Comput. Methods Appl. Mech. Engrg. 269 (2014) 334–355

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

GPU accelerated computation of the isogeometric analysis stiffness matrix

A. Karatarakis *, P. Karakitsios, M. Papadrakakis

Institute of Structural Analysis and Antiseismic Research, National Technical University of Athens, Zografou Campus, Athens 15780, Greece

ARTICLE INFO

Article history:
Received 27 March 2013
Received in revised form 25 October 2013
Accepted 2 November 2013
Available online 19 November 2013

Keywords:
Isogeometric analysis
Gauss quadrature
NURBS
Stiffness matrix assembly
Parallel computing
GPU acceleration

ABSTRACT

Due to high regularity across mesh elements, isogeometric analysis achieves higher accuracy per degree of freedom and improved spectrum properties, among others, compared with finite element analysis. However, this inherent feature of isogeometric analysis increases the density of the stiffness matrix and requires more elaborate numerical integration schemes for its computation. For these reasons, the assembly of the stiffness matrix in isogeometric analysis is a computationally demanding task, which needs special attention in order to be affordable for real-world applications. In this paper we address the computational efficiency of assembling the stiffness matrix using the standard element-wise Gaussian quadrature. A novel approach is proposed for the formulation of the stiffness matrix which exhibits several computational merits, among them its amenability to parallelization and the efficient utilization of the graphics processing units to drastically accelerate computations.

1. Introduction

Isogeometric analysis (IGA) was recently introduced by Hughes et al. [1] and since then it has attracted a lot of attention for solving boundary value problems as a result of using the same shape functions adopted from CAD community for describing the domain geometry and for building the numerical approximation of the solution.

Despite IGA's promising methodology and superior features [1–4] compared with finite element analysis (FEA), the computation of mass, stiffness and advection matrices is more laborious, which increases the cost of IGA in real-world applications. Due to its higher inter-element continuity, IGA produces quite more elements than FEA for the same number of degrees of freedom. This leads to an increase of the number of Gauss points and consequently of the computational cost for assembling the characteristic matrices. This drawback dramatically increases the computational cost in the multivariate domains, especially in 3D analysis.

It has been shown [2,3] that standard element-wise Gauss rules are inefficient, because they do not take precise account of the preserved smoothness at the element boundaries in the case of higher-order NURBS and polynomial B-SPLines, and that the higher the inter-element regularity the fewer the required number of Gauss points per element. However, recently proposed integration rules, although optimal or nearly optimal in terms of the number of function evaluations, are either cumbersome to implement [2] or need special consideration to be given to the boundary elements [3]. In an effort to address the increased effort in the computation of IGA characteristic matrices, collocation methods have been introduced, requiring a minimum number of quadrature points [5,6].

* Corresponding author.
E-mail addresses: alex@karatarakis.com (A. Karatarakis), pkarak@hotmail.com (P. Karakitsios), mpapadra@central.ntua.gr (M. Papadrakakis).

0045-7825/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.cma.2013.11.008

Applications of graphics processing units (GPUs) to scientific computations are attracting a lot of attention due to their low cost in conjunction with their inherently remarkable performance features. Driven by the demands of the gaming industry, graphics hardware has substantially evolved over the years with remarkable floating point arithmetic performance. Unlike CPUs, GPUs have an inherent parallel throughput architecture that focuses on executing many concurrent threads "slowly", rather than executing a single thread very fast.

A number of studies in engineering applications have been recently reported on a variety of GPU platforms using implicit computational algorithms [7–17]. Linear algebra applications have also been a topic of scientific interest for GPU implementations [18–21]. GPU accelerated assembly in finite elements methods is reported in [22–25]. A hybrid CPU–GPU implementation of domain decomposition methods is presented in [26] where speedups of the order of 40× have been achieved with just one GPU.

The present work achieves a drastic reduction of the computational effort required for assembling the stiffness matrix of IGA by implementing a novel interaction-wise procedure recently proposed for the computation of the stiffness matrix in element-free Galerkin formulations [27]. This approach is amenable to parallel computations since it does not have race conditions or the need for synchronization and it is particularly suitable for massively parallel systems with GPUs. The numerical results indicate that the proposed methodology succeeds in overcoming the drawback of the quadrature cost associated with IGA by performing the assembly of the stiffness matrix in orders of magnitude less computation time than the standard element-wise Gauss quadrature scheme.

2. Basic ingredients of the isogeometric analysis method

2.1. Non-uniform rational B-SPLines (NURBS)

In IGA the exact geometry is always represented – even in the case of very coarse meshes – and thus there is no approximation in that regard. For the implementation of IGA three spaces should be defined: the physical space, the parameter space and the index space. For NURBS shape functions, the parameter space is very important as all calculations take place in this space, while the index space plays an auxiliary role. The input data is drawn from the physical space, which contains the Cartesian coordinates of the control points and their corresponding weights. The number of basis functions is equal to the number of degrees of freedom. The unknowns of the resulting algebraic equations correspond to the displacements of the control points, while the knots are the boundaries of the corresponding isogeometric elements. In the case of uniform knot vector, knot spans have the same size in the parameter space while in the physical space they can have any size depending on the corresponding control points and shape functions. The discretized NURBS-model is subdivided into patches which are subdomains with the same material and geometry type and consist of a full tensor product grid of elements. In this respect, they are analogous to elements in FEA as the basis functions are interpolatory at its boundaries.

A knot vector is a non-decreasing set of coordinates in the parameter space, written as $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, where $\xi_i \in \mathbb{R}$ is the ith knot, i is the knot index, $i = 1, 2, \ldots, n+p+1$, p is the polynomial order and n is the number of basis functions used to construct the B-SPLine curve. The knots partition the parameter space into elements. Element boundaries in the physical space are the projections of knot lines under the B-SPLine mapping. Fig. 1 illustrates the quadratic C^1 continuous B-SPLine basis functions, which are produced by the open uniform knot vector $\Xi = \{0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 9\}$. Control points are shown as circles, while knots as rectangles. The interval $[0, 9]$ is a single patch and consists of 9 elements and 11 control points, which correspond to 11 B-SPLine basis functions.

Given an open uniform knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, the B-SPLine basis functions $N_i^p(\xi)$ are defined by the Cox-de Boor recursion formula:

$$N_i^0(\xi) = \begin{cases} 1, & \text{if} \quad \xi_i \leqslant \xi < \xi_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_i} N_{i+1}^{p-1}(\xi) \tag{2}$$

Due to their higher regularity between inter-element boundaries, they exhibit greater overlapping in comparison with the shape functions of FEA. Their basic feature is their tensor product nature. In the case of polynomial B-SPLines, basis functions are used as shape functions, while in the case of NURBS, shape functions are produced from the following formula in 1D case:

$$R_i^p(\xi) = \frac{N_i^p(\xi) W_i}{\sum_{i=1}^n \{N_i^p(\xi) W_i\}} \tag{3}$$

in the 2D case:

$$R_{i,j}^{p,q}(\xi, \eta) = \frac{N_i^p(\xi) M_j^q(\eta) W_{i,j}}{\sum_{i=1}^n \sum_{j=1}^m N_i^p(\xi) M_j^q(\xi) W_{i,j}} \tag{4}$$

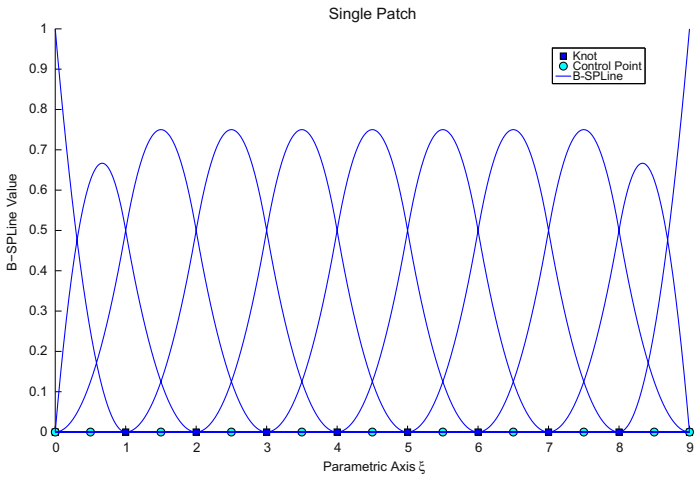

Fig. 1. C^1 continuous quadratic basis derived from open uniform knot vector $\Xi = \{0,0,0,1,2,3,4,5,6,7,8,9,9,9\}$.

in the 3D case:

$$R^{p,q,r}_{i,j,k}(\xi,\eta,\zeta) = \frac{N^p_i(\xi)M^q_j(\eta)L^r_k(\zeta)W_{i,j,k}}{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l}N^p_i(\xi)M^q_j(\xi)L^r_k(\zeta)W_{i,j,k}} \tag{5}$$

where W_i are weight factors with a full tensor product nature:

$$W_{i,j} = W_i W_j \tag{6}$$

$$W_{i,j,k} = W_i W_j W_k \tag{7}$$

The approximation of 1D displacement field in terms of control point variables can be written as

$$u(\xi) = \sum_{i=1}^{n}\{R^p_i(\xi)u_{CPi}\} \tag{8}$$

where $R^p_i(\xi)$ are the shape functions, n is the number of basis functions or control points, p is the polynomial order and u_{CPi} is the displacement of control point i. The exact geometry is described by

$$X(\xi) = \sum_{i=1}^{n}\{R^p_i(\xi)X_{CPi}\} \tag{9}$$

where X_{CPi} are the Cartesian coordinate(s) of the control point i.

There is a connection between polynomial basis order p, knot multiplicity m and continuity/regularity k, given by

$$k = p - m, \quad 1 \leqslant m \leqslant p+1 \tag{10}$$

Regularity -1 indicates discontinuity and it appears for the extreme knots of a single patch. In this case, basis functions are interpolatory at these extreme knots. Regularity 0 resembles the case of finite elements and is the minimum continuity for interior knots with basis functions interpolatory at those knots. The case of maximum continuity is $p-1$ and occurs when every interior knot is repeated only once.

In one-dimensional analysis with polynomial order p, multiplicity m and number of elements n^{el}, the corresponding number of control points n, which are directly linked to the number of degrees of freedom, is equal to

$$n = (p+1)n^{el} - (k-1)(n^{el}-1) \tag{11}$$

The corresponding knot vector has $n+p+1$ knot values. The external knots are repeated $p+1$ times and the interior knots m times.

2.2. Stiffness matrix formulation

A given domain is represented with several NURBS-based isogeometric models, depending on its geometry features. Every NURBS-based model is decomposed into subdomains, called patches, according to the variance of its geometry and material. The more abrupt the geometry is, the more subdomains are considered. They can be assumed as macro-elements consisting of a tensor product mesh of elements and they are assembled in the same way as in finite elements. The arrays for

the patches are constructed and assembled in element-by-element fashion by numerically integrating contributions over each element. In the parameter space, elements are rectangular.

