Recursive Singular Value Decomposition compression of refined isogeometric analysis matrices as a tool to speedup iterative solvers performance.

Mateusz Dobija[1][0000−0003−4557−3534] and Anna Paszynska[1][0000−0001−7766−6052]

Faculty of Physics, Astronomy and Applied Computer Science,
Jagiellonian University, Poland
anna.paszynska@uj.edu.pl

Abstract. The isogeometric analysis (IGA) uses higher-order and continuity basis functions as compared to the traditional finite element method. IGA has many applications in simulations of time-dependent problems. These simulations are often performed using an explicit time-integration scheme, which requires the solution of a system of linear equations with the mass matrix, constructed with high-order and continuity basis functions. The iterative solvers are most commonly applied for large problems simulated over complex geometry. This paper focuses on recursive decomposition of the mass matrix using the Singular Value Decomposition algorithm (SVD). We build a recursive tree, where submatrices are expressed as multi-columns multiplied by multi-rows. When we keep the mass matrix compressed in such a way, the multiplication of a matrix by a vector, as performed by an iterative solver, can be performed in $O(Nr)$ instead of $O(N^2)$ computational cost, where N is the number of rows of input matrix, r is the number of singular values bigger than given value. Next, we focus on refined isogeometric analysis (rIGA). We introduce the C0 separators into IGA submatrices and analyze the SVD recursive compression and computational cost of an iterative solver when increasing the patch size and the order of B-spline basis functions

Keywords: refined isogeometric analysis · hierarchically compressed matrix · matrix-vector multiplication · iterative solvers.

1 Introduction

Isogeometric analysis (IGA) [1–3] is a generalization of the traditional finite element method into higher-order and continuity basis functions. It has many applications in the simulation of time-dependent problems, from wind turbine simulations [4], drug transport [5], to tumor growth simulations [6, 7]. The refined isogeometric analysis [8–10] has been proposed to reduce the computational cost of IGA while keeping the high accuracy of IGA solutions. The time-dependent IGA solvers often rely on explicit dynamics formulations [6, 7, 11]. In the explicit

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

dynamics solvers, the system of linear equations with mass matrix constructed from IGA basis functions is solved in every time step. For large problems, iterative solvers are often employed [12]. In this paper, we discuss an algorithm for recursive compression of matrices. As an example, we consider the IGA mass matrix as applied in time-dependent simulations. Our algorithm employs a low-rank approximation of blocks, as performed by the Singular Value Decomposition algorithm [13]. This method follows the idea of hierarchical matrix compression as proposed by Hackbush [14]. The generation of isogeometric analysis mass matrices speeds up with low-rank approximation has been discussed in [15]. In our paper, we focus on the application of compression to speed up the iterative solver. We consider the refined isogeometric analysis and discuss the influence on the computational patch's different orders of approximations and dimensions.

2 Matrices of refined isogeometric analysis

B-spline functions are commonly used in computer design and simulations thanks to the growing popularity of so-called isogeometric analysis popularized by prof. T. J. R. Hughes [1]. The formulas for B-splines are defined using Cox-de-Boor rule [3] in the following way:

$$B_{i,0}(\xi) = 1 \text{ for } \xi_i \leq \xi \leq \xi_{i+1} \text{ and } 0 \text{ in the other case}$$
$$B_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} B_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} B_{i+1,p-1}(\xi) \tag{1}$$

Both linear B-splines and higher order B-splines can be described using the notation of the so-called knot vector. A knot vector will be a sequence of non-decreasing coordinates of points. Let's, for simplicity, assume points with integer coordinates. The degree of the basis functions is equal to the number of times the first (or last) point in the knot vector is repeated, minus one. In exemplary knot vector [0 0 0 0 1 2 3 4 5 5 5 6 7 8 9 10 10 10 10] the first and last points are repeated four times (0 0 0 0 and 10 10 10 10), so we introduce cubic basis functions of second continuity. There is a repeated knot in the vector (5 5 5), and the repetition of knot reduces the continuity at the point, so we have inserted the C0 separator at the center. In rIGA the C0 separators are inserted every l intervals. The two-dimensional B-splines are created by the tensor product of one-dimensional B-splines [3]. For example, the tensor product of the knot vectors [0 0 0 0 1 2 3 4 5 5 5 6 7 8 9 10 10 10 10] and [0 0 0 0 1 2 3 4 5 5 5 6 7 8 9 10 10 10 10] describes B-splines presented in Fig. 1, merging together four patches of 5x5 elements with cubic B-splines. The mass matrix is defined as the multiplication of two two-dimensional B-splines. The first B-splines are called trials and are used for the approximation of the solution. The former are called tests and are employed to generate different equations within the system (and for local approximation of the equation in the strong form).

