Finite Element Analysis of 3D Woven Composites Using Consumer Graphical Processing Units

BORYS DRACH

ABSTRACT

The focus of this paper is application of a Graphical Processing Unit (GPU) based solver to linearly elastic finite element analysis (FEA) of composites with three-dimensional (3D) woven reinforcements. Aspects specific to this material system including local material orientations, high contrast between elastic properties of constituents, large number of degrees of freedom, and simulation runtimes are discussed. Speedups offered by parallelization via GPUs and regularity of structured grids enable matrix-free implementation of FEA, which requires reassembly of the global stiffness at every iteration of solution of the system of linear equations, but in turn significantly reduces memory requirements. This makes linear analysis of composite structures with explicit reinforcement representation (tens of millions of degrees of freedom) possible on personal computers. Potential applications of this procedure include fast calculation of effective properties for design of novel 3D woven architectures and efficient solution of problems with high degrees of material nonlinearity requiring frequent stiffness matrix updates.

INTRODUCTION

Three-dimensional (3D) woven composites can be tailored to combine a unique set of mechanical properties required for a specific application [1]. However, the design process of new woven architectures is challenging due to the vast design space and complicated structure-property relationships. In theory, digital representations of new weave architectures can be generated and used to develop models for full mechanical characterization of new systems via finite element analysis (FEA). The composites community has made a great progress towards making this workflow possible. However, many challenges are yet to be overcome before such an approach is routinely used in engineering practice, especially in the case of composites with high fiber volume fractions.

Borys Drach, New Mexico State University, Department of Mechanical and Aerospace Engineering, 1040 S. Horseshoe Str., Las Cruces, NM 88003, U.S.A.

One of the major challenges is development of realistic (. as-woven") reinforcement geometry. During weaving and processing (e.g. in resin transfer molding), fiber bundles are deformed from contact interactions with each other and manufacturing equipment. As a result, there are substantial differences between the idealized (nominal) and the final woven geometry [2], which affect the mechanical behavior of the composites. In most cases, use of nominal geometry leads to significant overprediction of elastic properties of woven composites [3]. Several geometry generation methods that account for the local deformation of fiber bundles have been proposed to deal with this issue. In the author's opinion, the most promising among them is the digital element method [4], [5], which can be used to simulate the weave process and predict the as-woven geometry. A comprehensive approach to realistic geometry generation based on this methodology is presented in [6], [7]. However, digital element chain method requires input of multiple non-physical constants affecting the final solution, which undermines its predictive power and makes it reliant on reference information such as X-ray computed tomography data.

Another major challenge is related to FEA of digital representations of 3D woven composites. Due to the complicated reinforcement geometry of these composites, the smallest non-repeating portion of the material known as a unit cell (typically several mm in size) often requires hundreds of thousands or millions of elements for accurate geometric representation [8]. Even linearly elastic analysis of a unit cell becomes computationally challenging at such levels of discretization because of large amount of RAM memory required to store the global stiffness matrix. When possible, researchers employ periodic boundary conditions to approximate behavior of a large composite specimen with a single unit cell. Highly non-linear problems such as simulation of progressive damage and materials processing pose yet another obstacle – repeated stiffness matrix updates, which leads to simulation runtimes measured in many hours or even days. As such, there is a need for new computational approaches that would enable faster, more accessible FEA of 3D woven composites without sacrificing fidelity of geometric representation of reinforcement.

Due to their unique architecture graphical processing units (GPUs) offer a promising alternative to central processing units (CPUs) when it comes to large non-linear simulations [9]. Originally developed for acceleration of image rendering on a screen GPUs have evolved into complex systems with thousands of processing cores and gigabytes of memory. Nowadays, GPUs are used for a wide spectrum of applications including physics-based simulations, medical imaging, artificial neural networks, machine learning etc., see for example, review in [10]. Modern general-purpose GPUs are unsurpassed in the single instruction multiple data class of computing – their theoretical performance is measured in trillions of floating-point operations per second. However, their specialized architecture places certain limitations on numerical methods that can be successfully ported to GPUs; in addition, efficient implementations are highly application specific. Due to these challenges, the full potential of GPUs is yet to be realized in commercial engineering software packages including general-purpose FEA solvers.