The equilibrium equations applied to control points of the whole domain are expressed as

$$\mathbf{Ku} = \mathbf{f} \tag{12}$$

In order to formulate the domain/global stiffness matrix, the stiffness matrix of every patch $i = 1, \ldots, N_p$ has to be calculated:

$$\mathbf{K}^i = \int_V (\mathbf{B}^i)^T \mathbf{E}^i \mathbf{B}^i dV = \int\int\int_{\xi,\eta,\zeta} (\mathbf{B}^i)^T \mathbf{E}^i \mathbf{B}^i det\mathbf{J}^i d\xi d\eta d\zeta \tag{13}$$

where $\mathbf{E}^i$, $\mathbf{B}^i$ are the elasticity and deformation matrix of the patch i respectively.

The stiffness matrix formulation in 2D elasticity cases is presented below. For the 3D case, the formulation is analogous. Assuming n, m control points per parametric axis ξ, η respectively, the 2D control points are $N = nm$ (full tensor product) and the deformation matrix $\mathbf{B}$ is given by:

$$\underset{(3\times N)}{\mathbf{B}} = \underset{(3\times 4)}{\mathbf{B}_1}\ \underset{(4\times N)}{\mathbf{B}_2} \tag{14}$$

with

$$\underset{(3\times 4)}{\mathbf{B}_1} = \frac{1}{det\mathbf{J}(\xi)} \begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ 0 & 0 & -J_{21} & J_{11} \\ -J_{21} & J_{11} & J_{22} & -J_{12} \end{bmatrix} \tag{15}$$

and

$$\underset{(4\times N)}{\mathbf{B}_2} = \begin{bmatrix} R_{1,\xi} & 0 & R_{2,\xi} & 0 & \ldots & R_{N,\xi} & 0 \\ R_{1,\eta} & 0 & R_{2,\eta} & 0 & \ldots & R_{N,\eta} & 0 \\ 0 & R_{1,\xi} & 0 & R_{2,\xi} & 0 & \ldots & R_{N,\xi} \\ 0 & R_{1,\eta} & 0 & R_{2,\eta} & 0 & \ldots & R_{N,\eta} \end{bmatrix} \tag{16}$$

The Jacobian matrix is local to patches rather than to elements and is given by

$$\underset{(2\times 2)}{\mathbf{J}(\xi,\eta)} = \underbrace{\begin{bmatrix} R_{1,\xi}(\xi,\eta) & R_{2,\xi}(\xi,\eta) & \ldots & R_{N,\xi}(\xi,\eta) \\ R_{1,\eta}(\xi,\eta) & R_{2,\eta}(\xi,\eta) & \ldots & R_{N,\eta}(\xi,\eta) \end{bmatrix}}_{(2\times N)} \underbrace{\begin{bmatrix} X_{CP1} & Y_{CP1} \\ X_{CP2} & Y_{CP2} \\ \vdots & \vdots \\ X_{CPN} & Y_{CPN} \end{bmatrix}}_{(N\times 2)} = \underset{(2\times 2)}{\begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix}} \tag{17}$$

where $R_l(\xi,\eta)$ is the shape function that corresponds to the control point l, with Cartesian coordinates X_{CPl}, Y_{CPl}, and

$$R_{l,\xi}(\xi,\eta) = \frac{dR_l(\xi,\eta)}{d\xi}, \quad R_{l,\eta}(\xi,\eta) = \frac{dR_l(\xi,\eta)}{d\eta} \tag{18}$$

The derivatives in Eq. (18) are obtained by applying the quotient rule to Eq. (4):

$$\begin{aligned} \frac{dR^{p,q}_{i,j}(\xi,\eta)}{d\xi} &= \frac{\frac{dN^p_i(\xi)}{d\xi} M^q_j(\eta) W_{i,j} W(\xi,\eta) - N^p_i(\xi) M^q_j(\eta) W_{i,j} \frac{dW(\xi,\eta)}{d\xi}}{(W(\xi,\eta))^2} \\ \frac{dR^{p,q}_{i,j}(\xi,\eta)}{d\eta} &= \frac{N^p_i(\xi) \frac{dM^q_j(\eta)}{d\eta} W_{i,j} W(\xi,\eta) - N^p_i(\xi) M^q_j(\eta) W_{i,j} \frac{dW(\xi,\eta)}{d\eta}}{(W(\xi,\eta))^2} \end{aligned} \tag{19}$$

where

$$\begin{aligned} \frac{dW(\xi,\eta)}{d\xi} &= \sum_{i=1}^{n}\sum_{j=1}^{m} \left\{ \frac{dN^p_i(\xi)}{d\xi} M^q_j(\eta) W_{i,j} \right\} \\ \frac{dW(\xi,\eta)}{d\eta} &= \sum_{i=1}^{n}\sum_{j=1}^{m} \left\{ N^p_i(\xi) \frac{dM^q_j(\eta)}{d\eta} W_{i,j} \right\} \end{aligned} \tag{20}$$

2.3. Quadrature rule

The Gauss quadrature rule is applied to the non-zero knot spans as in FEA. However, the standard element-wise Gauss rule requires extensive function evaluations due to the increased support of the shape functions. According to [3], for the case of a one-dimensional function of order p, the optimal (minimum exact) number of Gauss points per element is equal

to $(p+1)/2$ or $(p+2)/2$, for odd and even p, respectively. For the computation of the stiffness matrix in case of 1D elasticity, the integrand's order is equal to $q = 2p - 2$ and the optimal number of Gauss points per element is equal to $(q+2)/2 = p$. In case of 2D and 3D elasticity, the integrand's order is equal to $q = 2p$ and the optimal number of Gauss points per element is equal to $(q+2)/2 = p+1$. The above rules are optimal for the case of minimum continuity. For higher continuity, new macro-element rules have been proposed [2,3], which are more efficient but also more involved and difficult to implement.

3. Element-wise formulation of the stiffness matrix

In order to build the stiffness matrix of a domain (or patch), the contributions of all Gauss points need to be added. The contributions are expressed by the following product:

$$\mathbf{K} = \sum_{G} w_G \mathbf{B}_G^T \mathbf{E} \mathbf{B}_G = \sum_{G} w_G \mathbf{Q}_G \tag{21}$$

where the deformation matrix $\mathbf{B}_G$ is computed at the corresponding Gauss point, and w_G is the weight factor of the Gauss point.

However, instead of adding each Gauss point contribution to the global stiffness matrix, the standard procedure is to first build the stiffness matrix of each element by adding the contributions of all Gauss points G of the element to its local stiffness matrix:

$$\mathbf{K}_E = \sum_{E_G} w_G \mathbf{B}_G^T \mathbf{E} \mathbf{B}_G = \sum_{E_G} w_G \mathbf{Q}_G \tag{22}$$

Since all Gauss points of an element modify the same group of values in the global stiffness matrix, making several additions locally and then applying the collective contribution to the stiffness matrix is beneficial. Each element's stiffness matrix is subsequently appended to the global stiffness matrix in the appropriate positions:

$$\mathbf{K} = \sum_{E} \mathbf{K}_E \tag{23}$$

The summation is performed for each Gauss point and affects all control points within its domain of influence. Compared with FEA, the amount of calculations for performing this task is significantly higher since the domains of influence of Gauss points are much larger than the corresponding domains in FEA. The areas influencing a control point are shown in Fig. 2 for various values of p in the 1D case. A comparison between the influencing areas of a control point/node in IGA and FEA is depicted in Fig. 3, for 2D case and for different p. It should be noted that the actual correlation is between control points and Gauss points. Elements provide a convenient grouping so the correlations are easier to handle and store. Throughout this paper we do not address the issue of accuracy when comparing the two methods with the same number of control points or elements.

In FEA, each Gauss point is involved in computations with nodes within its own element only. The shape functions and their derivatives are predefined for each element type and need to be evaluated on all combinations of nodes and Gauss points within the element. In IGA, however, each Gauss point is involved in computations with control points of the surrounding areas as well (Fig. 4), while the shape functions are not predefined and span across larger domains with a significantly higher amount of control point–Gauss point correlations.

The domain stiffness matrix is large and therefore needs to be stored with appropriate efficient storage methods. Depending on the method used for the solution of the resulting equations and on the constraints imposed by the hardware (e.g. memory limit), it may be in sparse skyline format, multi-diagonal (since the grid is structured) format, etc. The sparse matrix formats used in this work are specifically tailored for the assembly phase and a distinction is made between the implemented sparse matrix builders and formats like CSR/CSC more suitable for the solution phase.

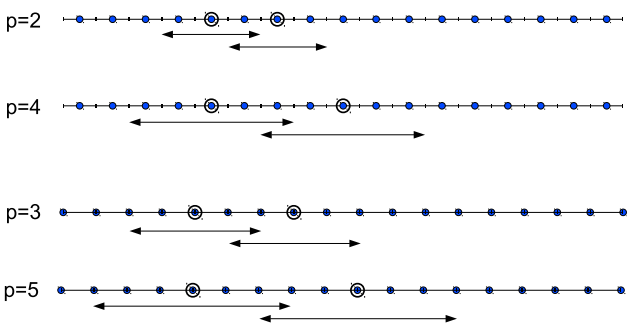

Fig. 2. IGA 1D domains of influence of control points ◉ for various values of p.

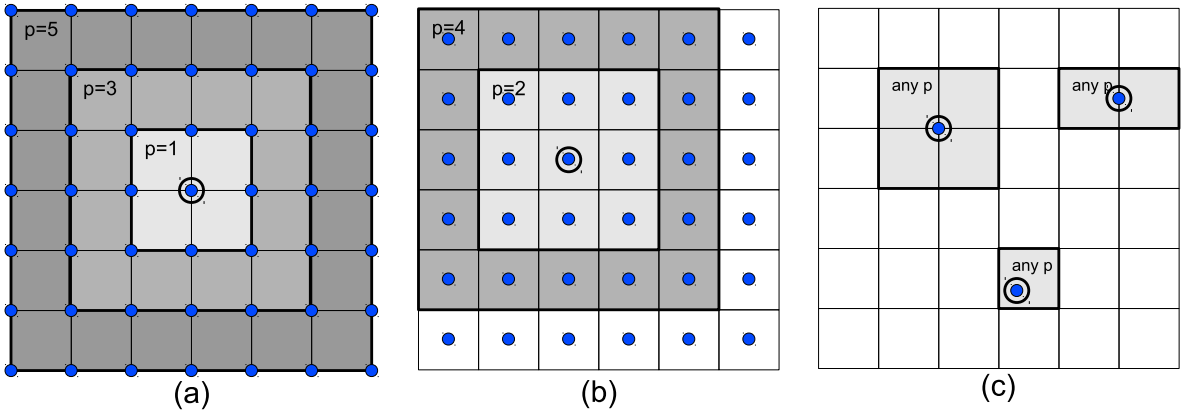

Fig. 3. Areas influencing control point ◉ in (a) IGA (p odd); (b) IGA (p even); (c) FEA. The influencing entities are the Gauss points in the shaded areas.

