Computationally efficient viscoelastic flow simulation using a Lagrangian-Eulerian method and GPU-acceleration

Journal of Non-Newtonian Fluid Mechanics 279 (2020) 104264

Contents lists available at ScienceDirect

Journal of Non-Newtonian Fluid Mechanics

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

Computationally efficient viscoelastic flow simulation using a Lagrangian-Eulerian method and GPU-acceleration

Simon Ingelsten[a,b,*], Andreas Mark[a], Klas Jareteg[a], Roland Kádár[b], Fredrik Edelvik[a]

[a] Computational Engineering and Design, Fraunhofer-Chalmers Research Centre for Industrial Mathematics, Gothenburg, Sweden
[b] Division of Engineering Materials, Department of Industrial and Materials Science, Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

Keywords:
Non-Newtonian flow
Computational fluid dynamics
Immersed boundary methods
High performance computing

ABSTRACT

A recently proposed Lagrangian-Eulerian method for viscoelastic flow simulation is extended to high performance calculations on the Graphics Processing Unit (GPU). The two most computationally intensive parts of the algorithm are implemented for GPU calculation, namely the integration of the viscoelastic constitutive equation at the Lagrangian nodes and the interpolation of the resulting stresses to the cell centers of the Eulerian grid.

In the original CPU method, the constitutive equations are integrated with a second order backward differentiation formula, while with the proposed GPU method the implicit Euler method is used. To allow fair comparison, the latter is also implemented for the CPU. The methods are validated for two flows, a planar Poiseuille flow of an upper-convected Maxwell fluid and flow past a confined cylinder of a four-mode Phan Thien Tanner fluid, with identical results.

The calculation times for the methods are compared for a range of grid resolutions and numbers of CPU threads, revealing a significant reduction of the calculation time for the proposed GPU method. As an example, the total simulation time is roughly halved compared to the original CPU method. The integration of the constitutive equation itself is reduced by a factor 50 to 250 and the unstructured stress interpolation by a factor 15 to 60, depending on the number of CPU threads used.

1. Introduction

Viscoelastic flows are present in various industrial processes, such as polymer processing, adhesive extrusion and 3D-printing. Hence, the ability to predict such applications with numerical simulation tools is desired. From an industrial point of view, the computational efficiency is particularly important for the simulations to be useful in practice.

Viscoelastic flows are however challenging to model numerically and simulations can be computationally expensive. Viscoelastic fluids may also require models with multiple relaxation modes to predict flows with sufficient accuracy. An increasing number of modes rapidly increases the number of equations to solve and thus also the computational cost.

Various constitutive models can be found in the literature, suitable for different types of viscoelastic fluids. Such models range from the simpler Upper-Convected Maxwell (UCM) and Oldroyd-B models [1] to nonlinear models as the Phan Thien Tanner (PTT) model [2]. Other examples of nonlinear models are the Finitely Extensible Nonlinear Elasticity (FENE) models, such as FENE-P and FENE-CR [3], and the Giesekus model [1].

Common approaches to simulate viscoelastic fluid flow include discretizing the flow equations in the Eulerian frame of reference using either finite volumes [4–6] or finite elements [7,8]. Due to numerical instabilities often arising on moderate to high Weissenberg numbers, commonly referred to as the High Weissenberg Number Problem (HWNP) [9], it is also common to use stabilization methods in the numerical model. Some approaches enhance the ellipticity of the mathematical problem through the diffusive terms in the transport equations. This can be done e.g. with Elastic-Viscous Stress Splitting (EVSS) [10] or both-sides diffusion, for example as proposed by Fernandes et al. [11]. More advanced approaches reduce the stiffness of the constitutive equation through reformulation. Examples are the Positive Definiteness Preserving Scheme (PDPS) by Stewart et al. [12], the Square Root Conformation Representation (SRCR) by Balci et al. [13], and the Log-Conformation Representation (LCR) proposed by Fattal and Kupferman [14,15]. The latter has grown particularly popular. A detailed comparison of the performance for the different approaches can be found in Chen et al. [16].

Another approach is to solve some or all of the equations in the Lagrangian frame of reference. Rasmussen and Hassager developed a La-

* Corresponding author at: Computational Engineering and Design, Fraunhofer-Chalmers Research Centre for Industrial Mathematics, Gothenburg, Sweden.
E-mail address: simon.ingelsten@fcc.chalmers.se (S. Ingelsten).

https://doi.org/10.1016/j.jnnfm.2020.104264
Received 14 October 2019; Received in revised form 14 January 2020; Accepted 9 March 2020
Available online 18 March 2020

grangian method where all equations were solved using finite elements in a deforming mesh [17]. Harlen et al. [18] proposed a Lagrangian-Eulerian method where the constitutive equation was solved in the nodes of a co-deforming mesh and coupled to a Eulerian finite volume solver for momentum and continuity. Halin et al. [19] presented the Lagrangian Particle Method (LPM), which was further refined to the Adaptive Lagrangian Particle Method (ALPM) [20] and the Backwards-tracking Lagrangian Particle Method (BLPM) [21].

In recent years, the overall interest has steadily increased for using Graphics Processing Units (GPU) for numerical simulations. The main reason is the possibility to distribute the calculation over thousands of parallel threads, offering great reduction in computation time. Consequently, various software libraries support different types of numerical computation on the GPU, e.g. cuBLAS [22] for basic linear algebra routines, cuSparse [23] for sparse matrix operations and the AmgX library [24] for solving linear systems of equations.

GPU:s are designed to execute a large amount of identical operations in parallel, and the performance is sensitive to memory usage and allocation. Memory bandwidth is particularly limited. The available memory on a single GPU can also be considerably lower than the typical workstation computer, and memory transfer between the GPU and the computer's RAM memory is a potential bottleneck. Furthermore, the architecture of a GPU constrains the way that algorithms can be constructed for optimal performance, to a higher extent compared to CPU code. One related example is the concept of divergence. All threads in a warp (typically 32), i.e. the smallest group of threads, are required to execute the same machine instruction. Such architectural limitations can clearly cause sub-optimal performance and should be avoided wherever possible. For the stated reasons, the applicability and the possible performance improvements therefore highly depend on the algorithm and the extent to which the calculations can be parallelized.

The implementation of GPU-acceleration for viscoelastic flow simulations has gained limited attention. Bergamasco et al. [25] presented a micro-macro method where a GPU-based Lattice-Boltzmann method for the configuration part of the Fokker-Planck equation on the micro-scale was coupled to a CPU-based finite volume method for the flow on the continuum scale. Feng et al. [26] developed a Smooth Particle Hydrodynamics (SPH) method for viscoelastic flow running on the GPU.

In a previous article, we presented a Lagrangian-Eulerian method for viscoelastic flow simulation [27]. The constitutive equations were solved in Lagrangian fluid elements convected by the flow and the stresses were interpolated to the cell centers of the Eulerian fluid grid using radial basis functions (RBF). Both the calculation of the stresses and the interpolation are naturally parallel operations and should thus be prime candidates for GPU-acceleration. The aim of the current work is therefore to investigate and demonstrate the effects of GPU acceleration on the computational performance, as well as to assess the suitability of our previously proposed method for the same.

In Fig. 1 an example is shown of the fractions of the computational cost for calculating the viscoelastic stresses accounted for by the three most demanding parts of the algorithm. This particular example is for flow of a four-mode PTT fluid past a symmetrically confined cylinder, which is discussed in more detail in the results section. The simulation is performed using four CPU threads with the Lagrangian-Eulerian method. Solving the ODE systems, i.e. the constitutive equation, accounts for roughly 50% of the computational time, followed by 30% for the stress interpolation. As a remark, the stress calculation accounted for 63 percent of the total computational time for this case, where the total time includes solving the momentum and continuity equations. Clearly, decreasing the time-consumption for these operations would significantly improve the performance of the whole simulation algorithm.

