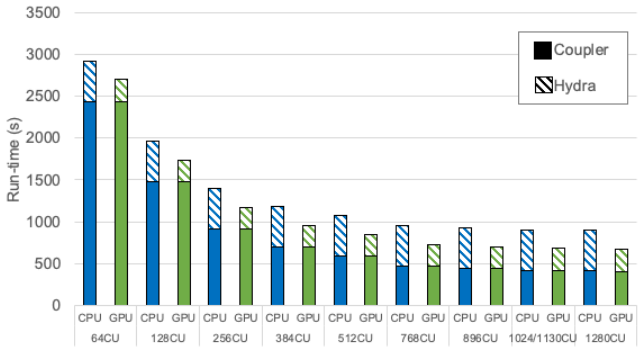

Fig. 10: The predicted run-times (s) for the 6B cell test case on CPU and GPU clusters. The number of Hydra ranks fixed at 3180 per HS for the CPU cluster and 40 GPUs per HS for the GPU cluster.

model may be too aggressive in assuming no speedup beyond 20,000 ranks, any additional speedup will likely be minimal if this is not the case. The coupler scaling is likely limited by the brute force nature of the search routine; the performance model states that even at 1280CUs, 89% of the run-time is made up of the search. The CFD bottleneck is likely memory or cache related, as it is a memory bound code [25].

It is important to note that the number of CUs cannot be increased indefinitely. As the CU number increases, the size of each partition decreases which can cause issues if there is no mapping between a cell in one interface and the opposite interface. Although the stated CU configurations are the optimal setups for these test cases, splitting the interface such that this issue does not occur is difficult. Whilst having larger partitions and a smaller number of CUs will show a slight performance decrease, as shown in Figure 10, it will be far easier to set up than the optimal configuration. Any additional time spent during run-time will be offset by the time saved during setup, as the partitions will likely be stable on the first attempt at partitioning. Therefore it may be sensible to allocate fewer CUs than optimal to ensure stability.

VI. Conclusions and Future Work

In this paper, we presented CPX, a mini-coupler designed to match the performance behaviour of Rolls-Royce's production coupler framework. Combined with the MG-CFD mini-app, CPX was used to create proxy configurations that are representative of production simulations carried out with RR Coupler and Hydra. We show that the mini-coupler can be used alongside MG-CFD to predict the run-time and scaling behaviour of Rolls-Royce's large scale coupled CFD simulations with high qualitative and quantitative prediction accuracy. Furthermore, we detailed the creation and testing of a performance model that is designed to predict the optimal resource configuration of a given coupled simulation for both CPX and the RR Coupler. We demonstrated the use of the model to theorise the optimal configuration for a large test case to be simulated on both a 100,000 core cluster and a 400 GPU cluster, comparing power equivalent CPU and GPU setups. Although the optimal configuration can be speculated, due to the need to partition the interfaces manually in the RR Coupler, increasing the number of partitions or CUs indefinitely can lead to stability issues.

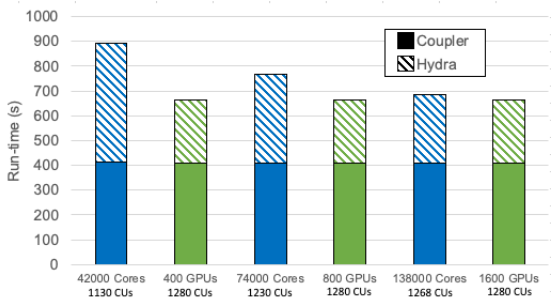

Fig. 11: The predicted run-times(s) for the 6B cell test on CPU and GPU power equivalent clusters.

Thus, slightly lowering the number of CUs from the proposed optimal configuration can lead to benefits with regards to setup time at the cost of a slight decrease in performance.

There are several areas that can be considered further in future work. One obvious area is to improve the search algorithm, as the brute force approach currently used is inefficient. Work has already begun on implementing a tree-based method which will significantly reduce this search time. A key issue with the RR coupler is the time-consuming process of manually partitioning the interfaces. Automating the partitioning process can improve the utility of the performance model, as well as significantly reducing the setup time. Additionally, since the RR Coupler supports the use of conjugate heat transfer as the interface between CFD models, CHT test cases can be explored in CPX by removing the need to execute the interface search routines that must occur when coupling sliding planes. Furthermore, we plan to extend the functionality of CPX so that it supports the creation of proxy configurations that are representative of more complex coupling scenarios. This includes the coupling of MG-CFD with a combustion mini-app to model the interaction between the engine compressor and the combustion chamber, as well as functionality that allows for coupling between CFD and FEM solvers. We also plan to extend the performance model to determine optimum run-time configurations for these new solvers, and carry out testing CPX with the GPU versions of MG-CFD to validate the GPU run-times predicted by the model. These extensions will allow for proxy configurations that are more representative (in terms of performance) of a complete gas turbine engine simulation to be created using CPX. Finally, the prediction accuracy of the mini-coupler motivates its use in predicting the cost and performance of simulations executed with the RR Coupler on various open platforms and systems, such as cloud instances, which is not possible with the full code due to licensing restrictions. The CPX coupler is available as open-source software at [27].

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] 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.

[2] 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.

[3] 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.

[4] 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.

[5] M. L. Shur, P. R. Spalart, M. K. Strelets, and A. K. Travin, "Synthetic turbulence generators for rans-les interfaces in zonal simulations of aerodynamic and aeroacoustic problems," *Flow, turbulence and combustion*, vol. 93, no. 1, pp. 63–92, 2014.

[6] "Joliot-curie supercomputer used to build first full, high-fidelity aircraft engine simulation," Accessed May 2021. https://cerfacs.fr/en/actualite/first-360-degrees-large-eddy-simulation-of-a-full-engine/.

[7] 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.

[8] "Strategic partnership in computational science for advanced simulation and modelling of engineering systems - asimov," Accessed May 2021. https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/S005072/1.

[9] "World's most efficient large aero-engine - trent xwb," Accessed May 2021. https://www.rolls-royce.com/products-and-services/civil-aerospace/airlines/trent-xwb.aspx#section-technology.

[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] D. Amirante, V. Ganine, N. J. Hills, and P. Adami, "A coupling framework for multi-domain modelling and multi-physics simulations," *Entropy - Special Issue: Computational Fluid Dynamics and Conjugate Heat Transfer*. Under Review.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] "Archer2," Accessed May 2021. https://www.archer2.ac.uk.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] R. Bunt, S. Pennycook, S. Jarvis, L. Lapworth, and Y. Ho, "Modelled optimisation of a geometric multigrid application," in *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 742–753, IEEE, 2013.

[23] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*, pp. 44–54, Ieee, 2009.

[24] G. Mudalige, M. Giles, I. Reguly, C. Bertolli, and P. Kelly, "Op2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures," in *2012 Innovative Parallel Computing (InPar)*, pp. 1–12, IEEE, 2012.

[25] I. Z. Reguly, A. M. 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 *International Conference on High Performance Computing*, pp. 391–410, Springer, 2021.

[26] O. B. Messer, E. DAzevedo, J. Hill, W. Joubert, S. Laosooksathit, and A. Tharrington, "Developing miniapps on modern platforms using multiple programming models," in *2015 IEEE International Conference on Cluster Computing*, pp. 753–759, IEEE, 2015.

[27] "Cpx source code," Accessed July 2021. https://github.com/warwick-hpsc/MG-CFD-app-OP2/tree/feature/coupler.