For equivalent meshes, the bandwidth is the same between the two methods but IGA has a larger amount of control point interactions and, consequently, denser stiffness matrices. Furthermore, the computation of each non-zero coefficient is more laborious because the control point pairs have a lot more shared elements (on average) and consequently significantly more Gauss points that contribute to the final values.

3.1. Comparison with FEA

Table 1 gives a quantitative comparison of the total number of elements and Gauss points in IGA and FEA for $n = 121$ control points/nodes in each axis, for 2D and 3D simulations. A $p+1$ integration rule is adopted in both IGA and FEA for the sake of comparison. For large numbers n of control points/nodes, the ratio of elements in IGA and FEA asymptotically approaches p^d, where d is the dimension of the problem (Table 2). Since each element has the same number of Gauss points in both IGA and FEA, the ratio holds for both elements and Gauss points.

Figs. 5 and 6 show a visual comparison when zooming in on a region of the domain with the same number of control points/nodes. Variations in the number of correlations in the boundary of the domain are ignored in the following analysis. Fig. 5 depicts the control points/nodes for even numbers of p for IGA and $p = 2$ for FEA, while Fig. 6 depicts them for odd numbers of p for IGA and $p = 3$ for FEA.

Table 3 shows the number of correlations of nodes with elements and Gauss points in 2D and 3D problems for FEA. The first column shows the order p while the second shows the number of Gauss points per element. The third column shows the number of nodes influenced by a Gauss point or element. The next columns show the number of elements and the number of Gauss points influencing a node. As can be seen from Figs. 5 and 6, there are variations as to how many elements/Gauss points influence a node. Corner nodes have the max number of correlations, while internal nodes are only affected by the Gauss points of the element they belong to, while side/edge nodes are in-between.

Table 4 shows the corresponding number of correlations of control points with elements and Gauss points in 2D and 3D problems for IGA. In this case, the way in which a control point is correlated with an element/Gauss point is different and there are no variations in the number of Gauss points or elements affecting a control point.

In FEA each Gauss point affects only the nodes within its own element and the number of nodes is increased with the order p. In IGA each Gauss point affects surrounding areas (range depending on p), but the number of control points affected by each Gauss point is the same as the number of influenced nodes in FEA. On the other hand, each control point is affected by more elements in IGA than in FEA and consequently by a lot more Gauss points. The correlations are increasing much faster in IGA than in FEA as p increases. Table 5 shows the number of Gauss points influencing a control point/node with respect to p, demonstrating the growth rate.

It should be noted that in FEA corner nodes are constantly 4 or 8 (for 2D and 3D analysis respectively) and the number of internal nodes increases faster than side/edge nodes. Thus, for higher values of p, most nodes are internal and therefore the average number of Gauss points/element influencing a node is closer to the *min* value than to the *max* value. Therefore, for large problems (where deviations in the boundary can be ignored) and for increasing values of p, the amount of correlations in IGA are approaching the square of those in FEA.

3.2. Computation of stiffness contribution for each element

3.2.1. $\boldsymbol{B^TEB}$ Calculation

A fast computation of the matrix product

$$\mathbf{Q}_G = \mathbf{B}_G^T \mathbf{E} \mathbf{B}_G \tag{24}$$

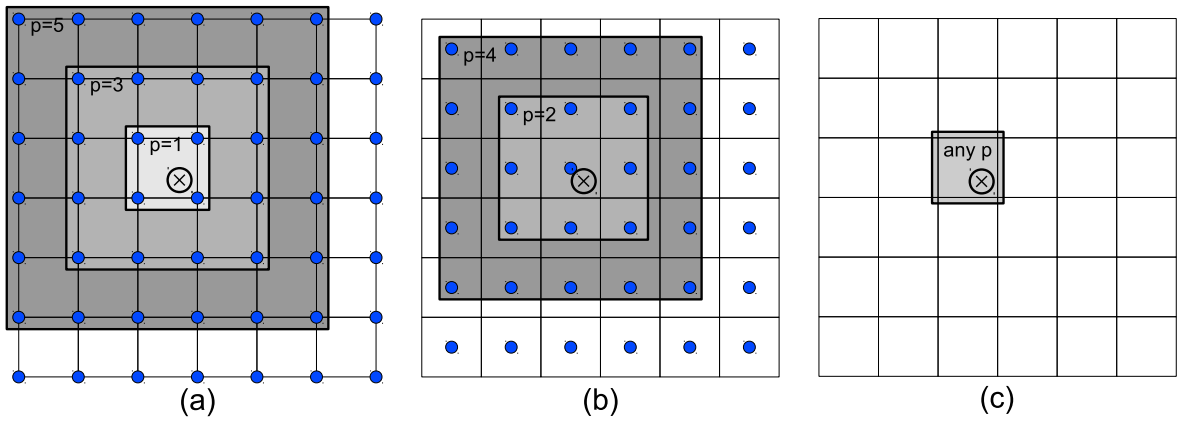

Fig. 4. Control points/nodes influenced by Gauss point ⊗ in (a) IGA (p odd); (b) IGA (p even); (c) FEA.

of Eq. (22) is important because it is repeated at each integration point. This may not be so critical in FEA compared to the total simulation time, but it is very important in IGA where the number of Gauss points and the number of influenced nodes per Gauss point are both significantly greater.

The computations of Eq. (24) can be broken into smaller operations for each combination of influenced control points i,j belonging to the domain of influence of the Gauss point under consideration:

$$\mathbf{Q}_{ij} = \mathbf{B}_i^T \mathbf{E} \mathbf{B}_j \tag{25}$$

Once a submatrix $\mathbf{Q}_{ij}$ is calculated, it is multiplied with the weight factor of the corresponding Gauss point and then added to the appropriate positions of the (element's) stiffness matrix. The computation related to $\mathbf{Q}_{ij}$ together with the associated entry updates of $\mathbf{K}$ are a significant part of the total effort for the formulation of the global stiffness matrix.

For an isotropic material in 3D elasticity, $\mathbf{Q}_{ij}$ takes the form:

$$\underset{(3\times3)}{\mathbf{Q}_{ij}} = \underset{(3\times6)}{\mathbf{B}_i^T} \underset{(6\times6)}{\mathbf{E}} \underset{(6\times3)}{\mathbf{B}_j} = \begin{bmatrix} \Phi_{i,x} & 0 & 0 & \Phi_{i,y} & 0 & \Phi_{i,z} \\ 0 & \Phi_{i,y} & 0 & \Phi_{i,x} & \Phi_{i,z} & 0 \\ 0 & 0 & \Phi_{i,z} & 0 & \Phi_{i,y} & \Phi_{i,x} \end{bmatrix} \begin{bmatrix} M & \lambda & \lambda & & & \\ \lambda & M & \lambda & & & \\ \lambda & \lambda & M & & & \\ & & & \mu & & \\ & & & & \mu & \\ & & & & & \mu \end{bmatrix} \begin{bmatrix} \Phi_{j,x} & 0 & 0 \\ 0 & \Phi_{j,y} & 0 \\ 0 & 0 & \Phi_{j,z} \\ \Phi_{j,y} & \Phi_{j,x} & 0 \\ 0 & \Phi_{j,z} & \Phi_{j,y} \\ \Phi_{j,z} & 0 & \Phi_{j,x} \end{bmatrix}$$

$$\underset{(3\times3)}{\mathbf{Q}_{ij}} = \begin{bmatrix} \Phi_{i,x}\Phi_{j,x}M + \Phi_{i,y}\Phi_{j,y}\mu + \Phi_{i,z}\Phi_{j,z}\mu & \Phi_{i,x}\Phi_{j,y}\lambda + \Phi_{i,y}\Phi_{j,x}\mu & \Phi_{i,x}\Phi_{j,z}\lambda + \Phi_{i,z}\Phi_{j,x}\mu \\ \Phi_{i,y}\Phi_{j,x}\lambda + \Phi_{i,x}\Phi_{j,y}\mu & \Phi_{i,y}\Phi_{j,y}M + \Phi_{i,x}\Phi_{j,x}\mu + \Phi_{i,z}\Phi_{j,z}\mu & \Phi_{i,y}\Phi_{j,z}\lambda + \Phi_{i,z}\Phi_{j,y}\mu \\ \Phi_{i,z}\Phi_{j,x}\lambda + \Phi_{i,x}\Phi_{j,z}\mu & \Phi_{i,z}\Phi_{j,y}\lambda + \Phi_{i,y}\Phi_{j,z}\mu & \Phi_{i,z}\Phi_{j,z}M + \Phi_{i,y}\Phi_{j,y}\mu + \Phi_{i,x}\Phi_{j,x}\mu \end{bmatrix} \tag{26}$$

Matrices $\mathbf{E}$ and $\mathbf{B}_i$ are never formed. Instead three values for $\mathbf{E}$, the two Lamé parameters λ, μ and the P-Wave modulus $M = 2\mu + \lambda$, as well as three values for $\mathbf{B}_i$, specifically $N_{i,x}$, $N_{i,y}$, $N_{i,z}$, are stored. Since some of the multiplications are repeated, the calculations in Eq. (26) can be efficiently performed with 30 multiplications and 12 additions.

3.2.2. Sparse matrix format for the element-wise approach

The global stiffness matrix is stored in sparse format because it requires considerably less memory. However, indexing time during the creation of the stiffness matrix is a common source of significant slowdown and must be handled appropriately. Thus, an efficient implementation for building the stiffness matrix in sparse format is required in the element-wise approach. The procedure requires updating previous values of the matrix, thus a sparse matrix type that allows lookups is needed.

A sparse matrix format suitable for this method is the dictionary of keys (DOK) [28]. The DOK format is inefficient for operations like matrix–vector products required in the solution phase of iterative solvers, but it is ideal for the assembly phase, offering O(1) lookup and only taking space proportional to the number of non-zero entries. Other formats considered, especially those more suitable for the solution phase, either do not have O(1) lookup time or do not have the same memory footprint as the DOK format. For the solution phase, a conversion to more appropriate formats (e.g. CSR/CSC) should be made.

