International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–22
© The Author(s) 2016
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1094342016671790
hpc.sagepub.com
SAGE

A fast massively parallel two-phase flow solver for microfluidic chip simulation

Martin Kronbichler[1], Ababacar Diagne[2] and Hanna Holmgren[2]

Abstract
This work presents a parallel finite element solver of incompressible two-phase flow targeting large-scale simulations of three-dimensional dynamics in high-throughput microfluidic separation devices. The method relies on a conservative level set formulation for representing the fluid-fluid interface and uses adaptive mesh refinement on forests of octrees. An implicit time stepping with efficient block solvers for the incompressible Navier–Stokes equations discretized with Taylor–Hood and augmented Taylor–Hood finite elements is presented. A matrix-free implementation is used that reduces the solution time for the Navier–Stokes system by a factor of approximately three compared to the best matrix-based algorithms. Scalability of the chosen algorithms up to 32,768 cores and a billion degrees of freedom is shown.

Keywords
Incompressible Navier–Stokes, multiphase flow, inf–sup stable finite elements, variable material parameters, matrix-free methods, parallel adaptive mesh refinement, generic finite element programming

1 Introduction

Numerical simulations are often the only available tool to understand flow in microfluidic devices. Three-dimensional effects of various flow configurations must be captured by highly resolved computational studies that in turn require large-scale computational facilities. In this work, we present a solver framework for soft inertial microfluidics involving particles with deformable surfaces (Wu et al., 2009). Simulations detailing the flow patterns in these devices are still rare but promise to reveal novel physics and to give better control over device design. This includes the ability to find flow configurations to sort particles of various sizes and material parameters as well as to monitor surface stresses. As a means for representing deformable surfaces, this work uses a model of immiscible incompressible two-phase flow with surface tension.

Detailed multi-phase flow simulations require a very high numerical resolution to track the evolution of free surfaces with a sudden jump in stresses and material parameters between the different fluid phases. Several competing methods exist for indicating the interface location between fluids, which can either be interface tracking methods (Peskin, 1977) such as front tracking (Unverdi and Tryggvason, 1992), or interface capturing methods such as level set methods (Osher and Sethian, 1988), the volume–of–fluid method (Hirt and Nichols, 1981), or phase field methods (Jacqmin, 2000). Two strains of developments can be distinguished for the representation of the interface forces in the incompressible Navier–Stokes equations. In so-called extended finite element methods (XFEM), the interface is represented in a sharp way. Suitable jump and kink enrichments are added to the pressure and velocity, respectively (Groß and Reusken, 2007; Fries and Belytschko, 2010; Rasthofer et al., 2011), as a means to exactly include jump conditions in the finite element spaces. In order to obtain stable and robust schemes, suitable jump penalty terms are added that avoid the otherwise deteriorating effect of small cut regions on condition numbers. The second strain of methods is

[1]Institute for Computational Mechanics, Technical University of Munich, München, Germany
[2]Division of Scientific Computing, Department of Information Technology, Uppsala University, Uppsala, Sweden

Corresponding author:
Martin Kronbichler, Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching b. München, Germany.
Email: kronbichler@lnm.mw.tum.de

given by continuous surface tension models in the spirit of the original work by Brackbill et al. (1992), where the surface tension forces and changes in material parameters are smoothly applied in a narrow band of width proportional to the mesh size. XFEM-based methods are generally more accurate for a given element size, but at the cost of an almost continuous change in maps of degrees of freedom as the interface passes through the domain. Besides the increased cost of integration in cut elements, the difficulty of efficiently implementing changing maps has confined most XFEM methods to serial and relatively modest parallel computations.

The contribution of this work is a highly efficient and massively parallel realization of a continuous surface tension model using a conservative level set representation of the interface (Olsson and Kreiss, 2005; Olsson et al., 2007). Parallel adaptive mesh refinement and coarsening is used to dynamically apply high resolution close to the interface. The algorithm is based on unstructured coarse meshes that are refined in a structured way using a forest-of-trees concept with hanging nodes (Burstedde et al., 2011). In many complex three-dimensional flows, choosing two additional levels of refinement around the interface merely doubles the number of elements. For a range of flow configurations, continuous surface tension models on such a mesh provide solutions of similar quality to those produced by state-of-the-art XFEM techniques; thus, our solver is expected to be competitive with good XFEM implementations in the present context. For time discretization, second-order accurate time stepping based on BDF-2 is used. In space, we choose inf–sup stable Taylor–Hood elements $\mathcal{Q}_2\mathcal{Q}_1$ for the representation of fluid velocity and pressure. These choices result in a considerably more accurate velocity representation as compared to the linear stabilized finite element case often used in the literature. Furthermore, we also consider so-called augmented Taylor–Hood elements $\mathcal{Q}_2\mathcal{Q}_1^+$, where an element-wise constant is added in order to guarantee element-wise mass conservation in the discretization of the incompressible Navier–Stokes equations (Boffi et al., 2012). These elements can provide additional accuracy, in particular with respect to the pressure representation, as compared to plain Taylor–Hood elements. Since these elements have not been studied in detail yet, we also present suitable iterative solvers for these elements.

On parallel architectures, many unstructured finite element solvers rely on (distributed) sparse matrix data structures, with sparse matrix-vector products dominating the run time. Unfortunately, these kernels are a poor fit for modern hardware due to the overwhelming memory bandwidth limit. Instead, our work replaces most matrix-vector products by fast matrix-free kernels based on cell-wise integration as proposed in Kronbichler and Kormann (2012). Fast computation of integrals on the fly is realized by tensorial evaluation for hexahedra (sum factorization) that has its origin in the spectral element community (Karniadakis and Sherwin, 2005; Cantwell et al., 2011; Basini et al., 2012). For element degree 2, however, integration still increases the number of arithmetic operations by about a factor of three over sparse matrix-vector products on the scalar Laplacian (Kronbichler and Kormann, 2012). Nonetheless, performance can be gained if the increase in computations does not outweigh the reduction of memory access. For systems of equations such as the incompressible Navier–Stokes equations with coupling between all velocity components (as they appear for variable material parameters and Newton linearization), fast integration has an additional advantage because the coupling occurs only on quadrature points. This enables matrix-free matrix-vector products that are up to an order of magnitude faster already on $\mathcal{Q}_2$ elements (Kronbichler and Kormann, 2012). These techniques are used in a fully implicit Navier–Stokes solver with a block-triangular preconditioner and a selection of algebraic multigrid and incomplete factorizations for the individual blocks as appropriate. This work will demonstrate that these components give a solver that features:

- massively parallel dynamic mesh adaptation;
- matrix-free solvers with good scalability and 2–4 $\times$ faster solvers compared to matrix-based alternatives;
- good memory efficiency, allowing one to fit larger problems into a given memory configuration.

We want to point out that many algorithms for the incompressible Navier–Stokes equations presented in the context of two-phase flow in microfluidic devices are also applicable in other contexts. Moreover, the solvers extend straight-forwardly to cubic and even higher-order polynomials, where the advantage over matrix-based algorithms is even more impressive. Finally, the higher arithmetic intensity and regular access structure makes these algorithms a promising development for future exascale hardware. The algorithms described in this manuscript are available as open source software on https://github.com/kronbichler/adaflo, building on top of the deal.II finite element library (Bangerth et al., 2016) and the p4est parallel mesh management (Burstedde et al., 2011).

The remainder of the paper is as follows. Section 2 presents the numerical model and discretization and Section 3 discusses the selected linear solvers and details of the fast matrix-free implementation. In Section 4, the microfluidic problem setting is introduced. Section 5 shows the performance results including strong and weak scalability tests. Section 6 gives a characterization of the algorithms for performance prediction on other systems, and Section 7 summarizes our findings.

2 Numerical model

We model the separation of species in a microfluidic device by the flow of two immiscible incompressible fluids as proposed in Wu et al. (2009). Surface tension at the fluid-fluid interface stabilizes the shape of the interface.

2.1 Incompressible Navier–Stokes equations

The motion of each fluid is given by the incompressible Navier–Stokes equations for velocity $\mathbf{u}$ and pressure p in non-dimensional form

$$\begin{aligned}\rho^* \frac{\partial \mathbf{u}}{\partial t} + \rho^* \mathbf{u}\cdot\nabla\mathbf{u} &= -\nabla p + \frac{1}{\mathrm{Re}}\nabla\cdot(2\mu^*\nabla^s\mathbf{u}) \\ &+ \frac{1}{\mathrm{Fr}^2}\rho^* g\mathbf{e}_g + \frac{1}{\mathrm{We}}\kappa\mathbf{n}\delta_\Gamma \\ \nabla\cdot\mathbf{u} &= 0\end{aligned} \tag{1}$$

Here, $\nabla^s\mathbf{u} = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$ denotes the rate of deformation tensor. The parameter Re denotes the Reynolds number, Fr the Froude number, and We the Weber number, which control the magnitude of viscous stresses, gravitational forces, and surface tension, respectively. The parameters ρ^* and μ^* denote the density and viscosity measured relative to the parameters of fluid 1

$$\rho^* = \begin{cases} 1 & \text{in fluid 1} \\ \frac{\rho_2}{\rho_1} & \text{in fluid 2}\end{cases} \qquad \mu^* = \begin{cases} 1 & \text{in fluid 1} \\ \frac{\mu_2}{\mu_1} & \text{in fluid 2}\end{cases}$$

2.2 Level set description of two-phase flow

We denote by Ω_1 the domain occupied by fluid 1, by Ω_2 the domain of fluid 2, and by Γ the interface between Ω_1 and Ω_2 as sketched in Figure 1. The computational domain Ω is the union of the two subdomains and the interface, $\Omega = \Omega_1 \cup \Gamma \cup \Omega_2$. The interface Γ is captured by the conservative level set method from Olsson and Kreiss (2005), i.e. by the zero contour of a regularized characteristic function Φ. Across the interface, Φ smoothly switches from -1 to $+1$ as depicted in Figure 1.

The evolution of Γ in time is via transport of the level set function Φ with the local fluid velocity $\mathbf{u}$

$$\partial_t\Phi + \mathbf{u}\cdot\nabla\Phi = 0, \qquad \Phi(\cdot,0) = \tanh\left(\frac{d(x,y)}{\varepsilon}\right) \tag{2}$$

At the initial time, the profile Φ is computed from a signed distance function $d(x,y)$ around the interface, where ε is a parameter that controls the thickness of the transition region.

2.2.1 Reinitialization procedure. To preserve the profile thickness and shape of Φ during the simulation despite the non-uniform velocity fields and discretization errors, a conservative reinitialization step is performed according to Olsson et al. (2007). The reinitialization seeks the steady state to the equation

$$\partial_\tau\Phi + \frac{1}{2}\nabla\cdot((1-\Phi^2)\mathbf{n}) - \nabla\cdot(\varepsilon(\nabla\Phi\cdot\mathbf{n})\mathbf{n}) = 0 \tag{3}$$

starting from the interface given by the advection equation (2), where τ is an artificial time. Using two to five pseudo time steps of equation (3) provides a good approximation of the steady state and ensures stable simulations.

2.2.2 Computation of surface tension. The evaluation of the surface tension in (1) requires the normal vector $\mathbf{n}$ of the interface as well as the interface curvature κ. These quantities are computed in terms of the level set function Φ

$$\mathbf{n} = \frac{\nabla\Phi}{|\nabla\Phi|} \quad \text{and} \quad \kappa = -\nabla\cdot\mathbf{n} \tag{4}$$

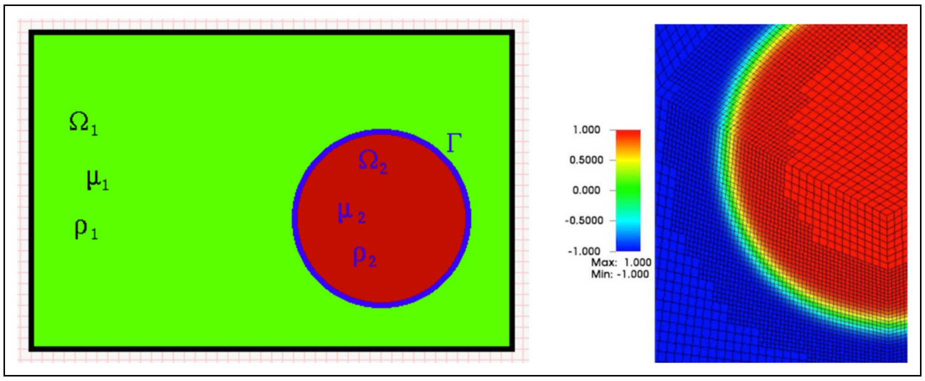

Figure 1. Left: The domain Ω occupied by two immiscible fluids separated by an interface Γ. ρ_i and μ_i denote respectively the density and viscosity in Ω_i. Right: A 2D snapshot of the level set function Φ and an adaptive mesh is depicted.

