Differentiable and Nonlinear Aircraft Loads at Large Scale Using an Accelerator-Based Concurrent Solution

Alvaro Cea*, Rafael Palacios†

Abstract

A novel framework for geometrically nonlinear aeroelastic analysis is augmented for large scale computations of aircraft-sizing loads. The software can be deployed on modern hardware architectures and the parallelisation has been tested both on CPUs and GPUs. While keeping computational times in the order of seconds, the presented approach unlocks new capabilities such as the following: uncertainty quantification analysis of the nonlinear response via massively-parallel Montecarlo simulations; construction of load envelopes of static and dynamic cases in full-aircraft models. Furthermore, differentiation of both Montecarlo simulations and load is achieved via JAX library automatic differentiation engine and collective primitives. We explore a high-aspect-ratio-wing aircraft configuration, showing the ability of the solvers to capture nonlinear effects for external forces inducing large deformations and also the importance of these effects when studying modern aircraft concepts. Discrete, manoeuvre and gust loads are assessed for multiple flow and parametric conditions running concurrently. Sensitivities of maximum wing stresses in the load-envelopes are computed and verified against finite differences.

1 Introduction

Aeroelastic analysis are expected to become critical in the very early phases of the wing design process: while the field was more important in post-design stages to ensure in-flight integrity, it now becomes paramount to capture the cross-couplings between disciplines. As highlighted in [1], formulations that include nonlinear effects should be developed that not only enhance current modelling techniques but that also allow rapid data turnaround for the industry. Real-time, hardware-in-the-loop flight simulators would also benefit of actively controlled, deformable airplane models. This leads to a more nonlinear landscape, where the overall aerodynamic performance needs to be calculated around a flight shape with large deformations [2]; the input for efficient control laws account for the steady state and nonlinear couplings [3]; and the loads ultimately sizing the wings are atmospheric disturbances computed in the time-domain [4]. A more holistic approach to the design also increases the complexity of the processes exponentially, and the trade-offs and cost-benefit

*Research Associate, CAGB 308, South Kensington Campus. (alvaro.cea-esteban15@imperial.ac.uk)

†Professor in Computational Aeroelasticity, CAGB 310, South Kensington Campus. AIAA Associate Fellow (r.palacios@imperial.ac.uk)

analysis may not be possible until robust computational tools are in-place to simulate the different assumptions. Certification of new air vehicles is another important aspect that requires 100,000s of load cases simulations [5], as it considers manoeuvres and gust loads at different velocities and altitudes, and for a range of mass cases and configurations. This poses another challenge for new methods that aim to include new physics since they normally incur in prohibitively expensive computational times. Lastly, the mathematical representation of the airframe, embodied in the complex Finite-Element Models (FEMs) built by organizations, encompasses a level of knowledge that is to be preserved when including the new physics mentioned above.

Those previous considerations set the goals for our work: 1) to be able to perform geometrically nonlinear aeroelastic analysis, 2) to work with existing generic FEMs in a non-intrusive manner, and 3) to achieve a computational efficiency that is equivalent to present linear methods (if not faster). 1) and 2) were considered in [6, 7]. In this work we explore the latest advances on accelerator's parallelisation and how to integrate them into our solution process to enable large scale aeroelastic simulations under geometrically nonlinear assumptions. Specifically, we set out to characterise the dynamics of highly flexible aircraft in response to the very large envelopes of in-flight loads encountered in the certification process. For that purpose, we have leveraged the numerical library JAX [**?**] to build a new simulation environment for time-domain nonlinear aeroelastic analysis that achieves two orders of magnitude speed-ups with respect to standard implementations [7], is suitable for modern hardware architectures such as GPUs [8], and that is also capable of computing derivatives via algorithmic differentiation [**?**]. The strength and suitability of JAX for scientific computation has been proved recently in fluid dynamics [9] and solid mechanics [10] applications. We want to go one step further by adding parallelisation and distributed computational capabilities to the codebase to tackle the very demanding task of calculating load envelopes while introducing new physics to account for the large displacements and rotations ultra-high-aspect-ratio wings undergo. In this multi-process environment, a Single Program Multiple Data (SPMD) paradigm is employed with the main computation spanning as many devices as available in the cluster and performing collective operations to communicate between devices. By addressing in one program a substantial part of scenarios during flight (manoeuvres and gust responses at different velocities and altitudes, and for a range of mass cases and configurations), we will be able to produce the critical loading characteristics of the aircraft at a fraction of time. Moreover, as future work we aim to differentiate the boundaries of these critical cases using the already demonstrated capabilities AD, thereby providing gradients for optimization studies as well as additional insights to the designer.

The paper is organised as follows: Sec. 2 gives and overview of the theoretical and computational developments that underpin this work with a focus on the new parallelisation capabilities. In sec. 3, a representative configuration of an ultra-high-aspect-ratio aircraft is studied under various loading scenarios that have been parallelised; namely structural static loads, manoeuvre cases for varying flow conditions and dynamic loads with multiple gusts running concurrently. This application of modern hardware architectures to aircraft nonlinear load analysis is novel and could potentially be introduced inside current industrial processes. We conclude in Sec. 4 with a summary of the main

advances and the future work that is needed to finalise a formulation that may run in parallel on modern hardware architectures as well as being differentiated.

2 Theoretical and computational background

The main aspects of the aeroelastic framework we have developed are presented in this section. The approach is built on a non-intrusive reduction order process combined with a nonlinear description of the dominant dimension for slender structures. It achieves a nonlinear representation of aeroelastic models of arbitrary complexity in a very efficient manner and without losing the characteristics of the linear model. We target the calculation of flight loads herein, but it can also be applied to the computation of aeroelastic stability phenomena such as flutter or divergence [6] and to broader multidisciplinary design optimisation problems, which are currently being explored. The key features of the formulation are:

- Geometrically nonlinear aeroelastic analysis using complex GFEMs: achieved via a three step process in which a condensed model is first produced, the dynamics of this reduced model are described by a system on nonlinear equations [11] written in material velocities and stresses, and a modal expansion of those variables is the final key step in seamlessly mapping the global FEM into the nonlinear description [12]. The overall process can be found in [13].
- Maximum performance: as a combination of a highly optimised and vectorised codebase, numerical library JAX with its JIT compiler and accelerator capabilities driving the calculations, and the newly added added parallelisation of load cases.
- Differentiation and sensitivity analysis: using JAX algorithmic differentiation toolbox, the entire process, from inputs to aeroelastic outputs can be differentiated [14].

