Finite element (FE) simulations are increasingly valuable in assessing and improving the performance of biomedical devices and procedures. Due to high computational demands such simulations may become difficult or even infeasible, especially when considering nearly incompressible and anisotropic material models prevalent in analyses of soft tissues. Implementations of GPGPU-based explicit FEs predominantly cover isotropic materials, e.g. the neo-Hookean model. To elucidate the computational expense of anisotropic materials, we implement the Gasser-Holzapfel-Ogden dispersed, fiber-reinforced model and compare solution times against the neo-Hookean model. Implementations of GPGPU-based explicit FEs conventionally rely on single-point (under) integration. To elucidate the expense of full and selective-reduced integration (more reliable) we implement both and compare corresponding solution times against those generated using underintegration. To better understand the advancement of hardware, we compare results generated using representative Nvidia GPGPUs from the three recent generations: Fermi (C2075), Kepler (K20c), and Maxwell (GTX980). We explore scaling by solving the same boundary value problem (an extension-inflation test on a segment of human aorta) with progressively larger FE meshes. Our results demonstrate substantial improvements in simulation speeds relative to two benchmark FE codes (up to 300× while maintaining accuracy), and thus open many avenues to novel applications in biomechanics and medicine.

Keywords: Finite element analysis; General purpose graphics processing unit; Anisotropic constitutive model; Gaussian integration

1. Introduction

Finite element (FE) simulations are increasingly valuable in assessing and improving the performance of biomedical devices and procedures, e.g. analyzing the performance of stents (Migliavacca et al. 2002; De Beule et al. 2008), optimizing devices for clamping arteries (Famaey et al. 2012), and studying brain contusions on impact (Horgan and Gilchrist 2003), among countless examples. Due to high computational demands such simulations may become difficult or even infeasible, especially when considering nearly incompressible and strongly anisotropic material models prevalent in computational analyses of soft tissue biomechanics. Additionally, speeding up the run-times of FE simulations for patient-specific studies in computational biomechanics may lead to increased clinical application of simulation tools.

*Corresponding author. Email: vule.strbac@mech.kuleuven.be
Thankfully, computation and memory throughput capacities of modern computing hardware is steadily increasing, loosely following Moore’s law. One such device is the general purpose graphics processing unit (GPGPU), offering considerable increases in arithmetic and memory throughput. GPGPUs have fundamentally different computing architectures than the more common Central Processing Units (CPUs). To take full advantage of the speed GPGPUs provide requires rewriting algorithms developed for CPUs. Adapting existing algorithms to novel computing technologies yields, as shown in this work, significantly reduced simulation times.

1.1 General Purpose Graphics Processing Units (GPGPUs)

GPGPUs emerged from the sphere of computer graphics, developed primarily for intrinsically highly-parallel operations such as rasterization and pixel color determination. In 2006 CUDA (formerly an acronym for Compute Unified Device Architecture) was introduced as a novel parallel computing platform and programming model that made fine-grained parallel programming more accessible. CUDA enabled increased control and efficiency, and thus general programming for GPGPUs increased in popularity. The architecture provides easy access to numerous (thousands) computing cores and to fast multi-layered memory, both of which enable the use of extremely high floating point performance and memory throughput. For more information on CUDA see Nvidia Corp. (2015d,e).

The foundation of parallelization for GPGPUs is distributing the work in small partitions and executing them in parallel. Algorithms are coded and executed through parallel functions called kernels. Contrary to conventional functions running on one or more CPU cores, kernels execute on the GPGPU, utilizing an array of GPU computing cores. The number of cores a kernel can utilize for its execution depends on the hardware (generation and model of a specific GPU), as well as the algorithm the kernel describes. Algorithms are developed in kernels (parallel functions) which execute on an array of computing cores. The numbers of cores and memory types depend heavily on the model and architecture of the GPGPU.

Programming on the CUDA GPGPU is done from the perspective of a single execution thread. The hardware runs and maintains a large number of threads in groups of 32 called warps. Warps are not inter-dependent and may switch their instruction stream over separate computing cores, providing more flexibility. Context-switching is significantly more expensive on a CPU. In order to exploit the fast context switching, sufficient parallelism must be exposed, i.e. the hardware must be saturated with enough warps to enable switching in the case of stalls, thus maximizing throughput by maintaining occupancy (the number of concurrently running warps). Additionally, in order to achieve high occupancy given a finite amount of memory (especially register memory), the memory requirements per thread should be small. Consequently, a small amount of work per-thread is best and leads to favorable performance in fine-grained parallelism – a paradigm at which GPGPUs excel.

Another important consideration in GPGPU programming is minimizing access to off-chip memory (e.g. global memory). Use of off-chip memory space, separated by a shared bus, significantly impacts execution speeds. Global memory transactions refer both to loading necessary data into the kernel and storing resulting data from the kernel back into global memory. Both are expensive and should be used sparingly. However, when data is contiguous in memory, using coalesced accesses, e.g. accessing position data of multiple consecutive nodes (as opposed to random nodes) simultaneously, improves access speeds. Additionally, when a thread’s assigned register memory is insufficient to retain loaded or generated data, the excess data is stored in local memory. This action becomes progressively more expensive as storage needs increase and data is implicitly stored in off-chip memory (register pressure). Obtaining data through global memory transactions, and retaining it through local memory for use in arithmetic or indexing are two important concepts guiding algorithms designed for GPGPUs.
1.2 GPGPU computations for soft tissue biomechanics

Nonlinear finite element (FE) methods are the current benchmark for problems in soft tissue biomechanics. GPGPU-specific algorithms for solving systems of equations based on FE methods can be divided into implicit and explicit FE implementations. As early as 2007, Göddeke et al. tackled issues of double-precision computation in implicit FE on GPUs (Göddeke et al. 2007). They highlighted the indirect risks of indiscriminate use of single-precision computation and presented comparisons among mixed- and double-precision solutions. Cecka et al. analyzed different implementations of implicit FE schemes for GPUs, and strategies for memory storage/access (Cecka et al. 2011). Their work illustrates the challenges and benefits of applying GPUs to implicit FE schemes on unstructured grids. Recently Wong et al. paid specific attention to the sparse matrix-vector multiplication present in implicit FE (Wong et al. 2015). One challenging problem with implicit FE schemes is inversion of the (often) large sparse global stiffness matrix. Due to its generality, this problem has received significant attention in literature leading to optimized implementations available in packages such as cuBLAS (CUDA Basic Linear Algebra Subroutines) (Nvidia Corp. 2015a), cuSPARSE (Nvidia Corp. 2015b), and MAGMA (Dongarra et al. 2014).