In this paper, we explore use of GPUs for linearly elastic analysis of a unit cell with orthogonal 3D woven reinforcement and implications of the approach for nonlinear analysis.

METHODOLOGY

GPU Implementation of FEA Solver

Linearly elastic FEA comes down to finding solution of a system of linear equations (SLE) of the form $\mathbf{Ku} = \mathbf{f}$, where $\mathbf{K}$ – is a symmetric positive-definite global stiffness matrix, $\mathbf{u}$ – is the vector of unknown displacements, and $\mathbf{f}$ – is the vector of applied nodal forces. If the system has only a few thousands of equations, it can be solved using direct methods such as Gaussian Elimination. However, for larger systems iterative approaches such as Conjugate Gradient Method (CGM) are typically used. Normally, $\mathbf{K}$ is assembled first then stored and used in the solution. For finite element models with millions of degrees of freedom, which result in millions of equations in the SLE, storage and manipulation of $\mathbf{K}$ requires a lot of memory and computation time.

To minimize memory requirements, matrix-free methods can be used. These methods only require storage of results of matrix-vector products at intermediate steps of an iterative procedure such as CGM. However, since CGM requires many iterations to find a converged solution, global stiffness matrix if not stored explicitly must be reassembled implicitly at every iteration. This dramatically increases computation time on CPUs. GPUs can be used to speed up the global stiffness matrix assembly process by hundreds of times. As a result, an efficient GPU FEA implementation has the potential to be faster and more memory-efficient than traditional CPU solvers. The main reason GPU FEA solvers are not widely used in engineering community is that implementations that would utilize all the benefits of GPUs are highly dependent on specific applications.

GPUs typically have several multiprocessors with each one capable of running thousands of threads simultaneously. This architecture enables GPUs to process independent portions of large data. Numerous strategies have been proposed for efficient parallelization of the matrix assembly computations to avoid race conditions, i.e. simultaneous changes of the same memory location by multiple threads. Such strategies include element-by-element (EbE) assembly with "coloring" (segmentation of the mesh into disjointed fragments that can be processed independently of each other), node-by-node (NbN), and degree of freedom-by-degree of freedom (DbD), see [11].

In addition, there are several kinds of on-chip memory including register and shared, and off-chip memory including global, local, constant and texture. Each memory type has a different latency of read and write operations associated with it. Minimization of low-bandwidth memory operations is one of the main optimization strategies in GPU programming, see for example, [12] for details.

Implementation in this paper uses the Compute Unified Device Architecture (CUDA) parallel computing platform by NVIDIA Corporation, which provides an interface for computation on general purpose NVIDIA GPUs. The approach presented here is based on the matrix-free implementation in [13]. One of the main strategies of the approach to improve performance of the assembly step is use of a fixed (or structured) mesh as opposed to a conformal mesh. This eliminates the need for a large element connectivity matrix and allows to reuse element stiffness matrix for identical elements in the model. It has previously been shown that structured meshes lead to good predictions of elastic properties of 3D woven composites when compared to

conformal meshes [8]. In the current approach, discretization results in each element belonging to either the matrix or the tows – there are no boundary elements defined by a mix of properties of two adjacent phases.

CGM with Jacobi preconditioner is used for solution of the FEA system, see Figure 1 for the iterative algorithm [13]. Scalars tol, k_{max} and k represent the user-controlled tolerance, maximum number of iterations and iteration counter, respectively; vector $\mathbf{u}_0$ represents the initial guess for displacement; vectors $\mathbf{a}, \mathbf{p}, \mathbf{r}$ and scalars α, β, ρ are quantities used in the CGM method; matrix $\mathbf{M}$ is the Jacobi preconditioner matrix, which is simply a diagonal matrix of the components K_{ii} (no summation) of the global stiffness matrix. This form of $\mathbf{M}$ makes finding its inverse as simple as calculating the reciprocal value of each component independently. Note that only non-zero elements of the matrix are stored. Other more efficient preconditioners exist, however, most of them require manipulations of the full stiffness matrix which is not available explicitly in the matrix-free implementation.