The paper is structured as follows. First the governing equations and the numerical method are presented, followed by an overview of the implementations used. It is then validated that the methods produce equivalent results. This is followed by an analysis of the computational performance and improvement. Finally, some conclusions are drawn and future work is discussed.

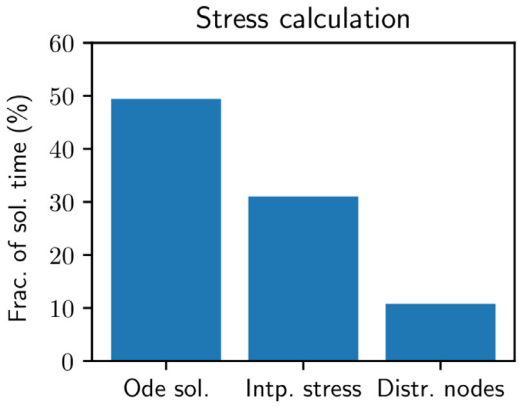

Fig. 1. Typical proportions of the stress calculation for the main components of the solution algorithm, computed for the confined cylinder flow described in detailed in Section 5 using 4 processor threads.

2. Governing equations

The viscoelastic flow is described by the incompressible momentum and continuity equations

$$\rho\Big(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\Big) = -\nabla p + \nabla\cdot(2\mu\mathbf{S} + \boldsymbol{\tau}) + \mathbf{f}, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{2}$$

where ρ is density, $\mathbf{u}$ velocity, p pressure, μ the Newtonian viscosity contribution, $\mathbf{S} = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)$ the strain rate, $\boldsymbol{\tau}$ the viscoelastic stress and $\mathbf{f}$ a body force. The viscoelastic stress is described by a constitutive equation on the form

$$\lambda\overset{\triangledown}{\boldsymbol{\tau}} + F(\boldsymbol{\tau})\boldsymbol{\tau} = 2\eta\mathbf{S}, \tag{3}$$

where λ is the relaxation time, η the polymeric viscosity and $F(\boldsymbol{\tau})$ depends on the constitutive model. Here $\overset{\triangledown}{\boldsymbol{\tau}}$ is the upper-convected derivative of $\boldsymbol{\tau}$,

$$\overset{\triangledown}{\boldsymbol{\tau}} = \frac{D\boldsymbol{\tau}}{Dt} - \nabla\mathbf{u}^T\cdot\boldsymbol{\tau} - \boldsymbol{\tau}\cdot\nabla\mathbf{u}. \tag{4}$$

The first term on the right-hand side is the material time derivative

$$\frac{D\boldsymbol{\tau}}{Dt} = \frac{\partial\boldsymbol{\tau}}{\partial t} + \mathbf{u}\cdot\nabla\boldsymbol{\tau}. \tag{5}$$

The constitutive models used in the current work are the UCM model, i.e. $F\equiv 1$, and the linear form of the PTT model,

$$F(\boldsymbol{\tau}) = 1 + \frac{\varepsilon\lambda}{\eta}\mathrm{Tr}(\boldsymbol{\tau}), \tag{6}$$

where ε is a dimensionless parameter and $\mathrm{Tr}(\boldsymbol{\tau})$ is the trace of $\boldsymbol{\tau}$.

For a multimode viscoelastic model, $\boldsymbol{\tau}$ is the sum of the modal stresses $\{\boldsymbol{\tau}_k\}_{k=1}^N$, i.e.

$$\boldsymbol{\tau} = \sum_{k=1}^{N}\boldsymbol{\tau}_k, \tag{7}$$

where each stress mode $\boldsymbol{\tau}_k$ is described by an equation on the form of (3).

3. Numerical method

The momentum and continuity equations are discretized with the finite volume method on a Cartesian octree grid. The pressure-velocity coupling is solved using the SIMPLEC algorithm. Interior boundaries are

modeled by the mirroring immersed boundary method [28,29], in which the velocity field is implicitly mirrored across the boundary surface such that the no-slip condition is satisfied for the converged solution.

The framework is implemented in IPS IBOFlow® [30], which has been employed for simulation of e.g. conjugated heat transfer [31], fluid-structure interaction [32] and two-phase flows of shear thinning fluids, including seam sealing [33,34], adhesive extrusion [35] and 3D-bioprinting [36].

3.1. Lagrangian-Eulerian stress calculation

Viscoelastic stresses are calculated using a Lagrangian-Eulerian method. In this section a presentation of the method is given in a condensed form. A more detailed description can be found in [27].

The constitutive equation is described in the Lagrangian frame of reference, represented by discrete nodes, or fluid elements, convected by the flow. The nodal position and stress are then described by the ordinary differential equation (ODE) system

$$\begin{cases} \dot{\mathbf{x}} &= \mathbf{u} \\ \dot{\boldsymbol{\tau}}_1 &= G_1(\boldsymbol{\tau}_1, \nabla\mathbf{u}) \\ &\vdots \\ \dot{\boldsymbol{\tau}}_N &= G_N(\boldsymbol{\tau}_N, \nabla\mathbf{u}) \end{cases}, \tag{8}$$

where $\mathbf{x}$ is the node position, $(\dot{\bullet})$ denotes time derivative and the functions $\{G_i\}_{i=1}^N$ follow from (3). The local properties required at the node positions are obtained by trilinear interpolation from the Eulerian fluid grid.

After calculating the stresses at each node, they are interpolated to the cell centers of the Eulerian grid using radial basis functions (RBF). When interpolating the stress to an arbitrary location $\mathbf{x}$, the nodes within a given distance R_s are identified. The interpolated stress components $\hat{\tau}_{ij}$ are then calculated as

$$\hat{\tau}_{ij}(\mathbf{x}) = \sum_{k=1}^{N_c} w_{ij}^{(k)} \phi(\xi|\mathbf{x} - \mathbf{x}^{(k)}|) + P_{ij}(\mathbf{x}), \tag{9}$$

where $w_{ij}^{(k)}$, $k = 1, \dots, N_c$, are the weights corresponding to the N_c neighboring nodes, $\mathbf{x}^{(k)}$ their positions, ϕ a radial basis function and ξ a scaling parameter. The last term is the polynomial

$$P_{ij}(\mathbf{x}) = v_{ij}^{(0)} + \sum_{k=1}^{d} v_{ij}^{(k)} x_k \tag{10}$$

where d is the spatial dimension. The coefficients $\{w_{ij}^{(k)}\}_{k=1}^{N_c}$ and $\{v_{ij}^{(k)}\}_{k=0}^{d}$ are obtained from the system

$$\begin{bmatrix} A & B \\ B^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{g} \\ \mathbf{0} \end{bmatrix} \tag{11}$$

$$A_{kl} = \phi(\xi|\mathbf{x}^{(k)} - \mathbf{x}^{(l)}|), \tag{12}$$

$$B = \begin{bmatrix} 1 & \cdots & 1 \\ \mathbf{x}^{(1)} & \cdots & \mathbf{x}^{(N_c)} \end{bmatrix}^T \tag{13}$$

$$\mathbf{g} = [\boldsymbol{\tau}_{ij}|_{\mathbf{x}^{(k)}} \cdots \boldsymbol{\tau}_{ij}|_{\mathbf{x}^{(N_c)}}]^T. \tag{14}$$

The vector $\mathbf{g}$ is unique for each stress component, while the matrices A and B depend only on the node positions $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N_c)}$ and therefore only needs to be calculated once per interpolation.

3.2. ODE solution

Due to the different inherent properties of CPU:s and GPU:s, different algorithms are suitable for each respective architecture. Classical CPU computations are not necessarily suitable for direct translation into GPU code. Two different algorithms for solving the ODE system (8) are therefore considered.