Advantageously, explicit FE schemes do not require large matrix inversions or assembly, and thus there are many GPGPU-specific algorithms applying this approach. Early work exploring explicit FE schemes for GPUs was influenced by biomedical applications, e.g. modeling both soft tissues (Picinbono et al. 2003) and surgery (Berkley et al. 2004). The Total Lagrangian Explicit Dynamic (TLED) algorithm by Miller et al. (Miller et al. 2007) established explicit FE in soft tissue biomechanics. This work showed the potential for reduced solution times, even without GPU support, by simply employing methods involving fewer operations. Joldes and coworkers worked on the algorithmic development of TLED for GPGPUs and documented, e.g. handling the hourglassing effect (Joldes et al. 2008), improving convergence through dynamic relaxation (Joldes et al. 2011a; Belytschko and Hughes 1983), and working with different element technologies (Joldes et al. 2011b). For an overview of TLED on (GP)GPUs see Joldes et al. (2009b) and references therein. The TLED algorithm and its (GP)GPU parallelization is also integrated in software suites such as SOFA (Comas et al. 2008) and NiftySim (Johnsen et al. 2014).

Implementations of GPGPU-based explicit FEs in the literature predominantly cover isotropic material models, e.g. linear Hookean (Bartezzaghi et al. 2014), neo-Hookean (Taylor et al. 2008; Joldes et al. 2010), and others (Johnsen et al. 2014), with some exceptions including transverse isotropy, e.g. (Comas et al. 2008) and (Han et al. 2012). However, current computational analyses of soft tissues, e.g. arteries (Famaey et al. 2012; Pierce et al. 2015b,a), tendons (Grytz and Meschke 2009), eyes (Pandolfi and Holzapfel 2008; Pinsky and Datye 1991), and cartilage (Pierce et al. 2013; 2015c), often rely on fiber-reinforced (strongly anisotropic) material models. This is the first comprehensive study implementing and testing the performance of the Gasser-Holzapfel-Ogden (GHO) dispersed fiber-reinforced model (Gasser et al. 2006) within GPGPU-based explicit FEs. To elucidate the relative computational expense of (strongly) anisotropic material models, we compare against corresponding results generated using the neo-Hookean model.

Additionally, implementations of GPGPU-based explicit FE algorithms conventionally use elements relying on Gaussian single-point (under) integration (UI). We extend our custom implementation of the TLED algorithm to include both full integration (FI) and selective-reduced (SR) integration. To elucidate the relative computational expense of these – more reliable but more complex – integration schemes we compare corresponding solution times against those achieved with single-point integration.

Strongly anisotropic material models like GHO rely on strict adherence to near-incompressibility to properly express anisotropic behavior (Helfenstein et al. 2010). This requirement is detrimental to the time-step size in explicit simulations via the well-established Courant-Friedrichs-Lewy condition (Courant et al. 1967; Belytschko 1976), which all explicit simulations must satisfy. Here the implicit approach is much better suited and therefore the convention, especially in conjunction with higher-order integration and the static nature of the solution. Consequently, for a more realistic...
comparison, we use the faster implicit FE approach as reference both for validation and run-time measurements (see also Strbac et al. (2017)).

The steady advancement of GPGPU technology – including double-precision computation, special memory spaces, and other core architectural changes – warrants revisiting the topic of performance-sensitive analyses in soft tissue biomechanics via GPGPU computing, cf. Strbac et al. (2015). To elucidate the steady advancement of hardware, we compare results generated using representative GPGPUs from three generations of Nvidia devices (oldest to newest): Fermi (C2075), Kepler (K20c), and Maxwell (GTX980). In all comparisons we explore the effects of scaling by solving the same boundary value problem with progressively larger FE meshes, i.e. meshes with an increasing number of elements.

2. Materials and Methods

2.1 Custom total Lagrangian explicit dynamic finite element code

We develop a custom explicit dynamic large-strain finite element code completely in house. Our code almost entirely adopts the total Lagrangian worldview, referencing most kinematic values to the initial configuration of the model, which naturally uses total measures of deformation and strain rather than rate forms, thereby reducing truncation error. For detailed pseudo-code of the algorithm refer to Joldes et al. (2010); Strbac et al. (2015).

With FE semi-discretization of the body under consideration, a system of second-order ordinary differential equations emerges, describing the balance of internal and external forces as

\[
[M]\{\ddot{u}\} + q[M]\{\dot{u}\} + \{f^{\text{int}}(\{u\})\} = \{f^{\text{ext}}\},
\]

where \([M]\) is the (diagonalized) mass matrix, \(\{u\}\) is the vector of displacements, \(\{f^{\text{int}}\}\) and \(\{f^{\text{ext}}\}\) are vectors of internal and external forces (respectively), \(q\) is the coefficient of mass-proportional damping, and \(\dot{\cdot}\) and \(\ddot{\cdot}\) are the first and second material time derivatives respectively. Diagonalizing (‘lumping’) the mass matrix decouples the equations and significantly simplifies the solution, essential for both implementation of explicit algorithms and parallelization for GPGPUs.

We use the classic central difference method for time integration. Applying a constant step to the displacements and dropping the bracket notation, we obtain (on a nodal level)

\[
t\dot{u} = \frac{t+\Delta t - t-\Delta t}{2\Delta t} u, \quad t\ddot{u} = \frac{t+\Delta t - 2t + t-\Delta t}{(\Delta t)^2} u,
\]

where the left superscript denotes the time point under consideration. Combining (1) and (2), and solving for the displacements we obtain

\[
t+\Delta t u = a(t^{\text{ext}} - t^{\text{int}}) + b(t u) - c(t-\Delta t u),
\]

with coefficients \(a\), \(b\), and \(c\) as

\[
a = \frac{2(\Delta t)^2}{(2 + q\Delta t)M_e}, \quad b = 1 + \frac{2 - q\Delta t}{2 + q\Delta t}, \quad c = \frac{2 - q\Delta t}{2 + q\Delta t},
\]

and \(q\) as

\[
q = \frac{2(1 - C_r^2)}{\Delta t(1 + C_r^2)}.
\]
Here $q$ is a function of the convergence rate $C_r$ and controlled by the user (see Joldes et al. (2011a)). Elemental masses $M_e$, as well as coefficients $a$, $b$, $c$, and $q$, are constant throughout a simulation, and can thus be precomputed and stored in the global memory prior to simulation.