$$M_{ij,k,l;p} = \int_\Omega B_{ij;p}(x,y) B_{kl;p}(x,y) dxdy = \int_\Omega B_{i;p}(x) B_{j;p}(y) B_{k;p}(x) B_{l;p}(y) dxdy \tag{2}$$

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

The mass matrix can be factorized in a linear cost only if the computational domain has a regular tensor product form. For the simulations on non-regular geometries in 2D, the direct solvers deliver $O(N^{1.5}p^2)$ computational cost, and the iterative solvers deliver $O(Nk)$ computational cost. Here N is the number of B-splines, p is the order of B-splines, and k is the number of iterations (depending on the geometry of the domain and the problem solved). The matrix-vector computations can be performed without forming a global matrix when we focus on refined isogeometric analysis and iterative solvers. This is called a matrix-free iterative solver algorithm [16]. The solution vector is obtained by assembling local elemental matrices multiplied by local portions of the right-hand side. In the context of refined isogeometric analysis, this can be generalized to patches of elements. In this paper, we focus on different patches of elements and different orders of B-splines, and we perform recursive SVD compression of the resulting mass matrix. Later, we compare the gain in terms of the number of floating-point operations when performing matrix-vector multiplication over the patch of elements, using the patches of rIGA.

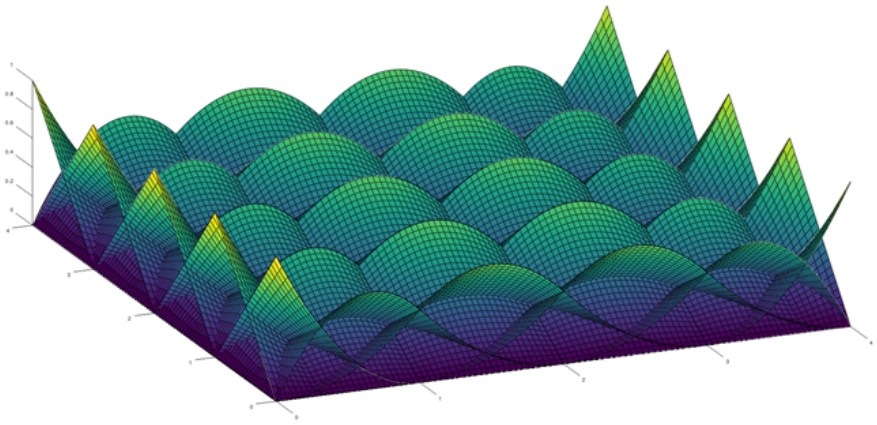

Fig. 1. Two dimensional B-splines described as tensor product of two one-dimensional B-splines defined by knot vector [0 0 0 0 1 2 3 4 5 5 5 6 7 8 9 10 10 10 10].

3 Compression algorithm

The main idea of the compression algorithm is the recursive dividing of the matrix into four smaller submatrices and performing the approximate singular value decomposition algorithm (SVD) for selected submatrices. The approximate SVD decomposes any matrix A into A=UDV, where D is s diagonal matrix with singular values sorted in a decreasing manner, U is the matrix of columns, an V is the matrix of rows. In the approximate SVD, we remove from D singular values

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

smaller than prescribed epsilon. Thus, we can only keep r columns of U and r rows of V, where r is the number of singular values left. Additionally, in order to save the memory, the coefficients of the calculated matrix V are multiplied by corresponding singular values of the diagonal matrix D. In other words, instead of storing UDV we store UV' where V' is DV. The gain from the SVD decomposition is that it enables matrix-vector multiplication in a computational cost of $O(Nr)$ instead of $O(N^2)$, if we perform the multiplications in order U*(V'*B) instead of A*B or (U*V')*B, see Fig. 2. For simplicity we call these matrices U and V (remembering that V is multiplied by D).

