Computer Assisted Methods in Engineering and Science
31(1): 51–66, 2024, doi: 10.24423/cames.2024.1011

Accelerating Computation of a Reduced Order Model of a Structural System Resulting from Craig–Bampton Reduction Using GPU Programming

Piotr GORECKI[1)], Miłosz KALINOWSKI[2)], Łukasz JEZIOREK[2)*], Jakub BRONISZEWSKI[1)], Tomasz KOZIARA[1)]

[1)] *General Electric Company Polska sp. z o.o., Al. Krakowska 110/114, 02-256 Warsaw, Poland*

[2)] *Łukasiewicz Research Network – Institute of Aviation, Engineering Design Center, Al. Krakowska 110/114, 02-256 Warsaw, Poland*

[*] *Corresponding Author e-mail: jeziorek.lukasz@ge.com*

The Craig–Bampton (CB) method is a well-known substructuring technique that reduces the size of a finite element model (FEM) using a set of vibration modes. For large FEA models, the reduction process could be computationally expensive since it requires algebra operations on FEM mode shapes and FEM system sparse matrices. In this paper, we investigate the potential of usage of GPU parallel processing to speed up solving the system of linear equations that results from the CB reduction process made for a model of cyclic structures. A Python based high-level approach, employing the CuPy, GinkGo and STRUMPACK libraries on the GPU, is compared with an optimized Fortran code. In side-to-side comparisons, employing the same inputs, the Python-GPU code is run on a single GPU device and the Fortran code is run on a multi-core compute node. The CB reduction process was split into several parts, each dealing with different kind of algebraic formulation of the problem. Performance comparisons were focused on the sparse system linear solver, since it turned out to be the most time-consuming part. The results suggest that the current GPU-based linear sparse solvers do not surpass the state-of-the-art CPU-based MKL PARDISO solver (at least up to 1M DOFs).

Keywords: GPU, CPU, reduced order model, structural model, CuPy.

1. Introduction

The calculation of a reduced order model (ROM) for a structural system, represented by a discrete finite element model, can be pursued in a variety of ways [1]. The selection of the proper reduced order model is primarily influenced by the FEM analysis type (static, dynamic, contact non-linearities, etc.). This work focuses on a particular reduced model – the CB method [2–4], which is one of the most popular component mode synthesis (CMS) techniques. While the CB method is typical and robust technique used in the industry, the most time-consuming part of this process is solving the resulting system of linear equations. The potential of speeding up this particular part of the process by the usage of high-performance computing (HPC) GPU hardware is explored. The utilization of existing high-level algebraic toolboxes, developed for the GPU hardware, is considered in this study. To this end, the Python-based library CuPy [5] is used as a formalism of focus for the quickest entry into writing algebraic code executed on the GPU.

As the ROM method in the study remains fixed, the main attention is paid on comparing various implementations of the GPU algebraic libraries and exploiting their efficiency for the ROM building process. The requirements of CB ROM application, used by the authors, substantially limited this study to several GPU-based algebraic toolboxes. The first limitation was the possibility of using the sparse representation of structural matrices. The aim of the work was to speed up the building process of high-fidelity FEM models, which can only be represented by sparse matrices due to RAM memory limitations. Another important limitation was the use of complex-valued algebra, e.g., a complex linear solver. The performance of GPU-based toolboxes was compared against a reference CPU implementation (Fortran code with the Intel MKL library).

At the beginning of the paper the CB and cyclic symmetry reductions are explained. Then, the FEM model and its various computational grid sizes are provided. In the next section, the tested algebra toolboxes are listed. Finally, performance metrics for both GPU and CPU implementation are provided, and the results are discussed in Sec. 5.

2. Structural reduced order modeling

Turbomachinery components such as bladed disks or blisks are subjected to time-varying loads, which are the primary contributors to High Cycle Fatigue failures (HCF). To study HCF, structural stress must be obtained from displacement response. For linear analysis of forced response, the displacements of the excited structure are obtained from the equation of motion (Eq. (1))

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{C}\dot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{F}(t). \tag{1}$$

In Eq. (1), $\mathbf{M}$, $\mathbf{K}$, $\mathbf{C}$ are respectively the mass, stiffness and damping matrices. These matrices are obtained from finite element analyses of a discretized model. $\mathbf{F}$ is the time-varying excitation force applied to the model degrees-of-freedom (DOFs) of the FEM model. In the design of turbomachinery components there is a strict requirement to construct high-fidelity FEM models to ensure the accuracy of the analysis. This results in a high number of DOFs, and, in consequence, very large dimensions of structural system matrices $\mathbf{M}$, $\mathbf{K}$, and $\mathbf{C}$. To handle such large matrices, sparse matrix representation has to be used (e.g. compressed row format). Otherwise, it could not be possible to provide computation resources (RAM memory) to solve a given model.