The surface tension force is defined by a slight modification of the continuous surface tension force according to Brackbill et al. (1992)

$$\mathbf{F} = \frac{1}{\mathrm{We}}\kappa\nabla H_\Phi \tag{5}$$

where H_Φ denotes a Heaviside function constructed from Φ, with the gradient ∇H_Φ replacing the term $\boldsymbol{n}\delta_\Phi$ in the original work (Brackbill et al., 1992; Olsson and Kreiss, 2005). The gradient form of surface tension has the advantage that the surface tension force for constant curvature, i.e. circular shapes, can be represented exactly by the pressure gradient without spurious velocities (Zahedi et al., 2012). The simple choice $H_\Phi = \Phi/2$ as proposed in Olsson and Kreiss (2005)[1] is undesirable, though, because $\nabla\Phi$ has support on the whole domain, albeit exponentially decaying for smooth Φ. For the adaptive meshes with a higher level of refinement around the interface according to Section 2.4 below, small distortions in the level set profile at faces with different refinement give rise to large non-physical curvatures and, thus, to spurious force contributions. To localize surface tension, we instead use the definition

$$H_\Phi \equiv \mathbb{H}_\varepsilon(d(\mathbf{x})), \quad d(\mathbf{x}) = \log\left(\frac{1+\Phi(\mathbf{x})}{1-\Phi(\mathbf{x})}\right) \tag{6}$$

where $d(\mathbf{x})$ denotes a signed distance function reconstructed from Φ, supplemented with suitable limit values for the regions where numerical errors in Φ yield values slightly outside the open interval $(-1, 1)$. The function $\mathbb{H}_\varepsilon$ denotes a one-dimensional smoothed Heaviside function that changes from 0 to 1 over a length scale proportional to ε. We choose the primitive function of the discrete delta function with vanishing first order moments derived in (Peskin, 2002, Section 6) for $\mathbb{H}_\varepsilon$, scaled such that the transition occurs in a band of width 2ε around the interface. Note that this region approximately corresponds to a band between the -0.76 and $+0.76$ contours of Φ.

To improve robustness, the equations for the normal vector field and the curvature (4) are each solved by a projection step of the level set gradient and normal vector divergence to the space of continuous finite elements, respectively, with mesh-dependent diffusion $4h^2$ added according to the discussion in Zahedi et al. (2012). Likewise, a projected normal vector n is computed before the pseudo time stepping of (3). We emphasize that these projection steps are essential for the robustness of the method on unstructured and 3D meshes. Slightly distorted normals, in particular the ones determining the curvature, can spoil the simulation.

2.3 Time discretization

For time stepping, an implicit/explicit variant of the BDF-2 scheme is used. In order to avoid an expensive monolithic coupling between the Navier–Stokes part (1) and the level set transport step (2) via the variables $\mathbf{u}$ and Φ, an explicit (time lag) scheme between the two equations is introduced. In each time step, we first propagate the level set function with the local fluid velocity, run the reinitialization algorithm, and then perform the time step of the incompressible Navier–Stokes equations with surface tension evaluated at the new time. Each of the two fields is propagated fully implicitly using BDF-2. To maintain second order of accuracy in time, the velocity field for the level set advection is extrapolated from time levels $n-1$ and $n-2$ to the new time level n

$$\mathbf{u}^{n,0} = 2\mathbf{u}^{n-1} - \mathbf{u}^{n-2} \tag{7}$$

or with suitable modifications when using variable time step sizes. Note that the splitting between the level set and Navier–Stokes parts corresponds to an explicit treatment of surface tension, which gives rise to a time step limit

$$\Delta t \le c_1\frac{\mathrm{We}}{\mathrm{Re}}h + \sqrt{\left(c_1\frac{\mathrm{We}}{\mathrm{Re}}h\right)^2 + c_2\mathrm{We}h^3} \tag{8}$$

where c_1 and c_2 are constants that do not depend on the mesh size h and the material parameters, see Galusinski and Vigneaux (2008). There exist methods to overcome this stability limit (Hysing, 2006; Sussman and Ohta, 2009). For the examples considered in this work, however, this only imposes a mild restriction with the first term dominating.

2.4 Space discretization and mesh adaptivity

We discretize all solution variables in space using the finite element method. To this end, the computational domain is partitioned into a set of elements. On each element, polynomial solutions of the variables are assumed, and continuity is enforced over the element boundaries. In each time step, the approximations for $\Phi_h^n, \mathbf{u}_h^n, p_h^n$ are of the form

$$\Phi_h^n(\mathbf{x}) = \sum_{j=1}^{N_\Phi}\Phi_j^n\varphi_j^\Phi(\mathbf{x}), \qquad \mathbf{u}_h^n(\mathbf{x}) = \sum_{j=1}^{N_u}U_j^n 2j_j^u(\mathbf{x}),$$
$$p_h^n(\mathbf{x}) = \sum_{j=1}^{N_p}P_j^n\varphi_j^p(\mathbf{x}) \tag{9}$$

where the coefficient values Φ_j^n, U_j^n, P_j^n are to be determined. When choosing the finite element ansatz spaces, i.e. the spaces spanned by the shape functions φ^Φ, φ^u and φ^p, respectively, we consider the following factors.

1. The function represented by Φ is a smoothed Heaviside function according to Section 2.2 the

width of which needs to be kept small for accurate pointwise mass conservation and interface positions. This sets high resolution requirements. Continuous linear functions on hexahedra, $\mathcal{Q}_1$, defined by a tensor product of 1D functions, are used for φ^{Φ}. With this choice sharp transitions are represented better than with higher order functions for the same number of degrees of freedom, because linears avoid over- and undershoots.

2. For the incompressible Navier–Stokes equations, the element selections for velocity and pressure need to fulfill the Babuška–Brezzi (inf–sup) condition (Girault and Raviart, 1986) unless stabilized formulations are used. We consider the following two inf–sup stable options:

- Taylor–Hood (TH): Shape functions of tensor degree q for each component of velocity and shape functions of degree $q-1$ for the pressure, denoted by $\mathcal{Q}_q\mathcal{Q}_{q-1}$, with $q \geq 2$ are used. The shape functions are constructed as a tensor-product of one-dimensional shape functions.
- Augmented Taylor–Hood (ATH): These elements use the same velocity space as TH elements but an extended space $\mathcal{Q}_{q-1}^{+}$ for the pressure where an element-wise constant function is added for φ^p. The additional constant function forces the velocity field to be element-wise conservative (divergence-free) (Boffi et al., 2012) and is consistent because no spatial derivatives on pressure variables appear in the weak form of the Navier–Stokes equations.

We use a mesh consisting of hexahedra that can be dynamically adapted by local refinement and coarsening (adaptive mesh refinement). This allows us to increase the mesh resolution close to the fluid-fluid interface where rapid changes in pressure as well as material parameters need to be captured. Moreover, a fine mesh around the interface keeps the approximation error in normals and curvatures from the level set variable Φ small. In order to meet the different resolution requirements for the Navier–Stokes variables on the one hand and the indicator-like level set function on the other hand, a finer mesh is used for the latter. We choose the level set mesh to be a factor of three to four finer than the Navier–Stokes mesh, depending on the desired balance between costs in the level set part and the Navier–Stokes part. In order to avoid a mismatch in pressure space and the term H_Φ according to Zahedi et al. (2012), we apply an interpolation of H_Φ onto the pressure space $\mathcal{Q}_1$ on the Navier–Stokes mesh before evaluating the surface tension. Note that the level set mesh could be coarser away from the interface in the spirit of narrow-band level set methods (Sethian, 2000); however, this is not done in the present work due to additional expensive data communication requirements between different Navier–Stokes and level set meshes.

As a mesh refinement criterion, we mark cell K for refinement if

$$\log\left(\max_K |\nabla\Phi|\varepsilon\right) > -4 \quad \text{or} \\ \log\left(\max_K |\nabla\Phi|\varepsilon\right) + 4\Delta t\frac{\mathbf{u}\cdot\nabla\Phi}{\varepsilon|\nabla\Phi|} > -7 \tag{10}$$

where the last term is evaluated in the center of the cell. Recall that ε controls the width of the transition region of the conservative level set function. In these formulas, the terms involving logarithms approximate the number of cells between cell K and the interface, with 0 indicating a cell cut by the interface and negative numbers the respective distance. Thus, the first criterion specifies that cells up to four layers away from the interface should be refined. The second formula makes the refinement biased towards the direction of the local flow field. A distance-only approach would adjust the mesh optimally to the current interface position and soon be outdated again. The second term heuristically adds a layer of approximately three mesh cells in downstream direction, which approximately doubles the time span over which the mesh remains viable and thus reduces the re-meshing frequency.

In the case where the distance to the interface is larger than the above values, cells are marked for coarsening. The mesh smoothing algorithms from p4est (Burstedde et al., 2011) ensure that the levels of neighboring cells in the adapted mesh differ at most by a factor 2:1 both over faces, edges, and vertices. In each time step, we check the cheap (and less strict) criterion of whether $\log(\max_K |\nabla\Phi|\varepsilon) > -3.5$, and in case there is at least one such cell, criterion (10) is evaluated for each cell and the mesh adaptation algorithm is called, including an interpolation of all solution variables to the new mesh. The frequency of mesh updates is typically between five and a few hundreds of time steps, depending on the flow field and the time step size.

Based on the mesh and the finite element shape functions, weak forms of the equations (1) and (2) are derived. In the usual finite element fashion, the equations are multiplied by test functions, divergence terms are integrated by parts and boundary conditions are inserted. For the discrete version of the incompressible Navier–Stokes equations (1), we implement the skew-symmetric form of the convective term $\rho^*\mathbf{u}\cdot\nabla\mathbf{u} + \frac{\rho^*}{2}\mathbf{u}\nabla\cdot\mathbf{u}$ in order to guarantee discrete energy conservation (Tadmor, 1984). Finite element discretizations of equations of transport type, such as the level set equation or the Navier–Stokes equations at higher Reynolds numbers, typically need to be stabilized. Here however, no stabilization is used since Reynolds numbers are moderate and the reinitialization will take care of possible oscillations in the level set field. Dirichlet boundary conditions, such as no-slip conditions on velocities or the prescribed level set at inflow boundaries, are imposed strongly by setting the respective values of U_j^n and Φ_j^n to the given values. For Neumann boundary conditions, e.g. for imposing non-zero pressure levels on outflow boundaries, boundary integrals are added to the equations.

2.5 Summary of solution algorithm

One time step of our solver consists of computing the coefficient values Φ_j^n, U_j^n and P_j^n in the finite element expansion (9) by performing the following steps.

1. Extrapolate all fields to the new time level using second order extrapolation according to (7), resulting in $\Phi^{n,0}, \mathbf{u}^{n,0}, p^{n,0}$, and apply boundary conditions at the new time step. The successive steps can then compute increments in time to these fields with homogeneous boundary conditions.
2. Compute increment $\delta\Phi^n$ by solving the weak form of the advection equation (2) with velocity $u^{n,0}$ and BDF-2 discretization in Φ. Then, we set $\widetilde{\Phi}^n = \Phi^{n,0} + \delta\Phi^n$.
3. Project $\nabla\widetilde{\Phi}^n$ onto the space of linear finite elements, including diffusion of size $4h^2$, and evaluate $\widetilde{\mathbf{n}}^n = \nabla\widetilde{\Phi}^n/|\nabla\widetilde{\Phi}^n|$ on each node of the level set mesh.
4. Perform $n_{\text{reinit\_steps}}$ reinitialization steps according to (3), based on the normal vector approximation $\widetilde{\mathbf{n}}^n$. The nonlinear compression term $\frac{1}{2}\nabla\cdot((1-\Phi^2)\widetilde{\mathbf{n}}^n)$ is treated in an explicit Euler fashion and the diffusion term $\nabla\cdot(\varepsilon(\nabla\Phi\cdot\widetilde{\mathbf{n}}^n)\widetilde{\mathbf{n}}^n)$ in an implicit Euler fashion. The result of this procedure is the final level set field Φ^n.
5. Project $\nabla\Phi^n$ onto the space of linear elements, including diffusion of size $4h^2$, and evaluate $\mathbf{n}^n$ on each node of the level set mesh.
6. Compute curvature κ by projecting $-\nabla\cdot\mathbf{n}^n$ onto the space of linear elements, including diffusion of size $4h^2$.
7. Compute the discrete Heaviside function H_Φ^n from Φ^n by evaluating (6) on each node of the finite element mesh. Interpolate H_Φ^n to the pressure finite element space.
8. Evaluate all forcing for the momentum equation, including surface tension according to (5). Evaluate the relative density ρ^* and viscosity μ^* based on the Heaviside function H_Φ^n at each quadrature point and store them for use in the Navier–Stokes solver.
9. Newton iteration for velocity and pressure, iteration index $k \geq 1$:
   (a) Compute nonlinear residuals of momentum and continuity equations.
   (b) Solve for increment $[\delta\mathbf{u}^{n,k}, \delta p^{n,k}]$ and add to $\mathbf{u}^{n,k-1}, p^{n,k-1}$.

   At convergence, we obtain the fields $\mathbf{u}^n$ and p^n at the new time level.

3 Solution of linear systems

After time discretization and linearization in the algorithm from Section 2.5, linear systems for the level set equations and for the Navier–Stokes equations need to be solved. The solution of linear systems represents the main computational effort in our solver and is therefore discussed in more detail. For the level set advection equation in step 2, the system matrix is

$$\left(\frac{3}{2\Delta t}M + C(\mathbf{u}^{n,0})\right)\delta\Phi = R_\Phi \qquad (11)$$

where M denotes the level set mass matrix and $C(\mathbf{u}^{n,0})$ the convection matrix depending on the current velocity. The vector R_Φ denotes the discrete residual of the level set equation, evaluated using Φ^{n-1}, Φ^{n-2}, and $\mathbf{u}^{n,0}$. Since the time step Δt is typically on the order $h/|\mathbf{u}|$ (constant CFL number), the condition number of the system matrix behaves as $\mathcal{O}(1)$ (Elman et al., 2005) and simple iterative solvers can be employed. Due to the non-symmetry, we choose a BiCGStab solver (Saad, 2003), preconditioned by the diagonal of the mass matrix M to account for the different scaling due to the non-uniform mesh. Typical iteration counts for the advection equation are between 5 and 20 for a relative tolerance of 10^{-8}.