The DOK implementation used in this work is based on a single hash-table [29] which hashes the keys by row and column and performs well even for situations where the Gauss point contributions are directly appended to the global stiffness matrix, which is the case of element-free Galerkin (EFG) methods [27]. In contrast to EFG, calculations in IGA are performed on the element level for all Gauss points of an element, and the combined contribution is then appended to the global stiffness

Table 1
Total elements and Gauss points in IGA and FEA, for $n = 121$ control points/nodes and different p, in 2D and 3D square and cubic domains.

$n = 121$			Total elements		Total Gauss points		Ratio
	p	GP per element	IGA	FEA	IGA	FEA	
2D	1	4	14,400	14,400	57,600	57,600	1.0
	2	9	14,161	3,600	127,449	32,400	3.9
	3	16	13,924	1,600	222,784	25,600	8.7
	4	25	13,689	900	342,225	22,500	15.2
	5	36	13,456	576	484,416	20,736	23.4
3D	1	8	1,728,000	1,728,000	13,824,000	13,824,000	1.0
	2	27	1,685,159	216,000	45,499,293	5,832,000	7.8
	3	64	1,643,032	64,000	105,154,048	4,096,000	25.7
	4	125	1,601,613	27,000	200,201,625	3,375,000	59.3
	5	216	1,560,896	13,824	337,153,536	2,985,984	112.9

Table 2
Total elements in IGA and FEA with respect to p.

	Total elements			
	IGA	IGA lim	FEA	FEA lim
1D	$(n–p)^1$	n^1	$[(n-1)/p]^1$	$(n/p)^1$
2D	$(n–p)^2$	n^2	$[(n-1)/p]^2$	$(n/p)^2$
3D	$(n–p)^3$	n^3	$[(n-1)/p]^3$	$(n/p)^3$

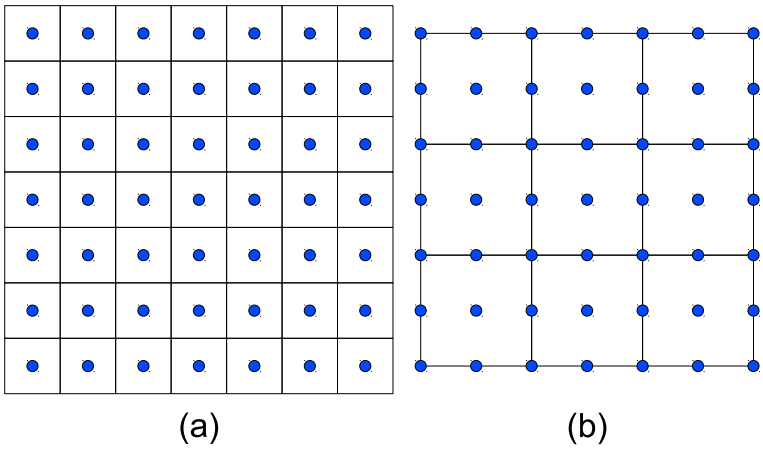

Fig. 5. Visual comparison between (a) IGA p = even; (b) FEA $p = 2$, for the same number of control points/nodes.

matrix. As a result, our implementation in IGA offers performance very close to that of a dense or skyline matrix, but with reduced memory cost.

3.3. Performance of the element-wise approach

The examples that will be presented throughout this work are detailed in Table 6, while the performance of the element-wise (EW) approach in the CPU (single core) is shown in Table 7. The examples are run on a Core i7-980X which has 6 physical cores (12 logical cores) at 3.33 GHz and 12 MB cache. The examples have no trivial knot spans in order to maximize the number of calculations. All floating point calculations are in double-precision arithmetic.

As it has been mentioned in previous sections, the formulation of the matrix in IGA is quite laborious and more code optimization is needed compared to a FEA code. Table 7 compares the element-wise approach for building the stiffness matrix when using sparse and skyline format. Although the skyline format offers very fast indexing times (almost the same as the dense format) it stores an increasingly higher number of stiffness elements, as shown in Table 8, and thus imposes restrictions in memory requirements of large-scale simulations. Tables 7 and 8 show that the proposed sparse matrix implementation is very close to the skyline format in terms of the required computing time, while storing a significantly lower amount of stiffness elements.

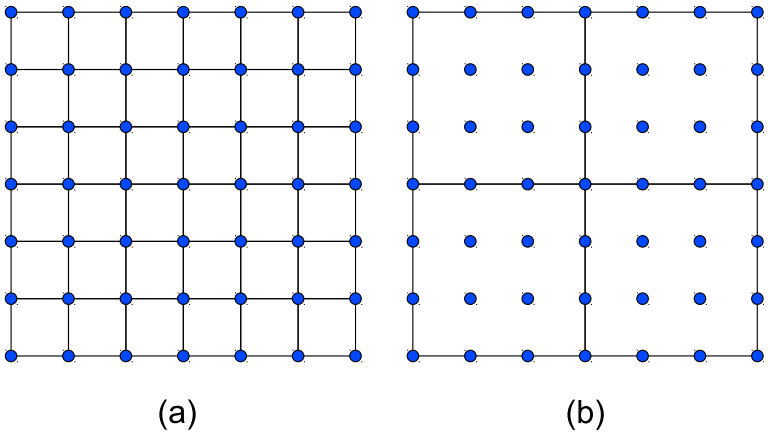

Fig. 6. Visual comparison between (a) IGA p = odd; (b) FEA p = 3, for the same number of control points/nodes.

Table 3
Correlations of nodes with elements and Gauss points for FEA.

	p	GP per element	Nodes influenced by GP/element	Elements influencing a node		Gauss points influencing a node	
				Min	Max	Min	Max
2D	1	4	4	4	4	16	16
	2	9	9	1	4	9	36
	3	16	16	1	4	16	64
	4	25	25	1	4	25	100
	5	36	36	1	4	36	144
3D	1	8	8	8	8	64	64
	2	27	27	1	8	27	216
	3	64	64	1	8	64	512
	4	125	125	1	8	125	1000
	5	216	216	1	8	216	1728

Table 4
Correlations of control points with elements and Gauss points for IGA.

	p	GP per element	Control points influenced by GP/element	Elements influencing a control point	Gauss points influencing a control point
2D	1	4	4	4	16
	2	9	9	9	81
	3	16	16	16	256
	4	25	25	25	625
	5	36	36	36	1296
3D	1	8	8	8	64
	2	27	27	27	729
	3	64	64	64	4096
	4	125	125	125	15,625
	5	216	216	216	46,656

Table 5
Number of Gauss points influencing a control point/node with respect to p.

Gauss points influencing a control point/node			
Problem type	IGA	FEA min	FEA max
1D	$(p+1)^2$	$(p+1)^1$	$2^1\,(p+1)^1$
2D	$(p+1)^4$	$(p+1)^2$	$2^2\,(p+1)^2$
3D	$(p+1)^6$	$(p+1)^3$	$2^3\,(p+1)^3$

4. Interaction-wise formulation of the stiffness matrix

An alternative way to perform the computation of the global stiffness matrix is the proposed interaction-wise approach (IW). This approach organizes calculations in a different way than the element-wise approach. More specifically, instead of iterating through the elements and making continual updates to the appropriate stiffness coefficients, the IW approach calculates the final value of a particular coefficient before moving on to another. This strategy carries a number of benefits discussed in the following sections.

The computation of the global stiffness coefficient K_{ij} is performed for all interacting $i-j$ control points and is formed from contributions by the shared Gauss points of their domains of influence. Two control points are interacting if there is at least one Gauss point that influences both control points. In IGA, it is more convenient to define two control points as being interacting if there is at least one element shared by both control points, but care must be taken in cases where there are trivial knot spans which have no Gauss points.

4.1. Interacting control point pairs and their shared elements

The IW approach initially identifies (a) the interacting control point pairs and (b) the shared elements of the interacting control point pairs. In our previous work on meshless EFG methods [27], we presented and analyzed general-purpose techniques for both of these steps. These techniques are also inherently parallel and the efficiency of their serial and parallel implementations were presented. These techniques apply here as well, but since IGA has structured grids, the interactions and shared elements can be derived from the grid thus obviating the need for searching. The interacting control points associated with a specific control point are those located within a fixed range dictated by the order p in each axis.

In FEA the nodes interact with nodes in adjacent elements only and thus the interacting node pairs can be easily defined from the element-node connectivity (Fig. 7b). In IGA, however, a control point pair contributes non-zero entries to the stiffness matrix, and therefore is active, if there is at least one (non-empty) element shared between the two control points (Fig. 7a). Thus, control point X interacts with B, C, D, but not with A or E. If the basis order is p, then the interacting control points extend up to p elements in all directions. This can be observed for $p=2$ in Fig. 7a. The gray shaded regions are the influence domains of each control point. The thick-lined rectangles in Fig. 7 include all control points/nodes that are interacting with the corresponding control point/node.

Fig. 8 shows the interactions for $p=3$. It also shows the case where there is a trivial knot span on the y-axis. This creates a row of elements that are empty. As a result, X and C are not interacting despite having a common element, since this element has no Gauss points. Control points X and D are still interacting because there is one shared element with Gauss points. The effect of a trivial knot span is that it limits the range of the interaction area in that direction, as can be seen in Fig. 8. Similarly, multiple trivial knot spans further limit the range of the interaction.

4.2. Comparison with FEA for equal number of freedom degrees

Tables 9 and 10 show the number of interactions per control point/node. The *max* case for FEA is equal to the case in IGA (assuming no trivial knot spans and ignoring boundaries). As it has already been mentioned, internal nodes are increasing faster than other nodes as p increases, so the average of FEA is going to be closer to the *min* values.

There are variations in the number of shared elements between two interacting control points/nodes in both IGA and FEA. Closer nodes have many shared elements, as is the case of points A and E, or only a single one, as is the case of distant elements like X and B in Fig. 7a and Fig. 8. In FEA, the variations are limited to less shared elements: internal nodes only have a single shared element with any of their interacting nodes, while non-internal nodes have at most 2, in 2D problems, or 4, in 3D problems, shared elements with other non-internal nodes they interact with.

From the above analysis it is clear that the computational effort for assembling the stiffness matrix in IGA is much higher than in FEA for the same number of freedom degrees.

4.3. Computation of global stiffness coefficients for each interacting control point pair

The computation of the stiffness elements for each interacting control point pair is split in two phases. In the first phase, the shape function derivatives for each influenced control point of every Gauss point are calculated as in the element-wise method. In the second phase, instead of continuing with the calculation of the stiffness matrix corresponding to a particular element, the stiffness matrix coefficients of each interacting control point pair is computed. For each interacting node pair ij, the matrix $w_G\mathbf{Q}_{ij}$ is calculated over all shared Gauss points and summed to form the final values of the corresponding entries of the global matrix:

$$\mathbf{K}_{ij}=\sum_{G} w_G\mathbf{Q}_{ij}=\sum_{G} w_G\mathbf{B}_i^T\mathbf{E}\mathbf{B}_j \tag{27}$$

The calculation of $\mathbf{Q}_{ij}$ matrices is performed as described in Section 3.2.1. The matrices $\mathbf{B}_i$, $\mathbf{B}_j$ contain the shape function derivative values calculated in the first phase and stored in order to be used in subsequent computations.