Using high-fidelity FEM models is not practical for advanced analysis, such as non-linear contact, because they require much more resources than standard linear analyses. In such situations, a reduced model is applied to lower the number of DOFs, while preserving the dynamic properties of the structure. In the next section, the two-level reduction of the FEM model will be explained – first, the cyclic reduction and then the CB reduction.

2.1. Cyclic reduction

Turbomachinery components such as bladed disks or blisks exhibit a cyclic symmetry property. This means that they consist of identical substructures (sectors), which are rotated by a fixed angle forming a full wheel geometry. The cyclic symmetry is characterized by the sector angle $2\pi/N$, where N is the number of sectors. The full wheel model analyzed in this study is shown in Fig. 1, including the extracted single cyclic sector.

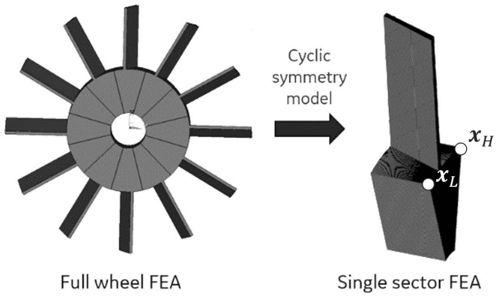

Fig. 1. Full wheel model and a corresponding cyclic sector.

In order to reduce the FEM model of a full wheel into a cyclic symmetry model, both the FEM structure and the subjected loads have to be cyclic. If this condition is fulfilled there is a specific relation of the common surfaces shared by cyclic sectors. These surfaces, for a single sector, are called high and low edges. In Fig. 1, a cyclic sector is shown with marked corresponding nodes on low and high edges. For such node pairs, there is a specific phase-lag relation of displacements $\mathbf{x}$

$$\mathbf{x}_H = \mathbf{x}_L e^{-i\alpha}, \tag{2}$$

where $\mathbf{x}_H$ and $\mathbf{x}_L$ denote the displacement vectors of high-edge nodes and low-edge nodes, respectively; α is called the interblade phase angle, and is dependant on the external excitation pattern (engine order excitation: EO). The interblade phase angle for a full wheel with N blades can be derived from:

$$\alpha = 2\pi \frac{EO}{N}. \tag{3}$$

The displacements of the entire cyclic sector can be split as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_H \\ \mathbf{x}_I \\ \mathbf{x}_L \end{bmatrix}, \tag{4}$$

where $\mathbf{x}_I$ denotes the displacement vector of internal nodes. Based on Eqs. (2) and (4), a cyclic transformation matrix can be derived as follows:

$$\begin{bmatrix} \mathbf{x}_H \\ \mathbf{x}_I \\ \mathbf{x}_L \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{I}e^{-i\alpha} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}_I \\ \mathbf{x}_L \end{bmatrix}. \tag{5}$$

Therefore, the cyclic transformation matrix becomes:

$$\mathbf{T} = \begin{bmatrix} \mathbf{0} & \mathbf{I}e^{-i\alpha} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{6}$$

For high-fidelity FEM models the $\mathbf{T}$ matrix is represented as a sparse matrix, given that the majority of entries are zero. The cyclic transformation introduced in Eq. (1), together with left-multiplying the equations by $\mathbf{T}^H$ yields:

$$\mathbf{T}^H\mathbf{M}\mathbf{T}\,\ddot{\overline{\mathbf{x}}}_{cyc} + \mathbf{T}^H\mathbf{C}\mathbf{T}\,\dot{\overline{\mathbf{x}}}_{cyc} + \mathbf{T}^H\mathbf{K}\mathbf{T}\,\overline{\mathbf{x}}_{cyc} = \mathbf{T}^H\mathbf{F}. \tag{7}$$

Thus, the cyclic matrices become:

$$\mathbf{M}_{cyc} = \mathbf{T}^H\mathbf{M}\mathbf{T}, \tag{8}$$

$$\mathbf{K}_{cyc} = \mathbf{T}^H\mathbf{K}\mathbf{T}, \tag{9}$$

$$\mathbf{C}_{cyc} = \mathbf{T}^H\mathbf{C}\mathbf{T}, \tag{10}$$

and the DOFs are reduced to:

$$\overline{\mathbf{x}}_{cyc} = \begin{bmatrix} \mathbf{x}_I \\ \mathbf{x}_L \end{bmatrix}. \tag{11}$$

The final form of the cyclic equation of motion becomes:

$$\mathbf{M}_{cyc}\,\ddot{\overline{\mathbf{x}}}_{cyc} + \mathbf{C}_{cyc}\,\dot{\overline{\mathbf{x}}}_{cyc} + \mathbf{K}_{cyc}\,\overline{\mathbf{x}}_{cyc} = \overline{\mathbf{F}}_{cyc}. \tag{12}$$