With the total Lagrangian worldview, we define internal forces within single elements via the nominal (first Piola-Kirchhoff) stress $P_{ji}$ in the initial $\Omega_0$ configuration as

$$f_{Ii}^{\text{int},e} = \int_{\Omega_0} \frac{\partial N_I}{\partial X_j} P_{ji} d\Omega_0,$$

(6)

where capital indices range over the number of nodes and lowercase indices range over the number of spatial dimensions, and employing summation notation. We differentiate the shape functions $N_I$ with respect to the initial configuration described by Lagrangian coordinates $X_j$. These derivatives are time-invariant and can also be precomputed.

For simplicity, we compute external forces in the current (Eulerian) configuration, i.e. we integrate tractions $t_i$ over the current boundary $\Gamma$ as

$$f_{Ii}^{\text{ext},e} = \int_{\Gamma} t_i N_I d\Gamma,$$

(7)

and ignore body forces. We compute internal and external forces using numerical quadrature with different element technologies, see Section 2.3.

### 2.2 Material descriptions

We implement two nonlinear hyperelastic material models based on convex strain-energy functions $\Psi$. To describe the kinematics we employ a multiplicative decomposition of the deformation gradient $F$ using $F = \left(\frac{J}{3} - 1\right)F$, where $J$ is the determinant of $F$. We employ an additive decomposition of $\Psi$ into volumetric and deviatoric contributions as

$$\Psi = U(J) + \Psi(F).$$

(8)

For both material models, we use $U = K(J - 1)^2/2$, where $K$ is the bulk modulus – a material parameter that, in the case of isochoric ($J = 1$) deformation, degenerates to a non-physical (positive) penalty parameter used to enforce incompressibility.

The Cauchy stress tensor $\sigma$ can be derived from the strain-energy functions $\Psi$ as

$$\sigma = J^{-1} \frac{\partial \Psi}{\partial F} F^T.$$

(9)

Decomposition of the Cauchy stresses allows us to calculate the hydrostatic ($\sigma_{ij}^{\text{hyd}}$)

$$\sigma_{ij}^{\text{hyd}} = \frac{1}{3} \sigma_{kk} \delta_{ij},$$

(10)

and deviatoric ($\sigma_{ij}$)

$$\sigma_{ij} = \sigma_{ij} - \sigma_{ij}^{\text{hyd}},$$

(11)

components. We obtain the nominal stress tensor $P$, useful in the total Lagrangian approach, through the transformation

$$P = JF^{-1}\sigma.$$

(12)
2.2.1 Neo-Hookean model

We implement the (isotropic) neo-Hookean model \( \Psi = \Psi^{iso} \), using
\[
\Psi^{iso} = \frac{\mu}{2} (\bar{I}_1 - 3),
\]
(13)
where \( \mu \) is the shear modulus and \( \bar{I}_1 \) is the first invariant of the deviatoric part of the left Cauchy-Green deformation tensor \( \mathbf{B} = J^{-2/3} \mathbf{B} \) where \( \mathbf{B} = \mathbf{F} \mathbf{F}^T \). To integrate stresses, we convert Cauchy stresses to nominal stresses using (12).

2.2.2 Gasser-Holzapfel-Ogden (GHO) model

We implement the Gasser-Holzapfel-Ogden (GHO) model, describing a (dispersed) fiber-reinforced material, \( \Psi = \Psi^{iso} + \Psi^{ani} \), using isotropic (elastin matrix) and anisotropic (reinforcing collagen) contributions respectively (Gasser et al. 2006). The isotropic contribution corresponds to (13), and the anisotropic contribution is
\[
\Psi^{ani} = \sum_{i=4,6} \frac{k_1}{2k_2} \left[ e^{k_2(\kappa \bar{I}_i + (1-3\kappa)\bar{I}_i - 1)^2} - 1 \right],
\]
(14)
where \( \bar{I}_4 = \bar{B} : \mathbf{M} \otimes \mathbf{M} \) and \( \bar{I}_6 = \bar{B} : \mathbf{M}' \otimes \mathbf{M}' \). The vectors \( \mathbf{M} \) and \( \mathbf{M}' \) denote the principal orientations of fiber reinforcement (e.g. the collagen fibers) in the reference configuration, with \( |\mathbf{M}| = |\mathbf{M}'| = 1 \). Furthermore, \( k_1 > 0 \) is a stress-like material parameter, \( k_2 \geq 0 \) (cf. Pierce et al. (2015b)) is a dimensionless parameter, while \( \kappa \in [0, 1/3] \) is a dimensionless parameter representing dispersion of the collagen fibers about the principal orientations \( \mathbf{M} \) and \( \mathbf{M}' \). When applied to model arteries, we determine the directions \( \mathbf{M} \) and \( \mathbf{M}' \) symmetrically with respect to the local circumferential direction of the artery, and thus both directions are uniquely defined by the structural parameter \( \varphi \), the angle between the circumferential direction of the artery and the fiber directions that lay in the plane of the tissue (i.e. a plane normal to the local radial direction). Tension/compression discontinuity in \( \Psi \) stems from a conditional statement that the anisotropic contribution \( \Psi^{ani} \) only contributes to the stored strain energy when \( \bar{I}_i > 1, i = 4, 6 \), such that the stress response is only non-zero in tension.

2.3 Element technologies

To calculate internal forces for trilinear hexahedral (‘brick’) elements we use quadrature to integrate element stresses. We implement three types of integration for the brick element: underintegration (UI), full (FI) and selective-reduced integration (SR), the latter two being higher-order quadrature schemes for this element. To apply pressure boundary conditions, we also implement a quadrilateral pressure element, where full integration is exclusively used to integrate the pressure field over and element’s surface, see Appendix A.