A global time step Δt refers to a full simulation step, i.e. the step length used for solving the momentum and continuity equations. In the ODE solver, the systems are solved from global time t to $t + \Delta t$ with N_{loc} local steps of length Δt_n, with $n = 1, \dots, N_{\text{loc}}$ and $\Delta t = \Delta t_1 + \cdots + \Delta t_{N_{\text{loc}}}$.

The first method considered is the second order backward differentiation formula (BDF), which was used in the original Lagrangian-Eulerian method in [27]. An approximate solution $\mathbf{y}_n$ at time t_n is then calculated by solving the discretized equation

$$\beta_n \Delta t_n \dot{\mathbf{y}}_n - \mathbf{y}_n + \alpha_1 \mathbf{y}_{n-1} - \alpha_2 \mathbf{y}_{n-2} = \mathbf{0}, \tag{15}$$

where subscript n denotes a property at time t_n and $\Delta t_n = t_n - t_{n-1}$. The coefficients β_n, α_1 and α_2 are uniquely determined given the recent history of the step size [37]. The system (15) is solved using functional iterations. The solution is considered converged when the relative and absolute weighted differences between consecutive solver iterations are below a given tolerance. The number of local time steps is estimated using local error estimation in the solver. The algorithm is executed on the CPU and is implemented in the Sundials CVode package [38]. Details on the solution algorithm can be found in [37].

The second algorithm considered is the implicit Euler method, identified to be more favorable for GPU computation in terms of memory size and bandwidth usage. An approximate solution $\mathbf{y}_n$ is calculated from

$$\mathbf{y}_n = \mathbf{y}_{n-1} + \Delta t_n \dot{\mathbf{y}}_n. \tag{16}$$

The local step length is determined as $\Delta t_n = \Delta t / N_{\text{loc}}$. The system (16) is solved using functional iterations and is considered converged when satisfying

$$R_{\text{rel}}^{(k)} < \varepsilon_{\text{rel}}, \tag{17}$$

$$R_{\text{abs}}^{(k)} < \varepsilon_{\text{abs}} \tag{18}$$

where $R_{\text{rel}}^{(k)}$ and $R_{\text{abs}}^{(k)}$ are the relative and absolute differences, respectively, of the solution vector between the kth and the $(k-1)$th iteration, and ε_{rel} and ε_{abs} are the corresponding tolerances. The differences between iterations are defined as

$$R_{\text{abs}}^{(k)} = \left\| \mathbf{y}_n^{(k)} - \mathbf{y}_n^{(k-1)} \right\|, \tag{19}$$

$$R_{\text{rel}}^{(k)} = \frac{R_{\text{abs}}^{(k)}}{\Delta t_n \|\dot{\mathbf{y}}_n^{(k)}\| + \varepsilon_{\text{abs}}}, \tag{20}$$

where $\|\bullet\|$ denotes the L_2-norm

$$\|\mathbf{r}\| = \sqrt{\sum_{i=1}^{N} r_i^2}, \quad \mathbf{r} \in \mathbb{R}^N. \tag{21}$$

In (20) ε_{abs} is added in the denominator to avoid issues of dividing by zero.

With the implicit Euler method, only the current and the last time step need to be stored simultaneously during the solution step. The memory usage is thus kept to a minimum, as well as the required bandwidth. The implicit Euler method is implemented both for the CPU and the GPU.

3.3. Node distribution

The resolution of the Lagrangian node set is defined relative to that of the Eulerian grid. Let the integer $n_{\text{split}} \geq 1$ be the number of subdivisions of a Cartesian cell in each direction. Upon initialization, each Eulerian cell is divided into n_{split} smaller segments in each coordinate direction such that n_{split}^d sub-volumes are created, where d is the spatial dimension. A node is then created at the center of each sub-volume.

Furthermore, nodes are added and deleted to maintain the quality of the distribution. Let $n_{\max}$ be the maximum allowed number of Lagrangian nodes in a sub-volume. In each global simulation step, nodes are added if a sub-volume is empty or deleted if the number of nodes in a sub-volume exceeds $n_{\max}$.

4. Implementation

The essential components of the implementation of the viscoelastic stress calculation in the numerical code are described in this section. All CPU code is implemented in C++ and the GPU-related code is implemented in CUDA/C++ using the Thrust library [39]. All calculations on both the CPU and the GPU are performed with double precision.

As previously mentioned, the BDF formula to solve the ODE systems is implemented for calculation on the CPU using the Sundials CVode C++ library. The implicit Euler method introduced in the current work is implemented both for calculation on the CPU and the GPU. The numerical integration tolerances are set to 10^{-6} for all methods.

The solution of the ODE systems on the GPU can be summarized as follows:

1. Copy updated position and stresses at Lagrangian nodes to GPU memory.
2. Copy velocities and velocity gradients in the Eulerian grid to GPU memory.
3. Solve ODE systems with the implicit Euler method on the GPU.
4. Copy position and stresses from GPU memory to CPU memory.

The time for copying data between the CPU and the GPU is minimized as the data only needs to be copied to and from the GPU once per global time step. Furthermore, the same data structures are used to store node position and stresses on both units. Hence, a straightforward memory copy can be performed. Unnecessary re-allocation of memory on the GPU is avoided by keeping the state of the Lagrangian nodes in GPU memory between global time steps. Consequently, only nodes that have been modified due to addition or deletion are copied from the CPU to the GPU. Re-allocation of GPU memory therefore only occurs if the stress and position arrays grow larger than the allocated memory.

The unstructured interpolation performed on the CPU is executed as follows. When the stress $\boldsymbol{\tau}$ is required at a cell center, all Lagrangian nodes within a distance $R_s = \sqrt{d}\Delta x$ are found using an R-tree data structure. Here d is the spatial dimension and Δx the local cell size. The choice of R_s implies that all Lagrangian nodes within the cell are included. The system (11) is then solved for each stress component and the interpolated stress $\hat{\boldsymbol{\tau}}$ is obtained. The execution on the GPU is similar. However, the R-tree is replaced by a Cartesian grid structure in which the Lagrangian nodes are sorted by which cell they reside in. The Cartesian grid allows for implementation with very low divergence between the GPU threads, as previously discussed. When the stress is required at a cell center, all nodes residing in the cell are found and used in the interpolation. The system (11) is then solved to obtain the interpolated stress $\hat{\boldsymbol{\tau}}$.

5. Results

The expected accuracy and convergence rate of the Lagrangian-Eulerian method has been assessed for pressure-driven channel flow and the flow around a confined cylinder in [27]. Therefore, it is first shown that the newly proposed implementation produces the same results for these flows. The computational performance is then evaluated for the different methods. For the implicit Euler methods $N_{\text{loc}} = 1$ is used if nothing else is explicitly stated.

5.1. Planar poiseuille flow

A single-mode UCM fluid, i.e $F(\boldsymbol{\tau}) = 1$ and $\mu = 0$, flowing in a planar channel with height $2H$ is simulated to validate the different methods. A constant pressure drop Δp is imposed through Dirichlet pressure conditions at the inlet and the outlet. Periodic boundaries are used for velocity and viscoelastic stress. Lagrangian nodes exiting through a periodic boundary is thus re-entering to the opposite side. The boundary conditions represent a channel of infinite length. No-slip is imposed at the channel walls. The length of the computational domain in the flow direction is H. The analytic solution for the steady flow reads [40]

$$u(y) = \frac{3yU}{2H}\left(2 - \frac{y}{H}\right), \tag{22}$$

$$\tau_{xx}(y) = \frac{18\lambda\eta U^2}{H^2}\left(1 - \frac{y}{H}\right)^2, \tag{23}$$

$$\tau_{xy}(y) = \frac{3\eta U}{H}\left(1 - \frac{y}{H}\right), \tag{24}$$

where y denotes the cross-channel direction and U is the mean velocity. The corresponding pressure gradient reads

$$\frac{\partial p}{\partial x} = \frac{3\eta U}{H^2}. \tag{25}$$

The flow is characterized by the Weissenberg number $\text{Wi} = \lambda U/H$ and the Reynolds number $\text{Re} = \rho U H/\eta$. For all simulations $U = 0.1m/s$ and $\text{Re} = 0.001$. The pressure drop Δp is calculated from (25).

