Fast and Robust Solvers for Pressure Correction in Bubbly Flow Problems [⋆]

S.P. MacLachlan, J.M. Tang, C. Vuik

Delft University of Technology, J.M. Burgerscentrum, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft Institute of Applied Mathematics, Mekelweg 4, 2628 CD Delft, The Netherlands

Abstract

We consider the numerical simulation of two-phase fluid flow, where bubbles or droplets of one phase move against a background of the other phase. Such flows are governed by the Navier-Stokes equations, the solution of which may be approximated using a pressure-correction approach. Within such an approach, the computational cost is often dominated by the solution of a linear system corresponding to a discrete Poisson equation with discontinuous coefficients. In this paper, we explore the efficient solution of these linear systems using robust multilevel solvers, such as deflated variants of the preconditioned conjugate gradient method, or robust multigrid techniques. We consider these families of methods in more detail and compare their performance in the simulation of bubbly flows. Some of these methods turn out to be very effective and reduce the amount of work to solve the pressure-correction system substantially, resulting in efficient calculations for two-phase flows on highly resolved grids.

Key words: deflation, multigrid, bubbly flows, Poisson equation, discontinuous coefficients, conjugate gradient
1991 MSC: 65F10, 65F50, 65M06, 65M55, 65N22, 76T10

1. Introduction

Computation of two-phase flows and, in particular, bubbly flows is a very active research topic in computational fluid dynamics (CFD), see, for instance, [12, 17, 49, 59–62] and, more recently, [25, 34, 47, 50]. Understanding the dynamics and interaction of bubbles and droplets in a large variety of industrial processes is crucial for economically and ecologically optimized design. Two-phase flows are complicated to simulate, because the geometry of the problem typically varies with time and the fluids involved can have very different material properties. A simple example is that of air bubbles in water, where the densities vary by a factor of about 800.

Mathematically, bubbly flows are modeled using the Navier-Stokes equations coupled with an interface advection equation, which can be approximated numerically using operator-splitting techniques. In these schemes, equations for the velocity and pressure are solved sequentially at each time step, and the fluid

[⋆] This research was supported by the Dutch BSIK/BRICKS project and by the European Community's Sixth Framework Programme, through a Marie Curie International Incoming Fellowship, MIF1-CT-2006-021927.

Email addresses: `scott.maclachlan@tufts.edu` (S.P. MacLachlan), `J.M.Tang@tudelft.nl` (J.M. Tang), `C.Vuik@tudelft.nl` (C. Vuik).

density is explicitly advected. In many popular operator-splitting methods, the pressure correction is formulated implicitly, requiring the solution of a linear system at each time step. This system takes the form of a Poisson equation with discontinuous coefficients. Solving these systems typically consumes the bulk of the computing time, even though the operator is elliptic.

Such operator-splitting approaches are amongst the oldest numerical schemes for solving the incompressible Navier-Stokes equations, dating back to the original work of Chorin [10, 11]. In the 1980's, this work was extended to second-order convergent methods for the velocities [6, 63]; here, we follow the approach of Van Kan [63]. Because we consider two-phase flow, these pressure-correction techniques require a coupled equation to describe the evolution of density field within the flow. Here, we use the mass-conserving level set approach developed in [61, 62]. Many other approaches are possible, both for the solution of the Navier-Stokes equations and the advection of the density field. Artificial compressibility techniques, for example, replace the incompressibility condition by one with a small compressibility term that vanishes when treated appropriately, recovering, in this limit, the original (incompressible) Navier-Stokes Equations [10,32]. While we will only consider the standard finite-difference discretization, other approaches are also possible; finite-element discretizations of Navier-Stokes are complicated by the need to satisfy a discrete inf-sup condition to give stable pressure discretizations [20, 22]. While we use an interface-capturing level-set scheme, other approaches include front-tracking techniques [59], the volume-of-fluid and marker-and-cell methods, as well as arbitrary Lagrangian-Eulerian techniques. Lattice Boltzmann techniques may also be used to model incompressible multiphase flow, with similar considerations arising [26].

The solution of the pressure-correction equation within such operator-splitting approaches has long been recognized as a computational bottleneck in fluid flow simulation, see, e.g., [9,25]. In the case of single-phase fluids, a common approach to overcoming this bottleneck is the use of multigrid solvers for this equation [58, 69]. Standard geometric multigrid techniques (cf. [58, 69]) offer optimal-scaling solution properties for the pressure-correction equations in a single-phase fluid. For multiphase fluids, however, large differences in the fluid densities can lead to dramatic deterioration in the multigrid performance. Other approaches, such as cyclic reduction [51] (FISHPACK) and FFT-based techniques, that are appropriate for constant densities cannot even be applied in this case [59]. In this paper, we consider alternate approaches to solving the pressure-correction equation that do not exhibit the same sensitivity to jumps in the material properties.

Krylov subspace iterations, such as the preconditioned conjugate gradient (PCG) algorithm, offer one possibility for the efficient solution of these systems. However, traditional preconditioners, such as the incomplete Cholesky (IC) factorization [35], typically do not lead to optimal-scaling solution algorithms, even for single-phase fluids. An alternative to PCG with single-level preconditioners is a deflated variant of PCG, called DPCG, see, e.g., [40, 66]. The addition of a deflation technique serves to remove components associated with small eigenvalues that cause the slow convergence of PCG. In many applications, DPCG has been shown to be an efficient method, such as in porous-media [65] and ground-water flows [38]. Recently, DPCG has also been applied successfully to bubbly flows [54–56].

Another option for the efficient solution of the pressure-correction equation is the use of multigrid techniques that are applicable in more general settings. The black box multigrid technique, first introduced in [3], uses geometrically structured coarse grids in combination with an interpolation operator designed to account for the effects of jumps in the diffusion coefficients to achieve fast multigrid convergence in many situations [5, 13, 15]. Algebraic multigrid, or AMG, is also known to be effective for elliptic problems with jumps in their coefficients [45, 48], achieving this efficiency by tailoring both the coarse-grid structure and interpolation operator to account for the jumps in the coefficients. While both of these solvers have been applied successfully in many cases, the modeling of bubbly flows provides some unique challenges. In particular, in simulations with bubbles that appear at (or below) the finest resolution of the grid, these techniques may encounter difficulties in treating such small-scale effects through adapting the coarse-scale models. In these cases, we have found that using the above multigrid algorithms as preconditioners within PCG easily restores their optimal convergence behavior at a minimal extra cost.