Table 6
Example details of 2D square ($n \times n$) and 3D cubic ($n \times n \times n$) domains.

Example		p	n	Control points	Dof	Elements	Gauss points
2D	P2–1	2	225	50,625	101,250	49,729	447,561
	P2–2	2	500	250,000	500,000	248,004	2,232,036
	P2–3	2	633	400,689	801,378	398,161	3,583,449
	P3–1	3	225	50,625	101,250	49,284	788,544
	P3–2	3	320	102,400	204,800	100,489	1,607,824
	P3–3	3	388	150,544	301,088	148,225	2,371,600
	P4–1	4	160	25,600	51,200	24,336	608,400
	P4–2	4	225	50,625	101,250	48,841	1,221,025
	P4–3	4	275	75,625	151,250	73,441	1,836,025
3D	P2–1	2	19	6859	20,577	4913	132,651
	P2–2	2	26	17,576	52,728	13,824	373,248
	P2–3	2	33	35,937	107,811	29,791	804,357
	P3–1	3	19	6859	20,577	4096	262,144
	P3–2	3	21	9261	27,783	5832	373,248
	P3–3	3	26	17,576	52,728	12,167	778,688
	P4–1	4	15	3375	10,125	1331	166,375
	P4–2	4	17	4913	14,739	2197	274,625
	P4–3	4	19	6859	20,577	3375	421,875

Table 7
Single core CPU computing time for the formulation of the stiffness matrix in the element-wise (EW) approach with sparse and skyline storage.

Example		Dof	Time (s)		Ratio
			Sparse	Skyline	
2D	P2–1	101,250	5	4	1.2
	P2–2	500,000	20	17	1.2
	P2–3	801,378	32	27	1.2
	P3–1	101,250	14	13	1.1
	P3–2	204,800	27	25	1.1
	P3–3	301,088	39	35	1.1
	P4–1	51,200	19	18	1.1
	P4–2	101,250	36	34	1.1
	P4–3	151,250	57	52	1.1
3D	P2–1	20,577	8	8	1.0
	P2–2	52,728	21	18	1.1
	P2–3	107,811	43	37	1.2
	P3–1	20,577	59	54	1.1
	P3–2	27,783	83	75	1.1
	P3–3	52,728	168	154	1.1
	P4–1	10,125	131	128	1.0
	P4–2	14,739	218	216	1.0
	P4–3	20,577	333	323	1.0

Both phases are amenable to parallelization, the first with respect to elements or Gauss points and the second with respect to interacting control point pairs. There are no race conditions or need for synchronization, which makes the interacting control point pairs approach an ideal method for massively parallel systems, like GPUs.

4.4. Sparse matrix format for the interaction-wise approach

The final values of each $\mathbf{K}_{ij}$ submatrix are calculated and written once in the corresponding positions of the global stiffness matrix instead of being gradually updated as in the element-wise approach. Apart from the reduced number of accesses to the global matrix, this method does not require lookups, which allows the use of simpler and more efficient sparse matrix formats.

The coordinate list (COO) format is appropriate for this method [28]. A simple implementation with three arrays, one for row indexes, one for column indexes and one for the value of each non-zero matrix coefficient is sufficient and can be easily applied both in the CPU and the GPU, while also requiring less memory than a format that allows lookups. Note that the interaction-wise method has no indexing time due to its nature, in contrast to the element-wise approach.

4.5. Performance of the interaction-wise approach

Table 11 shows the time required for the two phases described in Section 4.3, as well as the total computing time for the formulation of the stiffness matrix with the interaction-wise (IW) approach. Comparing the serial CPU times with the

Table 8
Number of stored stiffness elements for skyline and sparse storage.

Example		Dof	Number of stored elements		Ratio
			Skyline	Sparse	
2D	P2–1	101,250	91,071,675	2,554,947	36
	P2–2	500,000	999,744,000	12,690,072	79
	P2–3	801,378	2,028,680,811	20,359,251	100
	P3–1	101,250	136,226,475	4,936,563	28
	P3–2	204,800	392,296,720	10,030,368	39
	P3–3	301,088	699,568,656	14,773,776	47
	P4–1	51,200	64,992,000	4,058,400	16
	P4–2	101,250	181,177,875	8,090,675	22
	P4–3	151,250	331,150,875	12,129,675	27
3D	P2–1	20,577	43,366,569	3,182,649	14
	P2–2	52,728	209,681,004	8,606,172	24
	P2–3	107,811	693,626,571	18,142,461	38
	P3–1	20,577	63,172,473	7,982,313	8
	P3–2	27,783	104,801,445	11,085,579	9
	P3–3	52,728	308,053,200	22,134,864	14
	P4–1	10,125	24,421,500	6,849,000	4
	P4–2	14,739	46,342,884	10,594,236	4
	P4–3	20,577	81,740,508	15,503,568	5

corresponding values of the element-wise (EW) approach (Table 7), it can be observed that the IW approach performs better than the EW approach.

4.6. Parallelization features of the interaction-wise approach

The interaction-wise (IW) approach has certain advantages compared with the element-wise (EW) approach, the most important one being its amenability to parallelism. In the EW approach, each element contributes to different stiffness coefficients and the coefficients are continuously updated with the contributions of each element. The final value of a particular coefficient is formed once all contributions have been considered. Therefore, parallelizing the element-wise approach involves scatter parallelism, which is schematically shown in Fig. 9 for two elements *C* and *D*. Each part of the sum can be calculated in parallel but there are conflicting updates to the same coefficients of the stiffness matrix. These race conditions can be avoided with proper synchronization but in massively parallel systems like the GPU where thousands of threads may be working concurrently it is very detrimental to performance because all updates are serialized with atomic operations [30].

In the IW approach, the final values for the submatrix of each interacting control point pair are calculated and appended to the matrix, instead of continually updating the corresponding coefficients. For the calculation of a submatrix, all contributions of the Gauss points belonging to the intersection of the domains of influence of two interacting control points should be summed together. Thus, the IW approach utilizes gather parallelism as shown schematically in Fig. 10.

In a parallel implementation, each working unit, i.e. a thread or group of threads, prepares a submatrix $\mathbf{K}_{ij}$ related to a specific interacting control point pair *ij*. It gathers all contributions from the Gauss points and writes to a specific memory location accessed by no other thread. Thus, this method requires no synchronization or atomic operations. An important

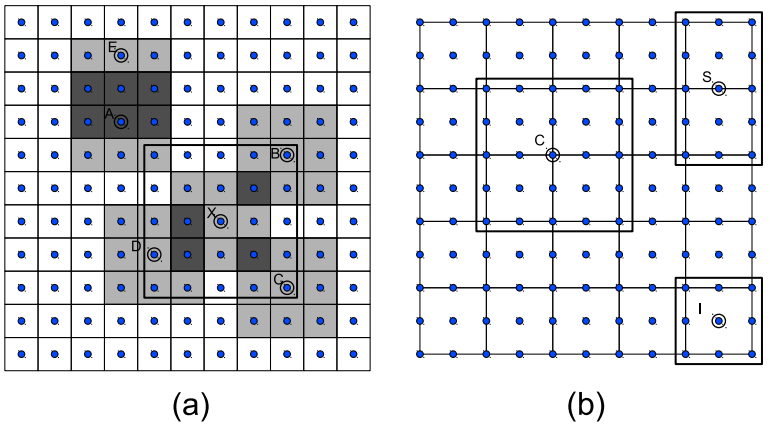

Fig. 7. Interacting control points/nodes for $p = 2$: (a) IGA; (b) FEA.

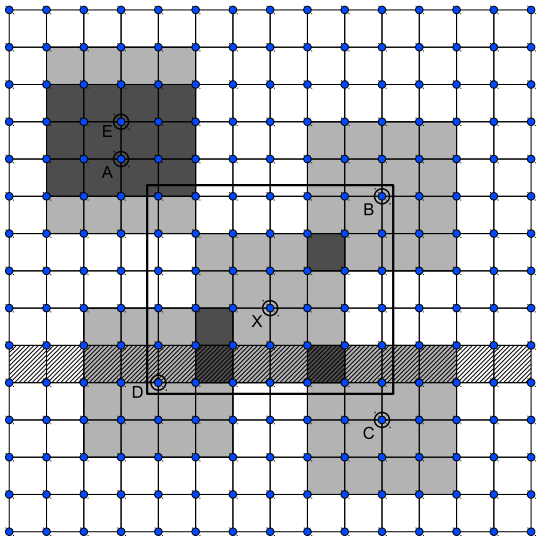

Fig. 8. IGA interacting control points for $p = 3$.

benefit of this approach is the indexing cost of the stiffness matrix elements. In the EW approach each stiffness matrix coefficient is updated a large number of times while in the proposed IW approach the final value is calculated and written only once.

5. GPU programming

Graphics processing units (GPUs) are parallel devices of the SIMD (single instruction, multiple data) classification, which describes devices with multiple processing elements that perform the same operation on multiple data simultaneously and exploit data level parallelism. GPUs have a large number of streaming processors (SPs), which can collectively offer significantly more gigaflops than current high-end CPUs.

5.1. GPU threads

The GPU applies the same functions, called kernels, on a large number of data. Kernels generate a large number of threads in order to exploit data parallelism, hence the single instruction multiple thread (SIMT) paradigm. Threads in GPUs take very few clock cycles to generate and schedule due to the GPU's underlying hardware support, unlike CPUs where thousands of clock cycles are required. All threads generated by a kernel define a grid and are organized in equally sized groups, commonly referenced as thread blocks [in CUDA] or thread groups [in openCL].

There is another type of thread grouping called warps which are the units of thread scheduling in the GPU. The number of threads in a warp is specific to the particular hardware implementation. In the present study, the GPUs used have warp size of 32. The purpose of warps is to ensure high hardware utilization. With a sufficient number of warps, the processors are likely to have a continuous workload in spite of the long-latency operations.

Table 9
Interactions per control point/node in IGA and FEA.

Interactions per control point/node				
	p	IGA	FEA min	FEA max
2D	1	9	4	9
	2	25	9	25
	3	49	16	49
	4	81	25	81
	5	121	36	121
3D	1	27	8	27
	2	125	27	125
	3	343	64	343
	4	729	125	729
	5	1,331	216	1,331

Table 10
Interactions per control point/node with respect to p.