2.2. The Craig–Bampton reduction technique

Model order reduction techniques are generally divided into three groups: modal reduction techniques, static reduction techniques and hybrid reduction techniques. The choice of the proper technique is mainly dependent on the specific application. However, the main goal of all reduction methods is the same, and it is to reduce the number of DOFs in the system. In model order reduction, the equation of motion is transformed into a new subspace, where the dimension n is much smaller than that of the original space N, i.e., $n \ll N$. The displacements of the reduced model are assumed to be a proper superposition of basis mode shapes. These basis mode shapes form the model reduction transformation matrix $\mathbf{T}$, and the weighting factors of basis coordinates are the reduced model displacements $\widetilde{\mathbf{x}}$. The full model displacements are related to reduced order displacements by the following relation:

$$\mathbf{x} = \mathbf{T}\widetilde{\mathbf{x}}. \tag{13}$$

The CB reduction technique, also called the fixed boundary method, is the most popular hybrid reduction technique. The CB basis consists of static modes and dynamic modes, where static reduction DOFs are fixed. The CB method requires to split all DOFs into boundary DOFs b and internal DOFs i. With such a DOFs distinction, the equation of motion is:

$$\begin{bmatrix} \mathbf{M}_{bb} & \mathbf{M}_{bi} \\ \mathbf{M}_{ib} & \mathbf{M}_{ii} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{u}}_b \\ \ddot{\mathbf{u}}_i \end{bmatrix} + \begin{bmatrix} \mathbf{K}_{bb} & \mathbf{K}_{bi} \\ \mathbf{K}_{ib} & \mathbf{K}_{ii} \end{bmatrix} \begin{bmatrix} \mathbf{x}_b \\ \mathbf{x}_i \end{bmatrix} = \begin{bmatrix} \mathbf{f}_b \\ \mathbf{0} \end{bmatrix}. \tag{14}$$

The physical displacements are transformed into CB displacement using the following transformation:

$$\begin{bmatrix} \mathbf{x}_b \\ \mathbf{x}_i \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{\Psi}_{CM,i} & \mathbf{\Phi}_i \end{bmatrix} \begin{bmatrix} \mathbf{x}_b \\ \boldsymbol{\eta}_i \end{bmatrix} = \mathbf{T}_{CB} \begin{bmatrix} \mathbf{x}_b \\ \boldsymbol{\eta}_i \end{bmatrix}, \tag{15}$$

where $\mathbf{\Psi}_{CM,i}$ is denoted as constraint modes (static modes) and $\mathbf{\Phi}_i$ is the fixed-interface vibration modes (dynamic modes).

Each constraint mode is computed by applying a unit displacement on a single boundary DOF and fixing all remaining DOFs to 0:

$$\begin{bmatrix} \mathbf{K}_{bb} & \mathbf{K}_{bi} \\ \mathbf{K}_{ib} & \mathbf{K}_{ii} \end{bmatrix} \begin{bmatrix} \mathbf{I}_{bb} \\ \mathbf{\Psi}_{ib} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{bb} \\ \mathbf{0}_{ib} \end{bmatrix}, \tag{16}$$

where $\mathbf{R}_{bb}$ are reaction forces. From Eq. (16) the constraint modes can be calculated as:

$$\mathbf{K}_{ib} I_{bb} + \mathbf{K}_{ii}\mathbf{\Psi}_{ib} = \mathbf{0}, \tag{17}$$

which can be transformed into a system of linear equations:

$$\mathbf{K}_{ii}\boldsymbol{\Psi}_{ib} = -\mathbf{K}_{ib}. \tag{18}$$

The dynamic modes – fixed-interface vibration modes – are calculated from the generalized eigenvalue problem, where the boundary DOFs are constrained ($\mathbf{x}_b = \mathbf{0}$):

$$(\mathbf{K}_{ii} - \boldsymbol{\omega}_{i,j}^2\mathbf{M}_{ii})\boldsymbol{\Phi}_{i,j} = \mathbf{0}. \tag{19}$$

The result is a set of eigenmodes which are constrained at all boundary DOFs. After computing both constraint modes and fixed-interface vibration modes, the equation of motion (Eq. (1)) can be reduced using $\mathbf{T}_{CB}$ similarly to Eq. (12):

$$\mathbf{M}_{CB}\,\ddot{\widetilde{\mathbf{x}}}_{CB} + \mathbf{C}_{CB}\,\dot{\widetilde{\mathbf{x}}}_{CB} + \mathbf{K}_{CB}\,\widetilde{\mathbf{x}}_{CB} = \widetilde{\mathbf{F}}_{CB}. \tag{20}$$

Here the CB reduced model matrices become:

$$\mathbf{M}_{CB} = \mathbf{T}_{CB}^H\mathbf{M}\mathbf{T}_{CB}, \tag{21}$$

$$\mathbf{K}_{CB} = \mathbf{T}_{CB}^H\mathbf{K}\mathbf{T}_{CB}, \tag{22}$$

$$\mathbf{C}_{CB} = \mathbf{T}_{CB}^H\mathbf{C}\mathbf{T}_{CB}. \tag{23}$$

From the given formulation of the CB reduction technique, it can be seen that the reduction process consists of three steps: constraint modes calculation, fixed-interface modes calculation and finally system matrix reduction. It has to be noted that if the solved system is cyclic then all matrices are complex-valued.