2.1 Nonlinear aeroelastic system

Given a general GFEM, a reduced model is obtained from a static or dynamic condensation that captures well the stiffness and inertia properties in the condensed matrices, $\boldsymbol{K}_a$ and $\boldsymbol{M}_a$. The eigenvalue solution of the FEM yields the modal shapes, $\boldsymbol{\Phi}_0$, and frequencies $\boldsymbol{\omega}$. A projection of the state variables, velocities $\boldsymbol{x}_1 = \boldsymbol{\Phi}_1 \boldsymbol{q}_1$ and stresses $\boldsymbol{x}_2 = \boldsymbol{\Phi}_2 \boldsymbol{q}_2$, and a Galerkin projection of the equations of motion leads to the system of ODEs that is solved in time domain. Aerodynamic forces are obtained via Generalised Aerodynamic Forces (GAFs) using a panel-based DLM solver and Roger's rational function approximation[15] to bring the forces to the time domain, resulting in a modal force component given as:

$$\boldsymbol{\eta}_a = Q_\infty \left(\boldsymbol{\mathcal{A}}_0 \boldsymbol{q}_0 + b\boldsymbol{\mathcal{A}}_1 \boldsymbol{q}_1 + b^2 \boldsymbol{\mathcal{A}}_2 \dot{\boldsymbol{q}}_1 + \boldsymbol{\mathcal{A}}_{g0} \boldsymbol{v}_g + b\boldsymbol{\mathcal{A}}_{g1} \dot{\boldsymbol{v}}_g + b^2 \boldsymbol{\mathcal{A}}_{g2} \ddot{\boldsymbol{v}}_g + \sum_{p=1}^{N_p} \boldsymbol{\lambda}_p \right) \tag{1}$$

where the $\boldsymbol{\mathcal{A}}_i s$ are real matrices, $b = \frac{c}{2U_\infty}$ with c the reference chord, $Q_\infty = \frac{1}{2}\rho_\infty U_\infty^2$ the dynamic pressure, $\boldsymbol{\lambda}_p$ the aerodynamic states and N_p the number of lags. Note these forces naturally follow the structure since the formulation is written in the material frame of reference. The coupling of the structure and aerodynamic equations combined with the aerodynamic lags, gravity forces, $\boldsymbol{\eta}_g$, and gust disturbances, $\boldsymbol{v}_g$, gives the final ODE system:

$$\begin{aligned}
\dot{\boldsymbol{q}}_1 &= \hat{\boldsymbol{\Omega}}\boldsymbol{q}_2 - \hat{\boldsymbol{\Gamma}}_1 : (\boldsymbol{q}_1 \otimes \boldsymbol{q}_1) - \hat{\boldsymbol{\Gamma}}_2 : (\boldsymbol{q}_2 \otimes \boldsymbol{q}_2) + \hat{\boldsymbol{\eta}} \\
\dot{\boldsymbol{q}}_2 &= -\boldsymbol{\omega} \odot \boldsymbol{q}_1 + \boldsymbol{\Gamma}_2^\top : (\boldsymbol{q}_2 \otimes \boldsymbol{q}_1) \\
\dot{\boldsymbol{\lambda}}_p &= Q_\infty \boldsymbol{\mathcal{A}}_{p+2}\boldsymbol{q}_1 + Q_\infty \boldsymbol{\mathcal{A}}_{p+2}\dot{\boldsymbol{v}}_g - \frac{\gamma_p}{b}\boldsymbol{\lambda}_p
\end{aligned} \tag{2}$$

where $\odot$ is the Hadamard product (element-wise multiplication), $\otimes$ is the tensor product operation and $:$ is the double dot product. In this system the aerodynamic added-mass effect has been moved to the left hand side such that $\mathbf{A}_2 = (\boldsymbol{I} - \frac{\rho c^2}{8}\boldsymbol{\mathcal{A}}_2)^{-1}$, and it couples all DoF in $\boldsymbol{q}_1$. Thus the natural frequency terms become $\hat{\boldsymbol{\Omega}} = \mathbf{A}_2 \text{diag}(\boldsymbol{\omega})$ and the nonlinear terms $\hat{\boldsymbol{\Gamma}} = \mathbf{A}_2\boldsymbol{\Gamma}$. The effect of all external forces, aero, $\boldsymbol{\eta}_a$, gravity, $\boldsymbol{\eta}_g$, and others, $\boldsymbol{\eta}_f$, are combined in such that $\hat{\boldsymbol{\eta}} = \mathbf{A}_2\left(\left(\boldsymbol{\eta}_a - \frac{\rho c^2}{8}\boldsymbol{\mathcal{A}}_2\dot{\boldsymbol{q}}_1\right) + \boldsymbol{\eta}_g + \boldsymbol{\eta}_f\right)$. The aerodynamic matrices $\hat{\boldsymbol{\mathcal{A}}}_{p+2}$ have also been scaled accordingly. The nonlinearities in the system are encapsulated in the modal couplings of the third-order tensors $\boldsymbol{\Gamma}_1$ and $\boldsymbol{\Gamma}_2$ (the former introduces the gyroscopic terms in the dynamics and the latter introduces the strain-force nonlinear relation).

Once the nonlinear solution of the condensed model is computed, the corresponding full 3D state is calculated via two postprocessing steps: firstly the displacements of the cross-sectional nodes linked to the reduced model via the interpolation elements are computed using the positions and rotations of the latter; secondly, Radial Basis Functions (RBFs) kernels are placed on those cross-sections, thus building an intermediate model that is utilised to extrapolate the positions of the remaining nodes in the full model. This paves the way for a broader multidisciplinary analysis where CFD-based aerodynamic loading could be used for the calculation of the nonlinear static equilibrium, and also with the transfer of the full deformed state back to the original FE solver to study other phenomena such as local buckling.

2.2 High performance implementation

The formulation described above has been made into the codebase FENIAX (Finite Element models for Nonlinear Intrinsic Aeroelastics in JAX) [1]. It has been thoroughly tested with currently 12 different models that amount to over 200 tests that run in minutes and are part of Continuous-Integration/Development (CI/CD) workflow. Moreover, a flexible software architecture allows for automatic analysis of generic models from standard input files, which can integrated with other computational tools. The Python library JAX has been used as the numerical engine for calculations and it also manages the parallelisation, therefore some details on the library are worth describing. JAX is designed for high-performance numerical computing with focus on machine learning

[1] Both implementation and examples can be found at `https://github.com/ACea15/FENIAX`.

activities [16]. It relies on XLA (Accelerated Linear Algebra), a domain-specific compiler for linear algebra that optimizes computations for both CPUs and GPUs. In fact XLA is platform-agnostic and achieves optimised performance on the target architecture orchestrating a complex process that encompassing a series of optimizations and transformations: the source code is first converted into HLO (High-Level Optimizer) code, an specialised language derived from a graph representation of the computations; XLA performs optimisations on the HLO code (geared towards high-level mathematical operations, particularly those in linear algebra and machine learning models), and are independent of the hardware architecture, such as operation fusion. It then carries optimisations for the particular architecture in use. From there the LLVM toolkit is leveraged to produce and Intermediate Representation (IR) that the LLVM compiler can understand, perform further optimisations and finally output the machine code. When it comes to leveraging the computational power of (NVIDIA) GPUs, the link between XLA and CUDA kernels is critical. On the one hand JAX utilises CUDA libraries such as cuBLAS for dense linear algebra; on the other hand, it is capable of generating custom CUDA kernels for operations that are not efficiently covered by standard libraries. In order to transform the high level Python to low level optimised code, the source code has to comply with various constraints and feature functional programming characteristics. With regards to the parallelisation, JAX follows a Single-Program Multi-Data (SPMD) parallelism, whereby a single program operates on multiple data sets in parallel. This means the same computation graph is compiled and executed across different devices. Inter-device communication and synchronization are managed internally by the library. For the implementation, the now deprecated `pmap` function maps a function across multiple input sets, distributing the workload across available GPUs. Thus being the parallel equivalent to the `vmap` function. The new standard for parallelisation is based on data sharding, either done automatically using the `shard_map` function or by sharding the data and passing it to a `jitted` function specifying input and output shape of the data to be partitioned. Inside the function, the compiler determines the necessary partitions of the data, synchronization, and communication. Collective operations like broadcasts and reductions are available within the `jax.lax` module. Internally JAX uses NVIDIA Collective Communications Library (NCCL) for low level communication across devices. The overall solution process and a description of the parallelisation strategy follow next.

