The 8th European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS Congress 2022
5-9 June 2022, Oslo, Norway

THE IPGDZ$^+$ TECHNIQUE FOR COMPRESSING PRIMAL SOLUTION TIME-SERIES IN UNSTEADY ADJOINT - APPLICATIONS & ASSESSMENT

A.-S.I. Margetis[a,1,*], E.M. Papoutsis-Kiachagias[a,2] and K.C. Giannakoglou[a,3]

[1] PhD Student, amargetis@mail.ntua.gr

[2] Research Scientist, vpapout@mail.ntua.gr

[3] Professor, kgianna@mail.ntua.gr and http://velos0.ltt.mech.ntua.gr/research/

[a] National Technical University of Athens (NTUA), School of Mechanical Engineering, Parallel CFD & Optimization Unit, 15772 Athens, Greece

Key words: Shape Optimization, Unsteady Adjoint, Data Compression Algorithms, Proper Generalized Decomposition, Computational Fluid Dynamics

Abstract. Gradient-based optimization for large-scale problems governed by unsteady PDEs, in which gradients with respect to the design variables are computed using unsteady adjoint, are characterized by the backward in time integration of the adjoint equations, which require the instantaneous primal/flow fields to be available at each time-step. The most widely used technique to reduce storage requirements, at the expense of a controlled number of recomputations, is binomial check-pointing. Alternatively, one may profit of lossless and lossy compression techniques, such as iPGDZ$^+$, this paper relies upon. iPGDZ$^+$ is a hybrid algorithm which consists of (a) an incremental variant of the Proper Generalized Decomposition (iPGD), (b) the ZFP and (c) the Zlib compression algorithms. Two different implementations of iPGDZ$^+$ are described: (a) the *Compressed Full Storage* (*CFS*) strategy which stores the whole time-history of the flow solution using iPGDZ$^+$ and (b) the *Compressed Coarse-grained Check-Pointing* (*3CP*) technique which combines iPGDZ$^+$ with check-pointing. Assessment in aerodynamic shape optimization problems in terms of storage saving, computational cost and representation accuracy are included along with comparisons with binomial check-pointing. The methods presented are implemented within the in-house version of the publicly available *adjointOptimisation* library of OpenFOAM, for solving the flow and adjoint equations and conducting the optimization.

1 INTRODUCTION

Time-dependent Computational Fluid Dynamics (CFD) solvers can be found in various applications to predict unsteady flows and run adjoint-based optimization. In the latter, the adjoint method computes the gradient of an objective function, usually cast in the form of a time integral, with respect to (w.r.t.) the design variables parameterizing the shape to be optimized. The great advantage of discrete or continuous adjoint, is that its cost is practically independent of the number of design variables. The unsteady adjoint PDEs are integrated backward in time and

require the instantaneous flow fields at each time-step. The two trivial ways to handle this issue are (a) full storage of the flow field time-series, provided that the available hardware is sufficient, and (b) repetitive flow recomputations, starting always from the same initial state. Binomial check-pointing [1] is the most frequently used middle-ground solution. As instantaneous flow fields are stored at predetermined time-steps (check-points) along the time-span, to retrieve the flow solution at time-instants other than check-points, the flow equations are integrated starting from the nearest previous check-point. Memory limitations, if any, can be overcome by storing some check-points on a larger, though slower, storage area (e.g. hard disk), instead of retaining all check-points in memory [2].

A viable alternative is to store the flow solution time-series in compressed form and, thus, avoid even a single flow recomputation. Lossless or lossy compression can be performed using relevant algorithms developed in the field of computer science, such as ZFP [3], SZ [4] and Zlib [5]. It is also possible to lossly compress the flow field time-series using cubic-splines, the Proper Orthogonal Decomposition (POD), the Gram-Schmidt Orthogonalization or the Proper Generalized Decomposition (PGD) techniques [6, 7, 8, 9].

In [6], the authors proposed and assessed the *Compressed Full Storage* (*CFS*) strategy in aerodynamic shape optimization, using lossy compression techniques, such as incremental PGD (iPGD) and ZFP. The iPGD may compress flow field snapshots incrementally, i.e. each time a new time-instant is computed, in contrast to standard PGD which requires the whole time-history to be available prior to its compression. To increase the efficiency of iPGD, the time-history of the flow problem is partitioned into non-overlapping, consecutive time-windows, which are individually compressed. An efficient synergistic use of iPGD and ZFP (referred to as iPGDZ) was also tried and proved to clearly outperform both iPGD and ZFP, achieving higher memory savings for the same error in the objective function gradient. *CFS* can further be enhanced by additionally utilizing the Zlib lossless compression algorithm [5], giving rise to a three-step compression. First, the iPGD algorithm compresses the flow field snapshots of a time-window. Then, the outcome of the iPGD for each time-window is lossly compressed using ZFP and, finally, the resulting ZFP stream is losslessly recompressed using Zlib. This will be referred to as the iPGDZ$^+$ algorithm, being the heart of the *CFS* strategy, as presented in this paper.

Based on the same compression kernel (iPGDZ$^+$), an alternative to *CFS* is also included and assessed in this paper. The co-called *Compressed Coarse-grained Check-Pointing* or *3CP* technique combines compression with coarse-level check-pointing. In particular, *3CP* firstly partitions the time-horizon into time-windows, similarly to *CFS*. Then, binomial check-pointing selects the time-windows in which the flow solution should be compressed using iPGDZ$^+$.