2.3. Modal assurance criterion as ROM verification

The accuracy of full model representation by the CB reduction technique greatly depends on the number of modes that form the CB subspace. In order to provide a quick assessment of the accuracy of the reduced model, the most handy criterion is the Modal Assurance Criterion (MAC) [11]. The MAC provides a metric for mode similarity. It is based on cosine similarity, with the difference that MAC values span from 0 to 1, where 1 indicates that modes are parallel and 0 means they are orthogonal. In reduced order verification MAC can be used to compare the natural modes of the full model against the neutral modes of the reduced model.

The modes of the full model (FM) are computed by solving the eigenvalue problem with system metrices:

$$(\mathbf{K}_{FM,j} - \boldsymbol{\omega}_{FM,j}^2\mathbf{M}_{FM,j})\boldsymbol{\Phi}_{FM,j} = \mathbf{0}. \tag{24}$$

On the other hand, the reduced order model's mode shapes are computed as:

$$(\mathbf{K}_{CB,j} - \boldsymbol{\omega}^2_{CB,j}\mathbf{M}_{CB,j})\boldsymbol{\Phi}_{CB,j} = \mathbf{0}. \tag{25}$$

Since the CB modes are defined in the reduced subspace, they have fewer DOFs than the full model modes. In order to define CB modes in the full model space, the CB modal reconstruction has to be used (Eq. (13))

$$\boldsymbol{\Phi}_{CB-FM,j} = \mathbf{T}_{CB}\,\boldsymbol{\Phi}_{CB,j}. \tag{26}$$

With the reconstructed CB modes, the MAC metric can be applied to each combination of the pair of full model vs. reduced mode:

$$\mathbf{MAC}_{(CB-FM,i),(FM,j)} = \frac{|\boldsymbol{\Phi}^H_{CB-FM,i}\boldsymbol{\Phi}_{FM,j}|^2}{(\boldsymbol{\Phi}^H_{CB-FM,i}\boldsymbol{\Phi}_{CB-FM,i})(\boldsymbol{\Phi}^H_{FM,j}\boldsymbol{\Phi}_{FM,j})}. \tag{27}$$

In the ROM verification process, MAC is used to observe if the full modes that are important for a given analysis are well represented in the reduced subspace, i.e., MAC values are close to 1.

3. Methodology

In Sec. 2, the complete process of model order reduction was presented. It can be inferred that there are several algebraic operations, which can be computationally expensive:

- cyclic reduction of system matrices (Eqs. (8)–(10)): sparse matrix multiplication, where the cyclic transformation matrix is complex-valued, and the system matrices are real-valued,
- constraint modes calculation (Eq. (18)): system of complex-valued linear equations,
- fixed-interface vibration modes (Eq. (19)): generalized eigenvalue problem of complex-valued matrices,
- CB reduction of system matrices (Eqs. (21)–(23)): complex-valued matrix multiplication, where the CB transformation matrix is complete, and system matrices are sparse,
- ROM check (Eq. (27)): multiple dot product of complex-valued vectors.

Since each algebraic operation is different, an initial ROM computation of a reference FEM model on the CPU code is conducted to obtain the percentage time breakdown of each item. In the further study, attention was paid only on the most time consuming part of model reduction.

3.1. Finite element model

The analyzed FEM is presented in Fig. 1. From the 12-sector wheel, a single cyclic sector was extracted. The cyclic sector is meshed with equally spaced brick elements. The model is rather academic, however for the current study the model is sufficient, since only computational performance is within the authors' interest. Table 1 includes four finite element models analyzed in the current work. The FEA models differ only by grid spacing, and in consequence, impact the number of DOFs. Also, each model has a different number of calculated constraint modes. This is due to the fact that on a coarse grid, fewer force/contact nodes can be fitted. So, the number of constraint modes has to be adjusted for each case. The authors decided to correlate the number of constraint modes with the grid spacing of each model.

Table 1. Sizes of FEA models used in the study, with the number of constraint modes.

	FEA nodes	FEA DOFs	CMs
1	16 321	50 058	42
2	50 058	145 152	114
3	127 909	374 292	150
4	324 625	957 264	390

3.2. Computational hardware

Table 2 shows details of the CPU and GPU architectures, which were used in the current work. In order to ensure as much objectivity in the performance comparison as possible, it was decided to compare the architectures based on the cost of the computational unit. At the time of writing this paper, the ratio of the cost of CPU to GPU architecture from Table 2 was about 4 : 3. Therefore, it was decided that for all CPU computations, the number of cores would be limited from 32 to 24, to equalize the cost of both architectures.

Table 2. Computational hardware.