To facilitate numerical integration we convert the integral in (6) to one over the element’s natural space (\( \text{\square} \)) as
\[
\int_{\text{\square}} \frac{\partial N_i}{\partial X_j} P_{ij} J^0 \xi d\text{\square},
\]
by introducing \( J^0_\xi = \det(\partial \mathbf{X}/\partial \xi) \), i.e. the volumetric change between the natural and initial configurations, and again where capital indices range over nodal numbers and lowercase indices range over spatial dimensions.
We calculate the deformation gradient $F$ as

$$F_{ij} = \frac{x_i}{X_j} = \delta_{ij} + \frac{\partial N_I}{\partial \xi_k} \frac{\partial X_j}{\partial \xi_k} u_{li},$$  \hspace{1cm} (16)$$

where we obtain the shape function derivatives with respect to reference coordinates by

$$\frac{\partial N_I}{\partial X_j} = \frac{\partial N_I}{\partial \xi_k} \left( \frac{\partial X_j}{\partial \xi_k} \right)^{-1}.$$  \hspace{1cm} (17)

We obtain the last term through the (inexpensive) inversion of

$$\frac{\partial X_j}{\partial \xi_k} = \frac{\partial N_I}{\partial \xi_k} X_{ij}.$$  \hspace{1cm} (18)

Herein we dropped the notation $\bigg|_{\xi(...)}$ for readability of the natural shape function derivatives. In the following sections we use this notation to indicate the location at which we evaluate variables.

### 2.3.1 Underintegrated, trilinear hexahedron

Classically used in explicit FE, (underintegrated) one-point quadrature uses a single point at the centroid of the element to perform the numerical integration. Thus, we perform the integration required in (15) as

$$f_{int,e}^{int,e} = 8 \frac{\partial N_I}{\partial X_j} \bigg|_{\xi_0} P_{ji} \bigg|_{\xi_0} J_0^0 \bigg|_{\xi_0},$$  \hspace{1cm} (19)$$

where $\xi_0$ indicates the location of the single centroidal integration point $\xi_0 = (0, 0, 0)$ in natural coordinates. Note that $8J_0^0 \bigg|_{\xi_0} = V_0$, the volume of the element in the initial configuration.

We compute the shape function derivatives in the initial configurations in the pre-simulation phase and load these data into the force calculation kernel. We also load displacements and thus calculate the deformation gradient through (16).

### 2.3.2 Fully integrated, trilinear hexahedron

Full Gaussian integration in the element is exact $(2 \times 2 \times 2)$ and captures all displacement modes of the brick element. We perform the integration required in (15) as

$$f_{int,e}^{int,e} = \sum_{Q_1=1}^{2} \sum_{Q_2=1}^{2} \sum_{Q_3=1}^{2} \frac{\partial N_I}{\partial X_j} \bigg|_{\xi_Q} P_{ji} \bigg|_{\xi_Q} J_0^0 \bigg|_{\xi_Q},$$  \hspace{1cm} (20)$$

where $\xi_Q = (\xi_1(Q_1), \xi_2(Q_2), \xi_3(Q_3))$, and $(Q_k) = \{ \pm \frac{1}{\sqrt{3}} \}$ with $k = 1, 2, 3$, indicates the locations of eight integration points.

We evaluate stresses and associated terms eight times, substantially increasing the arithmetic cost. Directly implementing this integration using (16) requires loading the shape function derivatives with respect to the initial configuration from global memory eight times, as well as retrieving the displacements, yielding 216 load requests. In the effort to reduce the transaction count, we instead use (18) as our starting point. This approach involves loading $\frac{\partial N}{\partial \xi} \bigg|_{\xi_0}$ and computing
\[ \partial N/\partial \xi \bigg|_{\xi_0} = f \left( \partial N/\partial \xi \bigg|_{\xi_0}, Q_i \right) \], which is only possible in the natural configuration. Additionally, we must load both the Lagrangian \( X_{ij} \) and \( U_{li} \), yielding a reduction in fetch instructions to 96 and an increase in arithmetic operations – a beneficial trade-off for GPGPU computing.

2.3.3 Selectively reduced integrated, trilinear hexahedron

Here, we perform full integration of the deviatoric stress and single-point quadrature of the volumetric stress response over the element volume. For this scheme we perform the integration required in (15) as

\[
\int_{\Omega} f_{ Ji} = \sum_{Q_1=1}^{2} \sum_{Q_2=1}^{2} \sum_{Q_3=1}^{2} \frac{\partial N_{I}}{\partial X_{j}} \bigg|_{\xi_Q} \mathcal{P}_{ji} \bigg|_{\xi_0} J_{0}^{0} + 8 \frac{\partial N_{I}}{\partial X_{j}} \bigg|_{\xi_0} P_{ji}^{hyd} \bigg|_{\xi_0} J_{0}^{0}, \tag{21}
\]

where \( \mathcal{P}_{ji} \) and \( P_{ji}^{hyd} \) are the deviatoric and hydrostatic components of the nominal stresses, respectively, obtained through the corresponding components of the Cauchy stresses (11) and (10) via (12).

2.4 GPGPU hardware generations

Since GPGPUs can differ significantly from generation to generation, we test representative Nvidia devices from three generations (architectures). These are, in order of release: the C2075 (Fermi, GF110 designation, cf. Nvidia Corp. (2012)), the K20c (Kepler, GK110, cf. Nvidia Corp. (2009)), and the GTX980 (Maxwell, GM204, cf. Nvidia Corp. (2014)).

2.5 Numerical experiments

To study the performance of our custom finite element code we perform numerical experiments using a representative boundary value problem (BVP). Motivated by the soft tissue biomechanics of arteries, we select an idealized segment of human aorta undergoing physiologically relevant inflation and axial extension. First we validate our custom FE code by comparing our results against those generated using two well-established (benchmark) FE codes. Next, we probe the effects of (1) material models, (2) element technologies and (3) GPGPU hardware generations on our code’s performance. In this context, examining the effect of mesh size (scaling) provides subtle insights into the performance of the code and facilitates approximate projections of simulation times based on mesh size. Thus, we solve the segment of human aorta using twenty different discretizations of the geometry, resulting in twenty meshes with an increasing number of elements. We measure the solution speed of our custom FE code using a high-precision hardware clock on the GPGPU exposed by CUDA.