CFS, *3CP* and binomial check-pointing are assessed in three external aerodynamic shape optimization cases with unsteady flows; the criteria are: (a) reduction in storage, (b) extra cost and (c) accuracy of gradients to ensure that the outcome of the optimization remains unaffected.

2 THE IPGDZ$^+$ ALGORITHM AND ITS CONSTITUENTS

The PGD algorithm [10], initially proposed for compressing structured data, has been adapted to fields $f = f(x,t)$ available on (2D or 3D) unstructured grids, where x is the (integer) cell-ID and t the time (time-steps' counter). This field is approximated by the sum of M products

of spatial $X(x)$ and temporal $T(t)$ modes, where the value of M is user-defined, as follows: $f(x,t) \simeq \sum_{\mu=1}^{M} X^{\mu}(x)\, T^{\mu}(t)$. Modes are computed iteratively by solving systems of algebraic equations. In its incremental variant (iPGD), the instantaneous flow fields are compressed at the end of each time-step by updating the previously computed modes; thereafter, previous instantaneous flow fields are no longer needed. For instance, once the solution field at the new $(L+1)$ time-step becomes available, a new element T_{L+1}^{m}, $m \in [1, M]$, is computed and added to all temporal modes, while updating modes T_k^m, $k \in [1, L]$ and X_i^m, $i \in [1, I]$ too. At each time-step, (X^m, T^m), $m \in [1, M]$ are computed iteratively, by minimizing the error [6]

$$E_m = \frac{1}{2} \sum_{i=1}^{I} \left[\sum_{\mu=1}^{m} X_i^{\mu} T_{L+1}^{\mu} - f_{i,L+1} \right]^2 + \frac{w}{2} \sum_{i=1}^{I} \sum_{k=1}^{L} \left[\sum_{\mu=1}^{m} X_i^{\mu} T_k^{\mu} - f_{i,k}^{iPGD} \right]^2 \tag{1}$$

$f_{i,k}^{iPGD}$ is reconstructed by computing the sum of products $\sum_{\mu=1}^{M} \widetilde{X}_i^{\mu} \widetilde{T}_k^{\mu}$, $i \in [1, I]$, $k \in [1, L]$. Note that the f fields at previous time-instants should be reconstructed on the fly. w is a user-defined weight factor; in all set-ups of this paper, $w = 1$. The first term on the r.h.s. of eq. 1 corresponds to the approximation error at the current $(L+1)$ time-step, whereas the second term to the cumulative error of the L previous time-instants. Unknown modes $\left(X_i^m,\ T_k^m \text{ and } T_{L+1}^m\right)$ are computed by satisfying $\partial E_m / \partial X_i^m = \partial E_m / \partial T_k^m = \partial E_m / \partial T_{L+1}^m = 0$, [6].

To retain the compression accuracy as the total number of time-steps increases, more modes (i.e. a higher value of M) are needed and this increases the extra cost per time-step. To deal with this issue, in [6], the time-domain is partitioned into consequtive, non-overlapping time-windows, with a user-defined number K of time-steps each (excluding, possibly, the last one). Each time-window may have a different set of modes, as this is compressed independently from the others.

Next to iPGD, the ZFP and Zlib algorithms are deployed. At the end of each time-window, the spatial and temporal modes for this window are lossly compressed and kept in memory by successively using ZFP [3] and Zlib [5]. Only the modes of the last time-window are stored in full precision, since there is no benefit from their compression. For the lossy compression by ZFP, each mode of random size N is transformed into a 2D array of size $4m$, where $m = \lceil N/4 \rceil$ and, then, compressed using the *fixed-precision* mode of ZFP with the same user-defined number (P) of bits. The resulting data stream is losslessly compressed using the fastest level of the Zlib library [5], which favors compression/decompression speed w.r.t. data reduction. The main steps of the iPGDZ$^+$ algorithm are sketched in fig. 1.

To get the most out of the available memory, the computational domain can be divided into (user-defined) sub-domains, in each of which flow fields are compressed using a different number M of modes. The rest of the compression parameters, namely K and P, are common in all sub-domains. This treatment allows greater memory savings, while retaining the accuracy of the computed sensitivity derivatives (SDs). Higher M values should be used in areas where strong time-varying local flow structures are expected, such as in the wake of a bluff body, whereas lower M values can be used elsewhere to avoid unnecessary storage. This feature is assessed in Section 4.