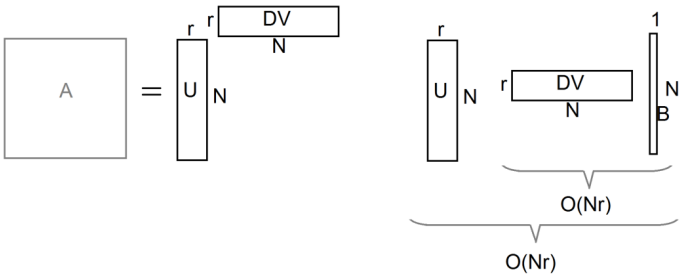

Fig. 2. The approximate SVD of matrix A, where we save r singular values, so we have N columns and N rows. The matrix-vector multiplication UDVB for the SVD compressed matrix A=UDV results in $O(Nr)$ computational cost.

The whole compression algorithm of a matrix can be seen as the recursive dividing of the matrix into four smaller submatrices and can be described in the following steps. If the submatrix consists of zeros, we remember its rank (zero) and its number of rows. If the submatrix has some nonzero values, the SVD algorithm is used. If the number of singular values found by SVD, which are bigger than desired epsilon, is zero, we also remember only the submatrix rank equal to zero and the number of its rows. If the number of singular values found by SVD, which are bigger than desired epsilon, is nonzero (denote this value by k), and additionally, (k <= r and k < numRows/2) or k == 1, where numRows is the number of rows in the matrix and r is the arbitrary boundary for rank, than we remember only the first k rows of matrix V and the first k column of matrix U found by the SVD algorithms. The algorithm, during recursively dividing of the matrix, creates the tree. Each node of the tree can be a leaf or it can have four sons representing four submatrices. In each node, some attributes are stored, like the rank of the corresponding matrix, the number of rows, or vector U and V found by the SVD algorithm. The input of the algorithm is the matrix A in a sparse form, the parameter epsilon, and the boundary rank value r. The algorithm of compression of the matrix is presented in Algorithm 1.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

Algorithm 1 Compress matrix(A, epsilon,r)

```
Create new node v
Divide Matrix A into 4 submatrices (quarters): A1, A2, A3, A4
for each submatrix B = A1, A2, A3, A4 do
  if number on nonzero elements in B is equal to zero then
    Create new node w
    w.rowsWithZero = numofRows of matrix B
    w.rank = 0
    append child (v, w)
  else
    //perform SVD for submatrix B
    [U, D, V] = svds(B)
    eigenvalues = diag(D)
    k =number of singular values bigger than epsilon
    if k == 0 then
      Create new node w
      w.rowsWithZero = numofRows of matrix B
      w.rank = 0
      append child (v, w)
    else if (k <= r) AND (k < (numRows/2) OR (k == 1)) then
      Create new node w
      w.rank = k
      w.Ucolumns = U(:, 1 : k)
      w.Vrows = V'(1 : k, :)
      for i = 1 : k do
        w.Vrows(i, :) = w.Vrows(i, :) * node.eigenvalues(i)
      end for
      append child (v, w)
    else
      w =Compress Matrix(B, epsilon, r)
      append child (v, w)
    end if
  end if
end for
```

4 Multiplication algorithm

The input for the algorithm for multiplication of the compressed matrix by a vector is the tree representing compressed matrix (the root node of the tree or its subtree) and the vector. The idea of the algorithm is the following. If the input node has no children, it means, that it represents a block of zeros or a block remembered as matrices $Ucolumns$ and $Vrows$ found by the SVD algorithm. In the first case, the resulting vector is a vector of zeros. In the second case, the result is the result of multiplication $node.Ucolumns * (node.Vrows * x)$. It must be underlined that the order of performing multiplication is critical, because it influences the computational cost of obtaining the results. In the last case, the input node has children. In this case, the partial multiplication for each

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

submatrix (each child of the input node) by the corresponding part of the input vector has to be performed by the recursive call of the algorithm, followed by calculating the final result of the multiplication.