2.5.1 Boundary value problem

Our representative boundary value problem models an extension-inflation test on a segment of human aorta mimicking in-vivo loading conditions, i.e. under 5% axial strain (prestretch) and 120 mmHg pressure (systolic blood pressure), cf. Kiousis et al. (2009). To establish the geometry of the aorta we set the inner diameter to 10 mm, and model the three-layered structure of the tissue using the median thicknesses of the intima, media and adventitia (0.68 mm, 0.94 mm and 1.07 mm) experimentally determined from nine human abdominal aortas (Weisbecker et al. 2012). To model each of the three layers we select the median constitutive parameters \( \mu, k_1, k_2, \varphi \) and \( \kappa \) for the intima, media and the adventitia of the human abdominal aorta, as determined from nine tissue samples (mean age 61.0 ± 11 yr), see Table 1 Weisbecker et al. (2012). Recalling
(near-)incompressibility we use $K \gg \mu$ as a user-specified and mathematically motivated penalty parameter. Using numerical experiments we selected $K$ to be three orders of magnitude larger than the matrix shear modulus, cf. Holzapfel et al. (2002).

We control the meshing process to ensure robust results. First, we design the meshes such that the aspect ratio of all elements remains close to unity. Additionally, all meshes contain three radial layers of elements per material layer, i.e. intima, media and adventitia. With this approach, we generate twenty different meshes with varying numbers of elements. The mesh with the lowest number of elements (Mesh 1, also used for validation studies) contains 2,052 hexahedral elements and 228 quadrilateral (pressure) elements, see Fig. 1(a). For the scaling study we generate 19 additional meshes by adding identical elements in the axial (Z) direction, ranging from 2,052/228 to 41,040/4,560 hexahedral/quadrilateral elements, see Fig. 1(b)-(c) for Meshes 10 and 20, respectively.

In terms of boundary conditions, we constrain all models axially at $Z = \min(Z) = 0$ (the left-most faces in Fig. 1) and apply an axial displacement $\Delta Z = 5\%$ at $Z = \max(Z)$ (the right-most faces in Fig. 1). Additionally, to prevent rigid-body rotation of the model, we constrain axial rows of nodes (at $0^\circ$ and $180^\circ$) such that $\Delta X = 0$, and axial rows of nodes (at $90^\circ$ and $270^\circ$) such that $\Delta Y = 0$. Finally, we apply an internal pressure of $p = 120$ mmHg to the inner surface of the aorta (the surface of the intima).

In our custom FE simulations, we impose both the displacement and pressure in 0.001 sec and obtain a static solution relatively quickly via (critical) damping. We control mass-proportional damping using a coefficient of $C_t = 0.997$, cf. (5). We use a significantly reduced time step of $8 \times 10^{-7}$ sec given the (substantial) effective stiffness of the material at full inflation.

2.5.2 Validation

We validate features of our custom FE code by comparison against benchmark codes Abaqus/Standard (Dassault Systèmes) and FEAP (University of California Berkeley). Since neither benchmark code has all of the required features by default, we use Abaqus to validate underintegrated (C3D8R) and selective-reduced (C3D8) elements, and FEAP to validate fully integrated elements (SOLID, FINITE, DISPLACEMENT). Additionally, we implement the neo-Hookean, (13), and GHO, (13) plus (14), models in our custom FE code and validate these using corresponding results in both Abaqus and FEAP.

We run both Abaqus and FEAP using an implicit scheme as is traditional for static problems of this type. In an attempt to standardize the performance of the benchmark codes we attempt
to solve our BVP using a single solution step, simultaneously extending by 5% and pressurizing to 120 mmHg. If the benchmark codes fail to converge, both codes employ the same polynomial loading curve (default in Abaqus, custom addition to FEAP) for reducing the solution step size and re-iterating.

In our custom FE code we monitor the root-mean-square (RMS) error of the displacement field (at all nodes) between our current solution and the previously computed reference solutions from Abaqus or FEAP, as well as the rate of change $\Delta$RMS. We set the termination criteria for all simulations such that RMS $< 0.0005$ mm and $\Delta$RMS $< 0.0001$ mm must both be satisfied. The $\Delta$RMS is included as a condition to preclude the solution from passing through the correct configuration without stabilizing.

### 2.5.3 Comparison of materials

To determine the performance costs of anisotropic material behavior (described by the GHO model), we complete simulations of our BVP using the GHO model with material parameters from Table 1. For context, we repeat the same simulation using the neo-Hookean model (and only $\mu$ from Table 1 for each tissue layer). To compare solution times we run the neo-Hookean simulations to the same number of steps as those with the GHO material. Note that comparing the resulting displacement fields does not have a meaningful interpretation.

### 2.5.4 Comparison of elements

By solving the same BVP using three element types (UI, FI, and SR) and a range of meshes, we compare the accuracy and execution speed of these element technologies. Underintegrated elements may be unreliable for material models including near incompressibility and strong anisotropy, even when employing hourglassing routines. In this light we compare solutions using the corresponding element technologies in the benchmark codes.

### 2.5.5 Comparison of hardware

By solving the same BVP using three generations of Nvidia GPGPUs (C2075, K20c, and GTX980) and a range of meshes, we compare the execution speed of these devices. Since double-precision computing cores are physically separate and fewer in number, the performance of GPGPUs is sensitive to the precision of number representations. Regardless, to eliminate quantization error artifacts and ensure stable solutions, we use double-precision computation throughout.

### 3. Results

#### 3.1 Validation

We run all codes on a single core of a Xeon E5645, 24GB RAM machine running Windows 7. All simulations using our custom FE code achieve results meeting our termination criteria, i.e. RMS $< 0.0005$ mm and $\Delta$RMS $< 0.0001$ when comparing the displacement field to that from the benchmark codes. All simulations stabilize robustly. We show the representative convergence behavior (for Mesh 20) in Fig. 2 where Fig. 2(a) plots the first principal stress on the final configuration, Fig. 2(b) details the evolution of the RMS and $\Delta$RMS convergence criteria as well as the magnitude of applied load, and Fig. 2(c) details the evolution of principal stresses and pressure.

#### 3.2 Comparison of materials

We show the relative cost of including anisotropic material models relevant to soft tissue biomechanics in Fig. 3(d), (e) and (f) using the GTX980, K20C and, C2075 GPGPUs respectively.
Figure 2.: Representative convergence behavior exemplified on Mesh 20: (a) the first principal stress on the final configuration, (b) the evolution of the RMS and ∆RMS convergence criteria as well as the applied load (L. Mag.), and (c) the evolution of principal stresses and pressure.