The projection systems for the normal vector $\mathbf{n}$ and the curvature κ, steps 3, 5, 6 in the algorithm, are all schematically of the form

$$(M + \gamma K)X = R_X \qquad (12)$$

where X denotes a selected component of the nodal values of $\mathbf{n}$ or the scalar field κ, K denotes the stiffness matrix, and γ is the amount of smoothing in the projection. The vector R_X contains the evaluated weak forms $\int_\Omega \varphi^\Phi \frac{\partial}{\partial x_i}\widetilde{\Phi}\mathrm{d}\mathbf{x}$ for the i-th component of the projected level set gradient and $\int_\Omega \nabla\varphi^\Phi \mathbf{n}\mathrm{d}\mathbf{x}$ for the curvature computation, respectively. The magnitude of γ is set to $4h_\text{I}^2$, where h_I denotes the maximum element size around the interface. With this choice, the final condition number of the matrix behaves similarly to that of a mass matrix, $\mathcal{O}(1)$. Thus, a conjugate gradient method preconditioned by diag(M) is viable. Step 3 uses a comparably coarse relative tolerance of 10^{-4} since it only enters in the interface "stabilization" part, whereas a more stringent tolerance of 10^{-7} is selected for steps 5 and 6. Mesh sizes in our complex applications are not exactly uniform around the interface, such that the smallest element size determining the condition number for stiffness matrices can be up to a factor of three smaller than h_I. Thus, the local conditioning is affected and iteration numbers between 20 and 100 are observed, about two to four times more than for plain mass matrices.

Finally, the equation system for the level set reinitialization, step 4 in the algorithm, is of the form

$$(M + \Delta\tau\varepsilon\widetilde{K}(\widetilde{\mathbf{n}}^n))\delta\Phi = R_R \tag{13}$$

where $\widetilde{K}(\widetilde{\mathbf{n}}^n)$ denotes the (degenerate) stiffness matrix with diffusion along the direction $\widetilde{\mathbf{n}}^n$ only and R_R is the residual of the reinitialization. We set the pseudo time step to $\Delta\tau = \frac{1}{d^2}h_{\min,\mathrm{LS}}$, where $d = 2,3$ is the spatial dimension and $h_{\min,\mathrm{LS}}$ is the minimum mesh size in the level set mesh. For this case, the matrix is dominated by the mass matrix and a conjugate gradient method, preconditioned by diag(M), is suitable. Typical iteration counts are between 8 and 35.

Turning to the Navier–Stokes equations, after time discretization and linearization, the following block system in velocity and pressure arises

$$\begin{pmatrix} A & B^{\mathrm{T}} \\ B & 0 \end{pmatrix}\begin{pmatrix} \delta U \\ \delta P \end{pmatrix} = \begin{pmatrix} R_{\mathbf{u}} \\ R_p \end{pmatrix} \tag{14}$$

where $A = \frac{3}{2\Delta t}M_\rho + C_\rho(\mathbf{u}) + \frac{1}{Re}K_\mu$ is a sum of a mass matrix, a convection matrix, and a stiffness matrix. Matrix B results from the term $\int_\Omega \varphi_i^p \nabla \cdot \varphi_j^u \,\mathrm{d}\mathbf{x}$. According to equation (1), the mass and convection matrices depend on the density contrast and the stiffness matrix on the viscosity contrast between the two fluids. The particular form of the convection matrix depends on the selected linearization method. For the fully implicit Newton method which is used in this work, it is the result of the following weak form

$$C_\rho(\mathbf{u})_{i,j} = \int_\Omega \rho^* \varphi_i \cdot \left[\mathbf{u}\cdot\nabla\varphi_j + \varphi_j\cdot\nabla\mathbf{u} + \frac{1}{2}\mathbf{u}\nabla\cdot\varphi_j + \varphi_j\nabla\cdot\mathbf{u}\right]\mathrm{d}\mathbf{x}$$

The block system (14) is of saddle point structure and solved by an iterative GMRES solver (Saad, 2003). For preconditioning, a block-triangular operator P^{-1} is applied from the right (Elman et al., 2005), defined by

$$P = \begin{pmatrix} A & B^{\mathrm{T}} \\ 0 & -S \end{pmatrix} \Leftrightarrow P^{-1} = \begin{pmatrix} A^{-1} & A^{-1}B^{\mathrm{T}}S^{-1} \\ 0 & -S^{-1} \end{pmatrix} \tag{15}$$

where $S = BA^{-1}B^{\mathrm{T}}$ denotes the Schur complement of the block system (14). If the inverse matrices A^{-1} and S^{-1} were formed exactly, the matrix underlying the GMRES iteration would be a block-triangular matrix with unit matrices on the diagonal. Thus, all eigenvalues would be of unit value with a minimum polynomial of degree two, for which GMRES can be shown to converge in at most two iterations (Elman et al., 2005; Benzi et al., 2005).

Approximations to A^{-1} and S^{-1} are used in our realization. The condition number of the velocity matrix A depends on the size of the time step relative to the size of the velocity and the strength of the viscous term. The time step size Δt is of order $h/|\mathbf{u}|$ and Reynolds numbers are moderate $\mathrm{Re} \le 50$, such that either the mass matrix term or the viscous term dominates. For the former case, we use an incomplete LU decomposition (ILU) (Saad, 2003) as an approximation of A^{-1}, whereas one V-cycle of an algebraic multigrid preconditioner (AMG) based on the software package ML (Tuminaro and Tong, 2000; Gee et al., 2006) is used for the latter case, both provided through the Trilinos library (Heroux et al., 2005). This choice will be evaluated in Section 6 below. For the Schur complement approximation, discretized differential operators on the pressure space are utilized (Elman et al., 2005). For the time-dependent incompressible Navier–Stokes equations, the action of S^{-1} is approximated by the sum

$$S^{-1} = \frac{3}{2\Delta t}K_{p,\rho}^{-1} + M_{p,\mu}^{-1} \tag{16}$$

where

$$\begin{aligned} (K_{p,\rho})_{i,j} &= \int_\Omega \nabla\varphi_i^p \tfrac{1}{\rho^*}\nabla\varphi_j^p \,\mathrm{d}\mathbf{x} && \text{(pressure Laplace matrix)} \\ (M_{p,\mu})_{i,j} &= \int_\Omega \varphi_i^p \tfrac{\mathrm{Re}}{\mu^*}\varphi_j^p \,\mathrm{d}\mathbf{x} && \text{(pressure mass matrix)} \end{aligned}$$

The pressure Laplace and mass matrices are scaled by the inverse density and viscosity, respectively, in order to take up the action of the velocity mass and stiffness operators (Cahouet and Chabard, 1988; Benzi et al., 2005). As boundary conditions for the pressure Laplacian, Neumann boundary conditions are imposed on Dirichlet boundaries for the velocity, and homogeneous Dirichlet conditions are imposed on velocity Neumann boundaries (e.g. outflow) (Turek, 1999). For the augmented pressure elements $\mathcal{Q}_1^+$, the discontinuous ansatz space necessitates the inclusion of face integrals for a consistent discrete Laplace operator. We realize this by a symmetric interior penalty discretization (Arnold et al., 2002), again weighted by the inverse density values. In the implementation of (16), approximations to $M_{p,\mu}^{-1}$ and $K_{p,\rho}^{-1}$ are used instead of exact inverses. For the Laplacian, we choose a V-cycle of the ML-AMG preconditioner with Chebyshev smoothing of degree three (Gee et al., 2006). For the mass matrix inverse, two different strategies are necessary for the Taylor–Hood and augmented Taylor–Hood cases, respectively, in order to guarantee convergence that is independent of the mesh size yet cheap to apply. For the former, an ILU is sufficient, whereas the condition number of the mass matrix in the ATH case scales as h^{-2} which is adequately approximated by a V-cycle of ML-AMG. For $\mathcal{Q}_1^+$, a near-null space of two vectors is specified for ML (Gee et al., 2006), including the continuous $\mathcal{Q}_1$ and discontinuous $\mathcal{Q}_0$ parts as separate components.

The linear solver for the Navier–Stokes system is terminated once a relative tolerance of 10^{-4} is reached. The nonlinear Newton iteration makes sure the final nonlinear residual reaches a value of 10^{-9} in absolute terms, which represents a similar accuracy as the relative residual of 10^{-8} on the level set part in the applications shown below.

3.1 Implementation

Our solver[2] is implemented in C++ based on the deal.II finite element library (Bangerth et al., 2007, 2016). The code is fully parallelized by domain decomposition with MPI using a framework that has been shown to scale to tens of thousands of processors (Bangerth et al., 2011), realizing adaptive mesh hierarchies on forests of octrees via the p4est library (Burstedde et al., 2011). Distributed matrix algebra, domain decomposition additive Schwarz methods for the extension of ILU methods to the parallel case, as well as the aforementioned ML-AMG are provided by the Trilinos library (Heroux et al., 2005).

Most of the iterative solvers described above spend the bulk of their computing time in matrix-vector products. A particular feature of our realization is the use of fast matrix-free methods from Kormann and Kronbichler (2011); Kronbichler and Kormann (2012) for matrix-vector products. For quadratic finite elements and systems of partial differential equations such as the system matrix of the incompressible Navier–Stokes equations linearized by a Newton method, the matrix-free kernels can be up to ten times as fast as matrix-based ones. Similar observations were made by May et al. (2014) in the context of the Stokes equations. The framework also enables the computation of the residual vectors R in the linear systems (11) to (14) at a cost similar to one matrix-vector product. Due to their small costs, all timings reported below include residual computations in the solver time. Besides increasing the performance of matrix-vector products, the matrix-free approach obviates matrix assembly (aside from the matrices needed for preconditioners), enabling very efficient Newton–Krylov solvers (Brown, 2010; Kronbichler and Kormann, 2012). In particular, for the variable-coefficient matrices of the Navier–Stokes matrix and the level set reinitialization equation, avoiding matrix assembly already saves up to one third of the global run time.

Of course, matrix-free operator evaluation only helps matrix-vector products and not other components in linear solvers. For example, the ILU preconditioners used for the approximations of the inverse of the velocity matrix still require explicit knowledge of matrix entries in order to build the factorization. To limit the cost of the matrix-based velocity ILU, we choose a simplified matrix that only contains the mass matrix, the convective contribution of a Picard iteration, and the velocity Laplacian. In this case, the velocity matrix is block-diagonal except for boundary conditions that mix different velocity components as, e.g. conditions that require the velocity to be tangential or normal to boundaries not aligned with the coordinate directions. In our solver, we use one and the same matrix for representing all three velocity components, increasing performance of the velocity ILU by a factor of more than 2 because matrix-vector multiplications are performed on three vectors at once through the Epetra matrix interfaces (Heroux et al., 2005). Since this matrix and the factorization are only used as preconditioners, they need to be updated only infrequently when the densities and viscosities have shifted too much. For the case where an AMG operator is used for the velocity, matrix-free evaluation can be used to a greater extent. On the finest and most expensive level of the multilevel hierarchy, a Chebyshev smoother involving only matrix-vector products and vector updates (Adams et al, 2003) can be realized. It only needs matrix diagonals and is thus possible to realize without explicitly forming a matrix. The matrix-free evaluation is supplemented with a matrix representation for computing the coarser hierarchies of the AMG, where the matrix drops coupling between velocity components. Similar techniques were also analyzed in Kronbichler et al. (2012); May et al. (2014).

4 The test problem: Flow in a microfluidic chip

In this article, we use a computational model to study the dynamics of a microfluidic chip that enables high-throughput separation of bacteria from human bloods cells. The flow in these chips is characterized by low Reynolds numbers, i.e. a laminar flow behavior, which makes it possible to handle and analyze a single particle at a time.

The device schematics are shown in Figure 2. Three inlets including sample fluid as well as a protecting

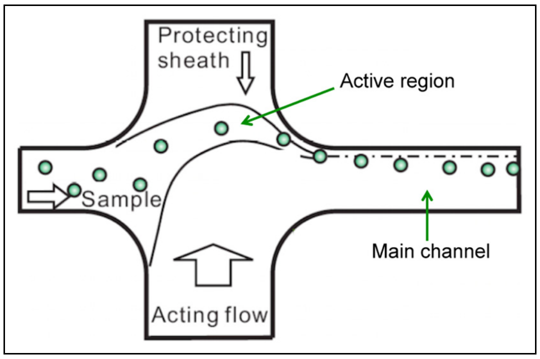

Figure 2. The microfluidic chip.

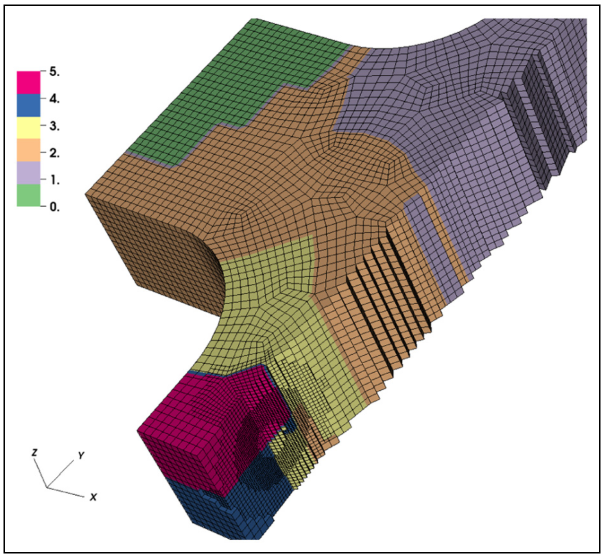

Figure 3. Visualization of the geometric partitioning of the grid on 6 processors. The computational domain in this example is sliced at $x = 0.5$ along the x-direction.

sheath and acting flow are joined in the active area. Three collectors for small particles, large particles, and waste are connected to the main channel in the downstream direction. Through this configuration, large particles are deflected from the original streamline while the path of small particles remains almost unchanged. Experiments (Wu et al., 2009) successfully demonstrated the separation of the bacteria (*Escherichia coli*) from human red blood cells (*erythrocytes*) through this robust and novel microfluidic process. As a result, a fractionation of two differently sized particles into two subgroups was obtained. The separated cells were proven to be viable. This sorting was based on the size of these bio-particles with size ratio between 3 and 8.