Processing unit	Hardware specification
CPU hardware	AMD EPYC™ 7543, L3 memory 256MB, Clockspeed 2.8 GHz, 32 Cores, 64 Threads, 2 Sockets
GPU hardware	NVIDIA A40, 10752 Cores, Memory size 48 GB

3.3. CPU implementation

A highly optimized Fortran 2008 code was used as a reference CPU implementation. The code takes advantage of compiler optimizations, such as vectorization. For sparse algebra computation the Intel MKL library is utilized [16], which has procedures optimized for both shared-memory (OpenMP) parallelism and distributed memory parallelism (Intel MPI).

At first, the reference run of the CPU implementation was conducted. FEA model no. 4 from Table 1 was run using 24 OpenMP threads and 2 MPI tasks. This run was performed to obtain the time breakdown of all algebra operations listed in Sec. 3. Here, it has to be noted that the fixed-interface vibration modes computation was carried out outside CPU code, i.e., using ANSYS software, due to challenges in implementing a custom complex-valued generalized eigenvalue problem for sparse matrices. Thus, fixed-interface vibration modes computation is excluded from the performance comparison studies.

Figure 2 shows the percentage time split of each computation. It can be observed that the vast majority of the computational time is taken by constraint mode computation. All remaining algebra operations takes only 5% of the total time.

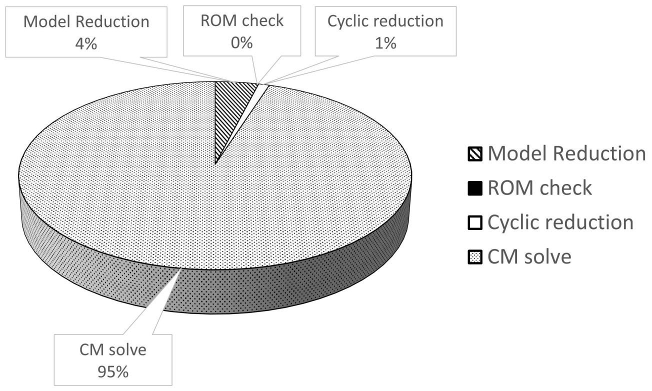

Fig. 2. CPU wall time breakdown for 957,264 DOFs FEA model (CPU solver).

Motivated by the runtime breakdown depicted in Fig. 2, the solution of the linear problem, constraint mode computation, becomes the quantitative focus in the further study of this paper. Other algebraic operations are omitted for now since the main goal of the study is to accelerate the overall time of the reduced order model building process.

3.4. GPU implementation

As mentioned in the previous section, the main attention is paid on the solution of the system of linear equations. Table 3 summarizes the linear solvers found in the literature that meet the requirements of the current work. The following libraries were used in this work:

- CuPy is an open-source library written in Python, intended for GPU acceleration of solving systems of linear equations with sparse matrices. Due to its support of NVIDIA CUDA platform, it allows to use NumPy and SciPy functionalities on GPU machines [5].
- Strumpack is a library written in C++, intended for delivering solvers and routines for solving linear systems of equations with sparse and dense matrices. It supports NVIDIA GPU machines with CUDA for sparse direct solvers [6, 7].
- Ginkgo is a library written in C++, intended for high performance linear algebra computing with special focus on the sparse linear systems. It supports GPU Machines through kernels implemented in CUDA, HIP and DPC++ [8–10].

The table also contains CPU solvers used in Subsec. 3.3. All the listed solvers can handle complex-valued sparse matrices.

Table 3. Compared linear solvers.

Technology	Processing unit	Type
Ginkgo parallel	GPU	iterative
Ginkgo sequential	GPU	iterative
CuPy gmres	GPU	iterative
CuPy cg	GPU	iterative
CuPy cgs	GPU	iterative
STRUMPACK 1	GPU	direct
STRUMPACK 2	GPU	direct
Intel MKL (1 core)	CPU	direct
Intel MKL (24 cores, 2 sockets) equivalent cost to NVIDIA A40	CPU	direct

The GPU implementation of model order reduction comprises Python code predominantly using the CuPy [5] library for the computational stages described in Sec. 2. Only in the case of the linear solve step other computational libraries, such as STRUMPACK [6] and Ginkgo [8], are used (see Table 3).

The implemented code reads from ANSYS FEA: sparse system matrices $\mathbf{M}$ and $\mathbf{K}$, normal modes (Eq. (24)), and fixed-interface modes (Eq. (19)). Also, high-edge and low-edge node numbers are read for cyclic constraints application. After the input stage, the data is kept on the GPU card memory, and only transferred to CPU memory if required by a specific library. Within the listed solvers in Table 3 there are both direct and iterative solvers. Each solver requires separate effort to integrate it with the GPU-based model order reduction Python code.

The conjugate gradient solver (CuPy cg) was diagonally preconditioned to reduce the number of iterations to converge, according to Listing 1. The remaining variants of the CuPy solver callbacks look virtually the same as Listing 1. Preconditioning is needed to keep the number of iterations relatively small. However, based on conducted tests, the use of seemingly more sophisticated preconditioners than the current diagonal deflation does not pay off computationally.