Figure 3.: Run-time comparisons illustrating ((a), (b) and (c)) general GPU/CPU execution, and ((d), (e) and (f)) the effect of using an anisotropic GHO material compared to an isotropic neo-Hookean model for all integration schemes and GPUs.

These figures illustrate the cost of running the GHO material model relative to the neo-Hookean model as a slow down factor (GHO run-time/neo-Hookean run-time in this context). The response of SR and FI is more pronounced when using the GHO material model. Selective-reduced and fully integrated elements respond similarly, inducing an average penalty of 5% to 35% depending on the GPU – a modest disadvantage considering the amount of additional math and memory operations
required for the GHO model.

### 3.3 Comparison of elements

We show the relative cost of selecting different element technologies, relevant to solving BVPs in soft tissue biomechanics, in Fig. 3(a), (b), and (c) using the GTX980, K20C and C2075 GPGPUs respectively.

These figures illustrate the cost of running different element technologies on the GPGPU relative to those in the benchmark codes as a speed up factor (GPGPU run-time/benchmark run-time of a particular element in this context). As is clear, significant speedups are possible, with the most recent Maxwell generation device performing fastest overall with speedups of: $333 \times$ (UI), $43 \times$ (FI) and $59 \times$ (SR). Different integration schemes did not affect convergence, and all GHO simulations terminate in the same 100-step window, ending at step 4000.

### 3.4 Comparison of hardware

We directly compare the performance of the GTX980, K20C and C2075 GPGPUs when including anisotropic material models relevant to soft tissue biomechanics (and using SR integration, Fig. 4(a)) and when selecting different element technologies (and using the GHO model, Fig. 4(b)).

Figure 4.: Overall performance results showing the cost of (a) anisotropy (GHO vs. neo-Hookean model) and (b) integration (running the most complex SR vs. the simplest UI element technology) vs. number of elements on different GPU hardware.

Figure 4(a) illustrates the performance cost of using different GPGPU hardware and running different material models (using only SR integration) as a slow down factor (run-time with GHO model/run-time with neo-Hookean model in this context). Similarly, Fig. 4(b) illustrates the performance cost of using different GPGPU hardware and running different element technologies (using only GHO model) as a slow down factor (run-time with SR integration/run-time with UI in this context). The most recent Maxwell architecture performs most poorly using the GHO model, with an average of 33% slow down. Conversely, the most costly GPU (K20C) performs best with an average of 11% slow down. Interestingly, (the oldest) Fermi architecture C2075 GPGPU is least affected by the more demanding integration scheme.
4. Discussion

As demonstrated by our results, our explicit FE code designed for GPGPU architectures gives substantial speed ups in solution times. It is important to note that we designed and implemented our custom code using only general GPU programming guidelines. No special architecture-specific considerations are employed. Therefore, all performance benefit or detriment is automatic. Our systematic study of the performance scaling shows a clear linear trend in the solution times, in contrast to implicit solutions. This indicates that larger speed ups are possible when solving larger meshes, i.e. meshes with larger numbers of elements. Our custom FE code performs worse than FEAP for meshes below a threshold size, consistent with basic parallelization principles on very small workloads.

4.1 Validation

We verified both the correctness of implementations and accuracy of solutions through simulation tests with an embedded termination criterion directly linked to reference solutions obtained with benchmark codes Abaqus and FEAP.

4.2 Comparison of materials

Within our custom FE code on the GPGPU, running simulations using the GHO material model (i.e. including fiber reinforcement and dispersion) affects the computation time surprisingly little. Solution times using underintegrated elements are generally affected less, with the exception of those obtained on the K20C – overall the best performing GPGPU. The average slow down (using UI) is approximately 12% on the K20C (Fig. 3(e)), 20% on the GTX980 (Fig. 3(d)) and only 5% on the C2075 (Fig. 3(f)) – a good result given the additional complexity of the GHO model. Overall, the latest Maxwell GTX980 performs most poorly with a range of 10–35% slow down due to anisotropy.

It is important to note that we do not reduce the material properties and especially the incompressibility constraint (here the penalty parameter $K$, and its ratio to $\mu$ always yield an initial Poisson’s ratio of $\nu = 0.4995$) to facilitate computation within the explicit FE approach.

4.3 Comparison of elements

In terms of performance, underintegration is the least accurate but most efficient approach. Given the small number of required arithmetic operations, and more importantly a relatively small number of required memory transactions, this approach allows excellent execution speeds. Lower-order integration, while significantly reducing computational cost, can suffer from instabilities in the displacement field generated from stress-less strain modes. These so-called hourglass modes may be counteracted artificially by applying viscous damping or (generally more effective) stiffness in the appropriate directions. BVPs dominated by bending require additional care because anti-hourglassing algorithms may unnaturally increase the bending stiffness of the elements. Permanent solution distortion is possible in case of inadequate hourglassing parameters [Schulz (1985)] – a problem which is exacerbated by strong anisotropy. Generally such problems are avoided by mesh refinement, resulting in larger numbers of elements [Belytschko et al. (2000)]. However, explicit FE approaches are particularly sensitive to increasing mesh densities because the solution time step must necessarily decrease. Consequently, in some cases the cost of FI elements may be justified by their increased reliability [Hallquist (2006)].

Note that including pressure boundary conditions, and thus pressure elements (A.3), is relatively expensive computationally. Thus, in simulations using underintegrated hexahedral elements, including pressure boundary conditions has the most noticeable impact on performance.
Contrary to the cost of material anisotropy, higher-order integration is substantially more expensive in the GPGPU-based FE code. With the improved accuracy, the fully integrated element is approximately ten times more costly in terms of instructions and twice as costly (conservative) in terms of memory requirements (versus the underintegrated element). In scenarios involving near-incompressibility (as shown here) this formulation often leads to volumetric locking, unnaturally inducing pressure stresses even for purely deviatoric deformations and thus deteriorating the quality of results (Hughes 2000). Bending dominated problems may also suffer from shear locking (Belytschko et al. 2000). Selective-reduced integration aims to fix both hourglassing and volumetric locking associated with other quadrature schemes. This element is the most computationally demanding, requiring nine evaluations of stresses and their integrations. The SR element, with improved robustness and accuracy, is approximately twelve times more costly in arithmetic operations and slightly more costly in memory requirements (versus the UI element).