Interactions per control point/node			
Problem type	IGA	FEA min	FEA max
1D	$(2p+1)^1$	$(p+1)^1$	$(2p+1)^1$
2D	$(2p+1)^2$	$(p+1)^2$	$(2p+1)^2$
3D	$(2p+1)^3$	$(p+1)^3$	$(2p+1)^3$

The number of threads in each block is subject to refinement. It should be a power of 2 and, in contemporary hardware, less than 1024. It is also recommended that the number of threads per block should be chosen as a multiple of the warp size [31].

5.2. GPU memory

GPUs have a variety of different memories that can be utilized by programmers in order to achieve high performance. Fig. 11 shows a simplified representation of the different memories. The global memory is the memory responsible for interaction with the host/CPU and is large in size and off-chip. The input data is first transferred from the host memory to the device global memory and the output data needs to be placed here before being passed over to the host. Constant memory also provides interaction with the host, but the device is only allowed to read from it and not write to it. It is small, but provides fast access for data needed by all threads.

There are also other types of memories which cannot be accessed by the host. Registers [CUDA] or private memories [openCL] are thread-bound meaning that each thread can only access its own registers. Shared memories [CUDA] or local memories [openCL] are allocated to thread blocks/groups instead of single threads, which allows all threads in a block to access variables in the shared memory locations allocated specifically for that block. Data in registers and shared memory can be accessed in a highly efficient manner.

6. GPU implementation of the interaction-wise approach

Each one of the two phases described Section 4.3 is calculated with its own kernel and exhibits different levels of parallelism. The implementations in this work are written in openCL for greater portability.

6.1. Phase 1 – calculation of shape function and derivative values

In the first phase, the shape functions and their derivatives are calculated for all influenced nodes of every Gauss point. All Gauss points of a particular element have the same influenced nodes. There are two levels of parallelism: the major over the elements and the minor over the Gauss points. A thread block/group is assigned to each element and each thread handles one Gauss point at a time and iterates over all influenced nodes. Since all threads iterate over the same number of influenced

Table 11
Single core CPU computing time for the formulation of the stiffness matrix in the interaction-wise (IW) approach.

Example		Dof	Time (s)		Total
			Shape functions	Assembly	
2D	P2–1	101,250	2	2	4
	P2–2	500,000	10	6	17
	P2–3	801,378	17	10	27
	P3–1	101,250	7	5	12
	P3–2	204,800	13	10	23
	P3–3	301,088	18	16	34
	P4–1	51,200	8	8	16
	P4–2	101,250	15	17	31
	P4–3	151,250	21	26	47
3D	P2–1	20,577	2	4	7
	P2–2	52,728	6	10	17
	P2–3	107,811	12	23	36
	P3–1	20,577	9	33	42
	P3–2	27,783	12	46	58
	P3–3	52,728	24	95	119
	P4–1	10,125	11	73	84
	P4–2	14,739	18	123	141
	P4–3	20,577	27	184	211

nodes, there is no thread divergence which would have a negative impact on performance. The thread organization is schematically shown in Fig. 12, where it is assumed that each thread handles a single Gauss point. As mentioned earlier, the number of threads of a block should be a power of 2. However, since the number of Gauss points is not necessarily a power of two, there will be a few wasted resources. For example, if there are 32 threads assigned for the computations of 27 Gauss points, 5 threads will not be involved in the computation.

For the most part of this phase all threads of a block are busy (Fig. 13). The exceptions are in the first step, which loads B spline values for each axis into the shared/local memory, and in the last step which rearranges the output values, as described in Section 6.3.

Since each element has its own thread block, all values related to a particular element are stored in the shared/local memory so they can be accessed efficiently by all threads in the block. In particular, all B-spline values and their derivatives (for all axes) that are relevant to the element assigned to this block are needed by each thread for processing all influenced nodes. The constant memory is used for storing values such as the number of influenced control points per element, the number of influenced Gauss points per element, the number of Gauss points per axis in each element, etc. As a result, many calculations are performed with data found in fast memories which is very beneficial from a performance point of view.

6.2. Phase 2 – calculation of the global stiffness coefficients

In the second phase, there are also two levels of parallelism, the major one being on the level of interacting control point pairs and the minor one on the Gauss points. A thread block/group is assigned to each control point pair and each thread of the block handles one Gauss point from the shared elements at a time. This is schematically shown in Fig. 14, where it is assumed that each thread handles only one Gauss point from each of the three shared elements shown.

In this phase, the threads of a block collectively iterate through the shared Gauss points of the node pair and calculate $w_G\mathbf{Q}_{ij}$ submatrices as described in Section 3.2.1. Each thread t of the block sums contributions from a different subset of the shared Gauss points and updates its own partial $\mathbf{K}_{ij}^t$ so there is no need for atomic operations. After all shared Gauss points have been processed, the partial $\mathbf{K}_{ij}^t$ matrices of each thread of the block are summed with a parallel reduction into the final values of the stiffness coefficients $\mathbf{K}_{ij}$ and transferred to the global memory. The process is shown in Fig. 15. The

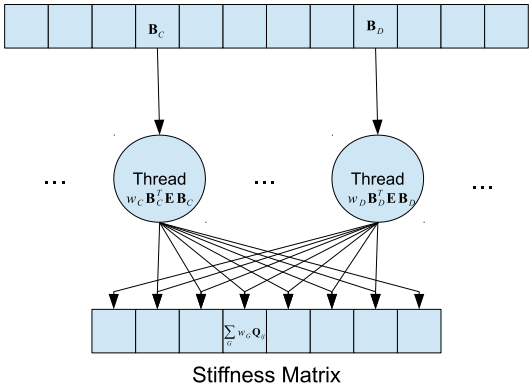

Fig. 9. Scatter parallelism required for the element-wise (EW) approach.

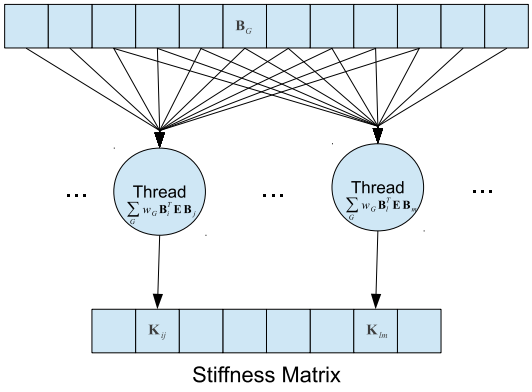

Fig. 10. Gather parallelism implemented in the interaction-wise (IW) approach.

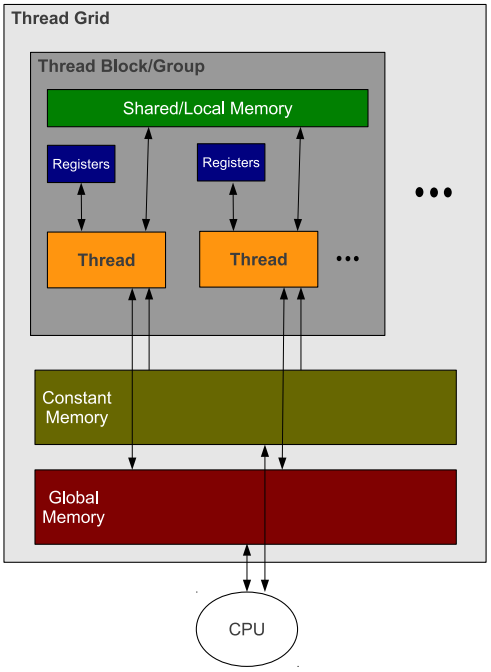

Fig. 11. Visual representation of GPU memory model and scope.

shape function derivatives, which is a large part of the input for phase 2, have been calculated in phase 1. Since they already reside in the GPU, they do not need to be transferred there.

In the last step of this phase, reductions need to be performed in order to calculate the sum of all $\mathbf{K}_{ij}^{t}$ into the final $\mathbf{K}_{ij}$. A serial summation operation can be trivially implemented by writing a simple loop to sum all the parts into the final $\mathbf{K}_{ij}$, but in a parallel environment a single accumulator would create an undesired synchronization point. Therefore, in a parallel implementation, the partial sums $\mathbf{K}_{ij}^{t}$ are gathered by summing pairs of $\mathbf{K}_{ij}^{t}$ concurrently. Each step of this pair-wise summation divides the number of partial sums by half and ultimately produces the final sum after $\log_2 N$ steps, as shown in Fig. 16.

It should be stressed, however, that the reductions required in the proposed implementation are only within a group/block of threads, not on a global GPU level. Furthermore they only exist because of an implementation choice: multiple threads are assigned to a control point pair so reductions are needed to reach the final values for the corresponding pair, but those reductions are isolated within that particular group of threads. If desired, a single thread can be assigned to each pair and this way no reductions would be required. However, a block/group of threads is assigned to each pair due to the amount of work per control point pair and in order to better utilize the intrinsic properties of the GPU. Only the fast shared/local memories are involved in the reductions.

6.3. Memory coalescing

The shape function derivative values are stored in a one dimensional array and are grouped together by element, i.e. all values of the same element are contiguous (Fig. 17). For each Gauss point of the element the derivative of each influenced

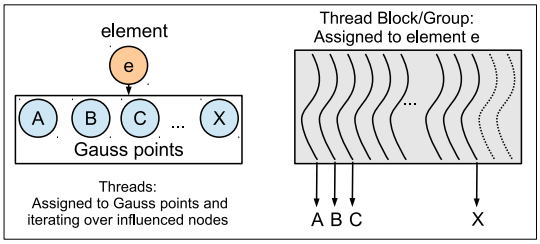

Fig. 12. IW approach: thread organization in phase 1.

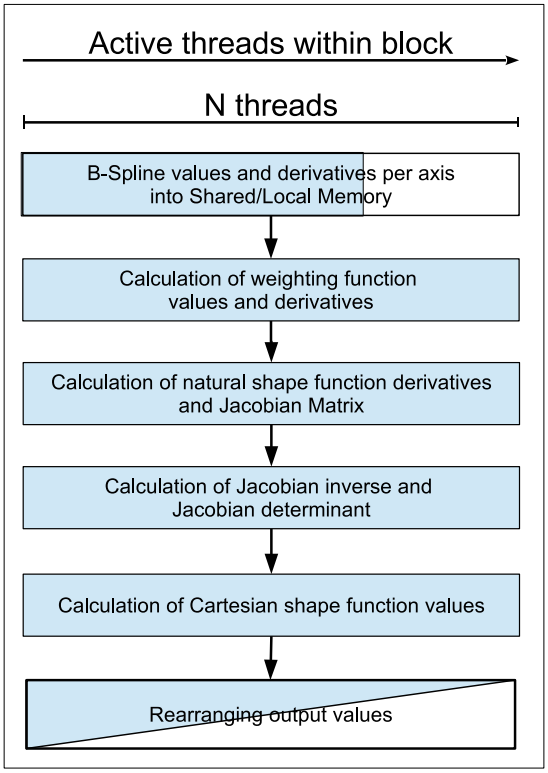

Fig. 13. IW approach: phase 1 – concurrency within a block/group of threads for the calculation of shape function values in the GPU.