The use of multigrid-preconditioned CG within pressure correction is not new. Indeed, multigrid was first considered for use as a preconditioner for discontinuous-coefficient problems very early in its history, see [7,29]. Under certain symmetry assumptions on relaxation and the coarse-grid operators, Tatabe demonstrated that multigrid always defines a positive-definite preconditioner (regardless of the number of pre- and post-relaxations) and, so, multigrid is acceptable as a preconditioner for PCG [57]. In the fluid dynamics literature, Tatabe's "MGCG" method has been adopted for the solution of the pressure-correction equation for variable-density flows [43, 49]. Similarly, the MUDPACK software package [1] has also been used for solving these equations [17, 18, 59], but this has been found to not offer robust performance to the large density jumps that appear in realistic simulations [18]. In contrast to many of these techniques, the multigrid algorithms considered here include two important features, Galerkin coarsening and operator-induced interpolation. Galerkin (or variational) coarsening creates the multigrid coarse-grid operators using matrix products that restrict the fine-grid operators onto the range of interpolation. This greatly simplifies the task of creating a consistent coarse-grid correction process for problems with density variations that are not resolved on the coarse scale. Operator-induced interpolation techniques further improve this approach by building interpolation operators tuned towards capturing the fine-scale modes that are slow to be reduced by simple relaxation on the variable-coefficient problem.

Algebraically, DPCG and PCG with a MG preconditioner are strongly related to each other, see [38,52,53]. In both algorithms, the same choices must be made. The fine-grid relaxation method or preconditioner is chosen to give effective treatment of certain modes of error. A complementary space is defined, in terms of a set of deflation vectors or the range of the multigrid interpolation operator, and an optimal correction over this space is computed using a variational approach. However, the preferred choices for these components are quite different between the two methods. While deflation techniques are typically based on a strong fine-scale preconditioner (e.g., IC) in combination with a coarse-scale correction over a very small space, multigrid techniques typically make use of a rather weak fine-scale relaxation technique (e.g. a Jacobi or Gauss-Seidel iteration) in combination with a coarse-scale correction over a space that is a large fraction of the fine-scale problem size. Furthermore, the treatment of the linear systems associated with the coarse scale are handled differently. Deflation techniques typically solve these systems using a direct or iterative method, whereas a recursive procedure is used in the multigrid approach.

In this paper, we make a detailed comparison between deflation and multigrid methods with their own typical parameters for problems of bubbly flow. While several previous papers have detailed the application of deflated PCG methods to bubbly flows [54–56], the use of advanced multigrid methods for these flow simulations has, to our knowledge, not been previously considered in the literature. As well, the systems of equations that arise in three-dimensional multiphase flows offer a good opportunity for comparison of these two families of solvers. In contrast to previous, theoretical comparisons, we focus here on evaluating each solver using its most advantageous selection of options.

The remainder of this paper is organized as follows. In Section 2, we review the solution of the Navier-Stokes equations using the standard operator-splitting approach. Section 3 presents the details of the deflation and multigrid methods considered here. Some implementation details of the two families of solvers are compared in Section 4; Section 5 presents numerical comparisons of the different solvers for three-dimensional simulations of bubbly flows in practical settings. Conclusions are given in Section 6.

2. Problem Setting

Bubbly flows are governed by the Navier-Stokes equations,

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho}\nabla \cdot \mu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right) + \mathbf{f}, \tag{1}$$

subject to an incompressibility constraint,

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

where $\mathbf{u} = (u, v, w)^T$ is the velocity vector, and ρ, p, μ and $\mathbf{f}$ are the density, pressure, viscosity and source vector (consisting of, for example, gravity and interface tension forces), respectively. Coupled with the Navier-Stokes equations is an interface advection equation, which describes how the boundary between phases evolves in time. We assume the density and viscosity are constant within each fluid. At the boundaries of the domain, Dirichlet boundary conditions are imposed on the velocity. Although accurate models of surface tension provide important details in the physical flows considered here, we neglect them in our experiments in Section 5, because these forces lead to simpler flow patterns than those given by the model without them. Our interest in this paper is to explore the robustness of various pressure-correction solvers to evolving density fields and, so, we look to consider bubbly flows that are both as realistic as possible, but also as challenging as possible; this is best accomplished by neglecting surface-tension terms.

Equations (1) and (2) are solved on an equidistant Cartesian grid in a rectangular domain using a pressure-correction method [63]. These equations are discretized using finite differences on a uniform staggered grid with n cells in each direction, where the grid points of the pressure variables are located at the cell centers, and the grid points associated with velocity components are located at the cell-face centers.

In the pressure-correction method, a tentative velocity vector, $\mathbf{u}^*$, is first computed from:

$$\frac{\mathbf{u}^* - \mathbf{u}^l}{\Delta t} = -\nabla \cdot \mathbf{u}^l\mathbf{u}^l + \frac{1}{\rho}\nabla \cdot \mu \left(\nabla \mathbf{u}^* + (\nabla \mathbf{u}^l)^T\right), \tag{3}$$

where $\mathbf{u}^l$ denotes the value of $\mathbf{u}$ at time step l. The resulting system of equations for unknown vector $\mathbf{u}^*$ is solved, for example, using the PCG method. The velocities at the new time step, $l+1$, are computed from

$$\frac{\mathbf{u}^{l+1} - \mathbf{u}^*}{\Delta t} = -\frac{1}{\rho}\mathcal{G}p + \mathbf{f},$$

under the constraint of (2). This yields

$$\begin{cases} \mathbf{u}^{l+1} &= \mathbf{u}^* + \Delta t \left(-\frac{1}{\rho}\mathcal{G}p + \mathbf{f}\right), \\ \mathcal{D}\mathbf{u}^{l+1} &= 0, \end{cases} \tag{4}$$

where $\mathcal{D}$ represents the discretization of the divergence operator, and $\mathcal{G}$ is the discrete gradient operator. Thus, Equation (4) gives

$$\mathcal{D}\frac{1}{\rho}\mathcal{G}p = \mathcal{D}\left(\frac{1}{\Delta t}\mathbf{u}^* + \mathbf{f}\right), \tag{5}$$

which is known as the pressure-correction equation and also as the Poisson equation with a discontinuous coefficient. Equation (5) can again be solved using, for example, the PCG method; however, solving (5) typically requires significantly more computing time than the solution of (3), since the convergence of the iterative process suffers from the highly discontinuous behavior of the coefficient, ρ, but is not ameliorated by the small time step, Δt. Without the use of advanced solvers for this step, the solution of the pressure-correction equations may require orders of magnitude more computing time per time step than the rest of the computation. Thus, we focus here on the efficient solution of Equation (5) alone.