Input: $\mathbf{K}, \mathbf{f}, \mathbf{u_0}, tol, k_{max}$
Output: $\mathbf{u}$
Algorithm:
01: $k = 0$
02: $\mathbf{u} = \mathbf{u}_0$
03: $\mathbf{r} = \mathbf{f} - \mathbf{K} \cdot \mathbf{u}$
04: $\mathbf{p} = \mathbf{M}^{-1}\mathbf{r}$
05: $\rho_0 = \mathbf{p}^{\mathrm{T}} \cdot \mathbf{r}$
06: while $\|\mathbf{r}\|_2 > tol \cdot \|\mathbf{f}\|_2$ do
07: $k = k + 1$
08: $\mathbf{a} = \mathbf{K} \cdot \mathbf{p}$
09: $\alpha_k = \frac{\rho_{k-1}}{\mathbf{a}^{\mathbf{T}} \cdot \mathbf{p}}$
10: $\mathbf{u} = \mathbf{u} + \alpha_k \cdot \mathbf{p}$
11: $\mathbf{r} = \mathbf{r} - \alpha_k \cdot \mathbf{a}$
12: $\rho_k = (\mathbf{M}^{-1}\mathbf{r})^{\mathrm{T}} \cdot \mathbf{r}$
13: $\beta_k = \rho_k / \rho_{k-1}$
14: $\mathbf{p} = \mathbf{M}^{-1}\mathbf{r} + \beta_k \cdot \mathbf{p}$
15: if $k = k_{max}$ then
16: break;
17: end
18: end

Figure 1. Conjugate gradient method with Jacobi preconditioner [13].

To minimize the number of data transmissions between the host (CPU) and the device (GPU), most of the one-time preprocessing operations are performed on the host. The program starts by reading the mesh, material properties and material orientations files. Since structured grid approach is used, mesh file represents a simple list of each element's phase in the grid – "0" is used for the matrix phase, positive integers correspond to the position of the material orientation vector of the element in the orientations file and represent the homogenized tow phase. Next, all element

stiffness matrices are calculated on the host and stored as a single vector. The total number of stiffness matrices is equal to the number of distinct material orientations present in the 3D woven tows plus one (for the matrix material). The matrix is assumed to be isotropic, tow elements are treated as transversely isotropic with the out-of-plane material orientation vectors aligned with tow centerlines. Matrix operations required for generation of element stiffness matrices [14] are programmed using Eigen C++ library [15]. The final size of the vector containing all element stiffness matrices affects the amount of memory occupied on the device, which means care must be used to keep the number of tow material orientations to a minimum without sacrificing the fidelity of the woven architecture representation. Once the element stiffness matrices are available, Jacobi preconditioner **M** can be calculated.

Next, all vectors used in the CGM algorithm are initialized and steps 4 and 5 of the algorithm are executed on the host. Matrix-vector product in step 3 and all vector operations in the "while" loop (steps 6 and beyond) are performed on the device. Memory for $\mathbf{a}, \mathbf{f}, \mathbf{M}, \mathbf{p}, \mathbf{r}, \mathbf{u}$ and for vectors containing all element stiffness matrices and composite mesh is allocated on the device and these quantities are copied from the host to the device's global memory. Steps 9, 10, 11, 12 and 14 represent simple vector operations, i.e. dot product, summation, multiplication by a scalar or their combinations. These steps are programmed using the CUDA-accelerated implementation of the Basic Linear Algebra Subprograms (BLAS) aka cuBLAS library [16].

Step 8 requires assembly of the global stiffness matrix, which depends on the physics of the problem, element type, boundary conditions etc. and must be implemented via a custom GPU subroutine (kernel). DbD scheme is used for the assembly to the avoid race condition – each degree of freedom is assigned to a separate thread and is only processed once. Note that in this case, contribution from the same element is added multiple times because an element is connected to multiple degrees of freedom, however, this is not a significant performance issue because all element stiffness matrices are precomputed in advance.