Listing 1. Python CuPy code impementing diagonally preconditioned conjugate gradient solution scheme.

```
from cupyx.scipy.sparse.linalg import cg, LinearOperator
from cupyx.scipy.sparse import dia_matrix
from tqdm import tqdm

def gpu_sam_solve_cg(K_sam, rhs, K_eps=1E-10):

        # Jacobi preconditioner
        dia = K_sam.diagonal()
        inv = 1.0/dia
        inv [dia == K_eps] = 0.0
        inv = dia_matrix((inv,[0]), shape = K_sam.shape)
        def matvec(v):
        return inv.dot(v)
        M = LinearOperator(inv.shape,matvec)

        # Solve for SAMs
        sam_modes = rhs.copy()
        for j in tqdm(range(rhs.shape[1]),desc = 'GPU␣cg'):
                x, info = cg(K_sam, rhs[:,j], M = M)
                sam_modes[:,j] = x
        return sam_modes
```

The CuPy library does support CPU-computed and GPU-applied sparse ILU-type preconditioning, which in the case of the current study was not proven to be effective. Hence, in all cases where the iterative solvers were compared (Cupy and Ginkgo), the same Jacobi-type diagonal deflation is used to precondition the iterations.

The Ginkgo solver is linked with the core model order reduction Python code via a custom C++ based Python module. The same technique is applied to provide the interface for the STRUMPACK solver. While the employed Ginkgo (conjugate gradient) solver is purely iterative and after all data is passed to it, the calculation is solely GPU-based, the STRUMPACK solver itself is a direct solver pre-computing the sparse symbolic elimination tree on the CPU and then traverses it, factorizing in parallel on the GPU unit. Hence, the STRUMPACK solver includes relatively complex data movement between the CPU and GPU sides of memory and computation.

In Table 3, the used libraries provide the following algorithms for solving system of linear equations:

- CG (Conjugate Gradient) method is an iterative algorithm for the solution of systems of linear equations if the system matrix is positive-definite. It is used for large sparse systems, because the algorithm needs only a formula for multiplying a sparse matrix by a given vector [12, 14, 17].
- CGS (Conjugate Gradient Squared) method is an improved CG algorithm with the same computational cost but better convergence [12, 13, 15].
- GMRES (Generalized Minimal RESidual) method is an iterative method for the solution of a system of linear equations with a nonsymmetric matrix. Using the Arnoldi iteration, this algorithm approximates the solution vector by a vector in the Krylov subspace with minimal residual [18, 19].

4. Results

4.1. Computational time

Figure 3 summarizes the computational runtimes of the linear solve stage for all tested solver types and all input problem sizes. It can be observed that among the CuPy-based solvers, the baseline conjugate gradient approach (cg) is the most effective one, while its squared version (cgs) and the gmres are slower. The Ginkgo versions of the conjugate gradient solver are consistently outperform-

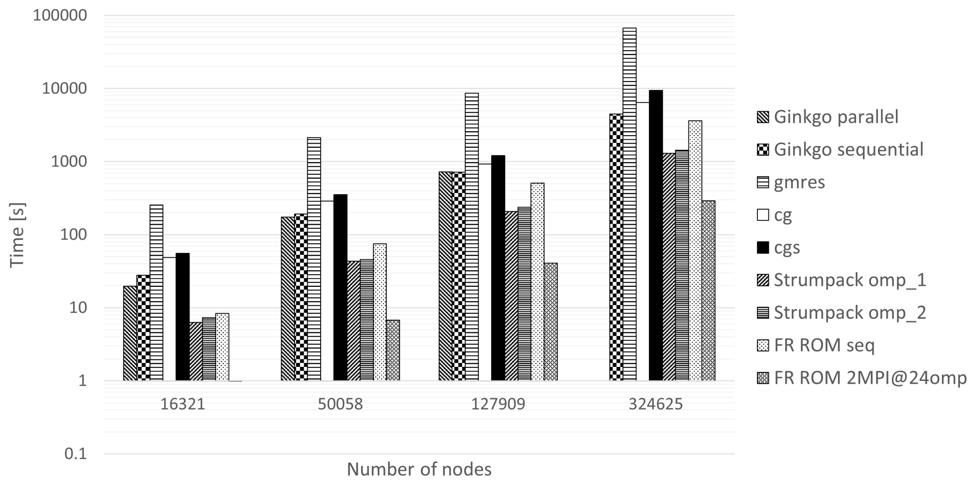

Fig. 3. Linear solver with multiple right-hand sides: computational time.

ing CuPy's cg by a small percentage, even though the invocation of the Ginkgo module involves the transfer of input and output algebra between the CPU and GPU memory. This points to the presumably more optimized nature of the implementation of this aspect of Ginkgo over CuPy's equivalent. STRUMPACK, on the other hand, consistently outperforms both iterative approaches. This is interesting and encouraging. Direct solvers, due to their greater numerical stability and simplicity of the application, are preferably used. Finally, the CPU implementation, employing Intel's MKL sparse solver (PARADISO), outperforms all of the GPU approaches given 24 CPU cores. This is perhaps "as expected", since this particular solver has been highly optimized over several decades. Nonetheless, the current study shows that the GPU solvers provide practical runtimes, in some cases comparable with CPU solver.