Finally, once the velocity, $\mathbf{u}^{l+1}$ has been found, the interface between the phases is explicitly advected using a mass-conserving level-set approach [61, 62]. In this approach, the interface between the phases is represented as the zero-valued level set of a marker function. Because the marker function is advected explicitly and the velocity is divergence-free, such an approach is automatically mass conserving. Further details about this approach to modeling bubbly flows can be found in [61, 62].

Due to the staggered grid, we do not have pressure points at the boundaries of the domain. Explicit pressure boundary conditions are, however, not required in the method, since, in Eq. (5), the velocity boundary conditions are naturally included in the discrete divergent operator, $\mathcal{D}$. It follows implicitly that Neumann boundary conditions hold for the pressure. In this case, the pressure is a relative variable, since the differences in pressure and not its absolute values are meaningful in the pressure-correction method.

Eq. (5) can be written as a symmetric and positive semi-definite (SPSD) linear system

$$Ax = b, \quad A = [a_{ij}] \in \mathbb{R}^{N\times N}, \tag{6}$$

for $N = n^3$, and a singular matrix, A. The solution, x, to (6) is, then, only determined up to a constant; i.e., if x_1 is a solution of (6), then $x_1 + c$ is also a solution for any arbitrary constant vector, $c \in \mathbb{R}^N$. This situation presents no real difficulty for the iterative solver, as long as

$$b \in \mathrm{Col}(A), \tag{7}$$

see [28]. It appears that Eq. (7) is always satisfied for our linear system (6), see [54] for more details.

Here, we consider two-phase bubbly flows with, for instance, air (a low-density phase) and water (a high-density phase). In this case, ρ is piecewise constant with a contrast, ϵ, which is the ratio of the two densities, i.e.,

$$\rho = \begin{cases} \rho_0 = 1, \ \mathbf{x} \in \Lambda_0, \\ \rho_1 = \epsilon \quad \mathbf{x} \in \Lambda_1, \end{cases}$$

where Λ_1 is the volume occupied by the low-density phase, namely air bubbles in the domain, Ω, and Λ_0 is the volume occupied by the high-density phase, namely the fluid domain around these bubbles.

3. Numerical Methods for the Pressure-Correction Equation

Solving (6) is complicated by the number of points necessary to accurately capture the evolving flow. Whether finite difference, finite element, or finite volume techniques are used for the discrete gradient and divergence operators in (5), the resulting matrix will be sparse, but with a bandwidth of n^2, in lexicographic ordering. For a discretization on a cube with $n = 100$ grid points in each direction, this means that the matrix is of dimension $N = 10^6$, with bandwidth $n^2 = 10^4$. It is well-known that direct solution techniques without reordering require a number of operations that scales as n^7 for a banded Cholesky decomposition, giving $\mathcal{O}(10^{14})$ operations in the example above. Thus, here, we consider the solution of (6) using iterative techniques.

The solution of (6) using iterative methods is, itself, complicated both by the matrix size and its condition number. Using a regular, Cartesian-product grid with n grid points in each direction guarantees that the matrix, of dimension $N = n^3$, also has $\mathcal{O}(N)$ non-zero entries. Thus, the cost of a single matrix-vector product is proportional to the number of degrees of freedom in the system and, as a consequence, the cost per iteration of most classical iterative techniques will also be $\mathcal{O}(N)$. The number of iterations required by such a technique to produce a converged solution, however, is dependent on the condition number of A. For any of the standard discretizations, $\kappa(A)$ depends in the usual way on the problem size, $\kappa(A) \propto n^2$. For multiphase flow, however, this simple bound is compounded by variations in the material parameters between the different phases. For example, given a density contrast, $\epsilon \leq 1$, $\kappa(A) \propto \frac{1}{\epsilon}$. Thus, solution of (5) is more complicated than solution of the corresponding homogeneous Poisson equation.

3.1. *Conjugate Gradient and Preconditioning*

Because of the symmetry and (semi-)definiteness of the matrix, A, a natural choice for the iterative solution of (6) is the conjugate gradient (CG) algorithm. However, because of the ill-conditioning of A, we expect CG to converge quite slowly. Indeed, using the standard bound [23],

$$\|x^{(L)} - x\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^L \|x^{(0)} - x\|_A, \tag{8}$$

where $x^{(L)}$ denotes the approximation to x, the solution of (6), after L steps of CG, $x^{(0)}$ denotes the initial approximation, and $\kappa(A)$ is the condition number of A, we see that the expected number of iterations required to achieve a fixed tolerance grows with $\kappa(A)$.

For this reason, CG is rarely applied directly to (6) but, rather, is typically applied in preconditioned form, which implicitly uses the matrix $M^{-1}A$ in place of A, for some SPD matrix, M, such that $M^{-1}r$ is easily computed for an arbitrary residual, r. The incomplete Cholesky factorization of A [35] is one commonly used preconditioner, which is known to greatly improve the performance of the CG iteration for moderate matrix sizes, at relatively little additional cost. Incomplete Cholesky preconditioned CG, or IC-CG, is not, however, an ideal iteration, as is well-known for the single-phase flow case, where the number of iterations required to solve (6) still grows with the problem size. It has also been observed that the performance of IC-CG diminishes under variation of some of the other problem parameters; see results in Section 5.

3.2. *Deflation Approaches*

Writing the linear system that is solved in IC-CG as

$$M^{-1}Ax = M^{-1}b,$$

where M denotes the Incomplete-Cholesky preconditioner, we could try to improve the performance of the IC-CG method by including a second operation in the preconditioner,

$$M^{-1}PA\tilde{x} = M^{-1}Pb, \tag{9}$$

for some operator, P. The choice of P should then, as in the original choice of M, be done to complement the spectrum of $M^{-1}A$. Deflation methods [40] choose P based on a two-grid projection approach that is easy to implement within an existing preconditioned CG code.