Algorithm 2 MultiplyMatrixByVector(node, x)

```
if node.noofchildren == 0 then
  if node.rank == 0 then
    result =vector consisting of node.rowsWithZero zeros
  else
    result = node.Ucolumns * (node.Vrows * x)
  end if
else
  numRows =number of rows of vector x
  x1 = v(1 : floor(numRows/2), :)//first part of vector x
  x2 = v(floor(numRows/2 + 1) : numRows, :)//second part of vector x
  //calculate the partial multiplication for each submatrix
  res1 = MultiplyMatrixByVector(node.children(1), x1)
  res2 = MultiplyMatrixByVector(node.children(2), x2)
  res3 = MultiplyMatrixByVector(node.children(3), x1)
  res4 = MultiplyMatrixByVector(node.children(4), x2)
  //calculate the final result of multiplication
  if res1 consist of zeros then
    res1res2 = res2
  else if res2 consist of zeros then
    res1res2 = res1
  else
    res1res2 = res1 + res2
  end if
  if res3 consist of zeros then
    res3res4 = res4
  else if res4 consist of zeros then
    res3res4 = res2
  else
    res3res4 = res3 + res4
  end if
  result = [res1res2; res3res4]
end if
```

The main idea of the recursive multiplication algorithm is the following. Let consider the compressed matrix consisting of four blocks denoted by C, D, E, and F, where each block is compressed by the SVD algorithm into matrices $C1$ and $C2$, $D1$ and $D2$, $E1$ and $E2$, $F1$ and $F2$, respectively, as presented in Fig. 3. The result of multiplication of this matrix by vector X ($X = [X1, X2]$) is a vector: $[C1 * (C2 * X1) + D1 * (D2 * X2), E1 * (E2 * X1) + F1 * (F2 * X2)]$. Let assume that the matrices $C2$, $D2$, $E2$, $F2$ are of size $r * N$ and the matrices $C1$, $D1$, $E1$, $F1$ are of size $N * r$. The vectors $X1$ and $X2$ have size $N * 1$. The cost

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

of performing multiplication of our input matrix by vector can be summarized step by step, as follows:

- The cost of multiplication C2*X1 is N*r, and the result has size r*1,
- The cost of multiplication C1*(C2*X1) is N*r*r, and the result has size N*1,
- The cost of multiplication D2*X2 is N*r, and the result has size r*1,
- The cost of multiplication D1*(D2*X2) is N*r*r, the result has size N*1,
- The cost of summing the results is N*1,
- The cost of multiplication E2*X1 is N*r, and the result has size r*1,
- The cost of multiplication E1*(E2*X1) is N*r*r, and the result has size N*1,
- The cost of multiplication F2*X2 is N*r, and the result has size r*1,
- The cost of multiplication F1*(F2*X1) is N*r*r, and the result has size N*1,
- The cost of summing the results is N*1.

Summing up, the cost of the multiplication our input compressed matrix by a vector is $N * r$ in the contrary to the classic multiplication of matrix by vector with the cost N^2.

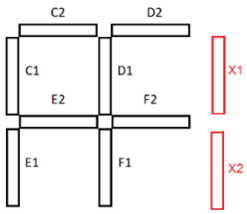

Fig. 3. Matrix-vector multiplication for the matrix recursively decomposed into four sub-matrices. The multiplications are performed $C1 * (C2 * X1) + D1 * (D2 * X2)$ and $E1 * (E2 * X1) + F1 * (F2 * X2)$ so the resulting computational cost is $O(Nr)$, where r is the number of rows and columns in the compressed matrices.

5 Results

The tests were performed comparing the number of floating point operations and accuracy for matrix by vector multiplication, for matrix compressed by the algorithm, and sparse matrix. The input matrix was the mass matrix in the compressed or sparse form. The tested mass matrices are generated for a different number of intervals, equal to 1, 2, 4, 8, 16, and 32, corresponding to different dimensions of patches in rIGA, and for different polynomial orders of approximation, equal 2, 3, 4 and 5. The compression algorithm was tested for epsilon equal to 0.00001, 0.0001, 0.001, 0.01 and 0.1 and r equal to 1, 2, 3, 4, 5, 6. For big values of *epsilon* (*epsilon* equal to 0.1) almost all singular values were smaller than epsilon and were omitted during the compression process thus the results of the multiplication of the compressed matrix by vector have low accuracy.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