2.2.1 Overall solution process

Algorithm 1 shows the main components in the solution process, highlighting the time and space complexities, $O(time, space)$, of the data structures being generated. We assume a single analysis is being run, for instance a dynamic simulation computing the response to multiple gusts that will be run in parallel for a total number of N_c cases. N_t time-steps are used in the integration scheme with a resolution of N_m modal shapes. The FE model has been condensed to N_N number of nodes. The intrinsic modes, $\boldsymbol{\phi}$, $\boldsymbol{\psi}$, are computed from the condensed FE nodal positions and matrices; subsequently, the nonlinear terms, $\boldsymbol{\Gamma}$, are obtained as the integral along the reduced domain of the modal couplings; the nonlinear system of equations is built and time-marched in time to yield

the solution in modal coordinates, $\boldsymbol{q}$; the intrinsic variables of the solution (velocities, $\boldsymbol{X}_1$, internal forces, $\boldsymbol{X}_2$ and strains, $\boldsymbol{X}_3$) are recovered from the modal coordinates and the intrinsic modes; finally the positional and rotational field, $\boldsymbol{r}_a$, $\boldsymbol{R}_a$, of the reduced model are computed via integration of the strain field.

Algorithm 1: Main components in solution process

input : Input file: settings.yaml; FE model: $\boldsymbol{K}_a$, $\boldsymbol{M}_a$, $\boldsymbol{X}_a$; Aerodynamic matrices: $\boldsymbol{\mathcal{A}}$
output: Nonlinear aeroealastic solutioxn
begin

$\boldsymbol{\phi}$, $\boldsymbol{\psi} \longleftarrow$ modes($\boldsymbol{K}_a$, $\boldsymbol{M}_a$, $\boldsymbol{X}_a$) ▷ Intrinsic modes: $\mathrm{O}(N_n^2 \times N_m;\, N_n \times N_m)$
$\boldsymbol{\Gamma} \longleftarrow$ couplings($\boldsymbol{\phi}$, $\boldsymbol{\psi}$) ▷ Nonlinear couplings $\mathrm{O}(N_n \times N_m^3;\, N_m^3)$
$\boldsymbol{q} \longleftarrow$ system($\boldsymbol{\Gamma}$, $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\phi}$, $\boldsymbol{X}_a$) ▷ Modal coordinates: $\mathrm{O}(\frac{N_c}{N_d} \times N_t \times N_m^3;\, N_c \times N_t \times N_m)$
$\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3 \longleftarrow$ ivars($\boldsymbol{q}$, $\boldsymbol{\phi}$, $\boldsymbol{\psi}$) ▷ velocity/strain fields: $\mathrm{O}(\frac{N_c}{N_d} \times N_t \times N_n \times N_m;\, N_c \times N_t \times N_n)$
$\boldsymbol{r}_a$, $\boldsymbol{R}_a \longleftarrow$ integration($\boldsymbol{X}_3$, $\boldsymbol{X}_a$) ▷ Positional/rotational fields: $\mathrm{O}(\frac{N_c}{N_d} \times N_t \times N_n \times N_m;\, N_c \times N_t \times N_n)$

2.2.2 Two-level parallelisation

Various parallelism models have been developed in the context of deep learning, for which JAX has been particularly designed, and we try to adapt here those methods to the problem at hand of solving a large system of nonlinear equations in parallel for multiple external forces, i.e. right hand side of the equations. Data Parallel (DP) consists of making the large batching into chunks that are fed to a single device, and allows scaling to large data batches. In Large Language Models (LLMs), the number of parameters can exceed that of input data, and therefore don't fit in a single device. In this case a tensor parallelism (TP) strategy is employed by which the tensor of weights that are to be optimised is sharded with synchronisation at the end of each step. Hybrid strategies are employed in production. In engineering applications, the number of designs variables would usually be between the tens to the few thousands, so tensor parallelism becomes less relevant. However, the number of simulations for different inputs, and the size of each one of them, can be very large. Therefore we opt for a DP strategy in which our batch of data becomes the multiple inputs that are used to build the external forces for which we want to compute the response. The strategy implemented first splits the input data along the leading axis according to the total number of devices available using a data sharding approach. Each device receives the subset of inputs, a closure function that is jitted is called with the respective inputs, and inside the closure the high level function that computes the response (solution of the static response or time marching of the dynamic equations) is vmapped with respect to the subset of inputs. This last vmap makes the inputs that go into each device, or CPU cores, to run in parallel. Note the parallelisation happens at the system of equations level, meaning previous steps such as computation of intrinsic modes or nonlinear couplings is only carried out once before the concurrent simulations. Algorithm 2 illustrates this process with psudo

code. The process by which inputs are split and sent to each device is presented in Fig. 1, which shows the two-level parallelisation.