In many applications, including the one considered here, it is the subspace associated with the small eigenvalues of A, and of $M^{-1}A$, that causes slow convergence of the conjugate gradient iteration. If this subspace is small, and if an approximate basis for this subspace is known explicitly, then P could take the form of an inexpensive projection of components in this subspace out of an approximation or a residual. Writing the matrix, Z, as the matrix of basis vectors,

$$Z := [z_1 \; z_2 \; \cdots \; z_K], \tag{10}$$

for vectors z_i, $i = 1, \ldots, K$, and choosing $P = I - AZE^{-1}Z^T$, with $E = Z^TAZ$, gives P as the orthogonal projection onto the complement of the range of Z in the A-inner product. With this choice of P, we can view the system given in (9) as a preconditioned system for the solution of the singular system, $PA\tilde{x} = Pb$. Because PA is singular, we must postprocess the solution, $\tilde{x}$, of (9) to get the solution of (6). Given Z, E, and P defined as above, it is easy to see that the solution, x, to (6) can be expressed as

$$x = ZE^{-1}Z^Tb + P^T\tilde{x},$$

see, e.g., [38].

If the subspace associated with the small eigenvalues is not known exactly, or the subspace is too large to efficiently treat exactly, deflation techniques based on other vectors may be more effective. Approximate eigenvectors [41,46], vectors based on Krylov subspace recycling processes [31,42], or vectors based on domain decomposition [33, 40] may all be used to define effective deflation approaches. Here, we consider vectors based on domain decomposition. In particular, we consider the division of each axis of the (discrete) domain into k non-overlapping equi-sized “super-cells”, and define $K = k^3$ subdomains using the tensor-product grid generated by these intervals. With each subdomain, we associate a single deflation vector, z_i, which takes the value 1 at each grid point included in subdomain i, and value 0 at all other grid points. With this choice for the deflation vectors, the deflation matrix, Z, as defined in (10), is quite sparse, and computations involving Z and P may be performed relatively efficiently. The subdomain vectors give good approximations to a large subspace associated with the small eigenvalues of the preconditioned system, $M^{-1}A$ [55].

If K becomes large, the coarse linear systems involving E become more costly to solve and, in particular, use of standard sparse direct solvers may be inefficient. Instead, an iterative solver can be adopted to deal with these coarse linear systems. In this paper, we use a variant of Deflated IC-CG, or DIC-CG, where the coarse linear systems are, themselves, solved using IC-CG [54]. Of course, as K increases, even standard iterative solution of the coarse-level systems can become quite expensive, and deflation-like techniques should also be incorporated into the coarse-level IC-CG algorithm, leading to a recursive multi-level deflation method [16]. This is, however, not considered in the results presented here.

3.3. *Multigrid Approaches*

Since the equation for the pressure closely resembles a diffusion equation, another class of techniques to consider for the solution of (5) is the family of multigrid methods. In the case of single-phase flow, in particular, the equation for pressure reduces to a constant-coefficient Poisson equation, for which geometric multigrid methods are known to provide optimal solution techniques [58, 69]. For multiphase fluids, more complicated multigrid techniques are necessary to achieve optimal performance; such techniques are well-known in other fields. Here, we present the details of these methods, and their specialization to solving (5).

Just as deflation methods use a correction over a small subspace to account for the deficiencies of a classical preconditioner, all multigrid methods combine the use of a coarse-scale (or coarse-grid) correction process that is aimed at correcting modes that a fine-scale iteration (or relaxation) is slow to resolve. Different multigrid methods differ in the choices made in these two processes, particularly in the details of how the coarse-scale space is chosen and how that correction is computed. Here, we will concentrate on multigrid methods that make use of simple, pointwise relaxation techniques, such as the Jacobi or Gauss-Seidel iterations, and consider four different choices for the coarse-grid correction procedure. A brief introduction into classical multigrid techniques is presented in the following subsection; for more details, consult, e.g., [58, 69].

3.3.1. *Geometric Multigrid*

For the case of single-phase flows discretized on a regular grid, the use of geometric multigrid techniques has long been studied (cf. [58, 69]). These methods are based on the realization that, while simple iterations, such as weighted-Jacobi and Gauss-Seidel, do not efficiently resolve all modes of the solution, they do quickly and effectively damp a large subspace of errors for a large class of matrices, including those considered here. In particular, for single-phase flow (constant density), the errors that are not quickly damped by these simple iterations are dominated by so-called "smooth" error modes; errors that vary slowly between neighboring grid points. It is, thus, the job of the coarse-grid correction process to attenuate exactly these modes.

An important difference between the coarse-grid correction processes in multigrid and deflation techniques is the size of the subspace employed for coarse-grid correction. While deflation aims for a correction over a much smaller subspace than the fine-scale problem size, the size of the coarse-grid problem in a multigrid method is specifically chosen to be a relatively large fraction of the fine-grid size; typical coarsening rates are by a factor of 2 or 3 in each dimension. Such slow coarsening is justified by considering the convergence behavior of the complementary stationary iteration; for any fixed reduction tolerance, the number of error modes which are reduced in magnitude by a factor larger than that tolerance (i.e., the number of slowly converging modes) increases with the size of the fine-grid matrix. Thus, to achieve convergence that is truly independent of problem size using classical stationary iterations requires that the size of the coarse-grid problem always remains a fixed fraction of the fine-grid problem size.

Here, we consider a geometric multigrid approach using Gauss-Seidel relaxation. For maximum efficiency, the algorithm is specialized to grids with the number of grid points in each dimension of the form $p2^q$, where p is a small integer (typically, $p = 1$ or 3), so that the coarse grid may be chosen by reducing the grid size by a factor of 2 in each direction at all levels. Interpolation is trilinear cell-centered interpolation, while restriction is taken as cell-centered piece-wise constant restriction [36]. Relaxation and residuals on the coarse grid are realized using direct discretization of the fine-scale homogeneous problem on the coarse grid. This is possible because of the assumption of constant-density for the single-phase (incompressible) flow; for the two-phase flows considered here, more sophisticated techniques, described below, are needed to discretize, on the coarse scale, a fine-scale flow that may have phase boundaries that cannot be represented on the coarse grid.