Obtained results for r equal to 1 and 6 and small *epsilon* values (0.00001) are summarized in this section. Fig. 4 presents the number of floating point operations performed in the matrix by vector multiplication algorithm for compressed matrix and matrix in sparse form. The mass matrix was generated for quintic B-splines ($p = 5$), and the number of intervals was equal to 1, 2, 4, 8, 16, 32. The compression was performed for epsilon equal to 0.00001 and r=1 (blue line) and r=6 (grey line). The maximal error in coefficient in obtained vector (the result of matrix by vector multiplication) is presented in Table 1 and Table 2. The presented results show that for epsilon=0.00001 and a smaller number of intervals (1,2,4) the compression performed for r=1 gives bigger number of floating point operations for matrix by vector multiplication than classic approach. For r=1 and number of intervals equal to 8 the number of floating point operations is almost the same, for 16 and 32 intervals, the number of floating point operations for multiplication of a compressed matrix by vector is smaller than for classic multiplication for sparse matrix. For r=6 and epsilon=0.00001, the number of floating point operations for compressed matrix by vector multiplication algorithm is up to five times smaller than the number of floating point operations performed by a sparse matrix by vector multiplication algorithm. The accuracy of a vector obtained by the compressed matrix by vector multiplication algorithm is two orders of magnitude better for r=1 than for r=6. However, the accuracy obtained for r=6 is also satisfactory. In Fig. 5, the compressed mass matrices for quintic B-splines (p=5) and the number of intervals equal to 2 (first and second panel) and 32 (third and fourth panel) are shown. The compression was performed for epsilon equal to 0.00001, r=1 (first and third panel) and r=6 (second and fourth). The white color in the matrix denotes blocks of zeros. For the case of performing the SVD algorithm for block and remembering only k-rows and k-columns, the block is represented as k rows and k columns denoted by black color and white color in other places of the block. It can be seen that for r=1 the mass matrices were less compressed. For r=6 the matrices were compressed into four big blocks, for the case of the mass matrix for 2 and 32 intervals.

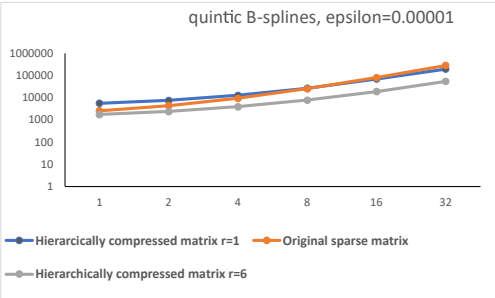

Fig. 4. The number of floating point operations in matrix by vector multiplication. The mass matrix generated for quintic B-splines, and the number of intervals equal to 1, 2, 4, 8, 16, 32. The compression performed for epsilon=0.00001, r=1 and r=6.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

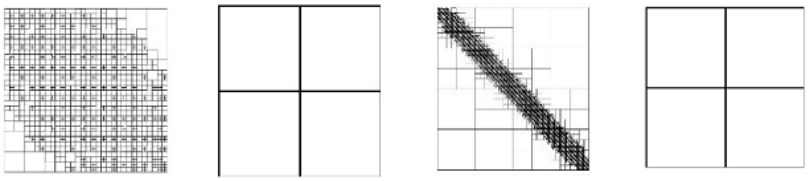

Fig. 5. The compressed mass matrix for quintic B-splines (p=5) and the number of intervals equal to 2 (first and second panel) and 32 (third and fourth panel). The compression performed for epsilon equal to 0.00001, r=1 (first and third panel) and r=6 (second and fourth panel).

Table 1. The number of elements in a patch in a single direction, the epsilon used in the approximate svd compression, the maximal number of compressed rows and columns r=1, the error of the approximate matrix-vector multiplication, the number of floating point operations for the approximate matrix - vector multiplication, the number of floating point operations for original sparse matrix-vector multiplication. The compressed mass matrix was generated for quintic B-splines.

number of intervals	epsilon	r	max difference	flops	orginal flops
1	0.00001	1	7,04E-06	5674	2592
2	0.00001	1	1,49E-05	7657	4418
4	0.00001	1	2,26E-05	12971	9522
8	0.00001	1	2,41E-05	26826	25538
16	0.00001	1	2,34E-05	70412	80802
32	0.00001	1	3,01E-05	196064	284258