In simulations requiring a high degree of accuracy, or where underintegration proves unreliable, full and selective-reduced integration provides a solution, albeit with a performance penalty. From a practical perspective, we compute the stress distribution and resulting displacements for a mesh consisting of approximately 40,000 elements within 27 seconds, with solutions effectively identical to those obtained from the benchmark codes. If one accepts the solution accuracy of underintegrated elements, we calculate corresponding solutions within 3.6 seconds. Finally, if one deems single precision accuracy sufficient, we calculate corresponding solutions within 2 seconds (results not shown). For detail on the single- vs. double-precision performance see previous work Strbac et al. (2015).

4.4 Comparison of hardware

Despite slow downs in simulations using anisotropic material models and/or higher-order integration schemes, simulation speeds on newer GPGPU hardware are still greatly increased, with a speed up factor of 59 (versus benchmark code) on the GTX980 using the most time-consuming selectively-reduced integration.

The increase in total solution time for all GPGPUs is roughly linear. Since all meshes require an approximately equal number of solution steps to achieve convergence, we expect the solution times to increase linearly as well. Thus, we see that larger speed ups are possible when solving meshes with larger numbers of elements. Depending on the available memory, scheduling resources and execution configuration, GPUs process work in batches of various sizes. Once a batch size is determined (e.g. 4992 elements are processed at once on the K20C for the internal force kernel), all elements of a given model are processed in one or several batches serially. Consequently, models with a similar element count may fall into the same ‘batch size’ and exhibit similar solution times, as most clearly exemplified in the K20C solutions using consecutive meshes, e.g. 2 and 3, 5 and 6, 7 and 8, and others in Appendix B, Fig. 7.

4.5 Custom TLED FE code for GPGPUs

4.5.1 GPU performance considerations

GPGPUs can differ significantly between generations. Overall architectural decisions, such as: number of single- or double-precision cores, register or shared memory, number of streaming processors or the caching policy/capacity, etc., all impact performance in different ways. For example, Fermi devices contain the highest register count per core (Strbac et al. 2015), making them resilient to a larger increase in register pressure versus competing GPUs, cf. Fig. 4(b). Conversely, register pressure can be reduced with better caching performance, introduced in Kepler (cf. Nvidia Corp. 2009), allowing better comparative results (cf. Fig. 4(a)). For a more complete list of relevant hardware features see Strbac et al. (2015).
Ascribing performance particularities among GPUs to specific architectural features is challenging, especially without detailed profiling, and presently out of scope. Nonetheless, we identify the algorithm as memory-bound, specifically — latency bound. The main performance limiter of our algorithm is the lack of availability in requested data to the computing subsystems, causing stalls and making the total solution times sensitive primarily to the number of registers per core. However, as evidenced here (and previously Strbac et al. (2015)), overall performance increases as GPUs evolve, despite our algorithm’s sensitivity to particular architectural features.

4.5.2 Implementation

Within or custom FE code, the kernel computing internal forces takes up the majority of computation time (> 80%) for computing the deformation gradient and the stress tensor, and integrating the stresses. We run this kernel on a per-element basis. Costly global memory accesses include those associated with displacements, shape function derivatives, initial positions (depending on the integration scheme), element connectivity, and material properties. Retrieval of elements’ displacements depends on connectivity. In unstructured grids this results in slow (non-coalesced) access to non-contiguous memory. An analogous problem appears after we compute the resulting forces and store them in global memory, with the additional difficulty that multiple elements add to the same nodes’ memory addresses. In this light, we perform all aggregation of data from multiple sources using atomic operations so as to avoid race conditions (i.e. multiple threads writing results to a single location simultaneously). Atomic operations are indivisible and, by definition, guarantee only one active executing thread. We circumvent the fact that double-precision atomic operations are not natively supported on the presented GPGPUs, resulting in a penalty on the kernel’s storage performance (cf. Nvidia Corp. (2015d)). Within our custom FE code, we process pressure loads using a (pressure) quadrilateral element, see Appendix A. Similar to the computation of internal forces, this kernel runs on an element level – that of the quadrilateral. We perform all input and (atomic) output operations, including full integration, analogous to the hexahedral elements. Since pressure is a deformation-dependent (follower) load we include special considerations to recompute area normals at each solution step. Deviating from the total Lagrangian approach we calculate tractions in the current configuration \( x \). This approach simplifies the computation, but requires accessing the current positions of relevant nodes. Thus we require an auxiliary kernel performing \( x_i = X_i + U_i \), an inexpensive nodal operation. This kernel is relatively expensive considering the data-dependent input \( x \), the number of fetches for \( N \) and \( \partial N / \partial \xi \), the data-dependent and atomically stored output \( f_{ext,e} \), as well as the required computations (A.3).

In our time-marching phase, the granularity of execution is one node. In this phase, nodes are displaced according to their dynamic behavior, i.e. aggregate total forces \( (f_{ext} - f_{int}) \) and mass.

The central difference scheme requires loading only the current and previous displacements, and the internal and external forces. At this nodal stage, these ingredients are vectors, yielding only 12 values retrieved from memory. This facilitates fast execution and limits overall impact on solution times.

All memory access, aside from the data-dependent displacements, has regular memory locality in our code and is coalesced. Divergence (through the ‘if’ statement in this case) is present only in this kernel, using the GHO material. In this situation, we execute the threads of a warp matching a condition while other threads are idle. Once completed, the opposing threads execute analogously. Given that two fiber families are tested for fiber recruitment individually (cf. 2.2.2), some impact from divergence is present.

Our robust implementation of explicit FE for GPGPU computing, using the concepts described above, opens many avenues for novel applications. A major application space is clinical practice and clinical workflow, including computer aided surgery systems where the state of the soft tissues is predicted inter-operatively in (near) real-time Joldes et al. (2009a) or reported using technologies of augmented reality Katić et al. (2013). Second generation virtual reality surgery simulators also demand fast execution (e.g. Cueto and Chimestra (2014); Courtecuisse et al. (2010)) well suited to
GPGPU-based FE simulations. Another application space is material parameter or geometry identification [Rausch et al. (2013)], sensitivity analyses [Krishnamurthy et al. (2008, 2009)], or anything that requires the systematic execution of a large number of FE simulations.