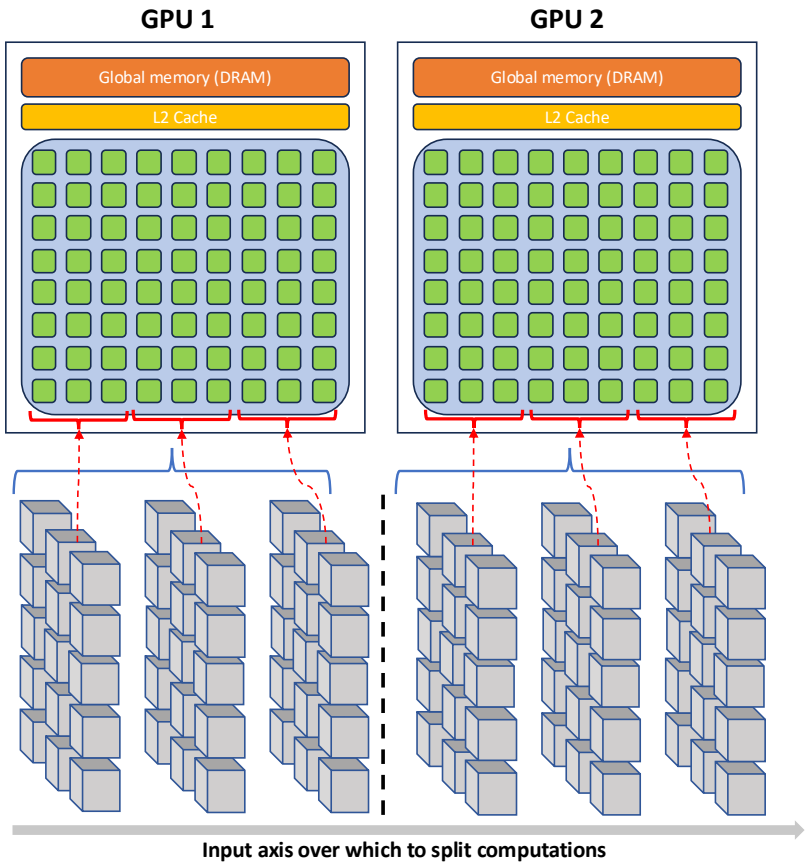

Figure 1: Input distribution example for multi-GPU runs

The inputs are tensors of arbitrary shape from which input data to the solution is built, with the only condition that the first axis being the one over which to run the parallelisation. For the monoeuvre and gust cases below, for instance, the tensor of inputs is a matrix with the second axis being a vector with the combination of flow conditions and gust parameters. In the figure we can see each GPU has a global memory and L2 cache, and in addition cores in the GPU are packed into the so-called streaming processors, each with its own registers and L1 caches. The strength of these chips is in the large number of cores, in the thousands, that can run in parallel, thus after the inputs are initially divided, many computations can run in parallel even within each GPU.

2.2.3 Differentiable-parallel load cases

Once a parallel system was in place to compute hundreds of load cases, the next step was to obtain the derivatives of the critical loads coming from the parallel analysis. Since those are calculated using AD, all the operations need to be available in memory. We encountered two major issues: the memory required for the gust cases was already in the limit of a single device (over 60 GB of RAM), to which the AD normally duplicates the requirement. As the software can now be run

Algorithm 2: Parallelisation multiple load cases

```
begin
    Function y_aeroelastic(inputs):
        Function y(input):
            ...
            (nonlinear aeroelastic computation)
            return q, X1, X2, X3, ra, Rab
        yvmap = jax.vmap(y)
        q_multi, X1_multi, X2_multi, X3_multi, ra_multi, Rab_multi ⟵
        yvmap(inputs)
        return dict(q=q_multi, X1=X1_multi, X2=X2_multi, X3=X3_multi,
        ra=ra_multi, Rab=Rab_multi)
    num_devices ⟵ jax.device_count()
    mesh ⟵ jax.sharding.Mesh(
    devices=jax.experimental.mesh_utils.create_device_mesh(
    (num_devices,)), axis_names=('x'))
    inputs = jax.device_put(inputs, jax.sharding.NamedSharding(mesh,
    jax.sharding.PartitionSpec('x')))
    y_aeroelastic ⟵ jax.jit(y_aeroelastic)
    sol ⟵ y_aeroelastic(inputs)
```

on multiple devices, each with its own memory, this is not a completely restrictive factor. The second issue was simply a lack of implementation of the needed collective operations in JAX, as with the maximum function (most of the data generated by such a maximum are zeros not needed anyway). The solution found has been named the Forager Pattern and is depicted in Fig. 2. The code launches many simulations concurrently with the predefined load-cases. The solutions of all these simulations are collected (hundreds of cases, hundreds of nodes, thousands of time steps make for a single field of interest like the stress to have a size of the order of 10^7). A filtering step consists of a selection of monitoring points of interest (nodes in the FEM), and then a double reduction operation in both time and load cases, for example the maximum of the selected field in time and across cases, and the output is a selection of the most problematic load cases according to the predefined metric in the input file. For these critical points the program builds the inputs for the cases previously run in parallel but now with AD and on a much smaller basis, and finally more FENIAX process are spawn for the AD computations. In this way we have created a meta-program that can automatically create programs based on the results, although at this stage is still very limited on the implemented possibilities.

3 Results

In this section we show the main strengths of our solvers to: a) run a representative aircraft model undergoing very large nonlinear displacements; b) leverage on modern hardware architectures and

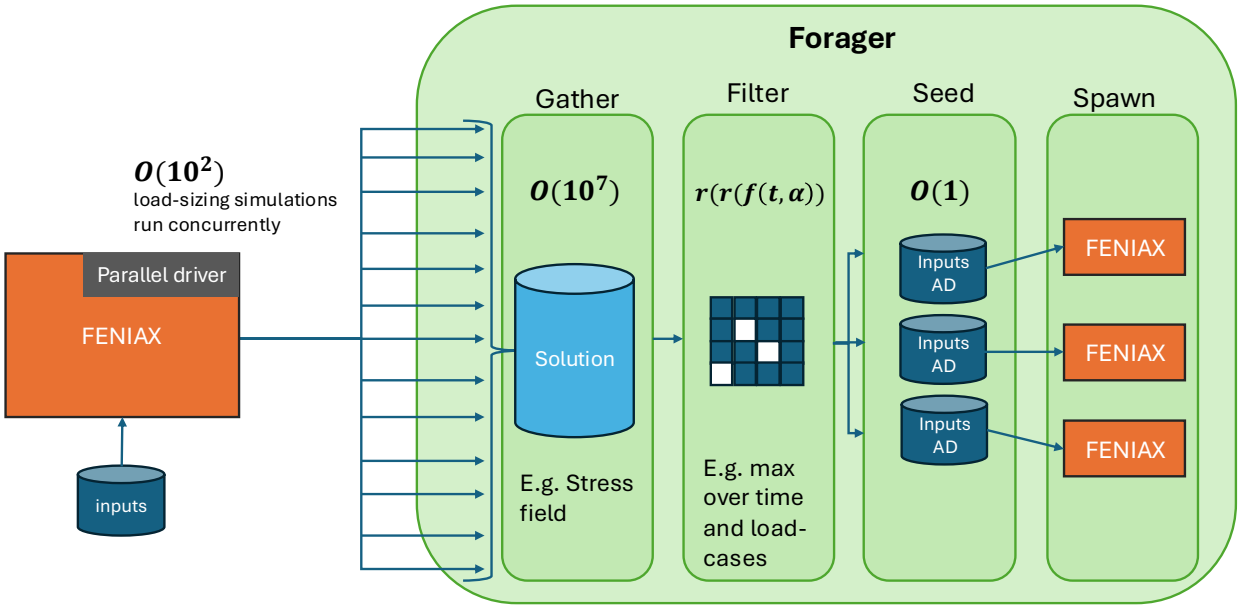

Figure 2: Forager pattern for differentiable-parallel simulations

a parallelisation across devices to unlock problems such as quantifying the uncertainties in the nonlinear response given a loading field that is not fully determinate; c) build load envelopes of static and dynamic aeroelastic simulations. The University of Bristol Ultra-Green (BUG) aircraft model [17] is the chosen platform to demonstrate these capabilities as it showcases high-aspect ratio wings that are built using a representative GFEM of current industrial models and it is not based on proprietary data. The main components of the aeroelastic model have been presented in

Structural and aeroelastic static simulations follow, all solved via a Newton-Raphson solver with tolerance of 10^{-6}, as well as an assessment of the aircraft dynamics in response to a gust. Calculations are carried out on a CPU Intel Xeon Silver 4108 with 1.80GHz speed, 6 cores and a total 12 threads, as well as on an Nvidia GPU A100 80GB SXM.