In Figure 3, we show the partitioning of approximately one third of the computational domain for 6 cores (MPI ranks) where each contiguously colored segment correspond to a core's subdomain. The computational domain is cut along the x-direction in order to visualize the refined mesh around the interface (see subsection 2.4). We emphasize that the aim of this paper is not to investigate the effectiveness of our method to predict the sorting between particles with respect to their size but to rather present the parallel performance of the selected algorithms. Therefore, we only show results for one configuration. We consider a particle with radius $r = 0.25$ centered at $(x_c, y_c, z_c) = (0.5, 0.5, 0.5)$. The viscosity and the density ratios are set to 20 and 10, respectively. The nondimensional Weber number is $\mathrm{We} = 3$ and the Reynolds number is $\mathrm{Re} = 1.5$ (measured by the sample channel diameter and sample inflow velocity). Figure 4 shows snapshots of a particle moving through the microfluidic device as well as the flow pattern. The particle position and shape are visualized by the level set field with the particle colored in red. Streamlines passing from the sample inlet to the main channel are included in the plots. As expected, the particle deflects due to the strong acting flow and enters the main channel from the curved trajectory. In the main channel, the bubble velocity is about a factor of 30 larger than at the inlet. Figure 4(b) also shows how the velocity field

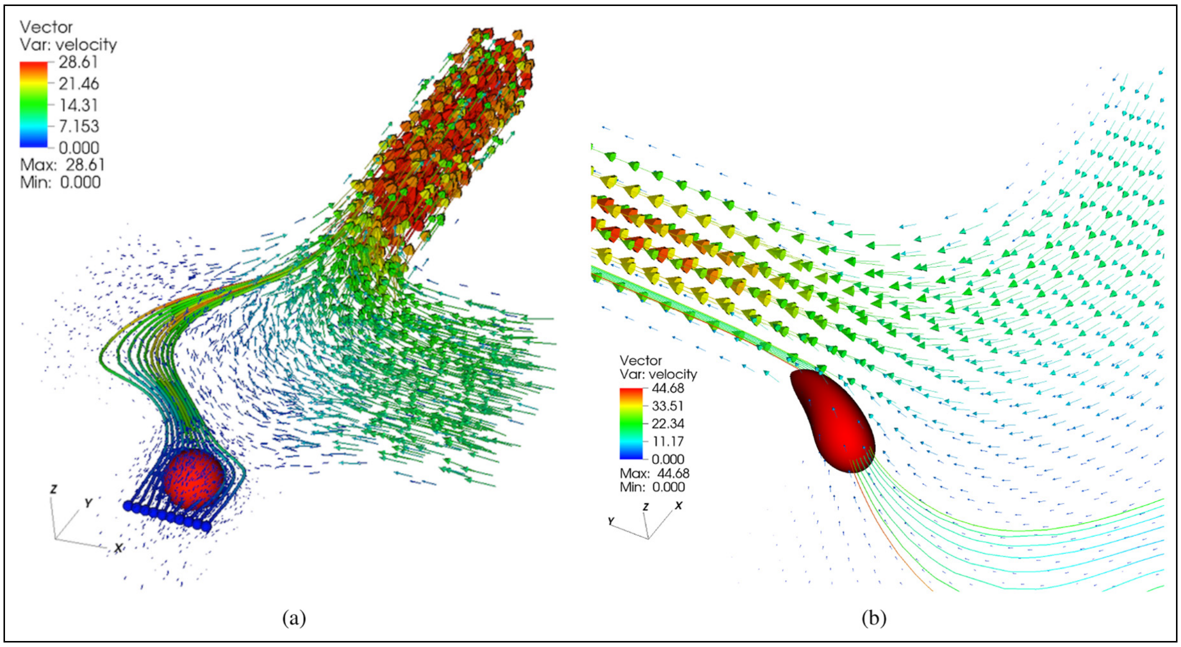

Figure 4. Streamline, velocity field, and the moving particle in the device: (a) after a few time steps; (b) at $t = 1.6$ (range of large acceleration forces).

Table 1. Relative error against reference result on $h = 0.002$ as a function of the vertical position $\bar{y}$ of center of mass of the particle.

	Q_2Q_1			$Q_2Q_1^+$		
h	$e_{\bar{x}}$	$e_{\bar{u}}$	$e_{\bar{\Psi}}$	$e_{\bar{x}}$	$e_{\bar{u}}$	$e_{\bar{\Psi}}$
0.016	5.09%	2.09%	7.58%	5.12%	2.11%	7.58%
0.008	1.68%	0.78%	2.15%	1.68%	0.78%	2.16%
0.004	0.45%	0.22%	0.40%	—	—	—

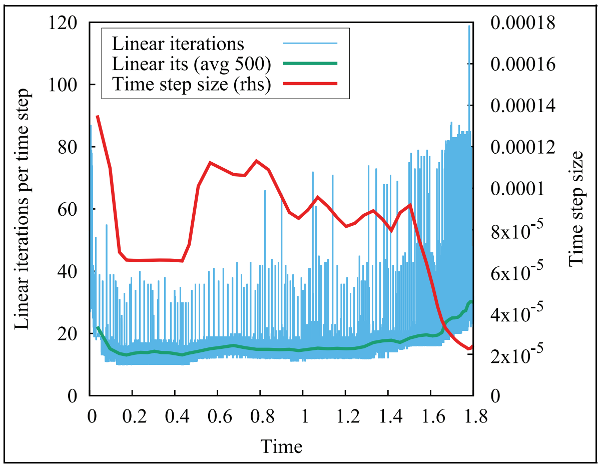

Figure 5. Number of iterations per time step (including the average over 500 time steps as a bold line) and time step size over a complete cycle of the particle through the microfluidic device using more than 25,000 time steps.

is modified by the rising particle in the main channel. The trend of the predicted streamline and shape of the particle show a good agreement with experimental results in Wu et al. (2009).

In addition, our numerical model characterizes the flow mechanism in the device with much more detail and predicts various features of the particles reasonably well.

4.1 Evaluation of particle dynamics

In Figure 5, we show statistics related to the solver for a realistic application–finding the complete particle path through the microfluidic device. The results are based on a mesh with two levels of global refinement and three more levels of adaptive refinements around the interface, using approximately 1.6 million elements and 43 million degrees of freedom for the velocity. This corresponds to a finest mesh size of around 0.004, spanning almost three orders of magnitude compared to the global geometry of width 4, length 8, and height 1. For the simulation, variable time step sizes have been used with the step size set to meet the following two goals:

- the CFL number in the advection equation should not exceed 0.5;
- condition (8) regarding the capillary time step limit must be fulfilled.

Up to $t \approx 1.55$, the latter condition dominates, where the variations in the time step size are due to variations in the local mesh size around the bubble. At later times when the bubble reaches the main channel, the particle is strongly accelerated and the CFL condition forces the time step size to decrease to approximately 2.5×10^{-5}. In total, more than 25,000 time steps have been performed. In the last phase, strong forces act on the particle, which lead to an increase in the number of linear iterations.

Due to the fine mesh, an AMG preconditioner for the velocity block has been selected. Overall, the mesh was adapted 3500 times in order to follow the bubble motion. The total simulation time on 1024 cores was 40 hours, with about 34% of time spent in the Navier–Stokes solver and 24% in the AMG preconditioner setup of both pressure and velocity (which was called approximately every second time step). 21% of the computational time was spent in the level set reinitialization, 16% in the other level set computations, 3% in mesh refinement and solution transfer functions, and the remaining 2% of time in solution analysis and input/output routines.

A mesh convergence study considering the path, velocity, and shape of the particle is given in Table 1. Taking the solution on a mesh of size 0.002 in the region around the interface with Q_2Q_1 elements as a reference, the relative numerical error e of the following particle quantities is considered: the center of mass $x = (\bar{x}, \bar{y}, \bar{z})$ of the bubble, the average rise velocity $u = (\bar{u}, \bar{v}, \bar{w})$, and the sphericity Ψ as the ratio between volume and surface area as compared to the data of a ball (see definitions in Adelsberger et al. (2014)). The time step has been set proportional to the mesh size according to the CFL number and the capillary time step limit as specified above. Due to the large variation in the particle velocity, the errors are evaluated as a function of the vertical position $\bar{y}$ of the center of mass rather than time. We observe estimated convergence orders close to two for all quantities. The augmented Taylor–Hood element produces results very close to the Taylor–Hood element, despite a 10–15% higher overall cost.

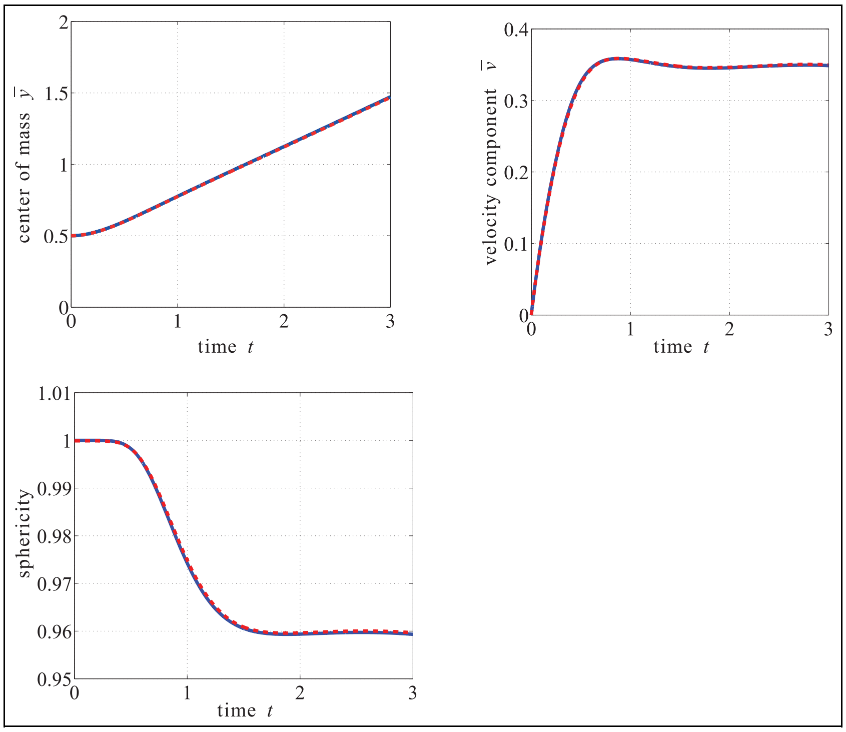

Figure 6. Blue line: Rising droplet quantities using the present two-phase solver. Red dashed line: Rising droplet quantities using the NaStsDGPF code in Adelsberger et al. (2014).

4.2 Verification of two-phase flow solver

For verification of the physics underlying the microfluidic device simulator, a 3D benchmark problem consisting of a two-phase flow representing a rising droplet from Adelsberger et al. (2014) is considered. The problem consists of a cuboid tank $\Omega = [0,1] \times [0,2] \times [0,1]$ and a droplet $\Omega_2 = \Omega_2(t) \subset \Omega$ which is lighter than the surrounding fluid $\Omega_1 = \Omega \backslash \Omega_2$. The initial shape of the droplet is a sphere of radius $r = 0.25$ and center point $x_c = (0.5, 0.5, 0.5)$. The initial fluid velocity is prescribed to zero and no slip boundary conditions are set on all walls. Furthermore a gravitational force $g = (0, -0.98, 0)$ is applied. During the simulation the droplet rises and changes its shape due to buoyancy effects. The densities and viscosities of the two fluids are $\rho_1 = 1000$, $\rho_2 = 100$, $\mu_1 = 10$, $\mu_2 = 1$, and the surface tension is $\tau = 24.5$.

We use an adaptively refined grid with $h = \frac{1}{640}$ at the finest level for the Navier–Stokes solver and a time step of 0.0005. The level set mesh is a factor of three finer than the Navier–Stokes mesh. To compare our results with the data from Adelsberger et al. (2014), we again measure the center of mass $\bar{\mathrm{x}} = (\bar{x}, \bar{y}, \bar{z})$, the rise velocity $\bar{\mathrm{u}} = (\bar{u}, \bar{v}, \bar{w})$, and the sphericity Ψ of the droplet. In Figure 6 the quantities of interest are depicted both for the simulation performed here and for the simulation with the NaSt3DGPF code from Adelsberger et al. (2014). The results are in close agreement. In Adelsberger et al. (2014) the diameter of the droplet had also been considered. Calculations of the diameter have been performed (using the high-order quadrature method for implicitly defined surfaces in Saye (2015)) that agree well with the reference data but are not presented here.