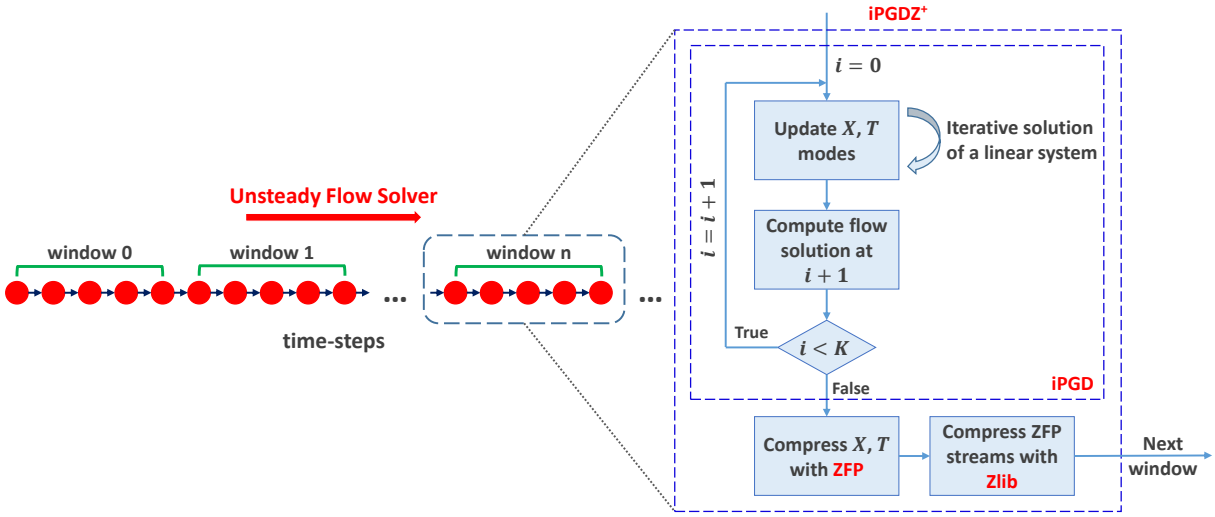


Figure 1: Flowchart of the iPGDZ$^+$ algorithm. Red circles denote instantaneous flow fields; i is the "local" (i.e. within this time-window) time-step counter. Each time-window has K time-steps.

3 TWO DIFFERENT IMPLEMENTATIONS OF IPGDZ$^+$: CFS & 3CP

The implementations of iPGDZ$^+$ in *CFS* and *3CP* differ, fig. 2. In the former, iPGDZ$^+$ is used to compress and store the whole flow solution in memory. In the latter, the flow solution is compressed by iPGDZ$^+$ only at a subset of the time-windows, to be referred to as "check-windows". The number S_w of the check-windows is defined by the user, depending on the available memory. Check-windows along the time-span are selected using the binomial check-pointing algorithm [1] applied to coarse grains (time-windows), rather than single time-steps, subject to the constraint of storing the last time-window. To retrieve the flow solution at time-instants not belonging to a check-window, the flow solver starts integrating forward from the last time-step of the nearest previous check-window. The *3CP* technique is sketched in fig. 3.

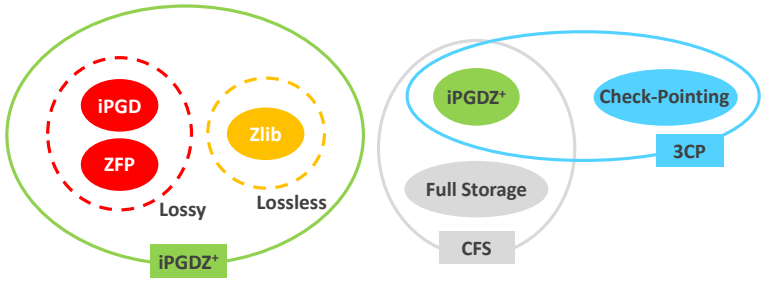


Figure 2: Left: The three constituent techniques of iPGDZ$^+$. Right: *CFS* and *3CP* techniques sharing the same core (iPGDZ$^+$).

In Section 4, iPGDZ$^+$ (used within *CFS* or *3CP*) is denoted by iPGDZ$^+$(S_w,M,K,P) to specify the used parametric values. If not stated otherwise, the whole computational domain is compressed using the same (different for each set-up) number M of modes.

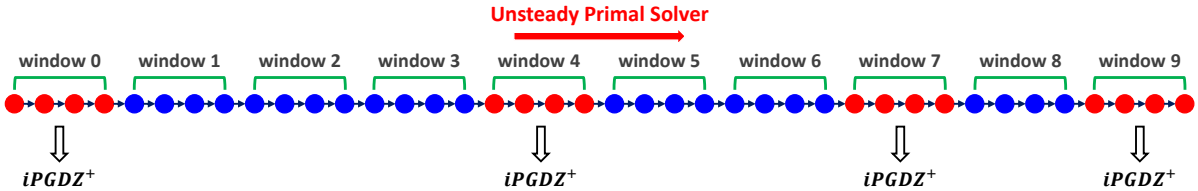


Figure 3: The *3CP* algorithm. In this example, the time-domain with 40 time-steps in total is partitioned into 10 time-windows. Flow solution at $S_w = 4$ check-windows (in red) is compressed using iPGDZ$^+$ and stored; blue circles correspond to time-instants of the flow that are recomputed during the adjoint solution.

4 APPLICATIONS

Benefits from the use of *CFS* and *3CP* in unsteady adjoint are demonstrated in three external aerodynamics cases. Cases description, including grids and parameterization, are given below (see also Table 1). **Case 1** is dealing with the shape optimization of a motorbike's fairing, whereas **Case 2** with the DrivAer car model [11]. Here, the fast-back DrivAer configuration with smooth underbody, mirrors and wheels is used as starting shape. Half of the car is modeled; a rotating wall boundary condition is imposed on the tires and wheel rims. **Case 3** performs the same for the ID.3 passenger car; its geometry was 3D-scanned by A2MAC1 [12] and the grid was generated by AirShaper [13]. Starting and optimized shapes in all cases are shown in fig. 4.