Finally, the scalar operations used for calculation of the convergence criterion are performed on the host.

RESULTS AND DISCUSSION

A model of an orthogonal 3D woven carbon/epoxy composite unit cell of the size $5.08 \times 5.08 \times 4\, mm^3$ is used to illustrate performance of the GPU FEA solver presented above. Reinforcement geometry of the unit cell was extracted from X-ray computed microtomography data and resampled to obtain a structured grid finite element mesh of the size $126 \times 126 \times 100$ (1,587,600 total) hexahedral elements, see Figure 2. Volume fraction of the tows in the unit cell is 67.6%. Elastic properties of the RTM6 epoxy matrix and homogenized tows (20% RTM6 epoxy matrix and 80% IM7 carbon fiber by volume) are provided in Table I. Note that local material orientations aligned with the tow centerlines are applied to all elements in the tows. Details of the model preparation approach are presented in [8].

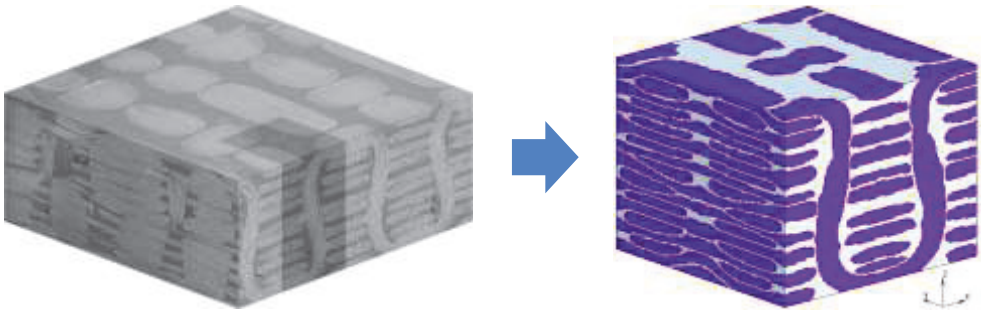


Figure 2. (a) X-ray computed microtomography data of an orthogonal 3D woven composite; (b) structured finite element grid generated from the data [8].

Table I. Elastic properties of the matrix and homogenized tow materials.

Material	E_1 (GPa)	E_2 (GPa)	G_{12} (GPa)	ν_{12}	ν_{23}
Matrix: RTM6 epoxy	2.89	-	-	0.350	-
Tow: 20% RTM6 epoxy + 80% IM7 carbon fibers by vol.	221	13.18	7.17	0.350	0.350

In this analysis, displacements of all nodes on the negative x-plane are fixed in all three directions and nodes on the positive x-plane are prescribed a displacement of $0.00508\ mm$ (0.1% strain) in the positive x-direction. For comparison, the same load case is simulated in the commercial FEA software MSC Marc. CASI Iterative solver with tolerance of 10^{-8} is used in Marc (https://www.mscsoftware.com/product/marc). Same tolerance is used in the GPU FEA code analysis.

Figure 3 presents contour plots of the displacement components x, y and z obtained using Marc. Figure 4a shows a comparison of the displacement plots between Marc and the GPU FEA code for the path along the positive x-axis and for the coordinates $y = z = 0$. Absolute errors normalized by the applied displacement (0.00508) are shown in Figure 4b. Path plots look identical. Upon closer examination of the error plots, the greatest error (0.29%) is observed in the component y of displacement. Error in the component x is much lower (0.038%) because both faces of the specimen, i.e. $x = 0\ mm$ and $x = 5.08\ mm$ are prescribed applied displacements in the x-direction. Comparison of reaction forces in the x-direction gives a relative error of 0.71%: MSC Marc predicts a value of $1486.6\ N$, GPU FEA – $1476.0\ N$.