Table 2. The number of elements in a patch in a single direction, the epsilon used in the approximate svd compression, the maximal number of compressed rows and columns r=6, the error of the approximate matrix-vector multiplication, the number of floating point operations for the approximate matrix - vector multiplication, the number of floating point operations for original sparse matrix-vector multiplication. The compressed mass matrix was generated for quintic B-splines.

number of intervals	epsilon	r	max difference	flops	orginal flops
1	0.00001	6	9.198856941664627e-04	1764	2592
2	0.00001	6	0.004692543537821	2401	4418
4	0.00001	6	0.006970278448885	3969	9522
8	0.00001	6	0.008889747812938	7921	25538
16	0.00001	6	0.004687370741346	18945	80802
32	0.00001	6	0.001832486	55201	284258

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

Figure 6 presents the number of floating point operations performed in the matrix by vector multiplication algorithm for compressed matrix and matrix in sparse form. The mass matrix was generated for quartic B-splines (p=4), and the number of intervals was equal to 1, 2, 4, 8, 16, 32. The compression was performed for epsilon equal to 0.00001 and r=1 (blue line) and r=6 (grey line). In Fig. 7, the compressed mass matrices for quartic B-splines (p=4) and the number of intervals equal to 2 (first and second panel) and 32 (third and fourth panel) are shown. The compression was performed for epsilon equal to 0.00001, r=1 (first and third panel) and r=6 (second and fourth).

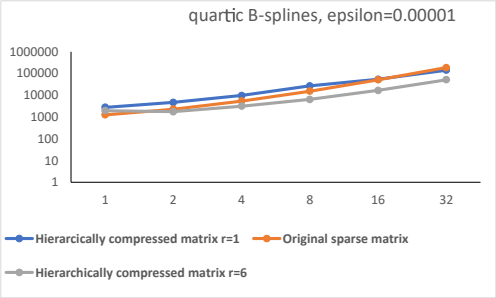

Fig. 6. The number of floating point operations in matrix by vector multiplication. The mass matrix generated for quartic B-splines, and the number of intervals equal to 1, 2, 4, 8, 16, 32. The compression performed for epsilon=0.00001, r=1 and r=6.

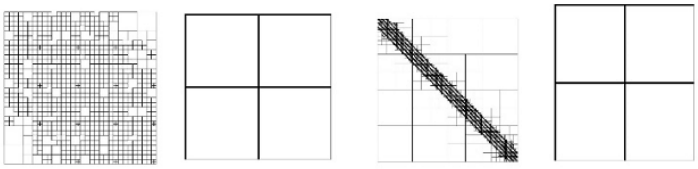

Fig. 7. The compressed mass matrix for quartic B-splines (p=4) and the number of intervals equal to 2 (first and second panel) and 32 (third and fourth panel). The compression performed for epsilon equal to 0.00001, r=1 (first and third panel) and r=6 (second and fourth panel).

Figure 8 presents the number of floating point operations performed in the matrix by vector multiplication algorithm for compressed matrix and matrix in sparse form. The mass matrix was generated for cubic B-splines (p=3), and the number of intervals was equal to 1, 2, 4, 8, 16, 32. The compression was performed for epsilon equal to 0.00001 and r=1 (blue line) and r=6 (grey line). The obtained

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

results show that for the mass matrix generated for one interval or two intervals and cubic B-splines, the number of floating point operations for matrix by vector multiplication algorithm for matrix in compressed form for r equal to 1 and 6 is bigger than for the matrix in sparse form. For bigger number of intervals, for r=6 and epsilon=0.00001 the number of floating point operations for compressed matrix by vector multiplication algorithm is smaller than the number of floating point operations performed by sparse matrix by vector multiplication algorithm. In Fig. 9, the compressed mass matrices for cubic B-splines (p=3) and the number of intervals equal to 2 (first and second panel) and 32 (third and fourth panel) are shown. The compression was performed for epsilon equal to 0.00001, r=1 (first and third panel) and r=6 (second and fourth).

