Parallel algorithm for concurrent integration of three-dimensional B-spline functions

Anna Szyszka[1][0000−0002−3179−7863] and Maciej Woźniak[1][0000−0002−5576−5671]

AGH University of Science and Technology, Kraków, Poland
macwozni@agh.edu.pl

Abstract. In this paper, we discuss the concurrent integration applied to the 3D isogeometric finite element method. It has been proven that integration over individual elements with Gaussian quadrature is independent of each other, and a concurrent algorithm for integrating a single element has been created. The suboptimal integration algorithm over each element is developed as a sequence of basic atomic computational tasks, and the dependency relation between them is identified. We show how to prepare independent sets of tasks that can be automatically executed concurrently on a GPU card. This is done with the help of Diekert's graph, which expresses the dependency between tasks. The execution time of the concurrent GPU integration is compared with the sequential integration executed on CPU.

Keywords: Trace Theory · Concurrency · Isogeometric Finite Element Method · Numerical Integration

1 Introduction

The isogeometric analysis (IGA-FEM) [7,8] is a modern technique for the integration of geometrical modeling of CAD systems with engineering computations of CAE systems. The IGA-FEM has multiple applications from phase field modeling [10],shear deformable shell theory [3], wind turbine aerodynamics [21], phase separation simulations [16], in compressible hyperelasticity [12], to turbulent flow simulations [6] and biomechanics [20,5]. IGA-FEM computations consist of two phases: (1) generation of the system of linear equations and (2) execution of an external solver algorithm of the global system of linear equations. The generated system of linear equations for elliptic problems is solved with multifrontal direct solvers [13,14] such as MUMPS [2], SuperLU [22] or PaStiX [18], or iterative solvers. It is possible to obtain linear computational cost for two-dimensional h-refined grids with point or edge singularities [1,15,17].

This work is a summary of [26,24] dealing with concurrent integration. We have parallelized the integration routines of the three-dimensional IGA code. This is done by localizing basic undividable tasks and finding sets of tasks that can be executed concurrently [11]. We concentrate on B-spline basis functions employed in a 3D isogeometric finite element method L^2-projection problem. The presented analysis can be applied to any elliptic problem in 3D with analogous results.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_57

2 Integration algorithm

2.1 Formulation of the model problem

The goal of this study is to evaluate the cost of using different integration methods to build IGA matrices. To illustrate this, we will use the heat equation discretized in time using the forward Euler method and focus specifically on the cost of assembling the Mass matrix. Find $u \in \mathcal{C}^1\left((0,T), H^1(\Omega)\right)$ such that $u = u_0$ at $t = 0$ and, for each $t \in (0,T)$, it holds:

$$\int_\Omega \frac{\partial u}{\partial t} v \, dx = -\int_\Omega \nabla u \cdot \nabla v \, dx, \qquad \forall\, v \in H^1(\Omega). \tag{1}$$

For simplicity, we consider a discrete-in-time version of the problem employing the forward Euler method.

$$\int_\Omega u_{n+1} v \, dx = \int_\Omega u_n v \, dx - \Delta_t \int_\Omega \nabla u_n \cdot \nabla v \, dx, \qquad \forall\, v \in H^1(\Omega). \tag{2}$$

Then, the matrix element is computed as:

$$A^{\alpha}_{\beta,\delta} = \sum_{n_1=1}^{P_1} \sum_{n_2=1}^{P_2} \sum_{n_3=1}^{P_3} \omega^{n_1} \omega^{n_2} \omega^{n_3} \, \Pi(x^n) \, J(x^n), \tag{3}$$

This study focuses on using 3D tensor B-spline basis functions with uniform polynomial degree order and regularity on the interior faces of the mesh for ease of demonstration. However, it should be noted that the techniques presented can be easily adapted to other types of basis functions. For the construction of B-spline basis functions, we used Cox-de-Boor recursive formulae [4].

2.2 Algorithms and computational cost

We compared two algorithms, the classical integration algorithm and the sum factorization algorithm. In the classical integration algorithm, the local contributions to the matrix on the left side A are represented as the sum of the quadrature points. For the classical integration algorithm, the associated computational cost scales, concerning the polynomial degree p as $\mathcal{O}(p^9)$ [19].