The polymeric viscosity $\eta = [1]$Pas is constant and the Weissenberg number is varied through λ. Both sides diffusion (BSD) with viscosity μ_a is used, which is applicable since the steady flow solution is of interest. The flow and material parameters used, respectively, are summarized in Table 1.

Table 1
Parameter sets simulated for the planar Poiseuille flow.

Wi	λ (s)	$\Delta t/\lambda$	μ_a (Pas)
0.1	0.01	10^{-2}	10^3
1	0.1	10^{-3}	10^4

The simulations are validated by comparing the normal stress, shear stress and velocity across the channel with the analytic solution at steady flow conditions. Since the numerical method is transient by construction, the flow is initiated from rest and simulate until a steady flow is obtained, defined with respect to the condition

$$\frac{||\phi_n - \phi_{n-1}||}{||\phi_n||} < \varepsilon_{\text{tol}}, \tag{26}$$

where ϕ_n is velocity or stress at global time step n and ε_{tol} is a tolerance. In the calculations performed, values below 10^{-10} were obtained for all quantities.

A series of spatial resolutions is used to validate grid convergence. In all cases $n_{\text{split}} = 2$ is used. Further, due to the regularity of the flow nodes are neither added nor deleted. Consequently, all values $n_{\max} > 0$ were found to give exactly the same node sets.

In Fig. 2 the velocity, normal stress and shear stress are shown across the channel for the three methods, obtained for $\text{Wi} = 1$ with the highest spatial resolution $H/\Delta x = 80$. The velocity is normalized by U and the stress with the corresponding analytic wall stresses $\tau_{xx,\,w}$ and $\tau_{xy,\,w}$, respectively. The simulated profiles overlap the analytic solution on the scale of comparison.

A detailed comparison with the analytic solution is performed by calculating the errors

$$E_\phi = \frac{||\phi_s - \phi_a||}{||\phi_a||}, \tag{27}$$

where ϕ_s and ϕ_a denote the simulated and analytic velocity or stress. In Fig. 3 the computed errors with respect to the analytic solution are shown for $\text{Wi} = 1$. The results for $\text{Wi} = 0.1$ are identical and have been omitted, along with the errors for the linear shear stress profile which are below 10^{-9} for all cases. Second order convergence for the velocity and normal stress is found for all cases, which is coherent with the results in previous work [27]. Moreover, the results obtained with the different methods are identical. For this flow it is thus concluded that the implicit Euler method on both the GPU and the CPU along with the

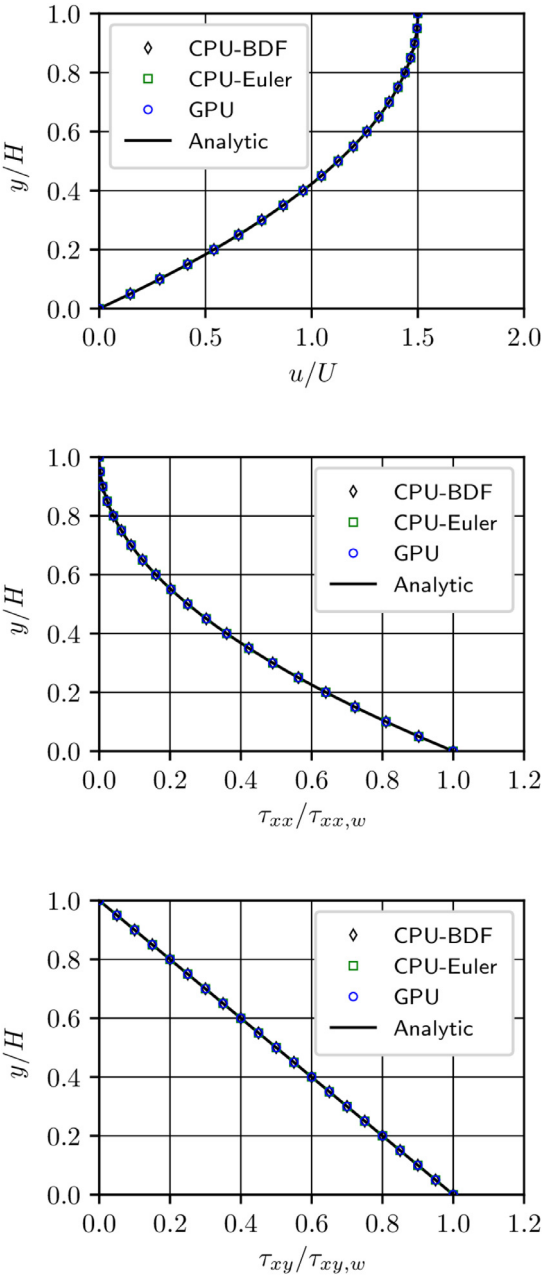

Fig. 2. Simulated fully developed velocity (top), normal stress (middle) and shear stress (bottom) calculated with the UCM model for Wi=1.

RBF interpolation on the GPU gives the same results as the previously validated method.

5.2. Confined cylinder flow

The second case is the flow of a four-mode PTT fluid over a cylinder in a two-dimensional channel. The flow geometry and the model parameters are the same as used by Baaijens et al. [7], which were used to validate the Lagrangian-Eulerian method in previous work [27].

A sketch of the flow geometry is shown in Fig. 4. The cylinder has radius R and is positioned at the channel centerline, at $(x, y) = (0, 0)$. The height of the channel is $4R$.

The flow is characterized by the Deborah number $\mathrm{De} = \bar{\lambda}U/R$ and by $\mathrm{Re} = \rho U R/\eta_0$. The solvent viscosity is $\mu = 0$ for the considered fluid, and the total viscosity η_0 and average relaxation time $\bar{\lambda}$ are respectively

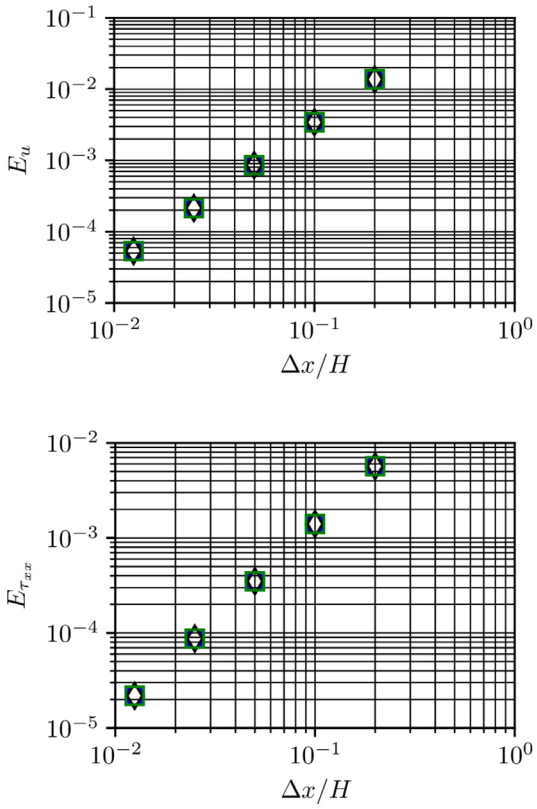

Fig. 3. Computed errors of velocity (top) and normal stress (bottom) with respect to analytical solution for Wi = 1, simulated using GPU (◦), CPU-BDF (⋄,) and CPU-Euler (□).