3.1 Uncertainty quantification of nonlinear response

uncertainty quantification is performed to the nonlinear response to a loading field that is non-deterministic. Hundreds to thousands of simulations are employed in Monte Carlo type of analysis to resolve for the statistics, for which parallelisation of the independent simulations become critical. The example resembles the workflow of flight loads and wing stress analysis in an industrial setup.

There will always be an element of uncertainty around the computed loads, and what we show here is how for nonlinear assumptions the statistics need to be computed for every distinct loading. And for this, having a parallisation strategy as the one presented could potentially allow the computation of complex correlations and averages that are more easily calculated under linear assumptions.

A constant loading force is prescribed along the wings consisting of follower forces in the z-

direction as well as torsional moments, with the characteristic that the force follows a normal distribution $N(\mu = 1.5 \times 10^4 \mu_0, \sigma = 0.15\mu)$ for the vertical forces and $N(\mu = 3 \times 10^4 \mu_0, \sigma = 0.15\mu)$ for the moments. Three scenarios are studied: one in which very large nonlinear deformations are induced with $\mu_0 = 1$, and two small loading with $\mu_0 = 10^{-2}$ and $\mu_0 = 10^{-3}$. The distribution of displacements is characterised by means of Montecarlo simulations that run in parallel for a total of 1600 simulations. The modal resolutions consists of 100 modes. Fig. 3 shows the equilibrium for the high loading calculations for two cases out of the 1600.

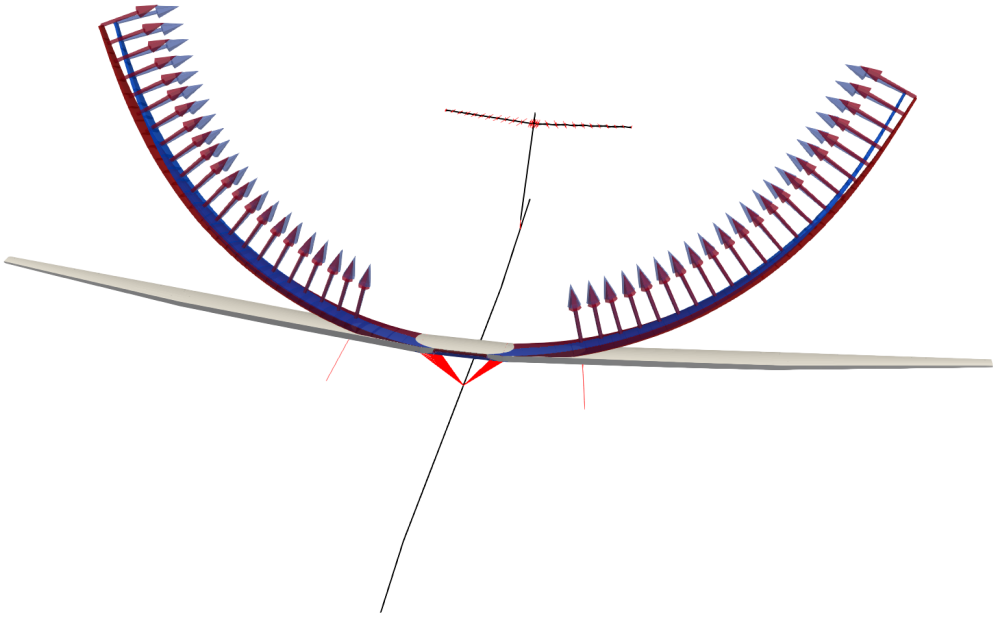

Figure 3: Static equilibrium for two cases of the random excitation ($\mu_0 = 1$)

Table 1 shows the statistics gathered from the response

Table 1: Tip displacement statistics

Case	Tip displacement mean (m)	Tip displacement std
Nonlinear ($\mu_0 = 1$)	11.57	1.35
Linear ($\mu_0 = 0.01$)	0.148	0.024
Very Linear ($\mu_0 = 0.001$)	0.0149	0.0023

We can see the statistics of the linear response are fully captured by one example, whereas for a nonlinear response such as $\mu_0 = 2$, the 1600 simulations would need to be computed again. Table 2 shows the times taken for the nonlinear case. The computation of 1600 independent simulations of Fig. 3, which presents deformations of over 40% the wing semi-span, in just over a minute, highlights the potential of this methodology in more complex uncertainty quantification problems.

Table 2: Computational times uncertainty quantification

Device	Time (sec.)
CPU (single)	$16.8 \times 1600 = 26880$
CPU (parallel)	317.4
GPU	67.6

3.1.1 Differentiation of statistical response

3.2 Steady manoeuvre loads

We extend the analysis to a static aeroelastic case for varying angles of attack that represent a manoeuvre scenario. The number of modes used was 100, more than necessary for this type of response, which indicates even faster calculations are possible on this type of analysis. We test the parallelisation by varying the flow density ($\pm 20\%$ of the reference density 0.41 Kg/ m^3) as well and the flow velocity ($\pm 20\%$ of the reference velocity 209.6 m/s). 16 different points for both density and velocity make a total number of 256 simulations. The Mach number is fixed at 0.7 corresponding to the reference flow condition values.

Fig. 4 illustrates the 3D equilibrium of the airframe at the reference flight conditions.

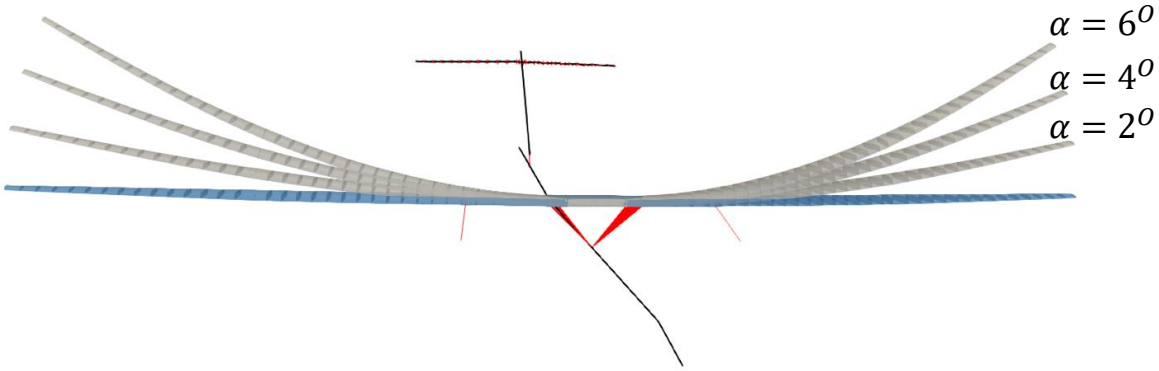

Figure 4: Aeroelastic steady equilibrium for increasing angle of attack

In Fig. 5 the tip of the wing in Fig. 4 is plotted for various angles-of-attach (AoA); the tip position falls down the linear projection between the 0 and 1 degrees AoA as expected. This highlights the potential need for geometrically nonlinear aeroelastic tools in future aircraft configurations under high loading scenarios.