4.2. RAM memory consumption

RAM memory consumption is crucial for high-fidelity FEA models, since it can easily become a limitation on the size of FEA model that can be solved on a given hardware. Even though GPU solvers are mainly based on GPU processing power and memory, they would still use CPU memory resources. The CPU and the GPU RAM memory consumption results are summarized in Figs. 4 and 5, respectively. An interesting observation, based on these figures, is that STRUMPACK does not seem to take up significantly more memory that the iterative solvers. In the course of running the tests, it was noticed that STRUMPACK is able to adapt dynamically to the available GPU memory size and if the memory threshold is about to be reached during the numerical factorization stage on the GPU, a recursive subdivision strategy is applied. The large

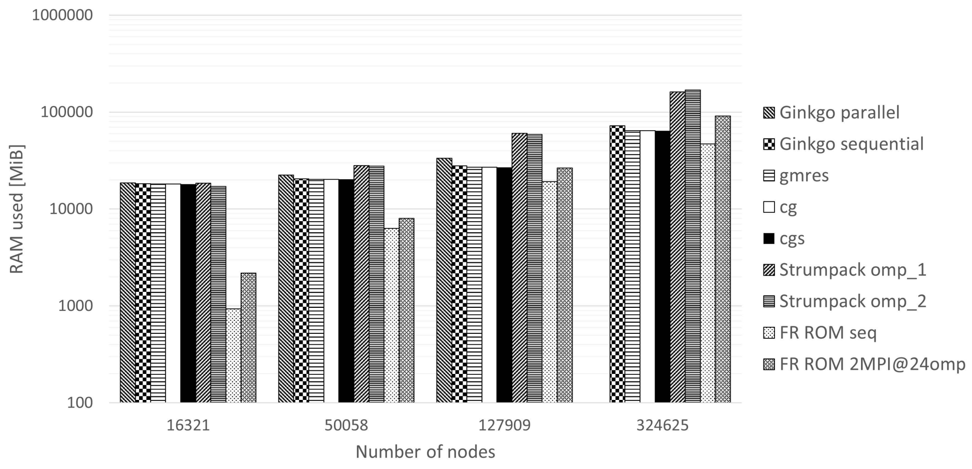

Fig. 4. Linear solver with multiple right-hand sides: consumed RAM memory on CPU.

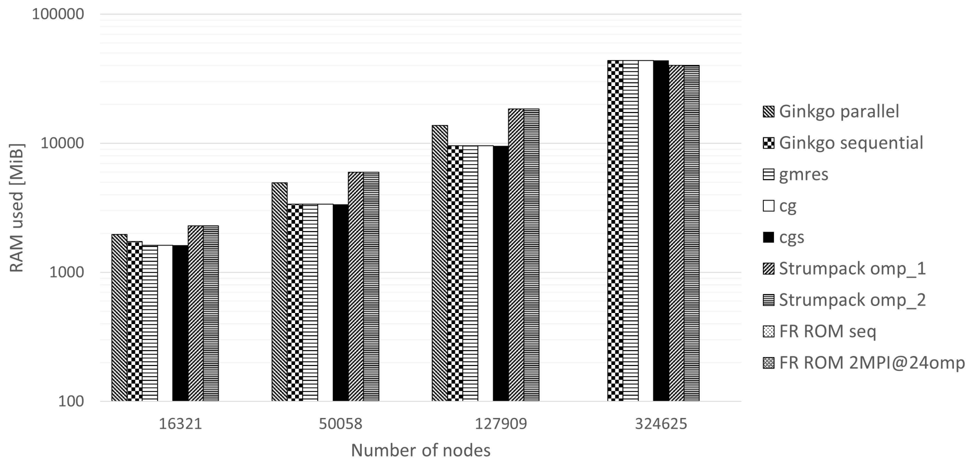

FIG. 5. Linear solver with multiple right-hand sides: consumed RAM memory on GPU.

memory consumption on the CPU for smaller models seems to be an artifact possibly due to the applied method of querying the operating system for the amount of memory used by a process. However, the authors were not able to clarify this.

5. Conclusions

In the analyzed CB reduction approach, the most time-consuming aspect was revealed to be the solution of the system of linear equations with multiple right-hand sides, i.e., constraint mode computation. Because of that, the effort to accelerate computation of reduced order model of a structural system using GPU programming was focused only on accelerating the solution of the system of linear equation. Several algebraic software libraries were selected to study their performance. From the results it can be concluded that currently there is no GPU solver as efficient as the CPU-based MKL PARDISO solver (at least up to 1M DOFs). It was also shown that Python CuPy and NumPy libraries require a substantial memory overhead for linear solvers (seemed to be a portion of the total memory). In future studies, the authors plan to focus on testing much larger models up to about 1B DOFs, with distributed multi-GPU architecture and low-level coding.