Volumetric B-splines lattices are used to morph both the part(s) of the shapes that are allowed to change and the surrounding CFD grid. The Cartesian coordinates of the control points of those lattices are the design variables. Particularly, in **Case 2**, all control points (CPs) placed on the symmetry plane were not allowed to move in the transversal direction. Also, in this case, bounding box constraints were imposed for the design variables; each CP is confined to move inside a rectangular parallelepiped, the edges of which pass through the midpoints of the edges connecting this and its adjacent CPs.

Common objective function in all cases is the time-averaged drag coefficient $J_{C_D} = \frac{1}{T_{of}} \int_{t^*}^{t^*+T_{of}} w\left(\frac{t-t^*}{T_{of}}\right) C_D(t)\, dt$, where $C_D(t) = \frac{\int_{S_W} [p(t) n_i - \tau_{ij}(t) n_j] r_i dS}{\frac{1}{2} A_{ref} U_{ref}^2}$ is the instantaneous drag coefficient, $\mathbf{r}$ is the unit vector parallel to the fixed far-field velocity, $\mathbf{n}$ is the unit vector normal to the vehicle boundary facing towards the solid, $\tau_{ij} = (\nu + \nu_t)\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)$ is the stress tensor, v_i are the velocity components, p is the pressure divided by the fluid density, ν is the bulk and ν_t the eddy viscocity, A_{ref} and U_{ref} are the frontal area and far-field velocity magnitude, respectively, and $w(\chi) = \frac{2}{3}\left[1 - \cos(2\pi\chi)\right]^2$, $\chi \in (0,1)$ is the Hann-Square-windowing function [14] used to regularize time-averaging. To exclude transient phenomena from the definition of the objective function, at each optimization cycle, J_{C_D} is integrated over $[t^*, t^*+T_{of}]$, where t^* is the warm-up time and T_{of} is the size of the integration window. The values of t^*, T_{of} and Δt for each case are summarized in Table 1.

The flow is governed by the Unsteady Reynolds-Averaged Navier Stokes (URANS) equations, including the Spalart-Allmaras [15] turbulence model and the Hamilton-Jacobi PDE computing distances Δ from the walls. The computation of gradient $\delta J_{C_D}/\delta \mathbf{b}$ of J_{C_D} w.r.t. the design variables $\mathbf{b}$ requires the numerical solution of the unsteady adjoint equations. The adjoint

system includes the adjoint to the mean-flow equations, the adjoint to the Spalart-Allmaras turbulence model PDE, the adjoint to the Hamilton-Jacobi equation and, also, the differentiated Spalding's law of the wall. Flow and adjoint equations can be found in [16] and [6]. Both the flow and the adjoint PDEs are discretized and numerically solved on unstructured grids for $t \in [0, t^* + T_{of}]$, using the cell-centered, collocated, finite-volume infrastructure provided by OpenFOAM, employing the PIMPLE algorithm.

All runs are performed on Intel Xeon CPU E5-2630 v3 cores at 2.40GHz, the number of which is defined separately for each case. At each time-step, the instantaneous fields of p, v_i, the Spalart-Allmaras turbulence model variable $\widetilde{\nu}$, computed at cell-centers, and volume fluxes ϕ at cell-faces, are stored for use by the adjoint solver and for restarting the flow solver from the last check-point during the adjoint solution. Due to the excessive memory requirements of full storage for real-world 3D applications, the CPU cost of the proposed technique is compared to the cost of "standard" binomial check-pointing [1]; the latter is denoted as *stdCP(S_p)*, where S_p is the number of check-points.

Quantitive comparisons are based on the following metrics:

$$CR = \frac{\text{uncompressed size of primal time-series}}{\text{compressed size of all check-windows}}, \quad \theta = cos^{-1} \frac{\frac{\delta J}{\delta \mathbf{b}} \cdot \frac{\delta J'}{\delta \mathbf{b}}}{\left\| \frac{\delta J}{\delta \mathbf{b}} \right\|_2 \left\| \frac{\delta J'}{\delta \mathbf{b}} \right\|_2}, \quad \varepsilon = \frac{\left\| \frac{\delta J}{\delta \mathbf{b}} - \frac{\delta J'}{\delta \mathbf{b}} \right\|_2}{\left\| \frac{\delta J}{\delta \mathbf{b}} \right\|_2} \quad (2)$$

where CR is the *compression ratio* and θ and ε are the angle and normalized difference between reference SDs computed using *stdCP* ($\delta J/\delta \mathbf{b}$) and SDs computed using iPGDZ$^+$ $\left(\delta J'/\delta \mathbf{b}\right)$, respectively. For both θ and ε, the ideal value is zero. CR measures savings in memory requirements when either of the two proposed techniques is used. For *stdCP*, *CR* is the ratio of the total number of time-steps and the number of check-points retained in memory.

Irrespective of the compression technique used, the performed optimizations reduce J_{C_D} by 6 to 7% in all cases, fig. 5. It is important though to show, first of all, that the use of lossy compression does not harm the accuracy of the computed SDs or the convergence of the optimization loop. Indeed, SDs are practically unaffected by lossy compression of primal data, for both *CFS* and *3CP*, with $\varepsilon < 0.6\%$ and $\theta < 0.3^\circ$ in all cases, table 2. The same holds for the evolution of J_{C_D} in the course of the optimization, left column in fig. 5. In the right column in the same figure, the instantaneous drag coefficient $C_D(t)$ at the starting shape of each case is plotted, and this confirms the unsteady nature of the flow around the examined bodies.