Figures 6 and 7 show the bending-shear-torsion diagrams at the wing-root for all the multiple points computed. Each point is coloured as $point = \rho_\infty/\rho_{max} + u_\infty/u_{max}$. The resulting graph is rather predictable with loads increasing with dynamic pressure. A more realistic case could be setup with a trim routine and varying flight condition but the aim of this section was to present the capability to compute multiple manoeuvre loads and compare it with the more interesting case of gust loads as shown in the next section.

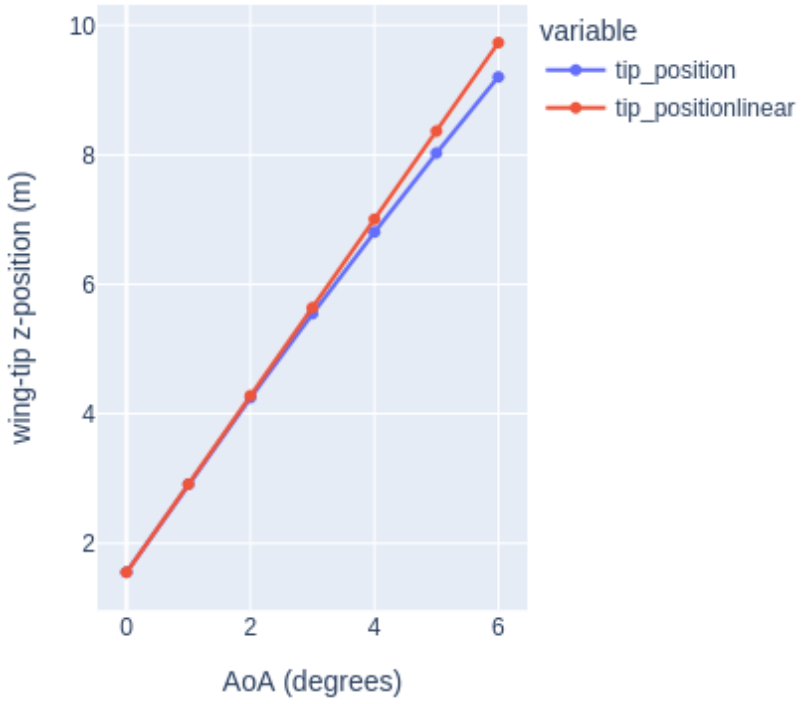

Figure 5: wing tip position for increasing angle of attack

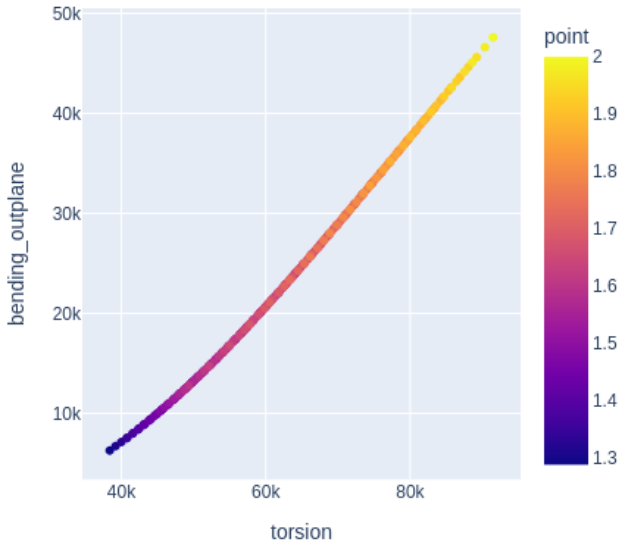

Figure 6: Manoeuvre case, bending-out-of-plane versus torsion

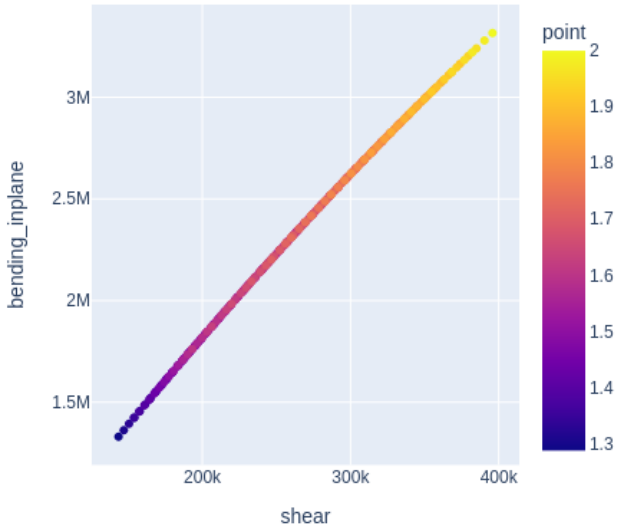

Figure 7: Manoeuvre case, bending-in-plane versus shear

Table 3 shows the computational times to run these simulations, which shows near no overhead in adding a few hundred of static calculations when moving from the single load case in the CPU to the GPU (nearly 8 seconds to 14 seconds, which amounts for 6 seconds cost when adding an extra 255 cases).

Table 3: Computational times for the multiple manoeuvre problem

Device	Time (sec.)
CPU (single)	$7.71 \times 256 = 1973.8$
CPU (parallel)	52.8
GPU	14.4

3.3 Dynamic loads at large scale

In this final example we perform a dynamic aeroelastic analysis to study the response of the aircraft to multiple 1-cos gusts for varying length, intensity and the density of the airflow. The mach number is kept constant at 0.7. A Runge-Kutta solver is employed to march in time the equations with a time step of 10^{-3} and the total number of modes used was 100. Note the large size of the aeroelastic ODE system: 2×100 nonlinear equations plus 5×100 linear equations for the aerodynamic states with 5 poles, plus 4 equations for the quaternion tracking the rigid-body motion, for a combined ODE system of 704 equations. In addition, a total of 512 gusts cases are run concurrently for all possible combinations of 8 gust lengths between 25 and 265 meters, 8 gust intensities between 1 and 30 m/s, and 8 airflow densities between 0.34 and 0.48 Kg/m^3. This means that $512 \times 704 = 360448$ equations are being marched in time, in this case for 2 seconds which is enough to capture peak loads. Figs. 8, 9 and 10 show the load diagrams for the wing root at the maximum gust intensity of 20, varying 16 gust lengths, L, in the range previously stated and 8 airflow densities, with the

points plotted as $point = L/L_{max} + \rho_\infty/\rho_{max}$. Different load pattern emerge which need further analysis but reflect the importance of running multiple of these simulations to assess the critical loads.