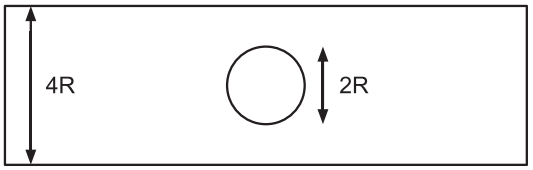

Fig. 4. Symmetrically confined cylinder channel geometry.

calculated from the individual modes as

$$\eta_0 = \sum_{k=1}^{N} \eta_k, \tag{28}$$

$$\bar{\lambda} = \frac{1}{\eta_0} \sum_{k=1}^{N} \eta_k \lambda_k. \tag{29}$$

Fully developed profiles for stress and velocity are imposed at the inlet, calculated from the semi-analytic solution by Cruz and Pinho [41]. At the channel walls the no-slip condition is used and at the outlet ambient pressure is set and velocities are extrapolated in the flow direction.

A uniform grid with cell size $\Delta x = R/40$ is used, which was the resolution used in the previous work. For the node set $n_{\mathrm{split}} = 2$ and $n_{\max} = 5$ are used. Also following the previous work, a small amount of BSD with $\mu_a = \eta_0$ is used.

The flow is simulated with mean inlet velocity $U = 0.1074 m/s$ following Baaijens et al. [7], corresponding to $\mathrm{De} = 2.32$ and $\mathrm{Re} = 0.174$. The PTT model parameters used are summarized in Table 2.

Starting from rest, the flow is advanced in time until a steady flow is obtained. The procedure is repeated for the three methods described in Section 3.2. In Figs. 5, and–7 the velocity, the first normal stress difference $N_1 = \tau_{xx} - \tau_{yy}$ and the shear stress, respectively, computed with

Table 2
Parameters for the PTT model used in the confined cylinder channel flows.

Mode	η(Pas)	λ (s)	ϵ
1	0.443	0.00430	0.39
2	0.440	0.0370	0.39
3	0.0929	0.203	0.39
4	0.00170	3.00	0.39

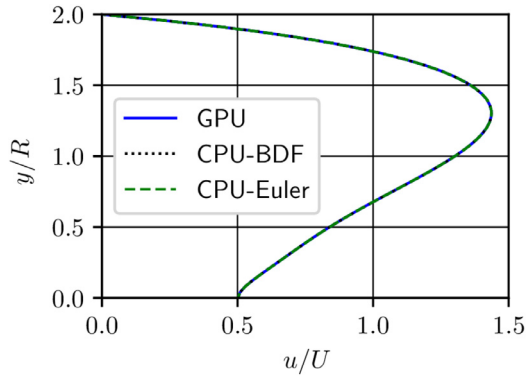

Fig. 5. Computed profiles of velocity across the confined cylinder channel at $x/R = 1.5$.

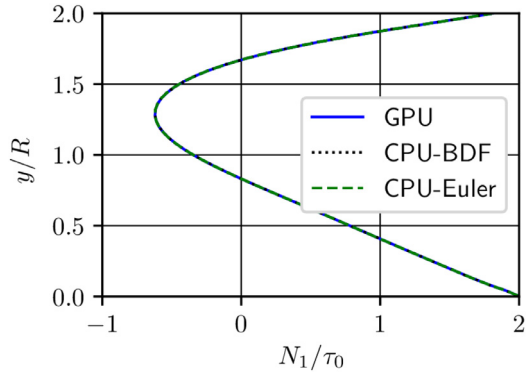

Fig. 6. Computed profiles of the first normal stress difference across the confined cylinder channel at $x/R = 1.5$.

the different methods are shown across the cylinder channel at $x = 1.5$. This location downstream of the cylinder is significant for the flow and should reveal possible discrepancies between the different methods. the velocity is normalized by U and the stresses with $\tau_0 = 3\eta_0 U/R$, following previous work.

In Fig. 8 the normal stress is shown along the channel centerline downstream of the cylinder. The profiles overlap and it is thus evident that the results obtained with the methods introduced in this work are equivalent to those obtained with the previously validated CPU-BDF method.

5.3. Simulation time

The cylinder channel flow is further used to compare calculation times for the different methods. The flow is simulated for 100 time steps and the average elapsed time is measured for the different components of the algorithm. All simulations are performed with an Intel(R) Xeon(R) Gold 6134 CPU with 8 3.20 GHz cores and with a Tesla V100 GPU with 32 Gb memory. The calculations are repeated for different grid resolutions and different numbers of CPU threads. At the highest resolution, $H/\Delta x = 80$, the Eulerian grid has 512 000 cells and the Lagrangian node

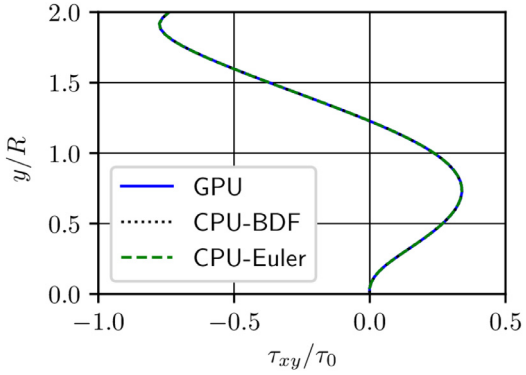

Fig. 7. Computed profiles of shear stress across the confined cylinder channel at $x/R = 1.5$.

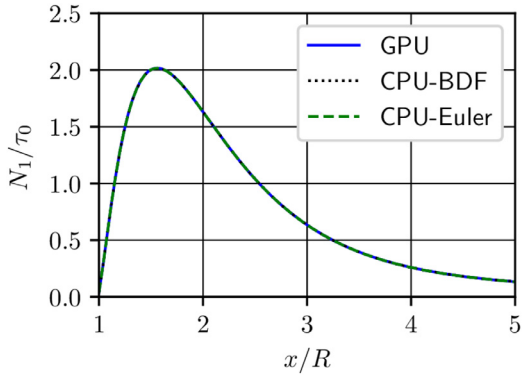

Fig. 8. Computed normal stress difference along the confined cylinder channel centerline.

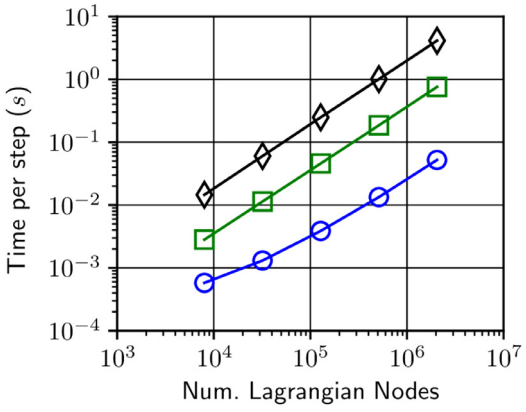

Fig. 9. Average time for solving the ODE systems for the four-mode PTT fluid in the confined cylinder channel for GPU (∘), CPU-BDF (⋄,) and CPU-Euler (□), with 4 CPU threads.

set consists of approximately two million nodes. The memory usage on the GPU for this case is around 400 Mb.

The calculation times for ODE solution and the unstructured stress interpolation are compared. In Fig. 9 the average times for solving the ODE systems (8) are shown for the different methods for varying grid resolution. The simulations were performed using four CPU threads. Compared to the CPU-BDF method there is a clear improvement in using the implicit Euler method for this flow, both on the CPU and the GPU. The computation times for the GPU computations are however much smaller than the times for both CPU methods.

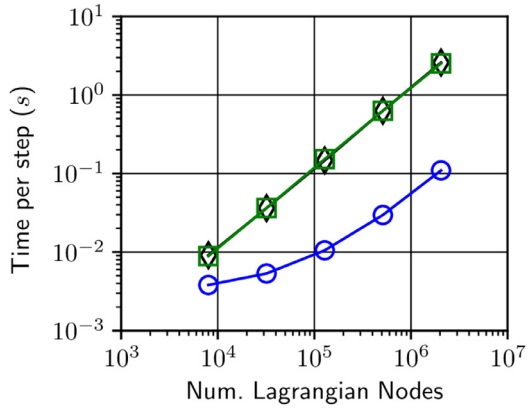

Fig. 10. Average time for interpolating viscoelastic stresses to fluid cell centers for the four-mode PTT fluid in the confined cylinder channel for GPU (◦), CPU-BDF (⋄,) and CPU-Euler (□), with 4 CPU threads.