After having shown that the compression/decompression process does not affect the optimization procedure, an interesting comparison in terms of memory requirements and CPU cost follows. In all cases, *stdCP* is used as the reference run; the number of the check-points is selected based on the available system memory on a case-by-case basis. Should the target be to avoid any recomputation, *stdCP* can be replaced by *CFS* which reduces the CPU cost per optimization cycle by $\sim 30\%$. Storage requirements are reduced by 2 to 3 orders of magnitude compared to full storage without compression ($CR = 450 \div 1200$), or $15 \div 35$ times compared to *stdCP*. If a higher reduction in memory is required, *3CP* can be used instead. In **Cases 1 and 2**, *3CP* achieves an impressive reduction in memory ($CR > 2000$ or ~ 60 times less memory demands than *stdCP*), which comes at a slightly higher cost compared to *CFS*. Even so, *3CP* has a $\sim 6\%$ lower cost than *stdCP*. *3CP* can be tuned to reach the right balance between

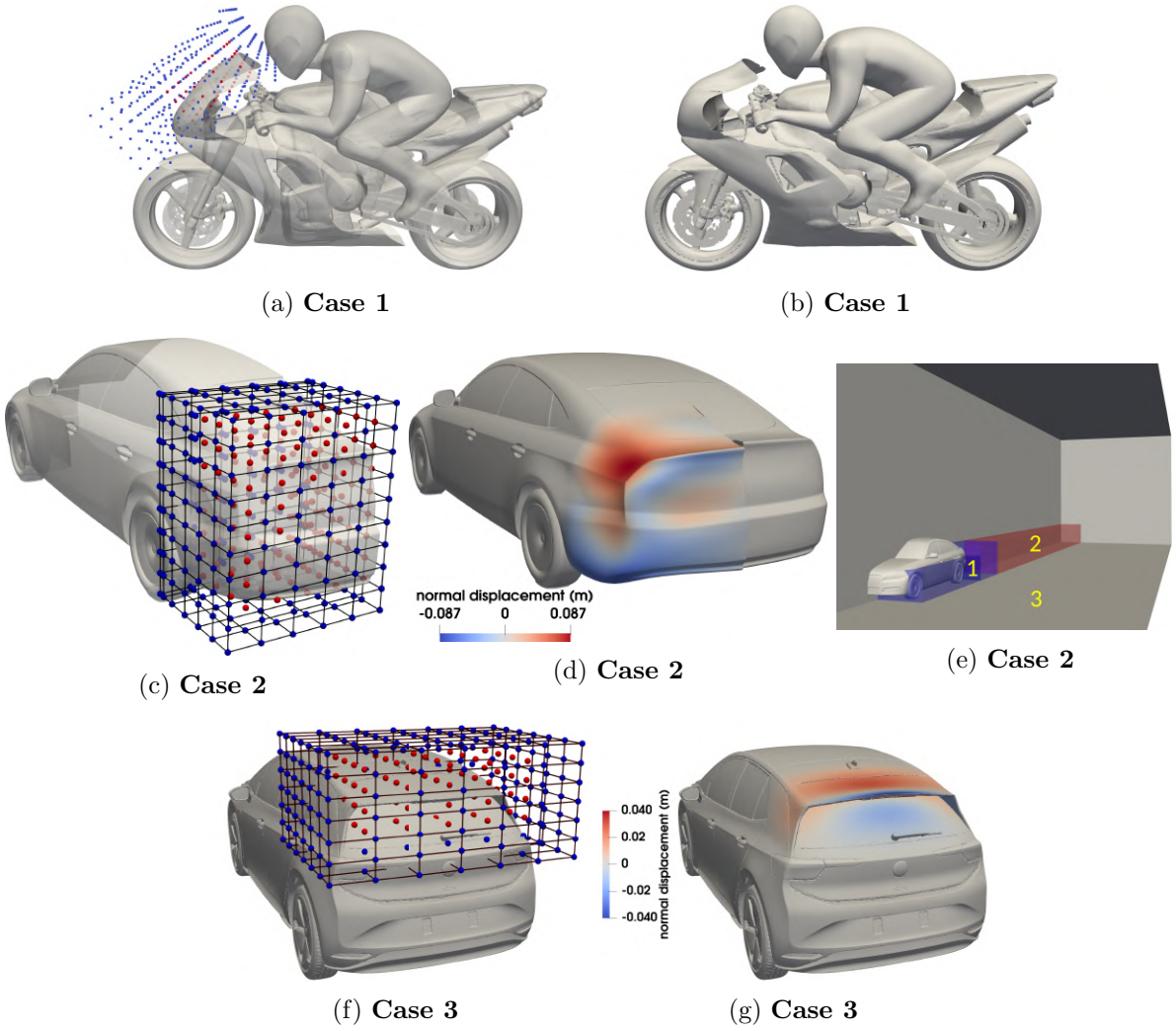
reduction in memory and CPU cost. This means, that the cost of *3CP* can further be reduced by increasing the memory footprint accordingly, fig. 6. In any case, both *CFS* and *3CP* have a clear advantage vis-à-vis to *stdCP* both in terms of CPU cost and memory requirements.