This was observed for the given case, e.g., a simplified blisk model. This finite element structural model was cyclic reduced what gave considered system of linear equation. While it cannot be stated that this observation is valid for other types of analysis, the investigation should be repeated and new conclusions should be drawn. However, the presented type of models is a very important part of CAE analysis in turbomachinery dynamics, and making is a wide and important area of interests for engineers and researchers.

Acknowledgements

The project was co-financed by the European Union within the European Regional Development Fund Smart Growth 2014–2020 Operational Programme Grant Agreement No. POIR.01.01.01-00-0479/20 "Development of process innovation in research and monitoring the life cycle of the aircraft engine components".

References

1. K. Lu *et al.*, A review of model order reduction methods for large-scale structure systems, *Shock and Vibration*, **2021**: 6631180, 2021, doi: 10.1155/2021/6631180.
2. R.R. Craig, Coupling of substructures for dynamic analyses: an overview, [in:] *Proceeding 41st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Atlanta, USA, AIAA-2000-1573, 2000, doi: 10.2514/6.2000-1573.
3. R.R. Craig, A.J. Kurdila, *Fundamentals of Structural Dynamics*, Wiley, New York, 2006.
4. D. de Klerk, D.J. Rixen, S.N. Voormeeren, General framework for dynamic substructuring: history, review, and classification of techniques, *AIAA Journal*, **46**(5): 1169–1181, 2008, doi: 10.2514/1.33274.
5. R. Okuta, Y. Unno, D. Nishino, S. Hido, C. Loomis, CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations, [in:] *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA, 2017, https://cupy.dev/.
6. P. Ghysels, R. Synk, High performance sparse multifrontal solvers on modern GPUs, *Parallel Computing*, **110**: 102897, 2022, doi: 10.1016/j.parco.2022.102897, https://github.com/pghysels/STRUMPACK.
7. A. Abdelfattah, P. Ghysels, W. Boukaram, S. Tomov, X. Li, J. Dongarra, Addressing irregular patterns of matrix computations on GPUs and their impact on applications powered by sparse direct solvers, [in:] *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, Dallas, TX, USA, pp. 1–14, 2022, doi: 10.1109/SC41404.2022.00031.
8. T. Cojean, Y.H. Tsai, H. Anzt, Ginkgo – A math library designed for platform portability, *Parallel Computing*, **111**: 102902, 2022, doi: 10.1016/j.parco.2022.102902, https://ginkgo-project.github.io/.
9. H. Anzt *et al.*, Ginkgo: A modern linear operator algebra framework for high performance computing, *ACM Transactions on Mathematical Software*, **48**(1): 2, pp. 1–33, 2022, doi: 10.1145/3480935.
10. H. Antz *et al.*, Ginkgo: A high performance numerical linear algebra library, *Journal of Open Source Software*, **5**(52): 2260, 2020, doi: 10.21105/joss.02260.

11. R.J. Allemang, The modal assurance criterion – Twenty years of use and abuse, *Sound and Vibration Magazine*, **August**: 14–20, 2003, http://www.sandv.com/downloads/0308alle.pdf.
12. Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, 2003.
13. D.R. Fokkema, G.L.G. Sleijpen, H.A. Van der Vorst, Generalized conjugate gradient squared, *Journal of Computational and Applied Mathematics*, **71**(1): 125–146, 1996, doi: 10.1016/0377-0427(95)00227-8.
14. D. Braess, *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*, Cambridge University Press, 2007, doi: 10.1017/CBO9780511618635.
15. A. Greenbaum, L.N. Trefethen, *GMRES/CR and Arnoldi/Lanczos as matrix approximation problems*, *SIAM Journal on Scientific Computing*, **15**(2): 359–368, 1994, doi: 10.1137/0915025.
16. A. Kalinkin, A. Anders, R. Anders, Intel® Math Kernel Library PARDISO* for Intel® Xeon Phi™ Manycore Coprocessor, *Applied Mathematics*, **6**(8): 1276–1281, 2015, doi: 10.4236/am.2015.68121.
17. M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, *Journal of Research of the National Bureau of Standards*, **49**(6): 409–436, 1952, doi: 10.6028/jres.049.044.
18. S.C. Eisenstat, H.C. Elman, M.H. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, *SIAM Journal on Numerical Analysis*, **20**(2): 345–357, 1983, doi: 10.1137/0720023.
19. Y. Saad, M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM Journal on Scientific and Statistical Computing*, **7**(3): 856–869, 1986. doi: 10.1137/0907058.

Received December 23, 2022; revised version December 12, 2023; accepted January 8, 2024; published online February 28, 2024.