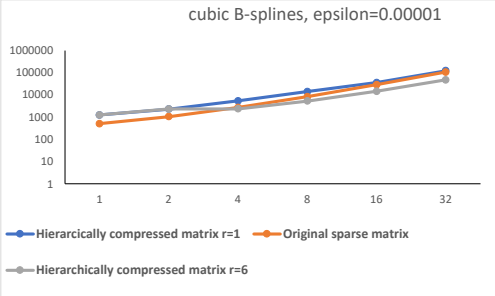

Fig. 8. The number of floating point operations in matrix by vector multiplication. The mass matrix generated for cubic B-splines, and the number of intervals equal to 1, 2, 4, 8, 16, 32. The compression performed for epsilon=0.00001, r=1 and r=6.

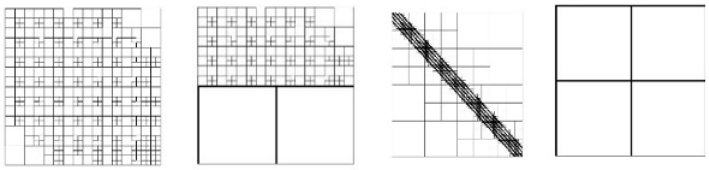

Fig. 9. The compressed mass matrix for cubic B-splines (p=3) and the number of intervals equal to 2 (first and second panel) and 32 (third and fourth panel). The compression performed for epsilon equal to 0.00001, r=1 (first and third panel) and r=6 (second and fourth panel).

Figure 10 presents the number of floating point operations performed in the matrix by vector multiplication algorithm for compressed matrix and matrix in sparse form. The mass matrix was generated for qadratic B-splines (p=2), and the number of intervals was equal to 1, 2, 4, 8, 16, 32. The compression was

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

performed for epsilon equal to 0.00001 and r=1 (blue line) and r=6 (grey line). The obtained results show that for the mass matrix generated for one interval or two intervals and quadratic B-splines, the number of floating point operations for matrix by vector multiplication algorithm for matrix in compressed form for r equal to 1 and 6 is bigger than for the matrix in sparse form. For bigger number of intervals, for r=6 and epsilon=0.00001 the number of floating point operations for compressed matrix by vector multiplication algorithm is slightly smaller than the number of floating point operations performed by sparse matrix by vector multiplication algorithm. It can be seen that for r=1 the mass matrices for 2 and 32 intervals as well as mass matrix for 2 intervals for r=6 were less compressed. For r=6 the mass matrix generated for 32 intervals was compressed into four big blocks. In Fig. 11, the compressed mass matrices for quadratic B-splines (p=2) and the number of intervals equal to 2 (first and second panel) and 32 (third and fourth panel) are shown. The compression was performed for epsilon equal to 0.00001, r=1 (first and third panel) and r=6 (second and fourth). Similar results, with better compression but slightly lower accuracy were obtained for compression with epsilon equal to 0.0001.

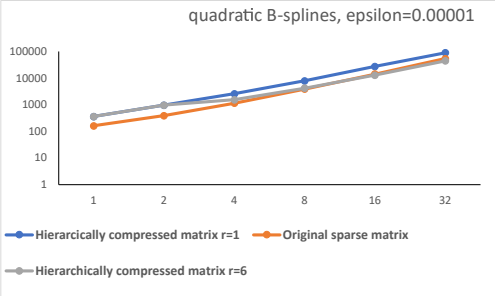

Fig. 10. The number of floating point operations in matrix by vector multiplication. The mass matrix generated for quadratic B-splines, and the number of intervals equal to 1, 2, 4, 8, 16, 32. The compression performed for epsilon=0.00001, r=1 and r=6.

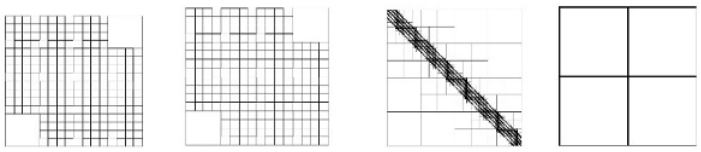

Fig. 11. The compressed mass matrix for quadratic B-splines (p=2) and the number of intervals equal to 2 (first and second panel) and 32 (third and fourth panel). The compression performed for epsilon equal to 0.00001, r=1 (first and third panel) and r=6 (second and fourth panel).

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

6 Summary of the results

The performed tests show that recursive compression of the matrix by SVD algorithm can speed up the process of matrix-vector multiplication. The best results were obtained for r=6 and epsilon=0.00001 for p=5 (quintic B-splines). Especially, the number of floating point operations for compressed matrix - vector multiplication algorithm (55201 floating point operations) is up to five times smaller than the number of floating point operations performed by a sparse matrix - vector multiplication algorithm (284258 floating point operations) -see Table 1. In general, increasing the order of B-splines results in better compression and faster matrix-vector multiplication.