The same memory savings with *CFS* can be obtained by using *stdCP*, but at a significantly higher cost, since the flow recomputations that are performed have the same cost as 1.4, 2.8, 14 complete flow solutions, respectively for each case. On the other hand, by properly selecting the check-points' number, *stdCP* may also match the memory requirements of *3CP*, but at a noticeably higher cost (flow recomputations are as expensive as 3.0 to 7.3 complete flow evaluations).

In **Case 2**, to optimally use the available memory, the computational domain is divided into three non-overlapping regions, fig. 4e, in which flow fields are compressed using $M=4$, 3 and 2, respectively. This way, regions where local time-varying flow-structures are expected to form, i.e. the wake and the region close to the underbody, are compressed with a higher accuracy than the rest of the domain. Using this set of M values, indexed by $\mathcal{C}$ in table 2, memory savings noticeably increase for both *CFS* and *3CP* ($CR=700$ and 2500, respectively) for the same error in SDs and CPU cost.

Case	***Re*** ($\times\mathbf{10^6}$)	***Cells*** ($\times\mathbf{10^6}$)	$\mathbf{t^*}$ ***(sec)***	$\mathbf{T_{of}}$ ***(sec)***	$\mathbf{\Delta t}$ ($\times\mathbf{10^{-4}}$ ***sec***)	***Control Lattice***	***Active CPs***	***Opt. Method***	***CPU Cores***	***Memory (GB)***
1	2.7	1.1	1.5	5.5	2.5	7×7×7	4×3×4	*Conjugate Gradient*	32	65
2	10	5.3	0.9	1.5	0.6	7×6×10	4×5×8	*SQP*	132	300
3	8.5	16.6	0.6	2.4	3	7×7×7	4×5×4	*Conjugate Gradient*	132	300

Table 1: Main settings for the solver and the optimization for each case.