The main advantage of this implementation is computational efficiency. Both simulations are run on an office machine with Windows 10 64-bit operating software, Intel i7-8700 (3.2 GHz) CPU, 32 GB of RAM, and NVIDIA GeForce GTX 1050 Ti 4 GB graphics card. The total runtime of the MSC Marc simulation is 345 seconds: 66 seconds was spent on the stiffness matrix assembly, 120 seconds – on the solution of the SLE. The total runtime of the GPU FEA code is 66 seconds with the assembly step taking on average 32 ms (stiffness matrix is assembled at every iteration). Considering matrix assembly alone, the speed increase achieved by the GPU FEA code is approximately 2,000 times, however, the total runtime is only 5 times shorter because Jacobi is an inefficient preconditioner and it takes GPU FEA many more iterations to

converge to the solution. The commercial code requires approximately 100 iterations, while GPU FEA – over 1000.

In terms of memory, GPU FEA code requires 973 MB of RAM and 400 MB of the dedicated GPU memory. MSC Marc uses approximately 17,000 MB of RAM, more than 17 times of the GPU FEA code requirement.

While linearly elastic analysis of a single composite unit cell is not particularly exciting – as demonstrated it can be performed using a commercial FEA package – the results presented in this paper lead to the following observations:

1) elastic analysis of very large models (tens of millions of elements) is possible on personal computers equipped with dedicated graphics cards if GPU-based matrix-free solvers are used.
2) nonlinear analysis which requires frequent reassembly of the global stiffness matrix can be sped up significantly with a GPU solver – assembly step on GPU takes a small fraction of the time required on CPU. For maximum performance, the complete global stiffness matrix could be stored, and an efficient preconditioner could be used. Note that this would increase the GPU memory requirements of the code.

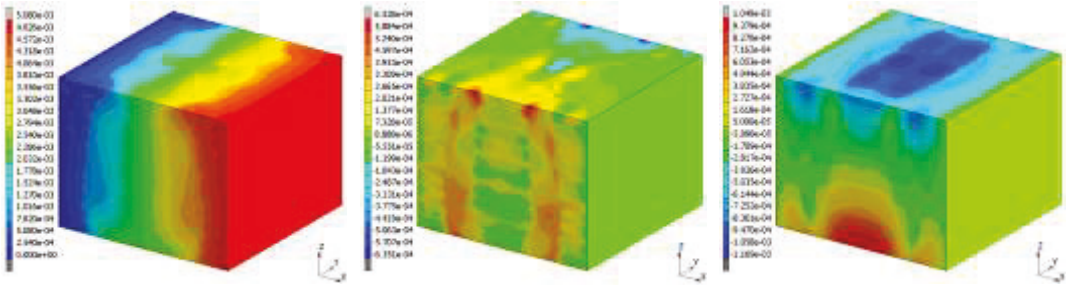
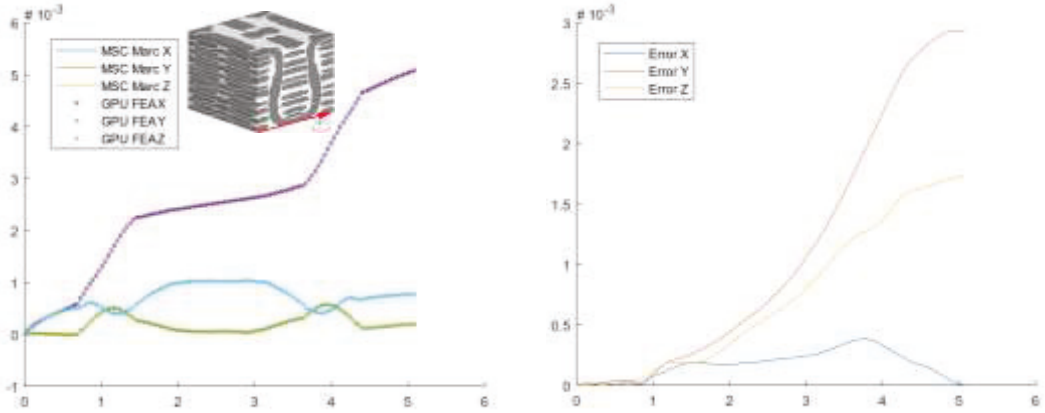


Figure 3. Contour plots of the displacement components x, y and z obtained using commercial FEA software MSC Marc.