After a relatively simple observation, we can reorganize the integration terms of Equation (3). In practice, Equation (3) is written as:

$$A^{\alpha}_{\beta,\delta} = \sum_{n_3=1}^{P_3} \omega_3^n \, B_j(x_3^n) \, B_{m;\,p}(x_3^n) \, C(i_2, i_3, j_2, j_3, k_1), \tag{4}$$

where buffer C is given by

$$C(i_2, i_3, j_2, j_3, k_1) = \sum_{n_2=1}^{P_2} \omega_2^n \, B_i(x_2^n) \, B_{l;\,p}(x_2^n) \underbrace{\sum_{n_1=1}^{P_1} \omega_1^n \, B_h(x_1^n) \, B_{k;\,p}(x_1^n) \, J(x^n)}_{D(i_3, j_3, k_1, k_2)}. \tag{5}$$

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_57

With the practical implementation, we end up with three distinct groups of loops and several buffers. As a consequence, the computational cost associated with sum factorization decreases to $\mathcal{O}(p^7)$ [19].

2.3 Concurrency model for integration

For both algorithms, we used the identical methodology described in [24]. We introduced four basic types of computational tasks.

- Computational tasks that evaluate the 1D basis function on the element E_α at the coordinate of the quadrature point.
- Computational tasks to evaluate the 3D basis function on the element E_α at the coordinate of the quadrature point.
- Computational tasks that evaluate the value of the product of two basis functions (from the test and trial space) on the element E_α at the quadrature point.
- Computational task that evaluates the value of the integral of the scalar product of two base functions (from the test and trial space) on the element E_α.

Next, we define an alphabet of tasks Σ and a set of dependencies between them D.

We also applied methodology to the sum factorization algorithm to obtain optimal scheduling and theoretical verification from the trace theory method. Next, we did a series of numerical experiments to measure parallel performance.

All the tasks mentioned in each layer of the Foata Normal Form are meant to be performed on a homogeneous architecture. All tasks within the particular Foata class should take nearly identical amounts of time, and can be effectively scheduled as a common bag. The method should be very useful with practical implementation for large clusters, such as modern supercomputers [9,25,23] with multiple GPUs per computational node.

2.4 Results

algorithm	t_{serial}	t_{OpenMP}	t_{GPU}
classical	11752.54	1125.72 (12 cores)	5.87
sum factorization	394.28	112.65 (4 cores)	10.78

Table 1:

We observed an unexpected performance behavior of the parallel sum factorization. Despite utilizing parallel loops across all elements, it was scaling only up to 4 cores. Beyond 4 cores, there was a plateau in speedup, indicating poor