For further verification, a convergence study of the rising droplet problem is performed. Five different grid sizes (uniform refinement) are used for three sets of simulations that differ in the elements used for the Navier–Stokes equations. In the first set of simulations the Taylor–Hood elements ($\mathcal{Q}_2\mathcal{Q}_1$) are used, in the second set augmented Taylor–Hood elements ($\mathcal{Q}_2\mathcal{Q}_1^+$), and in the third set the higher order elements $\mathcal{Q}_3\mathcal{Q}_2$. The three quantities of interest mentioned above are measured at the final time $T = 3$ and displayed in Table 2. Computing a reference solution (adaptive grid refinement with $h = 1/1280$ at the finest grid level and a time step of $1/3000$) gives $\bar{y} = 1.4725483$, $\bar{v} = 0.34874503$ and $\Psi = 0.95932012$. The rate of convergence when measuring the error in the droplet quantities against the reference solution is approximately two for all finite element pairs. As mentioned above, these results are in close agreement with the data reported in Adelsberger et al. (2014). For example, the vertical rise velocity at the final time $T = 3$ is $\bar{v} = 0.347298$ for the simulation in Adelsberger et al. (2014) (with the NaSt3DGPF code) which is between the two results we obtain with grid sizes $h = 0.025$ and

Table 2. Grid convergence study of the rising bubble test. Three set of simulations with different finite element pairs are performed: $\mathcal{Q}_2\mathcal{Q}_1$, $\mathcal{Q}_2\mathcal{Q}_1^+$, $\mathcal{Q}_3\mathcal{Q}_2$. Each set of simulations are performed on five different grid sizes h, where h is the size of the Navier–Stokes mesh. The level set variables are represented by $\mathcal{Q}_1$ elements on a mesh of size $\frac{h}{3}$.

	$\mathcal{Q}_2\mathcal{Q}_1$			$\mathcal{Q}_2\mathcal{Q}_1^+$			$\mathcal{Q}_3\mathcal{Q}_2$		
h	$\bar{y}$	$\bar{v}$	Ψ	$\bar{y}$	$\bar{v}$	Ψ	$\bar{y}$	$\bar{v}$	Ψ
0.05	1.406608	0.335588	0.976708	1.407760	0.335774	0.975883	1.412677	0.341571	0.976707
0.025	1.454877	0.345656	0.965096	1.455008	0.345581	0.965125	1.455475	0.345868	0.965122
0.0125	1.466830	0.347560	0.960585	1.466885	0.347530	0.960603	1.467088	0.347640	0.960615
0.00625	1.470800	0.348375	0.959598	1.470829	0.348364	0.959608	1.470916	0.348400	0.959614
0.003125	1.472023	0.348636	0.959399	1.472038	0.348628	0.959402	1.472096	0.348647	0.959384

$h = 0.0125$, respectively. The convergence history of our method thus confirms the correctness of our implementation.

The results using the higher order finite element pair $\mathcal{Q}_3\mathcal{Q}_2$ are just slightly better than those with the quadratic velocity elements. This is because the finite element space for the level set part is still $\mathcal{Q}_1$, with a major error contribution from the level set discretization. All simulations use a level set mesh that is a factor of three finer and thus shows the same error irrespective of using the $\mathcal{Q}_3\mathcal{Q}_2$ or $\mathcal{Q}_2\mathcal{Q}_1$ element pairs. For an alternative comparison, we relate the $\mathcal{Q}_2\mathcal{Q}_1$ element pair and a Navier–Stokes mesh size $h = 0.00625$ to the $\mathcal{Q}_3\mathcal{Q}_2$ element pair on a mesh of size $h = 0.0125$, both using the same level set mesh of size 0.003125. The error values for these choices are similar. For example, the bubble center of mass is $\bar{y} = 1.4692$ for $\mathcal{Q}_2\mathcal{Q}_1$ elements and $\bar{y} = 1.4689$ for $\mathcal{Q}_3\mathcal{Q}_2$. On the other hand, the overall computational time is 5.5 hours for the former (3.9 hours in the Navier–Stokes solver) compared to 4.3 hours for the latter (3.1 hours for the Navier–Stokes solver). In terms of solver cost per degree of freedom, the $\mathcal{Q}_3\mathcal{Q}_2$ solver is approximately 1.8 times as expensive as the $\mathcal{Q}_2\mathcal{Q}_1$ solver on these simulations. Given sufficient accuracy in the level set part, our results show that the higher order method can reach better efficiency. However, sharp interface (XFEM) techniques and higher order level set techniques are necessary to get convergence rates larger than two in the present context and thus to fully unleash the potential of higher order elements.

With regard to the augmented Taylor–Hood elements $\mathcal{Q}_2\mathcal{Q}_1^+$, the solution quality increases by 1–5% on the bubble benchmark tests (also when using an even finer level set mesh). However, the additional overall solver cost of up to 30–50% of the overall run time makes this element choice less efficient for the configurations considered here.

5 Performance results

5.1 Experimental setup

The tests are performed on two parallel high-performance computers, the medium-sized cluster *Tintin* and the large-scale system *SuperMUC*. *Tintin* is an AMD Bulldozer-based compute server at the Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX). Each compute server of *Tintin* consists of two octocore (four-module) Opteron 6220 processors running at 3 GHz. *Tintin* provides a total of 2560 CPU cores (160 compute nodes with dual CPUs) and the nodes are interconnected with QDR Infiniband. The cluster is operated with Scientific Linux, version 6.4, and GCC compiler version 4.8.0 has been used for compilation.

SuperMUC is operated by the Leibniz Supercomputing Center and uses 147,456 cores of type Intel Xeon Sandy Bridge-EP (Xeon E5-2680, 2.7 GHz). This cluster has a peak performance of about 3 Petaflops. The nodes are interconnected with an Infiniband FDR10 fabric and contain 16 cores each. GCC compiler version 5.1 has been used for compilation.

5.2 Strong (fixed-size) scalability tests

For a fixed problem size, we record the computational time spent on up to $N = 2048$ cores. The grid details are shown in Table 3. An adaptively refined mesh similar to the one in Figure 3 but with one more level of global refinements is used.

Figure 7 plots the run time for 65 time steps over the core count. This time frame includes at least one dynamic adaptation in the mesh and is therefore representative for the behavior of the solver over longer time intervals. We have verified that the testing time is long enough to keep run time variations negligible. The axes are logarithmically scaled. The results show a considerable speedup if computational resources are increased for both the TH and ATH solvers. We observe an almost perfect linear scaling from 32 to 512 cores: The global run time is reduced by a factor of 11.0 for the TH discretization and 10.3 for ATH with parallel efficiencies of 69% and 64%, respectively. Due to the more involved linear system (more iterations, slightly more unknowns), the computational cost is higher for the ATH finite element pair.

Table 3. Details of the 3D triangulation and number of degrees of freedom (dofs). TH and ATH designate the Taylor–Hood and the augmented Taylor–Hood element pairs, respectively.

Active cells	Velocity dofs		Pressure dofs		Level set	max(h)/min(h)
607,104	TH	15,013,701	TH	642,178	16,726,994	0.0209 / 0.0117
	ATH	15,013,701	ATH	1,249,282		

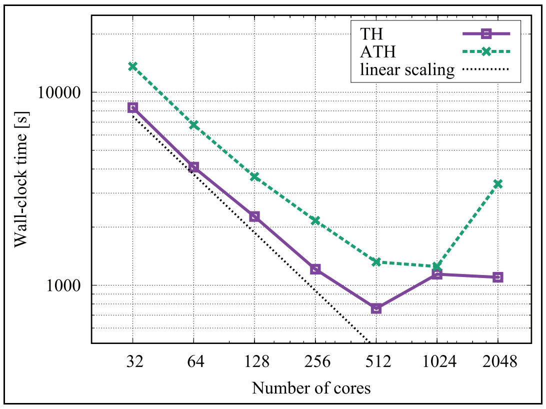

Figure 7. Wall-clock time for constant total work on a 3D test problem using the Taylor–Hood and the augmented Taylor–Hood element pairs on *Tintin*.

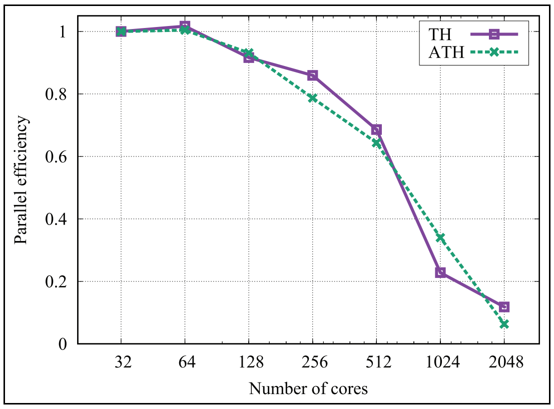

Figure 8. Parallel efficiency on 3D test problem using the Taylor–Hood and the augmented Taylor–Hood element pairs at constant total work (strong scaling) on *Tintin*.

The parallel efficiency of these runs as compared to the timings on 32 processors is presented in Figure 8. An efficiency of 1 indicates perfect isogranular efficiency, which is reached or even slightly exceeded at 64 cores (due to cache effects).

The main reason for saturation of scaling at 1024 and 2048 cores can be identified in Figure 9 which lists the proportion of the main solver components in the strong scaling study: The Navier–Stokes solver (NSSv) behaves significantly worse than the level set components and dominates at larger core counts. In order to isolate the mathematical aspects from the scaling of the implementation, Table 4 lists the average number of linear iterations of the Navier–Stokes solver per time step (accumulated over the nonlinear iteration) and the average time per linear iteration. The data allows us to identify the increase in the number of linear iterations as the main reason for suboptimal scaling up to 512 cores. This is due to the degradation of the domain decomposition additive Schwarz realization of the velocity ILU that does not include overlap. On the other hand, scalability issues beyond that level are due to the overwhelming cost of communication, mainly in the pressure AMG operator. For the 1024 core case, the number of unknowns per core is only around 15,000. More analysis on the communication cost is presented in Section 5.5 below. Figure 9 also demonstrates that the time spent in support of functionality such as the adaptive mesh refinement, assembling the preconditioner, or output analysis, remains below 20%.

5.3 Node-level performance

In this subsection we detail the computational behavior within shared memory inside a node in order to quantify the balance of computationally bound components versus memory bandwidth bound parts. To this end, we again consider strong scaling tests on *Tintin* and *SuperMUC*. We use a mesh consisting of 83,588 cells with one level of adaptive refinement and otherwise similar to Table 3.

Table 5 collects the overall run time of both the TH and ATH discretizations on *Tintin* and *SuperMUC* as well as a breakdown into the major solver components. On each system, we notice an improvement of more than a factor of 10 in the global compute time when going from 1 to 16 cores. Its worth pointing out that each component runs at least three times as fast on *SuperMUC* than on *Tintin*, indicating the higher performance of the Intel Sandy-Bridge processors as compared to the AMD Bulldozer ones. Moreover, the various components of the algorithm behave differently. The Navier–Stokes part takes 23% and 15% of the global computational time on *SuperMUC* and *Tintin*, respectively, when using the TH version of the algorithm, and 40% and 30% of the time for ATH. The share of level set computations is between 35% and 27% for the TH and ATH cases on both systems. On *Tintin*, reinitialization uses around 35% of the compute

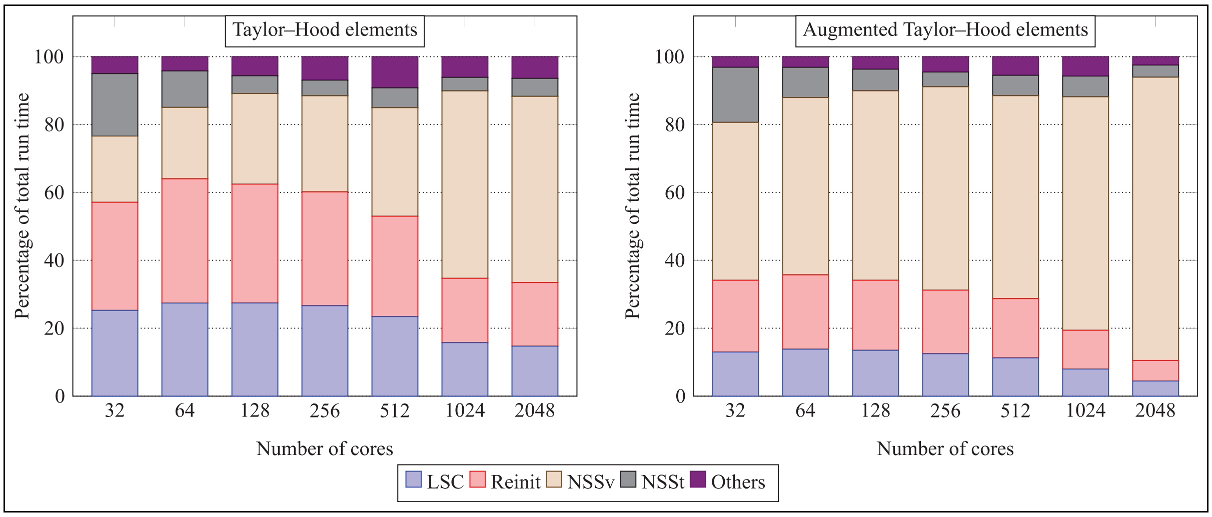

Figure 9. Distribution of the computational time spent in various blocks of the algorithm on *Tintin*. LSC: Level set computations (level set advance, normal, curvature, Heaviside, force calculations, i.e. steps 2, 3, 5–8 in the algorithm in Section 2.5), Reinit: reinitialization (algorithm step 4), NSSv: Navier–Stokes solver including residual computation (algorithm step 9), NSSt: Navier–Stokes setup (setup and assembly of matrices, computation of preconditioners). "Others" gathers the time for output, grid adaptation, and solution interpolation between different grids.