While geometric multigrid techniques often yield the fastest solvers for the constant-coefficient version of (5), several complications arise in extending these techniques to the case of variable density. The smoothing properties of simple relaxation techniques, such as Jacobi or Gauss-Seidel, are highly dependent on the density; in the case of non-constant density, the dominant errors after relaxation may exhibit sharp transitions and/or cusps, which must be accounted for in the coarsening process. Furthermore, if the variations in density have fine-scale features (as we expect for bubbly flows), it may not be clear how best to represent the equations on the coarser grid, as needed in the multigrid process. In the following subsections, we discuss several approaches for overcoming these obstacles.

Another approach to overcome these complications is to use the geometric multigrid method as a preconditioner for the PCG method. For a problem with bounded density, $\rho_{\min} \leq \rho \leq \rho_{\max}$, we expect

$$\frac{1}{\rho_{\max}} x^T A_1 x \leq x^T A_\rho x \leq \frac{1}{\rho_{\min}} x^T A_1 x,$$

where A_ρ is the discretization of (5) with density ρ, and A_1 is the discretization of the constant-coefficient Poisson problem. Thus, $\kappa(A_1^{-1} A_\rho) \leq \frac{\rho_{\max}}{\rho_{\min}}$, where we ignore the common null-spaces of A_1 and A_ρ. This means that, for small density contrasts, the constant-coefficient and variable-coefficient problems are spectrally equivalent, with a small spectral-equivalence bound. A good solver, such as geometric multigrid, for the constant-coefficient case is then expected to make a good preconditioner for the variable density case, so long as the density contrast is not too significant. Results for this approach are reported in Section 5 as method GMG-CG. All of the other multigrid approaches that we develop here may be applied both as a standalone solver and as a preconditioner for CG. In what follows, we discuss only the case of these techniques being used as standalone solvers; see Section 3.3.5.

3.3.2. *Galerkin Coarsening*

While GMG-CG (and geometric multigrid in general) performs well when the density contrast is small, its performance suffers greatly when problems with large density contrasts are considered (as shown in Section 5). Improving the multigrid performance requires improvement in one (or both) of the multigrid components, relaxation or coarse-grid correction. In GMG-CG, however, direct relaxation on (5) is replaced by relaxation on the constant-coefficient problem as a preconditioner for the variable-density problem of interest. Thus, a first step in improving the performance of GMG-CG would be to replace the relaxation on the homogeneous problem with that on the real problem of interest.

Making this improvement on the fine scale is simple to implement; both the constant-density and variable-density problems are well-defined on the fine scale, and it is relatively simple to replace relaxation on one with relaxation on the other. On coarse scales, however, we have no direct representation of the fine-scale problem, unless it is possible to represent the variation in the density naturally on the coarse scale. For many flows of interest, this will clearly not be the case for all coarse scales in a multigrid hierarchy. Thus, we need some indirect way to account for fine-scale variations in the density directly in relaxation on the coarse scales.

There are many possible ways to create a coarse-scale model with a fine-scale density distribution; the problem of numerical homogenization, or upscaling, has been studied in many disciplines [67]. While these techniques focus on defining an effective density coefficient that can be naturally represented on the coarse scale only, we will instead focus on the multigrid point of view that the primary purpose of relaxation on the coarse scales is not to represent the flow on those scales but, rather, to compute an appropriate correction to the errors in the fluid pressures on the fine scale.

Using the coarse-grid models to improve a fine-grid approximation to the pressure naturally leads to the question of how good a correction is possible from a coarse grid. Mathematically, we consider a fixed coarse grid and interpolation operator, Z, that maps from the coarse grid to the fine grid. Asking for the best possible correction from the coarse grid, i.e., the best correction in the range of Z, means that we wish to minimize some norm of the error, e, in our approximation, $x^{(L)}$, to the discrete pressure, x, that satisfies (6). Writing the corrected approximation as $\hat{x}^{(L)} = x^{(L)} + Zy$, for some vector y, this means that we wish to minimize

$$\|\hat{e}\| = \|x - (x^{(L)} + Zy)\| = \|e - Zy\|.$$

Both the minimum value and the coarse-grid vector, y, for which the minimum is achieved depend strongly on the norm chosen for the minimization. Choosing the A-norm, $\|w\|_A^2 = w^T A w$, implies that the optimal choice for y satisfies

$$(Z^T A Z)y = Z^T A e = Z^T (b - Ax^{(L)}).$$

That is, the best possible coarse-grid correction for a fixed choice of the multigrid interpolation operator, Z, may be expressed in terms of a coarse-grid matrix, $E = Z^T A Z$, and restriction of the fine-grid residual, $b - Ax^{(L)}$, using Z^T as the restriction map. This choice of coarse-grid and restriction operators is known in the multigrid literature as Galerkin coarsening [39, 58] because of its close relationship to Galerkin finite elements.

A first generalization of GMG-CG is then to consider the multigrid method with interpolation, Z, given as in GMG-CG, but with relaxation on all levels replaced by Gauss-Seidel relaxation on the finest-grid matrix and its Galerkin restrictions. This technique, which we will denote by CCMG, was first proposed for problems similar to (5) in [68] and later studied in [30, 64].

3.3.3. *The Black Box Multigrid Method*

While CCMG offers a great improvement over GMG-PCG in terms of its scalability, its performance still degrades as the density contrast increases (see, for example, Table 2 in Section 5). As CCMG arose through improvements to the relaxation phase in GMG-PCG, we now consider the role of interpolation in the performance of CCMG. Of particular importance in achieving consistent multigrid performance regardless of the density contrast or configuration of the flow is the principle of complementarity; as multigrid methods aim to reduce errors through two distinct processes, relaxation and coarse-grid correction, optimal multigrid performance can only occur when these processes are appropriately complementary.

While we could aim to improve the performance of CCMG by making further improvements to the relaxation routine, such improvements often dramatically increase the cost of the iteration. Instead, we aim to improve the multigrid performance by making a different choice for the interpolation operator, Z, to better complement the performance of lexicographical Gauss-Seidel relaxation. It has long been recognized that for problems with discontinuous coefficients, such as (5), the errors left after relaxation are not smooth, as in the case of constant-coefficient problems [3]. Thus, while coarse-grid correction with a fixed interpolation operator, such as those analyzed in [36] and discussed above, may be used to effectively complement relaxation for constant-coefficient problems, they are less appropriate when the problem contains large jumps in its coefficients.