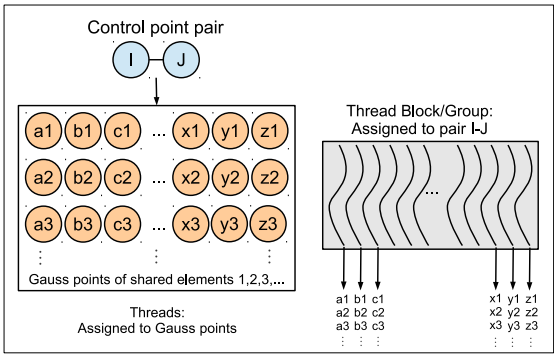

Fig. 14. IW approach: thread organization in phase 2.

control points is calculated. This yields a layout which for an element with 4 Gauss points and 4 control points is depicted in Fig. 18.

The derivative values are needed in phase 2 of the GPU implementation, but the values are accessed by control point in the IW approach instead of by Gauss point as in the EW approach. The IW access pattern is demonstrated in Fig. 19 and shows a GPU performance issue. The values are scattered in various memory locations and therefore reads from consecutive threads are not going to be coalesced. In general, when accessing global memory, peak performance utilization occurs when all threads in a warp access continuous memory locations.

Thus, the desired memory layout for an element with control points and 4 Gauss points is not the one in Fig. 18, but the one depicted in Fig. 20. After the shape function derivative calculation, a transformation between the two memory layouts is performed. The transformation is in essence an in-place matrix transpose and each thread swaps values of row i of the upper triangle with the values of column i of the lower triangle. Thus, threads have different amount of work to do and this is why Fig. 13 has a triangular shape in the last phase.

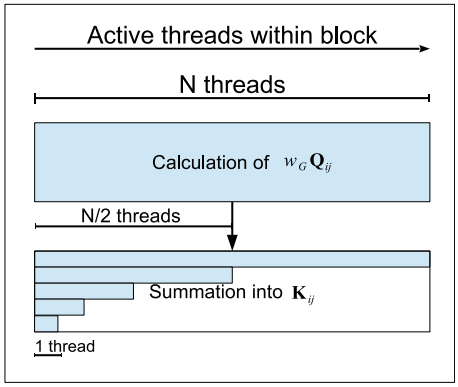

Fig. 15. IW approach: phase 2 – concurrency within a block/group of threads for the calculation of stiffness coefficients in the GPU.

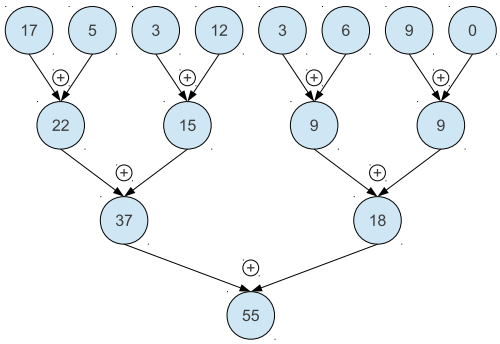

Fig. 16. Parallel summation.

Fig. 17. Element-level memory layout of shape function derivative values.

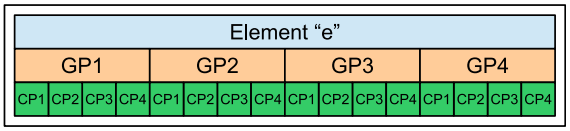

Fig. 18. Memory layout of shape function derivative values of an element “e”. The values are grouped together by Gauss point (GP) and each group contains values for a different control point (CP).

The transformation takes place during phase 1 of the GPU implementation because it has conducive levels of parallelism (block → elements, threads → Gauss points). The resulting layout ensures that consecutive threads access consecutive memory locations as shown in Fig. 21, thus achieving memory coalescence.

6.4. Performance of the GPU implementations of the interaction-wise approach

Tables 12 and 13 show the time needed for the non-coalesced and coalesced GPU implementations, respectively. A GeForce GTX680 with 1536 CUDA cores and 2 GB GDDR5 memory is used. The first phase takes a bit more time in the coalesced version in order to rearrange the values, but the benefits on the second phase are clear. Table 14 shows the speedup ratio between the two versions.

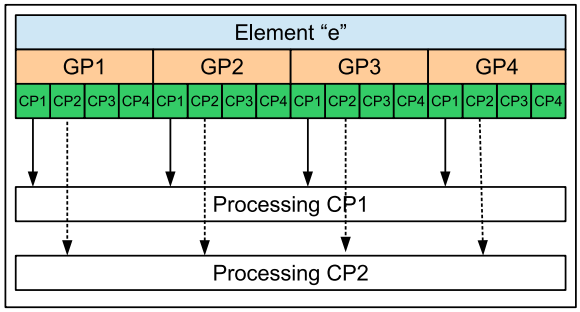

Fig. 19. Non-coalesced access pattern: consecutive threads access non-consecutive memory locations.

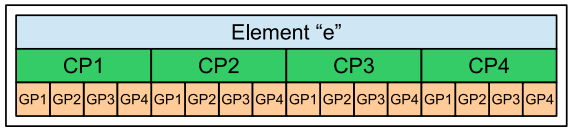

Fig. 20. Memory layout of shape function derivative values of an element “*e*”. The values are grouped together by control point (CP) and each group contains values for a different Gauss point (GP).

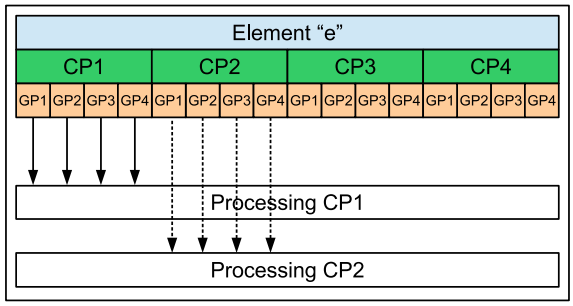

Fig. 21. Coalesced access pattern: consecutive threads access consecutive memory locations.

Table 12
Computing time for the formulation of the stiffness matrix in the non-coalesced GPU implementation of the interaction-wise (IW) approach with a GTX 680.

Example		Dof	IW GPU time (s)		Total
			Kernel 1	Kernel 2	
2D	P2–1	101,250	0.05	0.08	0.1
	P2–2	500,000	0.22	0.36	0.6
	P2–3	801,378	0.34	0.55	0.9
	P3–1	101,250	0.12	0.28	0.4
	P3–2	204,800	0.23	0.56	0.8
	P3–3	301,088	0.34	0.79	1.1
	P4–1	51,200	0.17	0.48	0.6
	P4–2	101,250	0.31	0.90	1.2
	P4–3	151,250	0.44	1.37	1.8
3D	P2–1	20,577	0.06	0.20	0.3
	P2–2	52,728	0.14	0.54	0.7
	P2–3	107,811	0.30	1.13	1.4
	P3–1	20,577	0.28	2.77	3.1
	P3–2	27,783	0.40	3.95	4.3
	P3–3	52,728	0.77	8.27	9.0
	P4–1	10,125	0.37	7.52	7.9
	P4–2	14,739	0.61	12.5	13.1
	P4–3	20,577	0.88	18.9	19.8

Table 13
Computing time for the formulation of the stiffness matrix in the coalesced GPU implementation of the interaction-wise (IW) approach with a GTX 680.

Example		Dof	IW GPU time (s)		Total
			Kernel 1	Kernel 2	
2D	P2–1	101,250	0.05	0.08	0.1
	P2–2	500,000	0.23	0.34	0.6
	P2–3	801,378	0.35	0.53	0.9
	P3–1	101,250	0.13	0.18	0.3
	P3–2	204,800	0.23	0.35	0.6
	P3–3	301,088	0.36	0.51	0.9
	P4–1	51,200	0.21	0.20	0.4
	P4–2	101,250	0.38	0.38	0.8
	P4–3	151,250	0.60	0.54	1.1
3D	P2–1	20,577	0.07	0.10	0.2
	P2–2	52,728	0.18	0.25	0.4
	P2–3	107,811	0.38	0.51	0.9
	P3–1	20,577	0.37	0.56	0.9
	P3–2	27,783	0.51	0.78	1.3
	P3–3	52,728	1.03	1.61	2.6
	P4–1	10,125	0.49	1.07	1.6
	P4–2	14,739	0.80	1.79	2.6
	P4–3	20,577	1.19	2.68	3.9

Table 14
Comparison between the non-coalesced and the coalesced GPU implementations with a GTX 680.

Example		Dof	GPU time (s)		Speedup ratio
			Non-coalesced	Coalesced	
2D	P2–1	101,250	0.1	0.1	1.0
	P2–2	500,000	0.6	0.6	1.0
	P2–3	801,378	0.9	0.9	1.0
	P3–1	101,250	0.4	0.3	1.3
	P3–2	204,800	0.8	0.6	1.4
	P3–3	301,088	1.1	0.9	1.3
	P4–1	51,200	0.6	0.4	1.6
	P4–2	101,250	1.2	0.8	1.6
	P4–3	151,250	1.8	1.1	1.6
3D	P2–1	20,577	0.3	0.2	1.5
	P2–2	52,728	0.7	0.4	1.6
	P2–3	107,811	1.4	0.9	1.6
	P3–1	20,577	3.1	0.9	3.3
	P3–2	27,783	4.3	1.3	3.4
	P3–3	52,728	9.0	2.6	3.4
	P4–1	10,125	7.9	1.6	5.1
	P4–2	14,739	13.1	2.6	5.1
	P4–3	20,577	19.8	3.9	5.1

7. Numerical results for 2D and 3D elasticity problems

Throughout this work, the two approaches for the computation of the stiffness matrix are tested in 2D and 3D elasticity problems. The geometric domains and the parameters of these problems maximize the number of correlations and consequently the computational cost for the given number of control points. The examples are run on the following hardware. CPU: Core i7-980X which has 6 physical cores (12 logical cores) at 3.33 GHz and 12 MB cache. GPU: GeForce GTX680 with 1536 CUDA cores and 2 GB GDDR5 memory. All floating point calculations are in double-precision arithmetic.