Table 4. Linear Navier–Stokes solver in a strong scaling setting (on *Tintin*). We present the time in seconds required to perform a single linear iteration.

Element	Number of cores	32	64	128	256	512	1024	2048
TH	Total time elapsed (s)	1570	825	595	412	246	637	612
	Lin. iterations/time step	30.2	31.5	34.9	37.4	39.5	43.4	35.8
	Time/linear iteration (s)	0.79	0.4	0.26	0.16	0.09	0.22	0.26
ATH	Total time elapsed (s)	6200	3510	2030	1300	797	874	2820
	Lin. iterations/time step	49.6	55.1	56.3	57.1	56.8	43.3	55.1
	Time/linear iteration (s)	1.92	0.97	0.55	0.35	0.21	0.31	0.78

time while on *SuperMUC* this percentage is reduced to around 20%, illustrating the more effective vectorization on the Intel processors.

Table 5 also shows that components which are dominated by matrix-free evaluation such as the reinitialization algorithm scale considerably better within a node than algorithms which involve sparse matrix kernels: ILU methods are used within the Navier–Stokes preconditioner and normal vector computation within the LSC part. This is despite the fact that many operations using sparse linear algebra (all but the pressure Poisson solver) operate on three vectors simultaneously.

5.4 Weak scalability tests

In this subsection, we assess the weak scaling of our solver on the *Tintin* and *SuperMUC* systems, respectively. For the weak scaling study, we simultaneously increase the problem size and the number of cores while keeping the problem size per core constant. Except for possible increases in solver iterations, the arithmetic load per processor is constant. Table 6 lists the problem sizes used in this test.

We increase the core count in steps of 8 from 4 to 2048 to reflect isotropic refinement of the base mesh, going from 8800 cells to 4.5 million cells. Run times for the two sets of tests are reported in Table 6 and Figure 10, where a breakdown into different parts of the solver is given.

We notice that the global run time increases slightly as the mesh is refined, with the most pronounced increase between 4 and 32 cores where memory bandwidth limitations within the node contribute substantially. The computing time for the level set parts increases only mildly. On the other hand, the Navier–Stokes solver scales considerably worse for both the TH and ATH discretizations, partly due to the already mentioned increase in linear iterations because of the velocity ILU. The parallel efficiency of the Navier–Stokes solver for the TH case drops to 20% on 256 cores of *Tintin* where all other parts achieve more than 40% parallel efficiency. On *SuperMUC*, the weak

Table 5. Comparison of run times in seconds within a node on both *SuperMUC* and *Tintin* systems for 83,588 cells and 65 time steps. For each version of the solver, the table reports the total wall-clock time (Global) and the timing for the major components of the solver such as, Level set computations (LSC), reinitialization (Reinit) and the Navier–Stokes solver (NSSv).

Element	# cores	*SuperMUC*				*Tintin*			
		Global	LSC	Reinit	NSSv	Global	LSC	Reinit	NSSv
TH	1	4510	1451	1110	1034	13900	5433	4760	2281
	2	2380	792	550	519	7620	2878	2490	1191
	4	1280	446	282	287	3760	1379	1150	638
	8	688	243	153	162	1770	650	492	299
	12	563	215	123	135	1670	661	514	277
	16	420	161	88	101	1360	533	410	243
ATH	1	5690	1455	1110	2202	16300	4641	4940	5133
	2	2990	797	557	1103	8660	2403	2570	2461
	4	1580	447	285	582	4020	1163	1130	1110
	8	859	246	153	327	2340	635	674	650
	12	717	217	124	285	1790	519	503	516
	16	546	163	89	221	1840	478	541	616

Table 6. Weak scaling experiments for Taylor–Hood (TH) and augmented Taylor–Hood (ATH) elements run over 65 time steps.

	Refinement 1	Refinement 2	Refinement 3	Refinement 4
Active cells	8800	70, 400	563, 200	4, 505, 600
Velocity dofs	234,651	1,782 099	13,884,195	109,598,787
Pressure dofs (ATH)	19,609	148,617	1,157,233	9,133,665
Pressure dofs (TH)	10,809	78,217	549,033	4,628,065
Level-set dofs	255,025	1,969,849	15,481,297	122,748,193
Time (s), TH, *Tintin*	469	828	1210	2770
Time (s), ATH, *Tintin*	414	817	1400	5030
Time (s), TH, *SuperMUC*	156	206	238	305
Time (s), ATH, *SuperMUC*	177	247	343	709

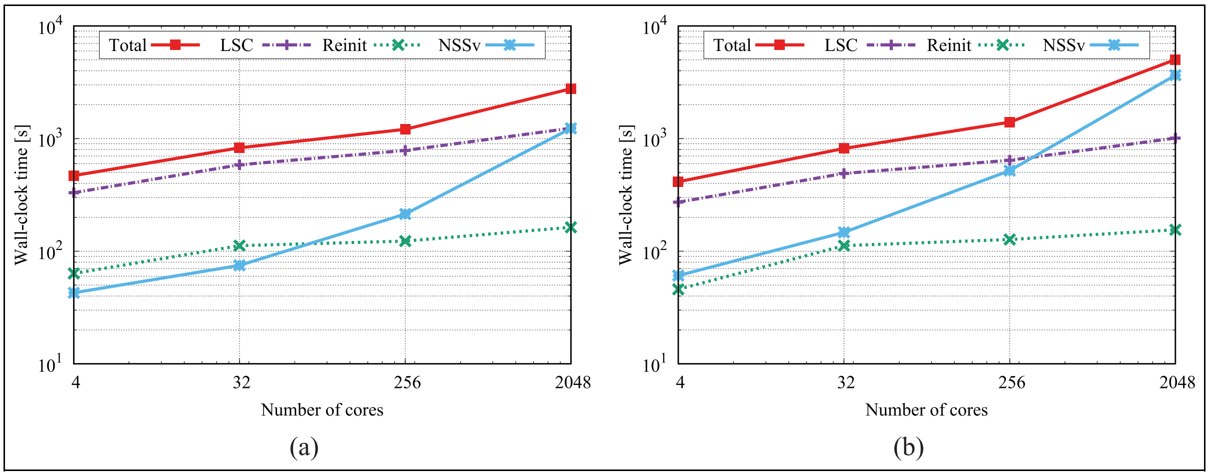

Figure 10. Weak scaling of the major solver components on *Tintin*: (a) Taylor Hood; (b) Augmented Taylor Hood. LSC: Level set computation (computing Heaviside, curvature, force, normal and advance the level set); Reinit: Reinitialization step; NSSv: Navier–Stokes solver.

scalability appears more favorable and can be attributed to a considerably better communication network than the one of *Tintin*.

Figure 11 shows a combined strong and weak scalability study, starting from refinement level 2 and going to level 5 with approximately one billion degrees of

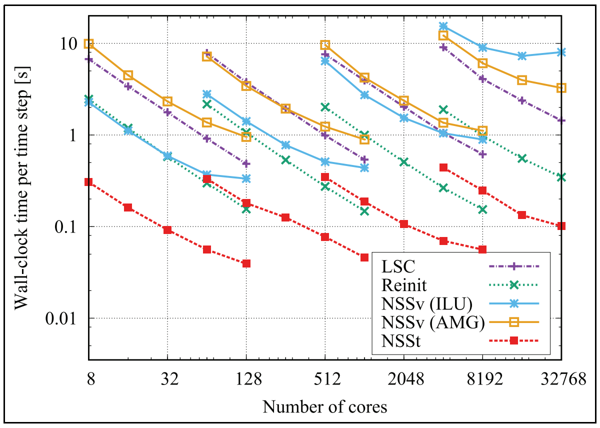

Figure 11. Scaling results on *SuperMUC* (TH elements) for $\Delta t = 0.0001$. Four different problem sizes at 70k, 563k, 4.5m, and 36m elements are shown for 5 processor configurations each (except for the largest size), involving between 232k and 14k Navier–Stokes dofs per core. The data points for strong scaling are connected, with breaks for weak scaling. Two solver variants for the Navier–Stokes solver (NSSv) are considered, one using a simplified ILU on the velocity block and one with an AMG.

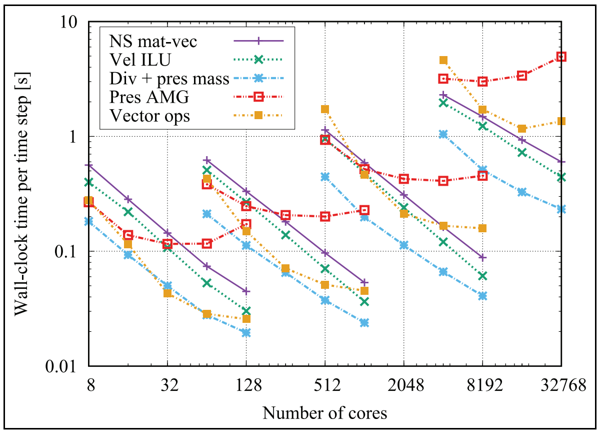

Figure 12. Scaling results for Navier–Stokes solver components according to the algorithm described in Section 3 on *SuperMUC*, where each set of lines is for problem sizes with 70k, 563k, 4.5m, and 36m elements, respectively. "NS mat-vec" refers to matrix-vector products with the Navier–Stokes Jacobian matrix, "Vel ILU" to multiplication with the ILU approximation of the inverse velocity matrix, "Div + pres mass" the multiplication by the transpose of the divergence matrix B^T and the inverse pressure mass matrix approximation by an ILU, "Pres AMG" the inverse pressure Poisson matrix approximation, and "Vector ops" the orthogonalization work done for the GMRES iterative solver, including global communication through inner products.

freedom in the Navier–Stokes system. The level set components (LSC, Reinit) scale almost perfectly up to the largest size. Thus, level set components that dominate the computational time at small problem sizes and low processor counts become very modest as compared to the Navier–Stokes part, see also Figure 9. Note that the level set mesh was chosen to be a factor of three finer than the Navier–Stokes mesh, giving the best overall accuracy in terms of computational input.

5.5 Analysis of Navier–Stokes solver

Now we consider the Navier–Stokes linear solver in more detail. This is the most important aspect of this work since the level set solver shows better scaling and its cost can be adjusted as desired by choosing the level of additional refinement beyond the Navier–Stokes mesh.

Figure 12 shows weak (different lines in the same color) and strong (along connected lines) scaling in the components of the Navier–Stokes solver individually. The matrix-vector products scale almost perfectly. For weak scaling from 32 to 16,384 processors, the time for a matrix-vector product goes from 5.3 ms to 5.65 ms, i.e. about 94% parallel efficiency. In terms of arithmetic performance, the matrix-vector product reaches between 30% and 40% of the theoretical arithmetic peak on *SuperMUC*. For instance, the matrix-vector product reaches 191 TFlops out of 603 TFlops possible at 32,768 cores. This number is based on the approximately 14,900 floating point operations that are done per cell in the linearized Navier–Stokes operator for our implementation (or about 600 operations per degree of freedom). The deviation from ideal scaling observed in Figure 12 is due to an increase in the number of linear iterations for larger problem sizes and larger processor counts. The ILU preconditioner time per application scales even slightly better than matrix-vector products because the no-overlap policy in the additive Schwarz domain decomposition operation removes coupling and thus operations once more processors are used; on the other hand, this is the main cause for the considerable increase in linear iterations as shown in Table 4.

We emphasize that a scalar ILU matrix representation for the velocity preconditioner and appropriate sparse matrix kernels are used in order to improve performane by reducing memory transfer by almost a factor of three. Despite this optimization, the matrix-vector product for the Navier–Stokes matrix is only approximately 1.3 to 1.5 times as expensive. Note that sparse matrix kernels for the Navier–Stokes Jacobian matrix are 6 to 10 times as expensive than the chosen matrix-free approach. Thus, the cost for the Navier–Stokes solver would increase by a factor of between two and three for moderate processor counts. Figure 13 shows a comparison of the proposed matrix-free solver with matrix-based variants for the problem with 14.4m and 114m degrees of freedom, respectively. Figure 13 also adds the time to recompute the system matrix in each Newton step as a metric of the total cost

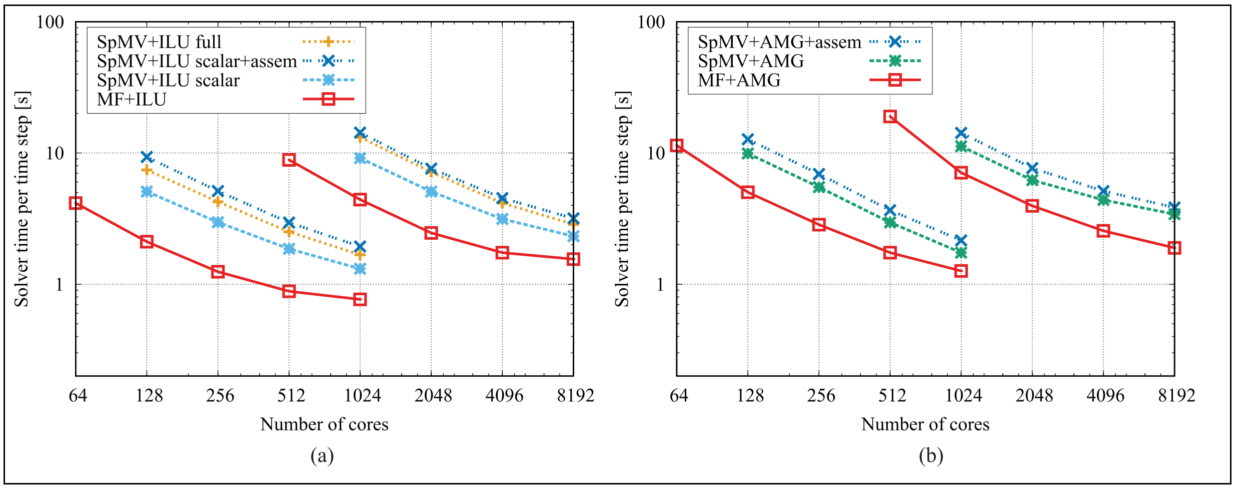

Figure 13. Scaling comparison of proposed matrix-free solver (MF) with matrix-based solvers (SpMV) on *SuperMUC* for $\Delta t = 0.0002$ on two meshes. For the matrix-based solver, the cost of actually assembling the sparse matrices approximately twice per time step is also included (data point " + assem"). The matrix-based ILU preconditioners are shown in two variants, one inverting the scalar convection operator and Laplacian (ILU scalar) as well as an ILU for the complete linearized velocity matrix (ILU full). The AMG is based on the vector Laplacian plus vector convection–diffusion operator without coupling between velocities. (a) velocity ILU preconditioner, (b) velocity AMG preconditioner.

of a matrix-based solver, using standard assembly routines from deal.II (Bangerth et al., 2016). We see a three- to fourfold improvement of the presented algorithms to large scales despite the close-to-ideal parallel scaling of matrix assembly. Moreover, the runs at 64 and 512 cores, respectively, did not complete for the matrix-based solver because the matrices and factorizations did not fit into the 2 GB memory per core available on *SuperMUC*. Obviously, we observe better strong scaling for the matrix-based solver because the component with the worst scaling, the pressure AMG, is the same in both cases.