The solution to the problem with large jumps in coefficients, first discussed in [3] and further developed in [13, 14], is to allow the coefficients of the interpolation operator, Z, to depend on the coefficients of the matrix, A. Such an operator-induced interpolation is better able to reflect the slow-to-converge errors of relaxation as these errors are, themselves, dependent on the variation in A. The technique of the Black Box Multigrid Method, or BoxMG [13], defines the coefficients of interpolation to a fine-grid point by combining the entries of the matrix in the rows corresponding to the grid point and its graph neighborhood.

In three dimensions, the BoxMG algorithm assumes that the fine-grid matrix comes from the discretization of an equation such as (5) on a logically rectangular grid [14]. The fine-grid operator is then assumed to have at most a 27-point connectivity structure; for each grid point, connection is only allowed to grid points that reside in grid point neighboring (possibly only at corners) that in which the grid point lies. The coarse grid is constructed by removing every other plane of grid points in each direction, in contrast to the coarsening used in GMG and CCMG, where finite volumes were aggregated in pairs in each direction. Interpolation in BoxMG then falls into four categories: fine-grid points may be themselves coarse-grid points (as the coarse grid is embedded in the fine grid), they may lie on the line segment connecting two coarse-grid points, they may lie in the same plane as four coarse-grid points, at the center of the square defined by these grid points, or they may lie in a plane with no coarse-grid points, at the center of the cube defined by 8 coarse-grid points.

Interpolation of corrections to embedded coarse-grid points is always done using injection, as the errors at these grid points are directly represented on the coarse grid. For fine-grid points lying between two coarse-grid points, the interpolation weights are defined by first adding the matrix entries in the row of A corresponding to the grid point along the planes orthogonal to the connecting line, collapsing the 27-point stencil to a 3-point stencil joining these three grid points. The correction to the grid point is then computed by setting this three-point stencil to zero, substituting in the injected corrections at the coarse-grid points and solving for the correction at the fine-grid points. A similar approach is used for fine-grid points lying at the center of a square in the plane of four coarse-grid points, however first the four other neighboring grid points in that plane are resolved using the first approach. In this way, the 27-point stencil need only be collapsed to a 9-point stencil, which can be treated using the previously computed corrections as in the 3-point stencil case. Finally, the correction to the grid point at the center of the cube defined by 8 coarse-grid points may be calculated directly, by satisfying the 27-point stencil at this grid point using the corrections computed at all of its neighbors [5].

The coarse-grid matrix in BoxMG is again defined using a Galerkin coarsening, with Z defined as described above. It can be verified that, if the fine-grid stencil has its non-zero connections confined to within a 27-point stencil pattern, then the coarse-grid operator also has 27-point connectivity. Thus, the BoxMG method may be applied recursively to define a full multigrid hierarchy. While there are many ways to use knowledge of the discrete operator, or of the density distribution itself, to define the multigrid hierarchy, the approach taken in BoxMG has been shown to be successful for a wide variety of problems. In [37], it is shown that one of the reasons for the success of BoxMG is that defining interpolation in this way approximately preserves the continuity of the normal flux, $\left(\frac{1}{\rho}\nabla p\right)\cdot\mathbf{n}$, across an interface with a jump in the density. Thus, BoxMG can be thought of as combining effective multigrid principles with useful physical insight in achieving a stable and efficient solution algorithm.

3.3.4. *Algebraic Multigrid*

With the early papers on BoxMG [3, 5, 13], it was recognized that the combination of operator-induced interpolation and Galerkin coarsening can lead to very robust methods for a wide class of problems. The algebraic multigrid method [8, 45], or AMG, is an algorithm based on a different implementation of the same principles, but which can be applied to an even larger class of problems. In particular, AMG can be applied to problems without a regular grid structure and allows for the choice of unstructured coarse grids regardless of the fine-grid operator structure.

The central idea of AMG is that all components of the coarse-grid correction cycle should be determined by the properties of the fine-grid operator. The first step in coarsening is then to determine whether two grid points that are connected in the fine-grid operator (grid points i and j are said to be connected if $a_{ij} \neq 0$) are connected in a significant way. Each grid point, i, is said to strongly depend on any of its neighboring grid points for which a_{ij} is of similar size as the largest entry in row i. For M-matrices, such as the natural discretization of (5), we define the set of grid points that i strongly depends upon as

$$S_i = \left\{ j : -a_{ij} \geq \theta \cdot \max_{J\neq i}\{-a_{iJ}\} \right\},$$

for some suitable θ, $0 < \theta \leq 1$. Once these strong connections have been identified, a coarse grid is formed by taking a maximal independent set of the graph created by the set of edges, a_{ij}, where $j \in S_i$.

To define interpolation in AMG, a similar strategy is used to collapse connections between grid points that appear only on the fine grid and define an interpolation operator. Choosing the coarse grid through the maximal independent subset algorithm described above implies that the coarse-grid points are embedded in the fine grid. For any fine-grid points, i, that is not also a coarse-grid point, interpolation can be defined by collapsing the connections from i to other fine-grid points, j, based on their common coarse-grid neighbors. Unlike BoxMG, this is done without using any intuition into the couplings involved; the elimination of these fine-fine connections is a purely algebraic operation. Each fine-fine connection is replaced using a weighted average of the coefficients connecting the fine-grid point, j, to the coarse-grid neighbors of grid point i; see [45, 48] for details.

Because both the choice of the coarse grid and of the interpolation operator in AMG are determined based on the fine-grid operator, we expect AMG to have convergence properties similar to, or possibly even better than, those of BoxMG. However, the price paid in AMG for this robustness is the use of completely unstructured matrix and vector data structures, as a result of the unstructured grid hierarchy. Additionally, the cost of the additional operations to compute the coarse grid (and, in fact, a more expensive computation of the interpolation weights) makes AMG an expensive alternative in situations where BoxMG (or CCMG or GMG) is expected to perform well. Nevertheless, AMG is often the method of choice in commercial codes where robustness is considered more important than achieving the fastest possible solution time. Indeed, in CFD, AMG solvers have been recognized as an important tool for achieving efficient solution in a wide variety of flow regimes [44].

3.3.5. *On the need for preconditioning*

While the multigrid methods discussed here are typically considered as standalone solvers, it is sometimes useful to also consider them as preconditioners for CG. Both BoxMG and AMG aim to directly treat the fine-scale structure of the density field through the use of operator-dependent interpolation algorithms. It is possible, however, that operator-dependent interpolation alone is not sufficient to yield an optimal solution algorithm in all situations.