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_57

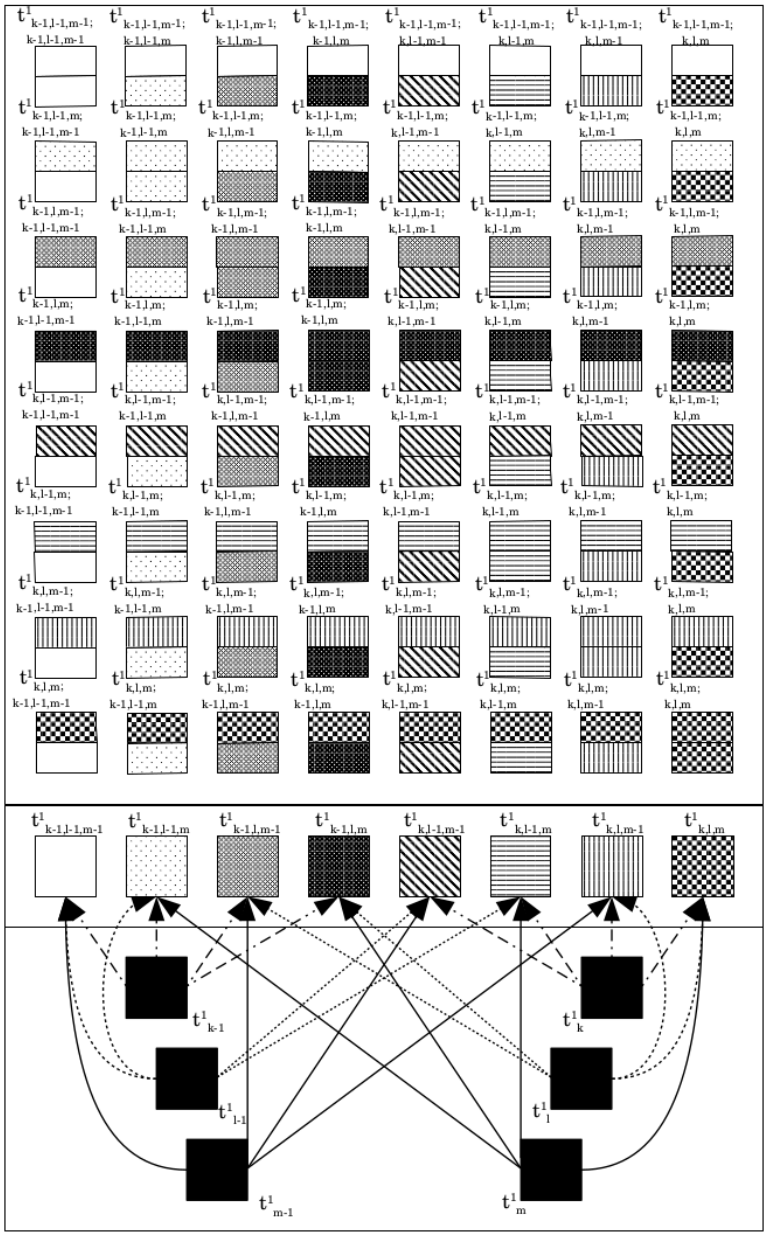

Fig. 1: Part of reduced dependency graph for calculating the frontal matrix using linear basis functions. To maintain the transparency of the chart, the most of relationships are marked with a fill.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_57

performance in a multicore environment. Note that sum factorization requires significantly more memory synchronization compared to the classical method.

We evaluated the performance of classical integration and sum factorization in various scenarios. We focus on $p = 9$ polynomial order basis functions as it was expected to be the most advantageous scenario for sum factorization. We examined three scenarios for a mesh size of : 1) single-core CPU execution, 2) shared-memory CPU computation, and 3) GPU execution. The classical integration on a single core took 11752.54 seconds, the 12-core OpenMP implementation took 1125.72 seconds, and estimated GPU implementation was expected to take 5.87 seconds. For sum factorization integration, single-core execution took 394.28 seconds, 4-core OpenMP implementation took 112.65 seconds, and the estimated GPU implementation was expected to take 10.78 seconds.

The performance of a single core shows that the classical algorithm scales as $\mathcal{O}(p^9)$, while sum factorization as $\mathcal{O}(p^7)$. However, sum factorization does not take advantage of concurrent implementations because of its nature of deep data dependencies. Even in the Figures presented in [26,24] we can observe a single reduction for the classical algorithm, and three such reductions for the sum factorization. In practical implementations, reduction costs $\mathcal{O}(\log n)$, where n is the number of threads / machines / cores from which we reduce. In the case of integration, n would mean the number of quadrature points.

Computations were performed on a Banach Linux workstation equipped with an AMD Ryzen 9 3900X processor, 64GB RAM, a GeForce RTX 2080 SUPER graphic card equipped with 8 gigabytes of memory, and 3072 CUDA cores. The code was compiled with nvcc and gcc, for GPU and CPU, respectively, and -O2 level of optimization.

3 Conclusions

Our approach validates the scheduling for the integration algorithm by utilizing trace theory. We compared the execution of the integration algorithm on a CPU, and a GPU. We can extrapolate its scalability for various elliptic problems. Furthermore, the trace-theory based analysis of concurrency in the integration algorithm can be adapted to different integration methods. The methodology is versatile and can be expanded to include higher-dimensional spaces.

Acknowledgement

The work of Maciej Woźniak was partially financed by the AGH University of Science and Technology Statutory Fund.

References

1. AbouEisha, H., Moshkov, M., Calo, V., Paszyński, M., Goik, D., Jopek, K.: Dynamic programming algorithm for generation of optimal elimination trees for multi-frontal direct solver over h-refined grids. Procedia Computer Science **29**, 947–959 (2014)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_57