The inverse pressure Poisson operator by ML-AMG and the vector operations scales considerably worse than the matrix-vector products since these involve global communication. In particular the pressure Poisson operator is affected by the small local problem size: The strong scaling shown in Figure 12 goes from approximately 9000 degrees of freedom per core down to 550 degrees of freedom per core with obvious implications on the communication cost. As also observed in Sundar et al. (2012), scaling algebraic multilevel preconditioners to the largest core counts represents a serious difficulty. Note that the re-partitioning settings of the AMG lead to slight deviations from expected scaling in some cases (e.g. the data point at 512 cores on the finer mesh in Figure 13(b)), but are essential for scaling to core counts beyond 2000. The assembly and setup times of preconditioners for the Navier–Stokes system, labeled "NSSt" in Figure 11, are very moderate because they need not be done every time step.

We note that the numbers presented here are summed separately on each core (without synchronization that would disturb the time measurements at that scale) and averaged over all processors afterwards. Operations that only involve nearest-neighbor communication can advance asymmetrically, leading to waiting times in later stages. This is partly responsible for the superlinear scaling in vector operations in the early stages of strong scaling (another factor is cache effects in the Gram–Schmidt orthogonalization of GMRES).

6 Performance prediction

The results collected in Section 5 allow us to generalize the behavior to other configurations. Due to the complex structure of the solver, we distinguish algorithmic reasons (iteration counts) and code execution reasons in the following two subsections. We concentrate on the linear solver for the Navier–Stokes part because it shows the most peculiar behavior. For the other components, the number of linear iterations and thus the cost per time step is almost proportional to the problem size and inversely proportional to the number of cores, as can be seen from Figures 10 and 11.

6.1 Characterization of Navier–Stokes linear solver

Besides the time step limit imposed by the two-phase flow solver (8), the time step size in the Navier–Stokes can not become too large for the Cahouet–Chabard Schur complement approximation to feature optimal iteration counts. For moderate Reynolds numbers, $0.1 \leq \mathrm{Re} \leq 100$, the CFL number in terms of $\Delta t = \mathrm{CFL} \cdot h / |\mathbf{u}|$ needs to be smaller than

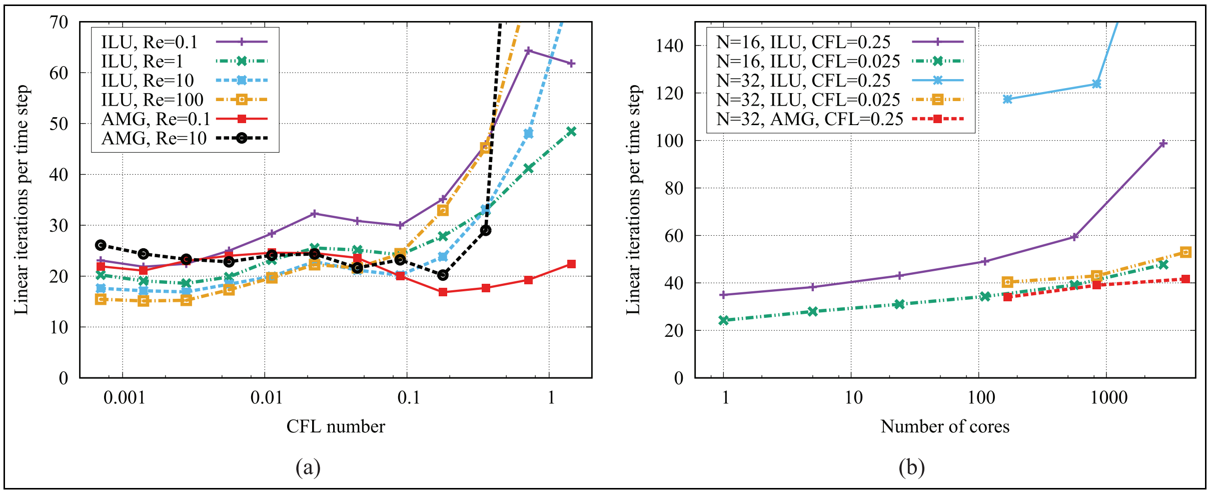

Figure 14. Average number of linear iterations per time step (accumulated over the nonlinear iteration) as a function of the CFL number on a serial computation (left) and as a function of the number of cores (right) for a typical run on unstructured meshes. (a) Serial computation, $h = \frac{1}{16}$. (b) Re = 1, element size $h = \frac{1}{N}$ in outlet cross section.

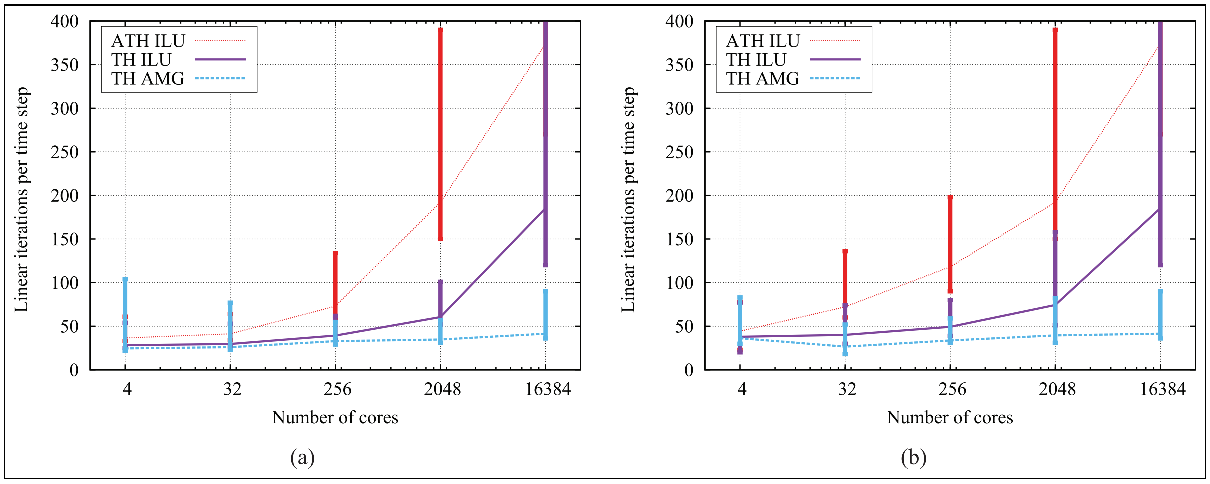

Figure 15. Average number of linear iterations per time step (lines) including the minimum and maximum number of iterations (vertical bars) in a weak scaling setting. Results are shown for Taylor–Hood element with velocity ILU and velocity AMG as well as the augmented Taylor–Hood element (ATH ILU). (a) Constant time step $\Delta t = 0.0001$. (b) Constant CFL = 0.45, $\Delta t = 0.0016 \ldots 0.0001$.

approximately 0.5, see Figure 14. This is an expected result because the Schur complement approximation does not take the convective term that dominates at larger CFL numbers into account. A viscous term that dominates over convection also results in a good Schur complement approximation, see the AMG results for Re = 0.1 and Re = 10, respectively. For finer meshes, the performance of the ILU degrades as the viscous term increases in strength. Nonetheless, the results in Figure 14 reveal that an ILU applied to the velocity block yields very competitive iteration counts as compared to an AMG over a wide range of time steps. Similar behavior was observed on other examples with large adaptive and unstructured meshes. This is because one linear iteration with the velocity AMG preconditioner is more than twice as expensive as with the velocity ILU. Compare also the two panels of Figure 13.

Figure 15 displays the number of Navier–Stokes solver iterations in a weak scaling experiment, illustrating that the degradation of the ILU in a block-Jacobi manner per processor subdomain seriously affects iteration counts. This needs to be contrasted to a velocity AMG preconditioner whose iteration count remains almost constant as the mesh is refined. Due to the difference in iteration counts, the AMG variant can surpass the solver time of the ILU variant on the largest mesh with 16,384 cores in Figure 15 at CFL = 0.45 and with 4192 cores at CFL = 0.25 in Figure 14. This result is confirmed by Figure 11. Note that the often-used technique to increase the quality of the velocity inverse by using an inner BiCGStab solver preconditioned by ILU does not lead to faster solver times. Even though the iteration count of the outer GMRES solver goes down, the increase in cost for the inner solver outweighs the gain.

Figure 15 also includes the minimum and maximum number of linear iterations taken in a specific time step. During the first time steps and whenever the mesh is refined/coarsened, the quality of the extrapolation according to the algorithm in Section 2.5 is reduced and more linear iterations are necessary. Likewise, due to the interface motion and the shift in material properties, the preconditioner quality degrades over time, until a heuristic strategy forces the re-computation of the preconditioner.

For the chosen preconditioner and implementation, one can expect iteration counts around 20–50 at low CFL and Reynolds numbers. If the iteration count exceeds approximately 70, replacing the velocity ILU by an AMG can restore optimal iteration counts. We recommend switching to multigrid for the velocity block in case the viscous term becomes strong with respect to mesh size, i.e. $\Delta t > \frac{1}{2}h^2 \mathrm{Re}/\max(1, \frac{\mu_2}{\mu_1})$ and/or when the number of cores exceeds a few thousand.

6.2 Prediction of run time

With respect to execution of the code, this section provides generalizations to other supercomputers with different machine properties than *Tintin* and *SuperMUC*. The most important ingredients are the memory bandwidth and communication latency and, to a somewhat lesser extent, the arithmetic throughput and suitability for general finite element-related code patterns such as fast indirect addressing. We distinguish three phases in the algorithm, namely:

- arithmetically heavy components, which are the matrix-free evaluation routines in matrix-vector products in the Navier–Stokes solver (14,900 arithmetic operations at 5800 bytes from main memory per $\mathcal{Q}_2\mathcal{Q}_1$ element on general meshes, giving an arithmetic balance of approximately 2.6 flops/byte) and in most of the level set computations (approximately 1.6 flops/byte);
- primarily memory bound components, such as the sparse-matrix based ILU or matrix-based solvers for the level set normal computation as well as the Gram–Schmidt orthogonalization in GMRES, (0.15–0.5 flops/byte);
- primarily latency bound components, such as the coarser levels of the algebraic multigrid preconditioner used in the pressure and possibly for the velocity.

The matrix-free kernels from Kronbichler and Kormann (2012) are mostly vectorized except for the indirect addressing into vectors inherent to continuous finite elements. Furthermore, additions and multiplications are not perfectly balanced and only approximately two thirds of the arithmetic operations are amenable to fused multiply-add (FMA) instructions for the given polynomial degrees. In absence of memory bandwidth restrictions, 40 to 70% of peak represents an upper limit. Depending on the machine balance in terms of memory bandwidth to flops ratio (Hager and Wellein, 2011), the algorithm ends up close to the ridge between computation bound and memory bound region according to the roofline performance model (Patterson and Hennessy, 2009).

To perform 30 Navier–Stokes linear iterations per time step on a mesh of 100k $\mathcal{Q}_2\mathcal{Q}_1$ elements, approximately 6×10^{10} floating point operations and 26 GB of main memory transfer in the arithmetically heavy parts (Navier–Stokes matrix-vector product, product by divergence matrix) are necessary and 46 GB of memory transfer in the memory bound parts (velocity and pressure ILU, pressure AMG with 3 smoother steps and operator complexity of 1.2, inner products and vector updates). Thus, one step requires 72 GB of transfer from main memory and about 7×10^{10} floating point operations. This theoretical prediction, only taking upper limits into account, is within a factor of 1.5 of the actually observed performance of 1.04 seconds for 70k elements on one node of *SuperMUC* according to Figure 11 (measured memory throughput 72 GB/s).