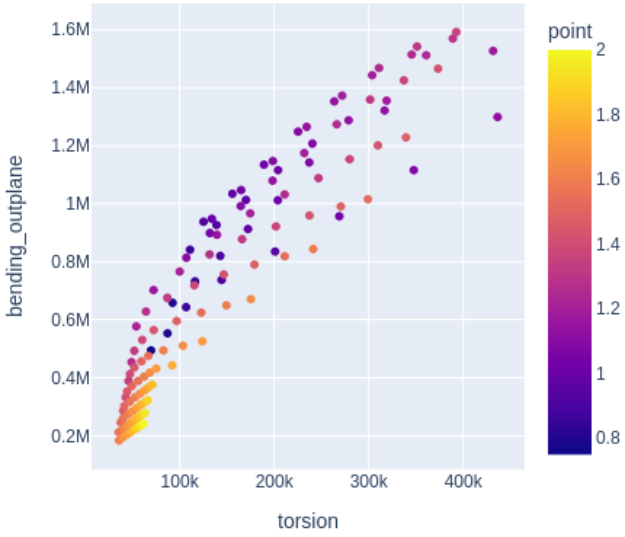

Figure 8: Gust case, bending-out-of-plane versus torsion

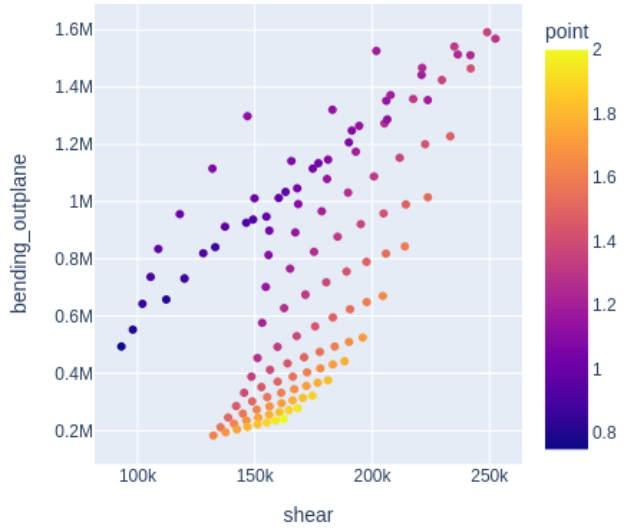

Figure 9: Gust case, bending-out-of-plane versus shear

As a validation of the parallelisation, Fig. 11 shows the wing tip time evolution for a gust of 150 m length, intensity of 20 m/s and flow density of 0.41 Kg/m^3. Both the results of a single simulation run and that of the 512 parallelised one are shown, which match perfectly.

In Fig. 12 the 3D reconstructed flight shape of the airframe is depicted for the simulation in Fig. 11.

Table 4 contains the simulation times of the calculation, which show one order of magnitude

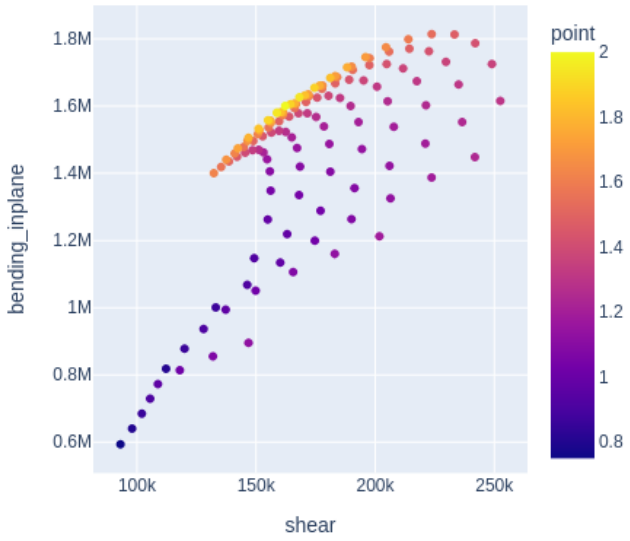

Figure 10: Gust case, bending-inplane versus shear

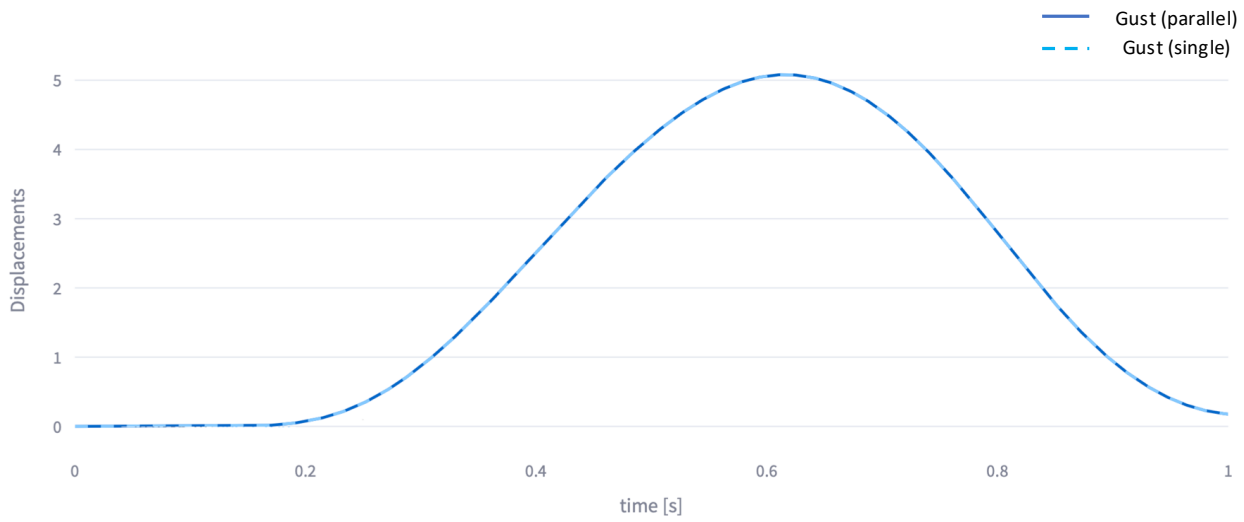

Figure 11: z-component of wing tip response to 1-cos gust excitation (concurrent and single simulation runs).