Consider a case with contrast ϵ, where a small bubble with density ϵ is embedded in a background medium with unit density. In both AMG and BoxMG, interpolation weights near the boundary of the bubble are determined to be proportional to the matrix entries in that region. Because the matrix comes from a discretization of (5), these matrix entries will be proportional themselves to the inverses of the densities; some matrix elements will have size $\mathcal{O}\left(\frac{1}{\epsilon}\right)$ and some will have size $\mathcal{O}(1)$. For small ϵ, both algorithms typically give small interpolation weights from a coarse-grid point that is located outside of a bubble to a fine-grid point inside of the bubble. Physically, this also makes sense. Across the bubble boundary, we expect to see continuity of both the pressure, p, and its normal flux, $\left(\frac{1}{\epsilon}\nabla p\right) \cdot \mathbf{n}$. When there is a large jump in ϵ between the bubble and the background, we expect to also see a large jump in ∇p across the bubble's edge, which makes interpolation across this edge difficult, particularly in the direction of high density (small ∇p) to low density (large ∇p).

Following this reasoning, we see that it is quite natural for AMG and BoxMG to not interpolate significantly across bubble boundaries. A natural requirement from this point of view would be to require, for each fine-scale bubble, a sufficient number of coarse-grid points to lie within the bubble, to allow for an accurate computation of a coarse-grid correction for all grid points within the bubble. In a real-life simulation, however, this may not be practical, due to bubbles and droplets that may only be resolved at the size of a single grid point on the fine scale. In Table 4, below, we see that the number of iterations for both BoxMG and AMG without the use of a Krylov wrapper increase as the number of bubbles increases. While such an increase is not dramatic, it can easily be attenuated by the use of these multigrid methods as a preconditioner for CG; in this case, the multigrid method gives good convergence for almost all types of errors, and the CG acceleration effectively resolves the few error modes associated with these small bubbles. In Section 5, we denote the experiments with CCMG, BoxMG, and AMG preconditioners for CG by CCMG-CG, BoxMG-CG, and AMG-CG, respectively.

4. Implementation and Computational Cost

In the numerical experiments in Section 5, we make use of standard implementations of the methods discussed above, when available. In the following subsections, we discuss the relative costs of these techniques, as well as their scalability for large problem sizes. Relative to CG by itself, or even to IC-CG, all of the other methods considered here have a larger cost per iteration. Their utility lies in the significant reduction in iterations possible using a multilevel technique as compared to a single-level method, such as IC-CG. Here, we stress the details of the relative costs of a single cycle of these algorithms, as a prelude to the numerical results in Section 5.

4.1. *Cost of Deflation*

The computational cost of the deflation method is discussed in [54]. The setup of the deflation method is rather cheap, since Z may be constructed independently of the problem matrix. Furthermore, it is not necessary to store Z explicitly in memory; AZ and E may be computed beforehand. In the 3D case, construction of AZ and E can each be done in $\mathcal{O}(n^2k)$ flops, assuming that E has dimensions $K \times K$, for $K = k^3$.

Moreover, DIC-CG needs only one more step than IC-CG at each iteration. The additional cost for the deflation step is $\mathcal{O}(N+\theta)$ flops, where θ is the number of flops required for the inner solves involving E. Using IC-CG as inner iterative solver, then each inner iteration costs $\mathcal{O}(K)$ flops, and at most $\mathcal{O}(k)$ iterations are required to achieve sufficient accuracy in the inner solve (based on a simple condition-number bound and

(8)), leading to $\theta = \mathcal{O}(k^4)$ operations. Note that, because AZ can be precomputed and is much sparser than A, there are no additional matrix-vector multiplications with A required at each iteration of DIC-CG.

While, in principle, DIC-CG may be applied in an unstructured manner, the implementation considered here is based on the assumption of a rectangular tensor-product grid. This allows significant savings in both the storage and the computational cost required by the iteration, as structured matrix data structures may be used in place of the more general (and more costly) storage required by an unstructured implementation. In this sense, the DIC-CG methods tested here are more comparable to a geometric multigrid approach than to AMG, although the DIC-CG algorithm, in principle, could be applied in the same unstructured settings as AMG.

It is also important to note that the implementation of DIC-CG is slightly different than for the other methods considered here. While all other methods are realized as PCG iterations for solving $Ax = b$ with various choices of preconditioners, DIC-CG is implemented as a preconditioned system for the deflated system, as in (9). This means that the stopping criterion within the DIC-CG iteration is based on the deflated residual, $Pb - PA\tilde{x}$, and not on the original residual, $b - Ax$. In exact arithmetic, $Pb - PA\tilde{x} = b - Ax$ and, so, this has no effect on the applicability of DIC-CG. It does, however, have some effect on the overall accuracy, as some additional numerical error is introduced in the post-processing of the solution generated by DIC-CG to get the solution to the original linear system, $Ax = b$.

4.2. *Cost of Multigrid*

The relative costs of the multigrid methods studied here, in principle, increase with the complexity of the algorithm. Geometric multigrid can easily be implemented in a very efficient manner. In fact, because GMG-CG uses relaxation only on the homogeneous problem, it may be implemented in a fully matrix-free manner. The same stencil is applied everywhere on each level, up to boundary conditions, and the simple transfer operators can be implemented again with constant stencils away from the boundaries. Thus, the true computational cost of a single iteration of GMG-CG is much smaller than that of a method with similar number of operations because of the optimization possible under the assumption of constant coefficients in the operator.

While multigrid methods use much finer coarse grids than typical deflation methods, their recursive treatment of these grids leads to an overall cost per iteration that remains $\mathcal{O}(N)$. Consider, for example, the cost of a single GMG V(1,1) cycle (that is, the cost of a single preconditioning step in GMG-CG). On each level of the multigrid hierarchy, 2 relaxation sweeps are performed at the appropriate resolution. On every grid, the cost of these relaxation sweeps is directly proportional to the size of that grid. Thus, $\mathcal{O}(N)$ operations are required for each sweep on the finest grid, $\mathcal{O}(\frac{N}{8})$ operations are required for each sweep on the first coarse grid (which has size $\frac{N}{8}$), and $\mathcal{O}(\frac{N}{64})$ operations per sweep are required on the next coarsest grid, etc. Overall, the total number of operations required to perform two relaxation sweeps on all levels is then bounded by $\frac{16}{7}$ times the cost of a single sweep of relaxation on the finest level, which is $\mathcal{O}(N)$. Similarly, the additional storage requirements for GMG-CG can also be bounded by a small constant times the fine-scale, $\mathcal{O}(N)$ storage requirement for CG.