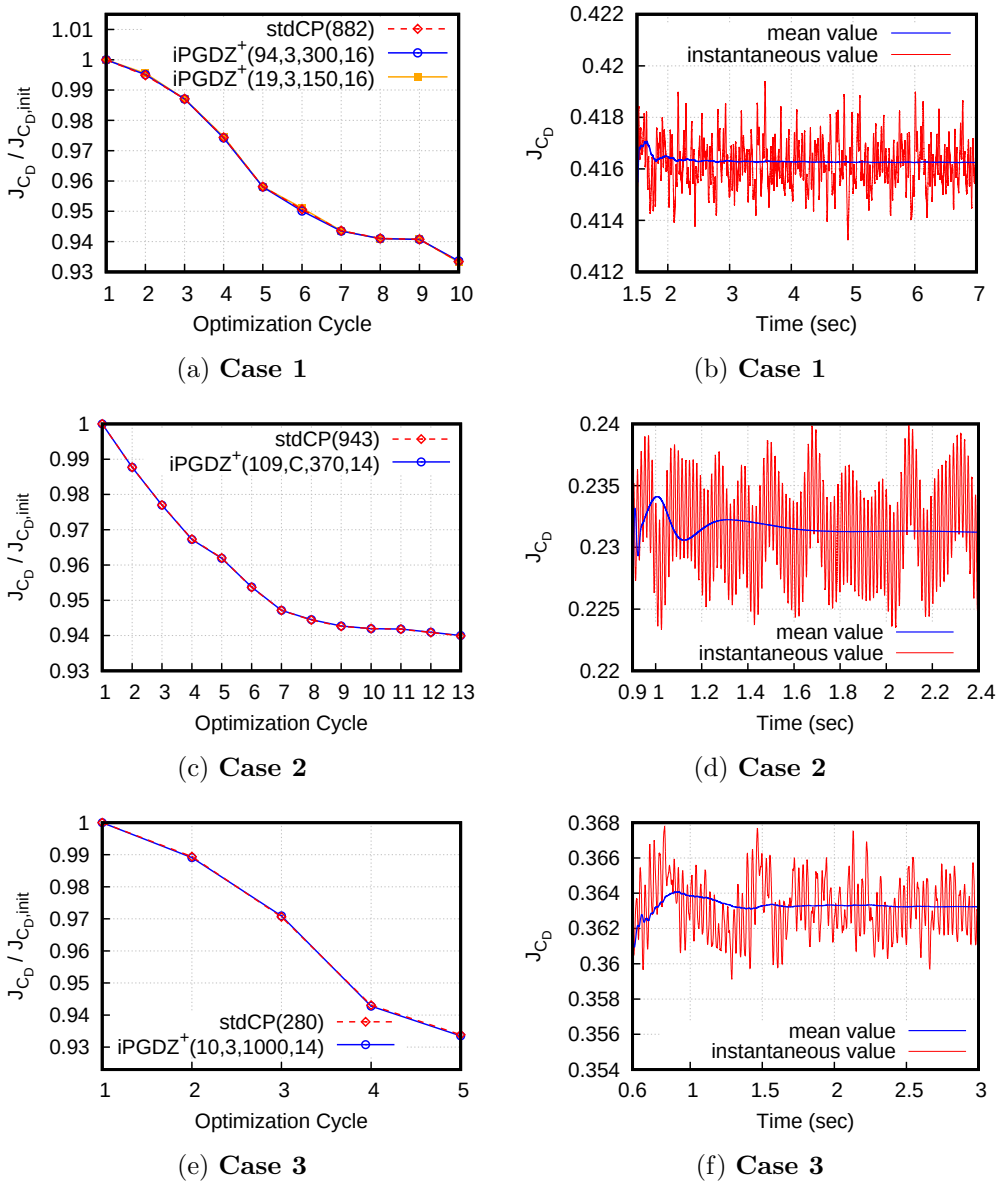


(a) **Case 1** (b) **Case 1**

(c) **Case 2** (d) **Case 2** (e) **Case 2**

(f) **Case 3** (g) **Case 3**

Figure 4: (a, c, f): Volumetric B-Splines lattice on the starting geometry of each case. CPs with at least one active degree of freedom are colored in red; non-active CPs are in blue. (e): Three non-overlapping regions, the union of which gives the whole computational domain, using different number M of modes for iPGDZ^{+}. Regions 1, 2 and 3 contain 2.41, 0.11 and 2.78 million cells, respectively. (b, d, g): Optimized shapes. The signed cumulative normal displacement fields plotted on the optimized shapes indicate directions in which surface points were displaced, either inwards (red) or outwards (blue).



(a) **Case 1** (b) **Case 1**

(c) **Case 2** (d) **Case 2**

(e) **Case 3** (f) **Case 3**

Figure 5: (a, c, e): Evolution of J_{C_D} in the course of the optimization for all cases. In **Cases 1 and 3**, J_{C_D} reduced by 6.7%, whereas in **Case 2** by 6.0%. Recall that the abbreviation iPGDZ$^+$(S_w,M,K,P) determines the number S_w of the check-windows, the number M of modes, the number K of time-steps per time-window and the number P of bits used in ZFP. In **Case 2**, $\mathcal{C}$ denotes the use of different values of M ($M=4,3,2$) in each sub-domain of fig. 4e. (b, d, f): Evolution of the instantaneous and mean J_{C_D} over T_{of} on the starting geometries.

Case	*Storage Strategy*		*Cost/cycle*		CR	*Mem. (GB)*	*SDs*	
			CPUh	*%*			$\theta(^\circ)$	ε
1		*stdCP(882)*	1.63*K*	100%	31.7	64.1	–	–
	CFS	*iPGDZ$^+$(94,3,300,16)*	1.19*K*	73.2%	451.5	4.50	0.30	0.57%
	3CP	*iPGDZ$^+$(19,3,150,16)*	1.52*K*	93.5%	1944	1.05	0.27	0.53%
2		*stdCP(943)*	5.89*K*	100%	42.4	298.0	–	–
	CFS	*iPGDZ$^+$(160,3,250,14)*	4.25*K*	72.2%	472.9	26.7	0.10	0.18%
		iPGDZ$^+$(109,$\mathcal{C}$,370,14)	4.25*K*	72.1%	702.0	18.0	0.08	0.15%
	3CP	*iPGDZ$^+$(30,3,90,14)*	5.60*K*	95.3%	2237	5.7	0.27	0.48%
		iPGDZ$^+$(27,$\mathcal{C}$,110,14)	5.59*K*	95.1%	2508	5.0	0.26	0.49%
3		*stdCP(280)*	7.77*K*	100%	35.7	297.3	–	–
	CFS	*iPGDZ$^+$(10,3,1000,15)*	5.37*K*	69.1%	1221	8.7	0.19	0.34%

Table 2: CPU cost, compression and SDs' accuracy metrics, at the first optimization cycle. The first line of each case corresponds to check-pointing; then, runs using *CFS* and *3CP* follow. Recall that the abbreviation iPGDZ$^+$(S_w,M,K,P) determines the number S_w of the check-windows, the number M of modes, the number K of time-steps per time-window and the number P of bits used in ZFP. In **Case 2**, $\mathcal{C}$ denotes the use of different values of M ($M=4,3,2$) in each sub-domain of fig. 4e.

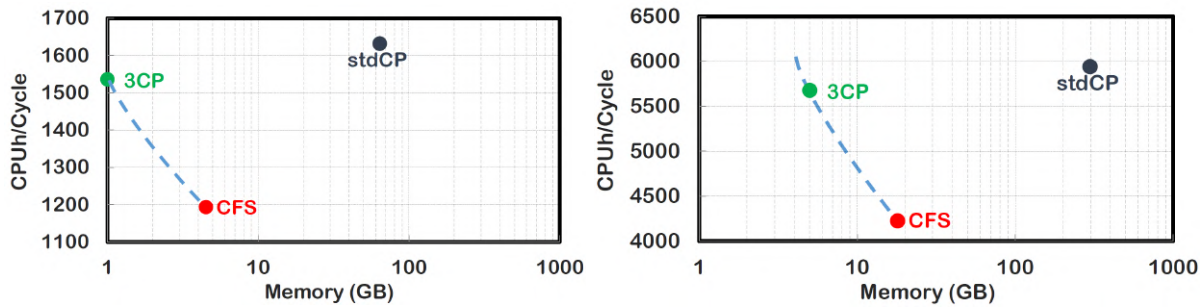


Figure 6: Memory requirements and CPU cost per optimization cycle. Left: **Case 1**. Comparison of *CFS* using iPGDZ$^+$(94,3,300,16), *3CP* using iPGDZ$^+$(19,3,150,16) and *stdCP* using 882 check-points. Right: **Case 2**. Comparison of *CFS* using iPGDZ$^+$(109,$\mathcal{C}$,370,14), *3CP* using iPGDZ$^+$(27,$\mathcal{C}$,110,14) and *stdCP* using 943 check-points.