7 Conclusions and future work

In this paper, we focused on recursive compression of isogeometric analysis mass matrices. We showed that having the matrix recursively compressed, we can speed up the computations of time-dependent problems with iterative solvers up to five times. We considered different orders of B-spline basis functions and different dimensions of patches as employed by refined isogeometric analysis. The future work will involve analysis for stiffness and advection matrices to implement the implicit time integration schemes, and considering adaptive non-regular grids, using e.g. T-splines [17].

References

1. Austin Cottrell, J. , Hughes, T. J. R., Bazilevs, Y.: Isogeometric Analysis: Toward Integration of CAD and FEA, John Wiley Sons, Computational and Numerical Methods, (2009)
2. Hughes, T. J. R. , Cottrell, J. A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Computer methods in applied mechanics and engineering 194(39),4135-4195 (2005)
3. Paszyński, M.: Classical and isogeometric finite element method. https://epodreczniki.open.agh.edu.pl/handbook/1088/module/1173/reader
4. Hsu, M.-C., Akkerman, I., Bazilevs, Y.: High-performance computing of wind turbine aerodynamics using isogeometric analysis. Computers and Fluids, 49(1), 93-100 (2011)
5. Hossain, S., Hossainy, S.F.A., Bazilevs, Y., Calo, V.M., Hughes, T.J.R.: Mathematical modeling of coupled drug and drug-encapsulated nanoparticle transport in patient-speciffic coronary artery walls. Computational Mechanics, 2, 213-242 (2011)
6. Łoś, M., Kłusek, A., Hassaan, M. A., Pingali, K., Dzwinel, W., Paszyński, M.: Parallel fast isogeometric L2 projection solver with GALOIS system for 3D tumor growth simulations. Computer Methods in Applied Mechanics and Engineering 343, 1-22, (2019)
7. Łoś, M., Paszyński, M., Kłusek,K., Dzwinel, W. :Application of fast isogeometric L2 projection solver for tumor growth simulations. Computer Methods in Applied Mechanics and Engineering 316, 1257-1269 (2017)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46

8. Garcia, D. , Pardo, D., Dalcin,L., Paszynski, M., Collier, N., Calo, V. M.: The value of continuity: Refined isogeometric analysis and fast direct solvers. Computer Methods in Applied Mechanics and Engineering 316, 586–605 (2017)
9. Garcia, D. , Pardo, D., Dalcin,L., M., Calo, V. M.: Refined isogeometric analysis for a preconditioned conjugate gradient solver. Computer Methods in Applied Mechanics and Engineering 335, 490–509 (2018)
10. Garcia, D. , Pardo, M., Calo, V. M.: Refined isogeometric analysis for fluid mechanics and electromagnetics. Computer Methods in Applied Mechanics and Engineering 356, 598–628 (2019)
11. Łoś, M., Woz, M., Paszyński, M, Dalcin,L., M., Calo, V. M.: Dynamics with Matrices Pos-sessing Kronecker Product Structure. Procedia Computer Science 51,286-295 (2015)
12. Saad, J.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics (2003)
13. Golub, G. H. ,Van Loan, C.: Matrix Computations, 3rd Edition. John Hopkins University Press, Baltimore, MD, (1996).
14. Hackbusch, W.: Hierarchical Matrices: Algorithms and Analysis, Springer (2015)
15. Mantzaflaris, A., Juttler, B., Khoromskij, B., Langer, U.: Matrix Generation in Isogeometric Analysis by Low Rank Tensor Approximation. International Conference on Curves and Surfaces, 321-340 (2015)
16. Langville, A.N., Meyer, C.D.: Google's PageRank and beyond: the science of search en-gine rankings, Princeton University Press (2006)
17. Bazilevs, Y., Calo, V. M., Cottrell, J. A. ,Evans, J. A. , Lipton, S., Scott, M. A., Sederberg, T. W.: Isogeometric analysis using T-splines. Computer Methods in Applied Mechan-ics and Engineering, 199 229-263 (2010)

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08754-7_46