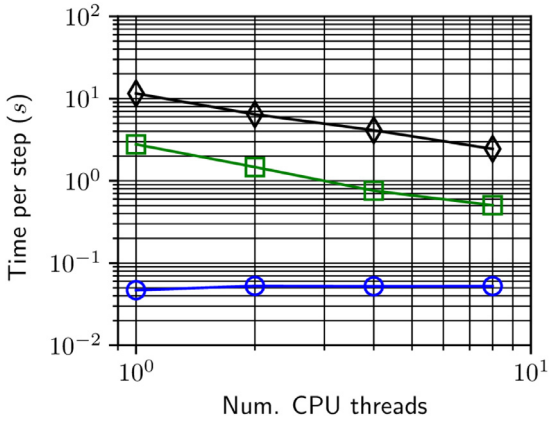

Fig. 11. Average time for solving the ODE systems for the four-mode PTT fluid in the confined cylinder channel for GPU (◦), CPU-BDF (⋄,) and CPU-Euler (□), at the highest grid resolution.

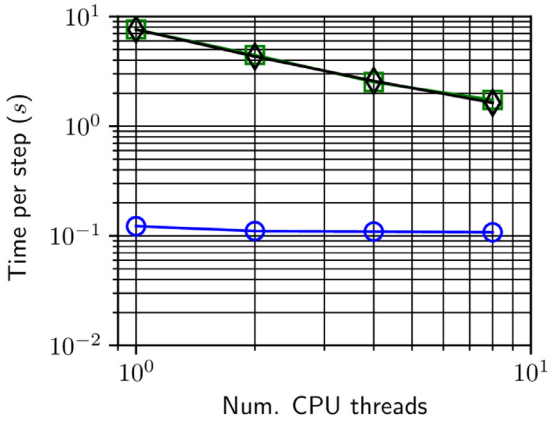

Fig. 12. Average time for interpolating viscoelastic stresses to fluid cell centers for the four-mode PTT fluid in the confined cylinder channel for GPU (◦), CPU-BDF (⋄,) and CPU-Euler (□), at the highest grid resolution.

In Fig. 10 the times for interpolating the viscoelastic stresses to the Eulerian cell centers are shown. The CPU methods are equally fast since they share the same implementation. For the GPU method the times are more than one order of magnitude smaller for the larger node sets.

The influence of the number of CPU threads is assessed. In Figs. 11 and 12 the times for the ODE solution and the interpolation are shown

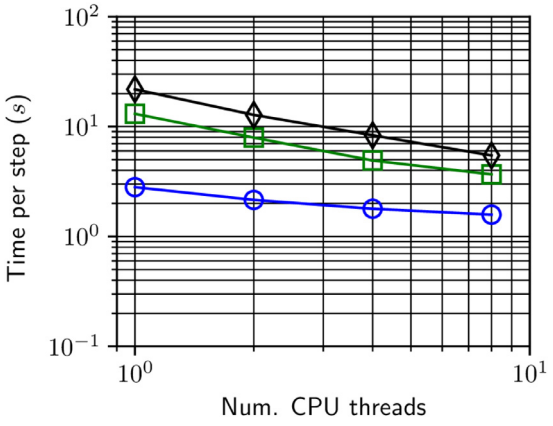

Fig. 13. Average time for calculating viscoelastic stresses for the four-mode PTT fluid in the confined cylinder channel for GPU (◦), CPU-BDF (⋄,) and CPU-Euler (□), at the highest grid resolution.

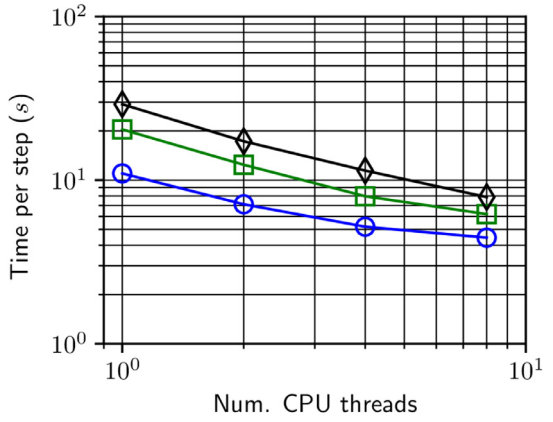

Fig. 14. Average time for a full time step for the four-mode PTT fluid in the confined cylinder channel for GPU (◦), CPU-BDF (⋄,) and CPU-Euler (□), at the highest grid resolution.

for different numbers of processor threads, obtained with the highest grid resolution. An improvement with the number of threads is observed for the CPU-based methods. The computational time for the GPU method is not improved by increasing the number of CPU threads, since the GPU parallelization is independent of the number of the processor threads. The computational times for the GPU method are however significantly lower compared to the CPU methods over the range compared.

In Fig. 13 the times for the full viscoelastic stress calculation are shown for the largest node set, including the ODE solution, the unstructured interpolation and the distribution of the nodes. The two CPU methods scale with the number of threads over the studied range. For the GPU method the observed decrease is smaller, since the main part of the algorithm is executed on the GPU and is thus independent on the number of CPU threads. Furthermore, the total computation times are compared. In Fig. 14 the times for a full simulation step, including solving the momentum and continuity equations, are shown for varying numbers of CPU threads. There is a clearly observable difference between the times obtained for the different methods. Particularly, the GPU method is always faster than the CPU methods.

To summarize, the improvements relative to the CPU-BDF method are presented for the largest Lagrangian node set for the CPU-Euler method in Table 3 and for the GPU method in Table 4. As previously discussed, the calculations perform the fastest with the GPU method for all cases. For a full simulation step the consumed time is roughly halved with the GPU method as compared to the original CPU-BDF method.

Table 3
Reduction of computational time relative to the CPU-BDF method for the largest Lagrangian node set in the cylinder flow for the CPU-Euler method.

CPU th.	1	2	4	8
Ode sol.	76.0%	77.2%	81.6%	79.4%
Stress sol.	40.1%	37.8%	41.1%	32.7%
Full step	29.9%	28.1%	30.3%	21.3%

Table 4
Reduction of computational time relative to the CPU-BDF method for the largest Lagrangian node set in the cylinder flow for the GPU method.

CPU th.	1	2	4	8
Ode sol.	99.6%	99.2%	98.7%	97.9%
Interp.	98.4%	97.4%	95.8%	93.4%
Stress sol.	87.1%	83.2%	78.6%	71.1%
Full step	62.2%	58.8%	54.6%	43.5%

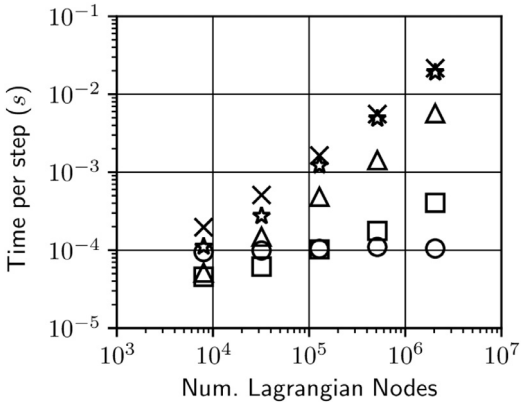

Fig. 15. Time per step with different number of Lagrangian nodes for arranging $\mathbf{u}$ and $\nabla\mathbf{u}$ on the CPU (*) and copy it to the GPU (⋄,), copy updated part of solution to GPU (□), solve ODE systems on GPU (○) and copy solution to CPU (×). Simulated with 4 CPU threads.