5 CONCLUSIONS

Two techniques that can greatly reduce the storage requirements and the CPU cost of the backward in time integrated unsteady adjoint equations, in gradient-based optimization, are compared to the widely used binomial check-pointing technique (*stdCP*). These are the *Compressed Full Storage* (*CFS*) and the *Compressed Coarse-grained Check-Pointing* (*3CP*) strate-

gies, which share the same core, i.e. the iPGDZ$^+$ lossy compression technique. The iPGDZ$^+$ technique synergistically applies lossy and lossless compression by means of: (a) the incremental Proper Generalized Decomposition (iPGD), (b) the ZFP and (c) the Zlib algorithms. Between the two proposed techniques, *CFS* achieves a higher reduction in the CPU cost per optimization cycle, whereas *3CP* a higher reduction in memory footprint and vice versa. In the three automotive applications, *CFS* reduced the CPU cost per optimization cycle by 30% and memory requirements by a factor of 15 to 35 compared to *stdCP*. On the other hand, *3CP* achieved an impressive 60 times reduction in memory compared to *stdCP*, which came also at a 6% lower CPU cost. This remarkable reduction in storage was achieved without practically affecting neither the computed sensitivity derivatives nor the outcome of the optimization. Note that though iPGDZ$^+$ is independent from the underlying primal equations and can be used either with continuous or discrete adjoint, the paper focuses solely on shape optimization methods in unsteady fluid mechanics, using continuous adjoint.

AKNOWLEDGEMENTS

Development of the core compression techniques was made in the context of a research project funded by the Bayerische Motoren Werke (BMW). The iPGDZ$^+$ technique and *CFS* were developed in the framework of the European High-Performance Computing Joint Undertaking (JU) (Exploitation of Exascale Systems for Open-Source Computational Fluid Dynamics by Mainstream Industry) under Grant Agreement No. 956416. The development of the *3CP* technique was made through a PhD scholarship to the first author offered by the Special Account for Research Funding (ELKE) of NTUA. Computational time was granted by the National Infrastructures for Research and Technology S.A. (GRNET S.A.) in the National HPC facility - ARIS - under project ID 10031. The authors would like to acknowledge the provision of the scanned geometry and computational mesh of the ID.3 car model by A2MAC1 and AirShaper, respectively.

REFERENCES

[1] A. Griewank and A. Walther. Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45, 2000.

[2] P. Stumm and A. Walther. Multistage approaches for optimal offline checkpointing. *SIAM Journal on Scientific Computing*, 31(3):1946–1967, 2009.

[3] P. Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[4] S. Di and F. Cappello. Fast Error-Bounded Lossy HPC Data Compression with SZ. pages 730–739. IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016.

[5] J. Gailly and M. Adler. Zlib Compression Library. http://zlib.net/.

[6] A.-S. Margetis, E. Papoutsis-Kiachagias, and K. Giannakoglou. Lossy compression techniques supporting unsteady adjoint on 2D/3D unstructured grids. *Computer Methods in Applied Mechanics and Engineering*, 387:114152, 2021.

[7] S. Walton, O. Hassan, and K. Morgan. Reduced order modelling for unsteady fluid flow using proper orthogonal decomposition and radial basis functions. *Applied Mathematical Modelling*, 37(20):8930–8945, 2013.

[8] E. Cyr, J. Shadid, and T. Wildey. Towards efficient backward-in-time adjoint computations using data compression techniques. *Computer Methods in Applied Mechanics and Engineering*, 288(C), 2014.

[9] L. Yang and S. Nadarajah. Data Compression Algorithms for Adjoint Based Sensitivity Studies of Unsteady Flows. Fluids Engineering Division Summer Meeting, 2018.

[10] F. Chinesta, R. Keunings, and A. Leygue. *The Proper Generalized Decomposition for Advanced Numerical Simulations: A Primer*. Springer, 2014.

[11] A. Heft, T. Indinger, and N. Adams. Experimental and Numerical Investigation of the DrivAer Model. *ASME, Symposium on Issues and Perspectives in Automotive Flows*, Volume 1: Symposia, Parts A and B:41–51, 2012.

[12] A2MAC1 Automotive Benchmarking. https://portal.a2mac1.com/.

[13] AirShaper. https://airshaper.com/.

[14] J. Krakos, Q. Wang, S. Hall, and D. Darmofal. Sensitivity analysis of limit cycle oscillations. *Journal of Computational Physics*, 231(8):3228–3245, 2012.

[15] P. Spalart and S. Allmaras. A One-Equation Turbulence Model for Aerodynamic Flows. *AIAA*, 439, 01 1992.

[16] E. Papoutsis-Kiachagias and K. Giannakoglou. Continuous Adjoint Methods for Turbulent Flows, Applied to Shape and Topology Optimization: Industrial Applications. *Archives in Computational Methods in Engineering*, 23:255–299, 2016.