Results of the single core element-wise (EW) and interaction-wise (IW) approaches in the CPU have been presented in Sections 3.3 and 4.5 for each of the two approaches, respectively, while results for the GPU implementations of the IW method have been presented in Section 6.4. An overview of the numerical results in this work is provided in Table 15. Speedup ratios of the coalesced GPU implementation compared with the proposed CPU implementations are given in Table 16.

Due to the parallelization features of the IW approach, all available hardware can be utilized: processing can be done by any available CPUs, GPUs or other processing units thanks to the large number of interacting control point pairs and the fact that control point pairs are completely independent. The control point pairs from the domain or all subdomains (if subdomains are used) are gathered in a pool and each processor is fed with groups of pairs to process as long as there are pairs remaining, thus achieving a load balanced implementation [26].

Table 15
Overview of the numerical results obtained with the element-wise (EW) and interaction-wise (IW) approaches in the CPU (single core) as well as with the coalesced GPU implementation of the IW approach.

Example		Dof	Time (s)		
			Element-wise (1-core CPU)	Interaction-wise (1-core CPU)	IW (GPU)
2D	P2–1	101,250	5	4	0.1
	P2–2	500,000	20	17	0.6
	P2–3	801,378	32	27	0.9
	P3–1	101,250	14	12	0.3
	P3–2	204,800	27	23	0.6
	P3–3	301,088	39	34	0.9
	P4–1	51,200	19	16	0.4
	P4–2	101,250	36	31	0.8
	P4–3	151,250	57	47	1.1
3D	P2–1	20,577	8	7	0.2
	P2–2	52,728	21	17	0.4
	P2–3	107,811	43	36	0.9
	P3–1	20,577	59	42	0.9
	P3–2	27,783	83	58	1.3
	P3–3	52,728	168	119	2.6
	P4–1	10,125	131	84	1.6
	P4–2	14,739	218	141	2.6
	P4–3	20,577	333	211	3.9

Table 16
Relative speedup ratios of the coalesced GPU (GTX 680) implementation compared to the single core CPU (Core i7-980X) implementations of the EW and IW approaches.

Example		Dof	Speedup ratios of GPU impl.	
			EW	IW
2D	P2–1	101,250	39	33
	P2–2	500,000	36	30
	P2–3	801,378	36	30
	P3–1	101,250	46	39
	P3–2	204,800	46	40
	P3–3	301,088	46	39
	P4–1	51,200	47	40
	P4–2	101,250	49	42
	P4–3	151,250	50	42
3D	P2–1	20,577	46	41
	P2–2	52,728	49	40
	P2–3	107,811	49	41
	P3–1	20,577	63	45
	P3–2	27,783	64	45
	P3–3	52,728	64	45
	P4–1	10,125	84	54
	P4–2	14,739	84	54
	P4–3	20,577	86	54

8. Concluding remarks

In the element-wise (EW) approach, the improved calculations and code optimizations in all parts of the process as well as the usage of an efficient sparse matrix format specifically tailored for the formulation phase of the stiffness matrix, achieve good performance while keeping the memory requirements to a minimum. Indexing of the sparse matrix is a factor that can greatly reduce performance, but our implementation handles this efficiently and achieves times almost as good as the sky-line format. The latter requires significantly higher memory, thus inhibiting large-scale applications and making the sparse format a preferable or even a mandatory choice.

The proposed interaction-wise (IW) approach has several benefits over the EW approach. The most important one is its amenability to parallelism especially in massively parallel systems like the GPUs. Each control point pair can be processed separately by any available processor in order to compute the corresponding stiffness submatrix. The interaction-wise approach can be characterized as "embarrassingly parallel" since it requires no synchronization whatsoever between control point pairs.

A GPU implementation is applied to the IW approach offering significant speedups compared with CPU implementations. The granularity of the IW approach offers ample parallelism and results in high hardware utilization which is evidenced by speedup ratios achieved with just one GPU in the test examples presented. The IW approach can be applied as is to any available hardware achieving even lower computing times. This includes using many GPUs, hybrid CPU(s)/GPU(s) implementations and generally any available processing unit. The importance of this feature becomes apparent when considering contemporary and future developments like heterogeneous systems architecture (HSA).

In conclusion, the parametric tests performed in the framework of this study showed that with the proposed implementation along with the exploitation of currently available low cost hardware, the expensive formulation of the stiffness matrix in IGA methods can be reduced by orders of magnitude. The presented IW approach enables the efficient utilization of any available hardware and can accomplish high speedup ratios, which convincingly addresses a shortcoming of isogeometric analysis, making it computationally competitive in solving large-scale problems in computational mechanics.

Acknowledgments

This work has been supported by the European Research Council Advanced Grant "MASTER – Mastering the computational challenges in numerical modeling and optimum design of CNT reinforced composites" (ERC-2011-ADG_20110209). The first author has also been supported by the John Argyris Foundation and the second author by the Alexander S. Onassis Foundation.

References

[1] T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Comput. Methods Appl. Mech. Eng. 194 (39–41) (2005) 4135–4195.
[2] F. Auricchio, F. Calabrò, T.J.R. Hughes, A. Reali, G. Sangalli, A simple algorithm for obtaining nearly optimal quadrature rules for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Eng. 249–252 (2012) 15–27.
[3] T.J.R. Hughes, A. Reali, G. Sangalli, Efficient quadrature for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Eng. 199 (5–8) (2010) 301–313.
[4] J.A. Cottrell, T.J.R. Hughes, Y. Bazilevs, Isogeometric Analysis: Toward Integration of CAD and FEA, first ed., Wiley, 2009.
[5] F. Auricchio, L. Beirão da Veiga, T.J.R. Hughes, A. Reali, G. Sangalli, Isogeometric collocation for elastostatics and explicit dynamics, Comput. Methods Appl. Mech. Eng. 249–252 (2012) 2–14.
[6] D. Schillinger, J.A. Evans, A. Reali, M.A. Scott, T.J.R. Hughes, Isogeometric collocation: cost comparison with Galerkin methods and extension to adaptive hierarchical NURBS discretizations, Comput. Methods Appl. Mech. Eng. 267 (2013) 170–232.
[7] I.C. Kampolis, X.S. Trompoukis, V.G. Asouti, K.C. Giannakoglou, CFD-based analysis and two-level aerodynamic optimization on graphics processing units, Comput. Methods Appl. Mech. Eng. 199 (9–12) (2010) 712–722.
[8] E. Elsen, P. LeGresley, E. Darve, Large calculation of the flow over a hypersonic vehicle using a GPU, J. Comput. Phys. 227 (24) (2008) 10148–10161.
[9] J.C. Thibault, I. Senocak, Accelerating incompressible flow computations with a Pthreads-CUDA implementation on small-footprint multi-GPU platforms, J. Supercomput. 59 (2) (2012) 693–719.
[10] M. De La Asunción, J.M. Mantas, M.J. Castro, Simulation of one-layer shallow water systems on multicore and CUDA architectures, J. Supercomput. 58 (2) (2011) 206–214.
[11] H. Zhou, G. Mo, F. Wu, J. Zhao, M. Rui, K. Cen, GPU implementation of lattice Boltzmann method for flows with curved boundaries, Comput. Methods Appl. Mech. Eng. 225–228 (2012) 65–73.
[12] A. Sunarso, T. Tsuji, S. Chono, GPU-accelerated molecular dynamics simulation for study of liquid crystalline flows, J. Comput. Phys. 229 (15) (2010) 5486–5497.
[13] J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, J. Comput. Phys. 227 (10) (2008) 5342–5359.
[14] E. Wadbro, M. Berggren, Megapixel topology optimization on a graphics processing unit, SIAM Rev. 51 (4) (2009) 707–721.
[15] D. Komatitsch, G. Erlebacher, D. Göddeke, D. Michéa, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, J. Comput. Phys. 229 (20) (2010) 7692–7714.
[16] T. Takahashi, T. Hamada, GPU-accelerated boundary element method for Helmholtz' equation in three dimensions, Int. J. Numer. Methods Eng. 80 (10) (2009) 1295–1321.
[17] G.R. Joldes, A. Wittek, K. Miller, Real-time nonlinear finite element computations on GPU – application to neurosurgical simulation, Comput. Methods Appl. Mech. Eng. 199 (49–52) (2010) 3305–3314.
[18] S. Tomov, J. Dongarra, M. Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems, Parallel Comput. 36 (5–6) (2010) 232–240.
[19] O. Schenk, M. Christen, H. Burkhart, Algorithmic performance studies on graphics processing units, J. Parallel Distrib. Comput. 68 (10) (2008) 1360–1369.
[20] J.M. Elble, N.V. Sahinidis, P. Vouzis, GPU computing with Kaczmarz's and other iterative algorithms for linear systems, Parallel Comput. 36 (5–6) (2010) 215–231.
[21] A. Cevahir, A. Nukada, S. Matsuoka, High performance conjugate gradient solver on multi-GPU clusters using hypergraph partitioning, Comput. Sci. – Res. Develop. 25 (1–2) (2010) 83–91.
[22] C. Cecka, A.J. Lew, E. Darve, Assembly of finite element methods on graphics processors, Int. J. Numer. Methods Eng. 85 (5) (2011) 640–669.
[23] A. Dziekonski, P. Sypek, A. Lamecki, M. Mrozowski, Finite element matrix generation on a GPU, Prog. Electromagn. Res. 128 (2012) 249–265.
[24] A. Dziekonski, P. Sypek, A. Lamecki, M. Mrozowski, Accuracy, memory, and speed strategies in GPU-based finite-element matrix-generation, IEEE Antennas Wirel. Propag. Lett. 11 (2012) 1346–1349.
[25] M.G. Knepley, A.R. Terrel, Finite element integration on GPUs, ACM Trans. Math. Software 39 (2) (2013).
[26] M. Papadrakakis, G. Stavroulakis, A. Karatarakis, A new era in scientific computing: domain decomposition methods in hybrid CPU–GPU architectures, Comput. Methods Appl. Mech. Eng. 200 (13–16) (2011) 1490–1508.
[27] A. Karatarakis, P. Metsis, M. Papadrakakis, GPU-acceleration of stiffness matrix calculation and efficient initialization of EFG meshless methods, Comput. Methods Appl. Mech. Eng. 258 (2013) 63–80.
[28] Sparse matrix, Wikipedia, The free encyclopedia, 2013.
[29] R. Sedgewick, K. Wayne, Algorithms, fourth ed., Addison-Wesley Professional, 2011.
[30] W.W. Hwu, D.B. Kirk, Parallelism scalability, in: Programming and tUning Massively Parallel Systems (PUMPS), Barcelona, 2011.
[31] NVIDIA Corporation, CUDA C Best Practices Guide, in NVIDIA GPU Computing Documentation | NVIDIA Developer Zone, 2012.