4.5.3 Opportunities and limitations

Our robust implementation of explicit FE for GPGPU computing, using the concepts described above, opens many avenues for novel applications. A major application space is clinical practice/work flow, including computer-aided surgery systems requiring inter-operative predictions in (near) real time [Joldes et al. (2009a)] or technologies of augmented-reality (Katić et al. 2013). Second-generation virtual-reality surgical simulators also demand fast execution (e.g. Cueto and Chinesta (2014); Courtecuisse et al. (2010)) well-suited to GPGPU-based FE simulations. Additional application spaces include material parameter or geometry identifications (Rausch et al. 2013) and sensitivity analyses (Krishnamurthy et al. 2008, 2009), essentially any that require systematic execution of large numbers of FE simulations.

However, custom GPU approaches to FE simulations exhibit several limitations. Primarily, consistent with explicit FE simulation — relatively ill-formed elements decrease a simulation’s stable step size, which may favor implicit solution schemes (cf. Strbac et al. (2017)). By design GPGPUs favor single-precision computation, whereas the small step size and high stiffness differences among materials shown here favor the use of double-precision. Further, in a complex FE model, inclusion of multiple material models creates difficulties in parallelization due to divergent control flow. Divergence likely forces a serialized implementation and execution over element groups of different material types, thus increasing run-times. Finally, we do not advise higher-order shape functions with our current approach, as these substantially increase both memory and arithmetic needs, thus inducing a disproportionate response in solution time similar to higher-order integration.

Competing interests

We have no competing interests.

Acknowledgment

This research was supported by the Research Foundation Flanders (G.0.932.11 and PDO/12) and the European Commission’s 7th Framework Programme FP7-ICT – under grant agreement No. 601021 (CASCADE).
References


Appendix A – Fully integrated, pressure quadrilateral

We implement pressure boundary conditions using bilinear quadrilateral elements and full (exact) integration. In our custom code the pressure \( p \) generates traction forces \( t_i = -pn_i \) which we insert into (7). To perform the integration over the 2-D surface of a parent element, we need the normal \( n_i \) and the Jacobian (i.e. the area change) from the current and natural coordinates \( \xi_1, \xi_2 \). Conveniently we calculate both terms as

\[
p n d\Gamma = p \frac{\partial x}{\partial \xi_1} \times \frac{\partial x}{\partial \xi_2} d\xi_1 d\xi_2,
\]

where \( J_\xi = |n| \). Inserting (A.1) into (7), and transforming the integral to one over the element’s natural space (cf. [15]), we obtain

\[
f_{\text{ext},e}^{i} = - \int_{-1}^{1} \int_{-1}^{1} p \varepsilon_{ijk} \frac{\partial x_j}{\partial \xi_1} \frac{\partial x_k}{\partial \xi_2} N_I d\xi_1 d\xi_2,
\]

where \( \varepsilon_{ijk} \) is the permutation tensor. Integrating (A.2) using second-order quadrature (full integration) we obtain

\[
f_{\text{ext},e}^{i} = \sum_{Q_1=1}^{2} \sum_{Q_2=1}^{2} p \varepsilon_{ijk} \frac{\partial x_j}{\partial \xi_1} \bigg|_{\xi_q} \frac{\partial x_k}{\partial \xi_2} \bigg|_{\xi_q} N_I \bigg|_{\xi_q}.
\]
Appendix B – Solution times from benchmark and custom codes

For reference, we provide the resulting solution times of the benchmark codes Abaqus and FEAP in Fig. 5 as well as from our custom code in Figs. 6, 7 and 8.

Figure 5.: Reference solution times obtained via benchmark FE codes Abaqus (underintegration - UI, selective-reduced integration - SR) and FEAP (full integration - FI).

Figure 6.: Reference solution times obtained via custom FE code using the C2075 GPGPU.
Figure 7.: Reference solution times obtained via custom FE code using the K20C GPGPU.

Figure 8.: Reference solution times obtained via custom FE code using the GTX980 GPGPU.
List of Tables:

<table>
<thead>
<tr>
<th></th>
<th>$\mu$ [MPa]</th>
<th>$k_1$ [MPa]</th>
<th>$k_2$ [-]</th>
<th>$\kappa$ [-]</th>
<th>$\varphi$ [°]</th>
</tr>
</thead>
<tbody>
<tr>
<td>intima</td>
<td>0.044</td>
<td>10.1</td>
<td>0.0</td>
<td>0.25</td>
<td>40.5</td>
</tr>
<tr>
<td>media</td>
<td>0.028</td>
<td>0.81</td>
<td>12.4</td>
<td>0.18</td>
<td>39.1</td>
</tr>
<tr>
<td>adventitia</td>
<td>0.010</td>
<td>0.38</td>
<td>3.35</td>
<td>0.11</td>
<td>40.6</td>
</tr>
</tbody>
</table>

Table 1.: Median material and structural parameters for the intima, media and adventitia of human abdominal aorta determined from nine tissue samples (mean age 61.0 ± 11 yr) [Weisbecker et al. (2012)].
List of Figures:

(1) Initial configurations of: (a) Mesh 1 (composed of 2,052/228 hexahedral/quadrilateral elements), (b) Mesh 10, and (c) Mesh 20 (composed of 41,040/4,560 elements).

(2) Representative convergence behavior exemplified on Mesh 20: (a) the first principal stress on the final configuration, (b) the evolution of the RMS and ΔRMS convergence criteria as well as the applied load, and (c) the evolution of principal stresses and pressure.

(3) Run-times comparisons illustrating general GPU/CPU execution ((a), (b) and (c)), and the effect of using an anisotropic GHO material compared to an isotropic neo-Hookean model ((d), (e) and (f)) for all integration schemes and GPUs.

(4) The overall performance results showing the cost of (a) anisotropy (GHO vs. neo-Hookean model) and the cost of (b) integration (running the most complex SR vs. the simplest UI element technology) vs. number of elements on different GPU hardware.

(5) Reference solution times obtained via benchmark FE codes Abaqus (underintegration - UI, selective-reduced integration - SR) and FEAP (full integration - FI).

(6) Reference solution times obtained via custom FE code using the C2075 GPGPU.

(7) Reference solution times obtained via custom FE code using the K20C GPGPU.

(8) Reference solution times obtained via custom FE code using the GTX980 GPGPU.