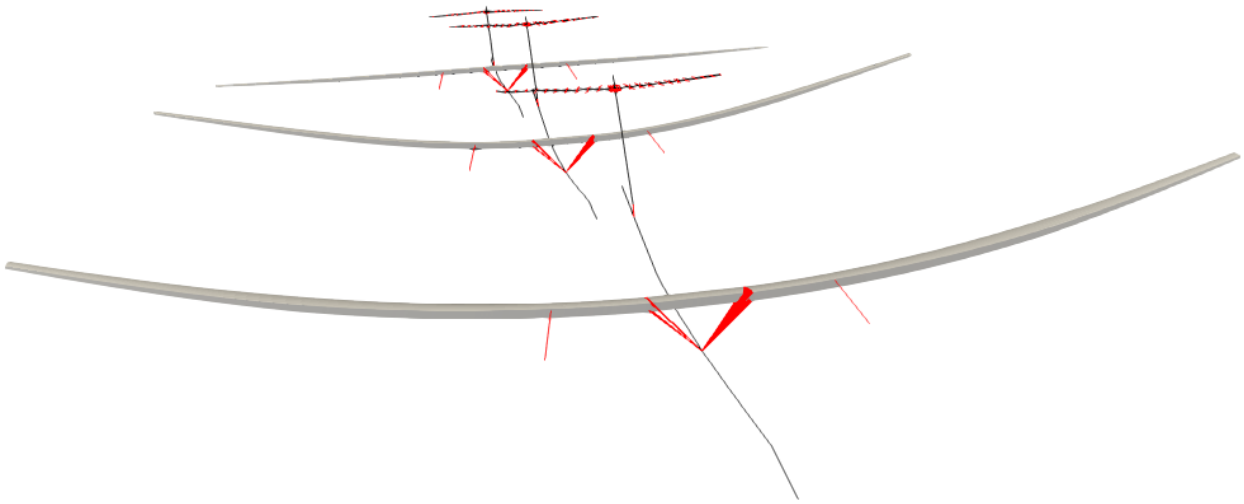

Figure 12: Full aircraft Dynamic response to 1-cos gust excitation

increase in performance when running in parallel in the CPU versus a complete single simulation running sequentially, and another order of magnitude when moving from the CPU to a modern GPU.

Table 4: Computational times multiple gust problem

Device	Time (sec.)
CPU (single)	$27.8 \times 512 = 14233.6$
CPU (parallel)	922.6
GPU	38.2

3.3.1 Differentiation of dynamic load envelopes

4 Conclusions

A modal-based, geometrically nonlinear formulation of the aircraft dynamics has been implemented for multiple load-cases parallelisation in modern hardware architectures. We have applied state-of-the-art techniques and tools employed for large problems in Deep Machine Learning to the computation and prediction of the sizing aeroelastic loads in commercial aircraft, which can expand thousands of simulations. Remarkable computational times of under a minute are achieved for 256 manoeuvres varying flow conditions and for 512 dynamic gust responses, including geometrically nonlinear effects in the simulations. Such a performance potentially unlocks two different applications: uncertainty quantification of the nonlinear aircraft response to a non-deterministic loading and integration of the software in larger multidisciplinary optimisation stuidies. The former has been demonstrated on a problem where a field of forces with an stochastic component induces very large deformation; it has been shown that while the statistics in the linear response can be easily forecast from one complete experiment, in the nonlinear case a Montecarlo simulation needs to be

carried out for each new set of loading scenario. For the latter, differentiation of the load envelopes via the AD capabilities within JAX will be the next step. Since we are already in the memory boundaries of a single GPU or CPU, this will require the use of multiple devices, for which we have already built the implementation. Scaling up the process to include various mass cases, as it is done in industrial scenarios, is also a feasible target. Thus combining prediction of sizing aeroelastic loads that include thousands of cases in commercial aircraft with the computation of their gradients with respect to design variables in a framework for multidisciplinary design optimization.

References

[1] Eli Livne. Aircraft Active Flutter Suppression: State of the Art and Technology Maturation Needs. *Journal of Aircraft*, 55(1):410–452, January 2018.

[2] Eirikur Jonsson, Cristina Riso, Bernardo Bahia Monteiro, Alasdair C. Gray, Joaquim R. R. A. Martins, and Carlos E. S. Cesnik. High-Fidelity Gradient-Based Wing Structural Optimization Including Geometrically Nonlinear Flutter Constraint. *AIAA Journal*, 61(7):3045–3061, July 2023.

[3] Marc Artola, Norberto Goizueta, Andrew Wynn, and Rafael Palacios. Aeroelastic control and estimation with a minimal nonlinear modal description. *AIAA Journal*, 59(7):2697–2713, 2021.

[4] Carlos E.S. Cesnik, Rafael Palacios, and Eric Y. Reichenbach. Reexamined Structural Design Procedures for Very Flexible Aircraft. *Journal of Aircraft*, 51(5):1580–1591, September 2014.

[5] Thiemo M Kier. An integrated model for lateral gust loads analysis and dutch roll flight dynamics using a 3d panel method. International Forum on Aeroelasticity and Structural Dynamics 2013, 2017.

[6] Alvaro Cea and Rafael Palacios. Geometrically Nonlinear Effects on the Aeroelastic Response of a Transport Aircraft Configuration. *Journal of Aircraft*, 60(1):205–220, January 2023.

[7] Alvaro Cea and Rafael Palacios. A Nearly-Real Time Nonlinear Aeroelastic Simulation Architecture Based on JAX. In *AIAA SCITECH 2024 Forum*, Orlando, FL, January 2024.

[8] Alvaro Cea and Rafael Palacios. Geometrically Nonlinear Analysis of Flexible Aircraft on Modern Hardware Architectures. *AIAA JOURNAL*, 2024 (pending publication).

[9] Deniz A. Bezgin, Aaron B. Buhendwa, and Nikolaus A. Adams. JAX-Fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows. *Computer Physics Communications*, 282:108527, January 2023.

[10] Tianju Xue, Shuheng Liao, Zhengtao Gan, Chanwook Park, Xiaoyu Xie, Wing Kam Liu, and Jian Cao. JAX-FEM: A differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science. *Computer Physics Communications*, 291:108802, October 2023.

[11] Dewey H. Hodges. Geometrically Exact, Intrinsic Theory for Dynamics of Curved and Twisted Anisotropic Beams. *AIAA Journal*, 41(6):1131–1137, June 2003.

[12] Rafael Palacios and Bodgan Epureanu. An intrinsic description of the nonlinear aeroelasticity of very flexible wings. *52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, (April):1–15, 2011.

[13] Alvaro Cea. *A Geometrically Nonlinear Approach for the Aeroelastic Analysis of Commercial Transport Aircraft*. PhD thesis, Imperial College London, 2021.

[14] Alvaro Cea and Rafael Palacios. Differentiable Aeroelastic Framework Suitable For Industrial Modelling Of Nonlinear Loads On Accelerators. *International Forum on Aeroelasticity and Structural Dynamics 2024, IFASD 2024*, 2024.

[15] Kenneth L Roger. Airplane math modeling methods for active control design. *Structural Aspects of Active Controls, AGARD CP-228*, pages 4.1–4.11, 1977.

[16] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018.

[17] O. Stodieck, J. E. Cooper, S. A. Neild, M. H. Lowenberg, and L. Iorga. Slender-Wing Beam Reduction Method for Gradient-Based Aeroelastic Design Optimization. *AIAA Journal*, 56(11):4529–4545, November 2018.