A similar calculation for the level set part using 2.7 million linear $\mathcal{Q}_1$ elements, i.e. using $h/3$ as compared to the size h of the Navier–Stokes mesh, involves approximately 125 GB of memory transfer. This number assumes 120 iterations on the normal vector, 50 iterations for the curvature calculation, 15 iterations for the advection equation, and 40 iterations in the reinitialization. Figure 11 reports 4.6 seconds for these operations on a mesh with 1.9m elements. In this case the model overestimates the actual performance by about a factor of three.

Going from intra-node behavior to inter-node behavior, the first two categories identified above show an almost perfect speedup since the messages are mostly nearest neighbor and of moderate size (up to a few tens of kilobytes for one matrix-vector product in typical settings). On the other hand, the communication network becomes essential for the global communication parts. This is particularly evident for the pressure AMG operator as seen in Figure 12. Modeling the network behavior is beyond the scope of this work; however, a general observation is that one AMG V-cycle cannot reach less than approximately 0.02 seconds on up to 500 nodes (point-to-point latency is 1–2 μs on *SuperMUC*, SpMV latency is around 250 μs), no matter how small the local problem size. In other words, strong scaling does not continue below 5k–20k matrix rows per MPI rank. If the mesh becomes finer and more levels in the multilevel hierarchy are necessary, the latency grows respectively. Note that the contribution of the pressure AMG to the memory traffic is approximately one sixth of the whole solver, so saturation in this component does not immediately stop scaling of the overall solver. Furthermore, if there is more work to do for the level set part (finer mesh on those variables), its better scaling makes the overall scaling more favorable.

7 Conclusions

In this work, a massively parallel finite element solver for the simulation of multiphase flow in a microfluidic device has been presented. Spatial discretizations of the incompressible Navier–Stokes equations with inf–sup stable Taylor–Hood and augmented Taylor–Hood elements have been used, respectively. For time discretization, the Navier–Stokes and the level set part have been split into two systems by extrapolation. Within each field, implicit BDF-2 time stepping has been used. First, the level set is advanced by a second-order extrapolated velocity field. The incompressible Navier–Stokes equations are evaluated with surface tension from the updated level set field and solved as one full velocity–pressure system with a block-triangular preconditioner. Scalability of the chosen algorithms up to approximately one billion degrees of freedom in each of the Navier–Stokes and the level system has been demonstrated. For fixed problem sizes, almost linear strong scaling has been observed as long as the local problem size is larger than approximately 30,000 degrees of freedom. Weak scaling tests show very good performance, with some degradation in the Navier–Stokes solver due to the non-optimal ILU preconditioner in the velocity. Better scaling is possible with multigrid-based preconditioners. Our experiments show that for low and moderate processor counts and small to moderate CFL numbers, ILU is the better choice in terms of computational time, in particular when the ILU is only based on the scalar convection–diffusion operator in velocity space. The picture only changes when the number of MPI ranks exceeds several thousands and the mesh size becomes small such that viscous effects become more pronounced, when the better algorithmic properties of the AMG encourage a switch. Future work will include the derivation of geometric multigrid components within the velocity and pressure variables in order to reduce the bottleneck from AMG at the largest core counts, using strategies similar to Sundar et al. (2012). Furthermore, alternatives to ILU for the velocity block that can leverage the matrix-free kernels in the higher order case need to be explored.

The key ingredient to our solvers are fast matrix-free implementations of matrix-vector products. Our results show an improvement of more than a factor of two in pure solver times over the best matrix-based algorithm. Taking into account the cost of matrix assembly, the advantage grows in favor of our algorithm. While this improvement either decreases time to solution or allows for larger problems on the same computational resources, it reaches communication limits earlier when used in a strong scaling setting. Detailed analysis of run times of selected solver components shows that as soon as the run time goes below approximately 0.5–1 s per time step in the Navier–Stokes solver (or approximately 0.2–0.5 s for the solution of one linear system), the communication cost in the algebraic multigrid preconditioner for the pressure Poisson operator prevents further scaling.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de) through project ID *pr83te*. Some computations were performed on resources provided by SNIC through Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) under project *p2010002*. The financial support from the eSSENCE strategic research program and the Swedish Research Council, as well as the technical assistance from UPPMAX are gratefully acknowledged.

Notes

1. The factor $\frac{1}{2}$ accounts for the range $[-1, 1]$ of the smoothed indicator-like function Φ in this work as opposed to the range $[0, 1]$ in Olsson and Kreiss (2005).
2. The computations shown in this manuscript have been performed on variations of the problem file applications/micro_particle.cc available from https://github.com/kronbichler/adaflo. Retrieved on 2016-05-08.

References

Adams M, Brezina M, Hu J, et al. (2003) Parallel multigrid smoothing: Polynomial versus Gauss–Seidel. *Journal of Computational Physics* 188: 593–610.

Adelsberger J, Esser P, Griebel M, et al. (2014) 3D incompressible two-phase flow benchmark computations for rising droplets. In: Oñate E, Oliver J and Huerta A (eds) *Proceedings of the 11th world congress on computational* *mechanics (WCCM XI)*, Barcelona, Spain, 20–25 July 2014, pp. 5274–5285. Barcelona: CIMNE.

Arnold D, Brezzi F, Cockburn B, et al. (2002) Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis* 39: 1749–1779.

Bangerth W, Burstedde C, Heister T, et al. (2011) Algorithms and data structures for massively parallel generic finite element codes. *ACM Transactions on Mathematical Software* 38(2): 14:1–14:28.

Bangerth W, Hartmann R and Kanschat G (2007) deal.II – A general purpose object oriented finite element library. *ACM Transactions on Mathematical Software* 33(4): 24: 1–24:27.

Bangerth W, Heister T, Heltai L, et al. (2016) The dealii library, version 8.3. *The Archive of Numerical Software* 4(100): 1–11.

Basini P, Blitz C, Bozdağ E, et al. (2012) SPECFEM 3D user manual. Technical Report, Computational Infrastructure for Geodynamics, Princeton University, University of PAU, CNRS, and INRIA.

Benzi M, Golub GH and Liesen J (2005) Numerical solution of saddle point problems. *Acta Numerica* 14: 1–137.

Boffi D, Cavallini N, Gardini F, et al. (2012) Local mass conservation of Stokes finite elements. *Journal of Scientific Computing* 52(2): 383–400.

Brackbill JU, Kothe DB and Zemach C (1992) A continuum method for modeling surface tension. *Journal of Computational Physics* 100: 335–354.

Brown J (2010) Efficient nonlinear solvers for nodal high-order finite elements in 3D. *Journal of Scientific Computing* 45(1-3): 48–63.

Burstedde C, Wilcox LC and Ghattas O (2011) p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing* 33(3): 1103–1133.

Cahouet J and Chabard JP (1988) Some fast 3D finite element solvers for the generalized Stokes problem. *International Journal for Numerical Methods in Fluids* 8(8): 869–895.

Cantwell CD, Sherwin SJ, Kirby RM, et al. (2011) From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements. *Computers and Fluids* 43: 23–28.

Elman H, Silvester D and Wathen A (2005) *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics*. Oxford: Oxford Science Publications.

Fries TP and Belytschko T (2010) The extended/generalized finite element method: An overview of the method and its applications. *International Journal for Numerical Methods in Engineering* 84(3): 253–304.

Galusinski C and Vigneaux P (2008) On stability condition for bifluid flows with surface tension: Application to microfluidics *Journal of Computational Physics* 227: 6140–6164.

Gee MW, Siefert CM, Hu JJ, et al. (2006) ML 5.0 smoothed aggregation user's guide. Technical Report 2006-2649, Sandia National Laboratories, USA.

Girault V and Raviart PA (1986) *Finite Element Methods for the Navier–Stokes Equations*. New York: Springer–Verlag.

Groß S and Reusken A (2007) An extended pressure finite element space for two-phase incompressible flows with surface tension. *Journal of Computational Physics* 224: 40–58.

Hager G and Wellein G (2011) *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton, FL: CRC Press.

Heroux MA, Bartlett RA, Howle VE, et al. (2005) An overview of the Trilinos project. *ACM Transactions on Mathematical Software* 31: 397–423.

Hirt CW and Nichols BD (1981) Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of Computational Physics* 39: 201–225.

Hysing S (2006) A new implicit surface tension implementation for interfacial flows. *International Journal for Numerical Methods in Fluids* 51: 659–672.

Jacqmin D (2000) Contact-line dynamics of a diffuse fluid interface. *Journal of Fluid Mechanics* 402: 57–88.

Karniadakis GE and Sherwin SJ (2005) *Spectral/hp Element Methods for Computational Fluid Dynamics*. 2nd ed. Oxford: Oxford University Press.

Kormann K and Kronbichler M (2011) Parallel finite element operator application: Graph partitioning and coloring. In: Laure E and Henningson D (eds) *Proceedings of the 7th IEEE international conference on eScience*, Stockholm, 5–8 December 2011, pp.332–339. Los Alamitos, CA: IEEE.

Kronbichler M, Heister T and Bangerth W (2012) High accuracy mantle convection simulation through modern numerical methods. *Geophysical Journal International* 191: 12–29.

Kronbichler M and Kormann K (2012) A generic interface for parallel finite element operator application. *Computers and Fluids* 63: 135–147.

May DA, Brown J and Le Pourhiet L (2014) pTatin3D: High-performance methods for long-term lithospheric dynamics. In: Kunkel JM, Ludwig T and Meuer HW (eds) *Supercomputing (SC14)*, New Orleans, LA, 16–21 November 2014, pp.1–11. Los Alamitos, CA: IEEE.

Olsson E and Kreiss G (2005) A conservative level set method for two phase flow. *Journal of Computational Physics* 210: 225–246.

Olsson E, Kreiss G and Zahedi S (2007) A conservative level set method for two phase flow II. *Journal of Computational Physics* 225: 785–807.

Osher S and Sethian JA (1988) Fronts propagating with curvature dependent speed: Algorithms based on Hamilton–Jacobi formulations. *Journal of Computational Physics* 79: 12–49.

Patterson DA and Hennessy JL (2009) *Computer Organization and Design*. 4th ed. Burlington, MA: Morgan Kaufmann.

Peskin C (1977) Numerical analysis of blood flow in the heart. *Journal of Computational Physics* 25: 220–252.

Peskin C (2002) The immersed boundary method. *Acta Numerica* 11: 479–517.

Rasthofer U, Henke F, Wall WA, et al. (2011) An extended residual-based variational multiscale method for two-phase flow including surface tension. *Computer Methods in Applied Mechanics and Engineering* 200: 1866–1876.

Saad Y (2003) *Iterative Methods for Sparse Linear Systems*. 2nd ed. Philadelphia, PA: SIAM.

Saye RI (2015) High-order quadrature methods for implicitly defined surfaces and volumes in hyperrectangles. *SIAM Journal on Scientific Computing* 37(2): 993–1019.

Sethian JA (2000) *Level Set Methods and Fast Marching Methods. Evolving Interfaces in Computational Geometry,* *Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge: Cambridge University Press.

Sundar H, Biros G, Burstedde C, et al. (2012) Parallel geometric-algebraic multigrid on unstructured forests of octrees. In: Hollingsworth J (ed.) *SC12: Proceedings of the international conference for high performance computing, networking, storage and analysis*, Salt Lake City, UT, 10–16 November 2012, pp.1–11. Los Alamitos, CA: IEEE.

Sussman M and Ohta M (2009) A stable and efficient method for treating surface tension in incompressible two-phase flow. *SIAM Journal on Scientific Computing* 31(4): 2447–2471.

Tadmor E (1984) Skew-self adjoint form for systems of conservation laws. *Journal of Mathematical Analysis and Applications* 103: 428–442.

Tuminaro R and Tong C (2000) Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. In: Donnelley J (ed.) *Super computing 2000 proceedings*, Dallas, TX, 4–10 November 2000, pp.1–21. Los Alamitos, CA: IEEE.

Turek S (1999) *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Berlin: Springer.

Unverdi SO and Tryggvason G (1992) A front-tracking method for viscous, incompressible, multi-fluid flows. *Journal of Computational Physics* 100: 25–37.

Wu Z, Willing B, Bjerketorp J, et al. (2009) Soft inertial microfluidics for high throughput separation of bacteria from human blood cells. *Lab on a Chip* 9(9): 1193–1199.

Zahedi S, Kronbichler M and Kreiss G (2012) Spurious currents in finite element based level set methods for two-phase flow. *International Journal for Numerical Methods in Fluids* 69: 1433–1456.

Authors biographies

Martin Kronbichler is a post-doctoral researcher at the Institute for Computational Mechanics, Technical University of Munich, Germany. He received a diploma degree (MS equivalent) in applied mathematics at Technical University of Munich in 2007 and a PhD degree in scientific computing with specialization in numerical analysis at Uppsala University, Sweden, in 2012. His research interests include finite element methods for flow problems, efficient numerical linear algebra, and parallel and high-performance implementations in modern scientific software. He leads research activities on fast matrix-free methods for continuous and discontinuous Galerkin methods within the deal.II finite element library.

Ababacar Diagne is a post-doctoral researcher at the department of Scientific Computing, Uppsala University, Sweden. He received a graduate degree in Applied Mathematics and Computer Sciences from Gaston Berger University of Saint Louis, Senegal, in 2007 and a PhD in Applied Mathematics with specialization in numerical analysis jointly at University Gaston Berger of Saint Louis and Royal Institute of Technology, Sweden, in 2012. His research interests include finite element methods for flow problems and control of hyperbolic systems with applications in environmental systems.

Hanna Holmgren is a PhD student at the Division of Scientific Computing at the Department of Information Technology, Uppsala University, Sweden. She received her Master of Science in Engineering at Uppsala University in 2012. Her research interests include finite element methods for two-phase flows and numerical models for simulation of moving contact lines.