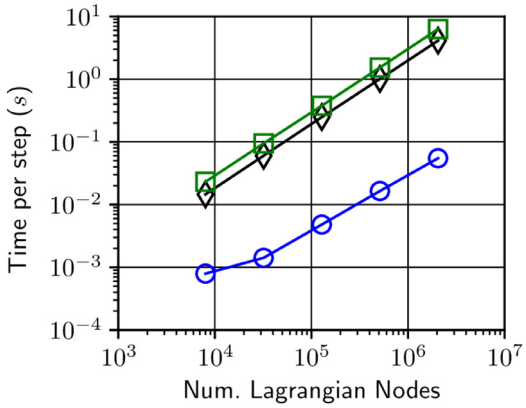

Fig. 16. Average time for solving the ODE systems with 10 substeps for the four-mode PTT fluid in the confined cylinder channel for GPU (○), CPU-BDF (⋄,) and CPU-Euler (□), with 4 CPU threads.

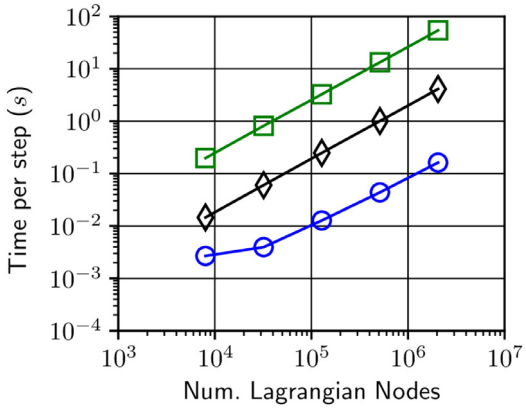

Fig. 17. Average time for solving the ODE systems with 100 substeps for the four-mode PTT fluid in the confined cylinder channel for GPU (○), CPU-BDF (⋄,) and CPU-Euler (□), with 4 CPU threads.

For the CPU-Euler method the corresponding time is reduced by around 30%.

The transfer of data between the CPU and the GPU memory may be a bottleneck for GPU algorithms. This is particularly true when the algorithm requires calculation on both units. This is true for the GPU method in the current work. The components of the GPU method are therefore studied in detail. In Fig. 15 the average times for the operations involved in the solution of the ODE systems are shown. Interestingly, the dominating component is not the solution of the ODE systems itself, but rather arranging and copying data between the GPU memory and the CPU memory. The total ODE solution time for the GPU method therefore scales with the number of nodes, since the copy operations do.

Since the BDF formula has higher order of accuracy than the implicit Euler method, it may in some cases be feasible to increase the number of local steps N_{loc} in the implicit Euler method to improve the accuracy. The impact on the computational time is therefore studied. In Figs. 16 and 17 the computational times for the ODE system solution using $N_{\text{loc}} = 10$ and $N_{\text{loc}} = 100$, respectively, are shown for the implicit Euler methods. The times for the BDF formula are the same as in the previous figures and are included as reference. For the implicit CPU-Euler method the time is proportional to the number of steps, showing a large increase when increasing this number. For the GPU method, the increase is less significant and the consumed time is significantly smaller than for both CPU methods. This is reasonable since the computational time for the GPU method is dominated by memory transfer to and from the GPU. The transfer time is equally large regardless the number of steps performed by the ODE solver, since the data is only transferred to the GPU prior to the first local step and back to the CPU after the last step.

6. Conclusion

A previously proposed Lagrangian-Eulerian method for simulating viscoelastic flow has been extended for parallel computation on the GPU. Two parts of the algorithm have been implemented for execution on the GPU, namely the integration of the constitutive equation at the Lagrangian nodes and the unstructured interpolation of the viscoelastic stress tensor from the nodes to the Eulerian fluid grid.

The original method integrates the constitutive equations using a second order backwards differentiation formula, while the GPU-algorithm uses the implicit Euler method. An equivalent implicit Euler method has therefore been implemented for the CPU to allow fair comparison of computation times. The introduced methods were compared to the original method for two flows, a planar Poiseuille flow and a flow past a confined cylinder. The results obtained with the three methods overlapped.

The computational times for the ODE solution and the stress interpolation were compared for the three methods, for different grid resolutions and numbers of CPU threads. The implicit Euler method proved to be faster than the BDF method both when executed on the CPU and the GPU. The GPU calculations were however significantly faster. The same is true for the unstructured interpolation. As a result, the simulation times were significantly lower for the GPU method for all cases studied. This includes the case of performing 10 or 100 local ODE steps

per global fluid time step, for which the computation times for the CPU-Euler increased proportionally.

While the current GPU method is evidently faster than the corresponding CPU method, adaptions that could potentially further extend the performance are conceivable. Memory transfer could be optimized by splitting into smaller parts. The computations for a small set of Lagrangian nodes could then be initialized without requiring the data transfer for all nodes to be completed. A clear improvement could be obtained from moving more components of the Lagrangian algorithm to the GPU, such as for example the redistribution of the Lagrangian nodes. Further, with the main part of the algorithm executing on the GPU, further optimization of the required memory transfer between the CPU and the GPU is possible. Another reasonable extension is to use an R-tree for the unstructured interpolation on the GPU, as this would improve the performance when used in combination with refined and adaptive Eulerian grids.

The results demonstrate the possibility to improve the performance of the simulation method through the use of GPU acceleration. They are also a promising contribution towards the goal of developing tools for viscoelastic flow simulation that are sufficiently efficient to simulate complex, real-life industrial applications.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has been partly carried out in a Centre for Additive Manufacturing Metal (CAM2) in a joint project financed by Swedish Governmental Agency of Innovation Systems (Vinnova), coordinated by Chalmers University of Technology. The work has also been supported in part by Vinnova through the FFI Sustainable Production Technology program, and in part by the Production Area of Advance at Chalmers University of Technology. The support is gratefully acknowledged.