(a) (b)

Figure 4. Comparison of displacements sampled along the red arrow from the commercial FEA code MSC Marc (solid lines) and GPU FEA (data points) results: (a) displacement components x, y and z; (b) errors in displacement values normalized by the applied displacement.

CONCLUSIONS

An implementation of a GPU-based FEA solver is presented and used for linearly elastic analysis of a 3D woven composite unit cell. The GPU-based solver offers significant performance improvements over the commercial FEA code: 5 times shorter runtime and 17 times lower RAM requirement. Due to reduced memory requirements, GPU FEA solvers enable linear analysis of models with tens of millions of elements on personal computers and show potential as efficient alternatives to CPU solvers in the case of highly nonlinear analysis where frequent global stiffness matrix updates are required.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation, USA through grant CMMI-1662098.

REFERENCES

[1] L. Tong, A. P. Mouritz, and M. K. Bannister, *3D Fibre Reinforced Polymer Composites*. Elsevier, 2002.

[2] I. Tsukrov *et al.*, "Finite Element Modeling to Predict Cure-Induced Microcracking in Three-Dimensional Woven Composites," *Int. J. Fract.*, vol. 172, no. 2, pp. 209–216, Dec. 2011.

[3] H. Moulinec and P. Suquet, "A numerical method for computing the overall response of nonlinear composites with complex microstructure," *Comput. Methods Appl. Mech. Eng.*, vol. 157, no. 1–2, pp. 69–94, 1998.

[4] G. Zhou, X. Sun, and Y. Wang, "Multi-chain digital element analysis in textile mechanics," *Compos. Sci. Technol.*, vol. 64, no. 2, pp. 239–244, Feb. 2004.

[5] L. Huang, Y. Wang, Y. Miao, D. Swenson, Y. Ma, and C.-F. Yen, "Dynamic relaxation approach with periodic boundary conditions in determining the 3-D woven textile micro-geometry," *Compos. Struct.*, vol. 106, pp. 417–425, Dec. 2013.

[6] A. Drach, B. Drach, and I. Tsukrov, "Processing of fiber architecture data for finite element modeling of 3D woven composites Dedicated to Professor Zdeněk Bittnar in occasion of his 70th birthday.," *Adv. Eng. Softw.*, vol. 72, pp. 18–27, 2014.

[7] B. Drach, A. Drach, I. Tsukrov, M. Penverne, and Y. Lapusta, "Finite Element Models of 3D Woven Composites Based on Numerically Generated Micro-Geometry of Reinforcement," in *Proceedings of the American Society for Composites 2014 - 29th Technical Conference on Composite Materials*, 2014.

[8] A. Ewert, B. Drach, K. Vasylevskyi, and I. Tsukrov, "Predicting the overall response of an orthogonal 3D woven composite using simulated and tomography-derived geometry," *Compos. Struct.*, vol. 243, no. October 2019, p. 112169, 2020.

[9] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[10] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Comput. Surv.*, vol. 47, no. 4, 2015.

[11] J. Martínez-Frutos, P. J. Martínez-Castejón, and D. Herrero-Pérez, "Fine-grained GPU implementation of assembly-free iterative solver for finite element problems," *Comput. Struct.*, 2015.

[12] NVIDIA, "CUDA Toolkit Documentation," 2020. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. [Accessed: 16-Jul-2020].

[13] J. Martínez-Frutos and D. Herrero-Pérez, "Efficient matrix-free GPU implementation of Fixed Grid Finite Element Analysis," *Finite Elem. Anal. Des.*, vol. 104, pp. 61–71, 2015.

[14] D. L. Logan, *A First Course in the Finite Element Method*, 6th ed. Cengage Learning, 2016.
[15] B. Jacob and G. Guennebaud, "Eigen C++ Library." [Online]. Available: http://eigen.tuxfamily.org/.
[16] NVIDIA, "cuBLAS Library: User Guide," 2020. [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html. [Accessed: 16-Jul-2020].