2. Amestoy, P.R., Duff, I.S., L'Excellent, J.Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers. Computer Methods in Applied Mechanics and Engineering **184**, 501–520 (2000)
3. Benson, D.J., Bazilevs, Y., Hsu, M.C., Hughes, T.J.R.: A large deformation, rotation-free, isogeometric shell. Computer Methods in Applied Mechanics and Engineering **200**, 1367–1378 (2011)
4. de Boor, C.: Subroutine package for calculating with b-splines. SIAM Journal on Numerical Analysis **14**(3), 441–472 (1971)
5. Calo, V.M., Brasher, N.F., Bazilevs, Y., Hughes, T.J.R.: Multiphysics model for blood flow and drug transport with application to patient-specific coronary artery flow. Computational Mechanics **43**, 161–177 (2008)
6. Chang, K., Hughes, T.J.R., Calo, V.M.: Isogeometric variational multiscale large-eddy simulation of fully-developed turbulent flow over a wavy wall. Computers & Fluids **68**, 94–104 (2012)
7. Cottrell, J.A., Hughes, T.J.R., Bazilevs, Y.: Isogeometric Analysis: Toward Integration of CAD and FEA. John Wiley & Sons, Ltd. (2009)
8. Cottrell, J.A., Hughes, T.J.R., Bazilevs, Y.: Isogeometric Analysis: Toward Integration of CAD and FEA. John Wiley & Sons, Ltd. (August 2009)
9. Cyfronet, https://kdm.cyfronet.pl/portal/Main_page: Cyfronet KDM
10. Dedè, L., Borden, M.J., Hughes, T.J.R.: Isogeometric analysis for topology optimization with a phase field model. Archives of Computational Methods in Engineering **19**, 427–465 (2012)
11. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific (1995)
12. Duddu, R., Lavier, L.L., Hughes, T.J.R., Calo, V.M.: A finite strain eulerian formulation for compressible and nearly incompressible hyperelasticity using high-order b-spline finite elements. International Journal for Numerical Methods in Engineering **89**, 762–785 (2012)
13. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear. ACM Transactions on Mathematical Software **9**, 302–325 (1983)
14. Duff, I.S., Reid, J.K.: The multifrontal solution of unsymmetric sets of linear equations. SIAM Journal on Scientific and Statistical Computing **5**, 633–641 (1984)
15. Goik, D., Jopek, K., Paszyński, M., Lenharth, A., Nguyen, D., Pingali, K.: Graph grammar based multi-thread multi-frontal direct solver with galois scheduler. Procedia Computer Science **29**, 960–969 (2014)
16. Gomez, H., Hughes, T.J.R., Nogueira, X., Calo, V.M.: Isogeometric analysis of the isothermal navier-stokes-korteweg equations. Computer Methods in Applied Mechanics and Engineering **199**, 1828–1840 (2010)
17. Gurgul, P.: A linear complexity direct solver for h-adaptive grids with point singularities. Procedia Computer Science **29**, 1090–1099 (2014)
18. Hénon, P., Ramet, P., Roman, J.: Pastix: A high-performance parallel direct solver for sparse symmetric definite systems. Parallel Computing **28**, 301–321 (2002)
19. Hiemstra, R.R., Sangalli, G., Tani, M., Calabrò, F., Hughes, T.J.: Fast formation and assembly of finite element matrices with application to isogeometric linear elasticity. Computer Methods in Applied Mechanics and Engineering **355**, 234–260 (2019). https://doi.org/https://doi.org/10.1016/j.cma.2019.06.020
20. Hossain, S.S., Hossainy, S.F.A., Bazilevs, Y., Calo, V.M., Hughes, T.J.R.: Mathematical modeling of coupled drug and drug-encapsulated nanoparticle transport in patient-specific coronary artery walls. Computational Mechanics **49**, 213–242 (2012)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_57

21. Hsu, M.C., Akkerman, I., Bazilevs, Y.: High-performance computing of wind turbine aerodynamics using isogeometric analysis. Computers & Fluids **49**, 93–100 (2011)
22. Li, X.S.: An overview of superlu: Algorithms, implementation, and user interface. TOMS Transactions on Mathematical Software **31**, 302–325 (2005)
23. ORNL, https://www.olcf.ornl.gov/summit/: Summit, Oak Ridge National Laboratory
24. Szyszka, A., Woźniak, M., Schaefer, R.: Concurrent algorithm for integrating three-dimensional b-spline functions into machines with shared memory such as gpu. Computer Methods in Applied Mechanics and Engineering **398**, 115201 (2022). https://doi.org/https://doi.org/10.1016/j.cma.2022.115201
25. TACC, https://portal.tacc.utexas.edu/user-guides/stampede2: Stampede2 User Guide
26. Woźniak, M., Szyszka, A., Rojas, S.: A study of efficient concurrent integration methods of b-spline basis functions in iga-fem. Journal of Computational Science **64**, 101857 (2022). https://doi.org/https://doi.org/10.1016/j.jocs.2022.101857

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_57