References

[1] R.G. Larson, Constitutive Equations for Polymer Melts and Solutions, Butterworths series in chemical engineering, Butterworth Publishers, 1988.
[2] N.P. Thien, R.I. Tanner, A new constitutive equation derived from network theory, J. Non-Newton Fluid Mech. 2 (4) (1977) 353–365.
[3] M. Herrchen, H.C. Öttinger, A detailed comparison of various fene dumbbell models, J. Non-Newton Fluid Mech. 68 (1) (1997) 17–42, doi:10.1016/S0377-0257(96)01498-X.
[4] P. Oliveira, F. Pinho, G. Pinto, Numerical simulation of non-linear elastic flows with a general collocated finite-volume method, J. Non-Newton Fluid Mech. 79 (1) (1998) 1–43, doi:10.1016/S0377-0257(98)00082-2.
[5] M. Alves, F. Pinho, P. Oliveira, The flow of viscoelastic fluids past a cylinder: finite-volume high-resolution methods, J. Non-Newton Fluid Mech. 97 (2) (2001) 207–232, doi:10.1016/S0377-0257(00)00198-1.
[6] M.A. Alves, P.J. Oliveira, F.T. Pinho, Benchmark solutions for the flow of Oldroyd-B and PTT fluids in planar contractions, J. Non-Newton Fluid Mech. 110 (1) (2003) 45–75, doi:10.1016/S0377-0257(02)00191-X.
[7] H.P. Baaijens, G.W. Peters, F.P. Baaijens, H.E. Meijer, Viscoelastic flow past a confined cylinder of a polyisobutylene solution, J. Rheol. 39 (6) (1995) 1243–1277.
[8] M.A. Hulsen, R. Fattal, R. Kupferman, Flow of viscoelastic fluids past a cylinder at high Weissenberg number: Stabilized simulations using matrix logarithms, J. Non-Newton Fluid Mech. 127 (1) (2005) 27–39, doi:10.1016/j.jnnfm.2005.01.002.
[9] R. Keunings, A survey of computational rheology, in: Proceedings of the XIIIth International Congress on Rheology, 1, Citeseer, 2000, pp. 7–14.
[10] M. Perera, K. Walters, Long-range memory effects in flows involving abrupt changes in geometry: part I: flows associated with i-shaped and t-shaped geometries, J. Non-Newton Fluid Mech. 2 (1) (1977) 49–81, doi:10.1016/0377-0257(77)80032-3.
[11] C. Fernandes, M. Araujo, L. Ferrs, J.M. Nbrega, Improved both sides diffusion (iBSD): a new and straightforward stabilization approach for viscoelastic fluid flows, J. Non-Newton Fluid Mech. 249 (2017) 63–78, doi:10.1016/j.jnnfm.2017.09.008.
[12] P.A. Stewart, N. Lay, M. Sussman, M. Ohta, An improved sharp interface method for viscoelastic and viscous two-phase flows, J. Sci. Comput. 35 (1) (2008) 43–61, doi:10.1007/s10915-007-9173-5.
[13] N. Balci, B. Thomases, M. Renardy, C.R. Doering, Symmetric factorization of the conformation tensor in viscoelastic fluid models, J. Non-Newton Fluid Mech. 166 (11) (2011) 546–553, XVIth International Workshop on Numerical Methods for Non-Newtonian Flows, doi:10.1016/j.jnnfm.2011.02.008.
[14] R. Fattal, R. Kupferman, Constitutive laws for the matrix-logarithm of the conformation tensor, J. Non-Newton Fluid Mech. 123 (23) (2004) 281–285, doi:10.1016/j.jnnfm.2004.08.008.
[15] R. Fattal, R. Kupferman, Time-dependent simulation of viscoelastic flows at high Weissenberg number using the log-conformation representation, J. Non-Newton Fluid Mech. 126 (1) (2005) 23–37, doi:10.1016/j.jnnfm.2004.12.003.
[16] X. Chen, H. Marschall, M. Schäfer, D. Bothe, A comparison of stabilisation approaches for finite-volume simulation of viscoelastic fluid flow, Int. J. Comput. Fluid Dyn. 27 (6–7) (2013) 229–250, doi:10.1080/10618562.2013.829916.
[17] H. Rasmussen, O. Hassager, Simulation of transient viscoelastic flow with second order time integration, J. Non-Newton Fluid Mech. 56 (1) (1995) 65–84, doi:10.1016/0377-0257(94)01274-L.
[18] O. Harlen, J. Rallison, P. Szabo, A split Lagrangian-Eulerian method for simulating transient viscoelastic flows, J. Non-Newton Fluid Mech. 60 (1) (1995) 81–104, doi:10.1016/0377-0257(95)01381-5.
[19] P. Halin, G. Lielens, R. Keunings, V. Legat, The Lagrangian particle method for macroscopic and micro-macro viscoelastic flow computations, J. Non-Newton Fluid Mech. 79 (2) (1998) 387–403, doi:10.1016/S0377-0257(98)00123-2.
[20] X. Gallez, P. Halin, G. Lielens, R. Keunings, V. Legat, The adaptive Lagrangian particle method for macroscopic and micromacro computations of time-dependent viscoelastic flows, Comput. Methods Appl. Mech. Eng. 180 (3) (1999) 345–364, doi:10.1016/S0045-7825(99)00173-5.
[21] P. Wapperom, R. Keunings, V. Legat, The backward-tracking Lagrangian particle method for transient viscoelastic flows, J. Non-Newton Fluid Mech. 91 (2) (2000) 273–295, doi:10.1016/S0377-0257(99)00095-6.
[22] cuBLAS, URL https://docs.nvidia.com/cuda/cublas/index.html.
[23] cuSPARSE, URL https://docs.nvidia.com/cuda/cusparse/index.htm.
[24] AmgX, URL https://developer.nvidia.com/amgx.
[25] L. Bergamasco, S. Izquierdo, A. Ammar, Direct numerical simulation of complex viscoelastic flows via fast lattice-Boltzmann solution of the Fokker-Planck equation, J. Non-Newton Fluid Mech. 201 (2013) 29–38, doi:10.1016/j.jnnfm.2013.07.004.
[26] H. Feng, M. Andreev, E. Pilyugina, J.D. Schieber, Smoothed particle hydrodynamics simulation of viscoelastic flows with the slip-link model, Mol. Syst. Des. Eng. 1 (2016) 99–108, doi:10.1039/C5ME00009B.
[27] S. Ingelsten, A. Mark, F. Edelvik, A Lagrangian-Eulerian framework for simulation of transient viscoelastic fluid flow, J. Non-Newton Fluid Mech. 266 (2019) 20–32, doi:10.1016/j.jnnfm.2019.02.005.
[28] A. Mark, B.G.M. van Wachem, Derivation and validation of a novel implicit second-order accurate immersed boundary method, J. Comput. Phys. 227 (2008) 6660–6680.
[29] A. Mark, R. Rundqvist, F. Edelvik, Comparison between different immersed boundary conditions for simulation of complex fluid flows, Fluid Dyn. Mater. Process. 7 (3) (2011) 241–258.
[30] IPS IBOFlow, http://ipsiboflow.com.
[31] A. Mark, E. Svenning, F. Edelvik, An immersed boundary method for simulation of flow with heat transfer, Int. J. Heat Mass Transf. 56 (12) (2013) 424–435, doi:10.1016/j.ijheatmasstransfer.2012.09.010.
[32] E. Svenning, A. Mark, F. Edelvik, Simulation of a highly elastic structure interacting with a two-phase flow, J. Math. Ind. 4 (1) (2014) 7, doi:10.1186/2190-5983-4-7.
[33] A. Mark, R. Bohlin, D. Segerdahl, F. Edelvik, J.S. Carlson, Optimisation of robotised sealing stations in paint shops by process simulation and automatic path planning, Int. J. Manuf. Res. 59 (1) (2014) 4–26.
[34] F. Edelvik, A. Mark, N. Karlsson, T. Johnson, J. Carlson, Math-based algorithms and software for virtual product realization implemented in automotive paint shops, in: L. Ghezzi, D. Hömberg, C. Landry (Eds.), Math for the Digital Factory, Springer-Verlag, Berlin, 2017, pp. 231–251.
[35] M. Svensson, A. Mark, F. Edelvik, J. Kressin, R. Bohlin, D. Segerdahl, J.S. Carlson, P.-J. Wahlborg, M. Sundbäck, Process simulation and automatic path planning of adhesive joining, Proc. CIRP 44 (2016) 298–303, 6th CIRP Conference on Assembly Technologies and Systems (CATS), doi:10.1016/j.procir.2016.02.113.
[36] J. Göhl, K. Markstedt, A. Mark, K. Håkansson, P. Gatenholm, F. Edelvik, Simulations of 3d bioprinting: predicting bioprintability of nanofibrillar inks, Biofabriaction 10 (2018), doi:10.1088/1758-5090/aac872.
[37] A.C. Hindmarsh, R. Serban, D.R. Reynolds, User Documentation for cvode v3.1.0 (sundials v3.1.0), Sundials,
[38] Suite of nonlinear and differential/algebraic equation solvers cvode, http://computation.llnl.gov/projects/sundials/cvode,
[39] N. Bell, J. Hoberock, Thrust: A productivity-oriented library for cuda, in: W. mei W. Hwu (Ed.), GPU Computing Gems, 2011, pp. 359–371.
[40] S.-C. Xue, R. Tanner, N. Phan-Thien, Numerical modelling of transient viscoelastic flows, J. Non-Newton Fluid Mech. 123 (1) (2004) 33–58, doi:10.1016/j.jnnfm.2004.06.009.
[41] Fully-developed pipe and planar flows of multimode viscoelastic fluids, J. Non-Newton Fluid Mech. 141 (2) (2007) 85–98, doi:10.1016/j.jnnfm.2006.09.001.