CCMG adds the cost of structured matrix storage and operations on all levels, as well as that of the Galerkin product in the setup stage of the algorithm. For the numerical results presented in Section 5, we use 64-point interpolation and restriction stencils, corresponding to bilinear interpolation and its adjoint for restriction. This results in some growth in the stencil size on coarser grids, but this growth can be easily quantified and still included in a structured-grid matrix data structure. Alternately, lower-order interpolation and restriction may be used, as in [30], to control the growth of the stencil on coarse grids. While these costs somewhat increase the cost of CCMG relative to GMG, the overall cost per cycle for CCMG remains $\mathcal{O}(N)$, as the added storage and computational costs on each level are bounded by a small constant times the number of unknowns on that level.

While variants of the CCMG algorithm are often used for variable-coefficient problems, it is also well-known that this algorithm does not offer perfect scalability as density ratio between the phases increases. Because of this, we do not focus on achieving an optimal implementation of the CCMG algorithm; in Section

5, we demonstrate that the iteration counts for CCMG clearly scale less well than those for DIC-CG, BoxMG, and AMG for the problems considered here. Instead, the CCMG method has been implemented within the AMG code, with a fixed choice of coarse grids and transfer operators. As a result, the implementation does not use the most efficient data structures and the reported CPU times in Section 5 are much larger than strictly necessary for CCMG, although we stress that the iteration counts and final residuals are the same for this implementation as they would be for a more efficient one. In practice, the setup costs for CCMG should be cheaper than those for BoxMG, due to the fixed interpolation pattern. However, the cost of a single iteration of CCMG is expected to require more operations than a single iteration of BoxMG, as BoxMG uses a 27-point interpolation stencil, compared to CCMG's 64-point stencil, leading to denser coarse-grid matrices for CCMG when compared with BoxMG.

In contrast to GMG and CCMG, BoxMG is not based on cell-centered data structures. Instead, BoxMG is, primarily, a node-based code; however, its robustness leads to successful results for our cell-centered discretization as well. The added cost in BoxMG is primarily in the setup phase of the algorithm, where coefficients of the operator on each level are used in determining interpolation to that level. Coarsening in BoxMG is also slightly different than that in CCMG and GMG, as it naturally takes nodal fine grids of $2^q + 1$ grid points in each dimension into nodal coarse grids with $2^{q-1} + 1$ grid points; however, because of the use of operator-induced interpolation, BoxMG is also able to successfully solve problems with grids of arbitrary sizes, as will be seen in Section 5, while maintaining multigrid's typical $\mathcal{O}(N)$ complexity per cycle.

Finally, AMG has the highest cost per iteration of the multigrid (and other) approaches considered here, because of the added cost of its unstructured grid processing and data structures, as well as a significant setup cost. This is to be expected; our choice of a structured-grid discretization naturally suggests that the best efficiency will be obtained with a structured-grid solver. Results for AMG are included to answer two interesting questions. First, it is interesting to see how much of a performance loss is seen with these unstructured data structures; results in Section 5 suggest that AMG is typically a factor of 10 to 15 times slower than BoxMG, when used as a preconditioner. Secondly, we note that while the CPU-time cost of AMG is significantly greater than that of the multilevel structured-grid codes (indeed, it is sometimes greater than that of simple IC-CG), the iteration counts for AMG-CG are quite good (typically comparable to those of BoxMG). This demonstrates the scalability and robustness seen with AMG, while highlighting the advantages of using a structured-grid algorithm when possible.

4.3. *Singularity of Coefficient Matrix*

As mentioned in Section 2, the coefficient matrix, A, is singular. A non-unique solution, x, always exists, because the linear system (6) is consistent due to (7). However, extra care should be taken in the implementation of the methods we compare. Matrix E is often singular, due to the singularity of A and the construction of Z. In this case, E^{-1} does not exist, and instead the pseudo-inverse, E^{+} should be used in the operator P. The iteration process using E^{+} does not cause any difficulties in DIC-CG, since the corresponding systems are consistent, see [54] for more details. The multigrid iterations are similarly insensitive to the singularity on all levels but the coarsest, as only iterative approaches are used in reducing errors at all other levels. On the coarsest level, the known form of the null space of E allows a simple perturbation and projection technique to be used in the direct solve of this system; see [15] for details.

4.4. *Parallelization*

While the tests performed in Section 5 are all done in a serial computing environment where, because of the efficiency seen in the best approaches, problems of up to 8 million degrees of freedom are easily handled, it is important to stress that this was done for convenience alone and not because of any inherent serial nature of the algorithms considered. Indeed, much effort has been invested in the parallelization of exactly the algorithms considered here. Parallelization of deflation solvers has been considered in [21], where it is shown that, with a sensible alignment of the subdomains and processor boundaries, deflation applied to block-incomplete Cholesky preconditioners can be implemented with limited increase in cost over that of the parallel matrix-vector multiplies already required by block-incomplete Cholesky preconditioned CG.

Parallelization of standard multigrid methods has been considered for many problems and many architectures; see, for example, [19, 27]. Similarly, parallel implementations exist for BoxMG [4] and AMG [24]. Because of the use of pointwise iterations in the relaxation step, virtually no parallel communication is necessary when block-Jacobi relaxation is used in place of pointwise Gauss-Seidel. In AMG, parallel communication and inherently serial algorithms is a well-studied issue, particularly with respect to the choice of coarse grid [2].

4.5. *Implementation*

While we stress that there is nothing "out of the ordinary" in the multigrid implementations considered here, it must be acknowledged that there have been many less-successful attempts to apply multigrid to these problems. Therein lies the attractiveness of the DIC-CG method; given an existing code, with existing data structures and single-level preconditioners, it is relatively simple to add an effective deflation step to the existing pressure solver. In contrast, use of the best-available multigrid codes (as considered here) requires some translation of the discrete problem into data structures that are more natural for multigrid treatment. This trade-off is at the root of the questions investigated here. While a greater investment of programming effort may be necessary to implement a robust, efficient multigrid solver, such as BoxMG, this effort appears to pay off in the reduced computing times seen in the following results.

5. Numerical Experiments

In this section, we present some numerical experiments with 3-D bubbly flow problems as described in Section 2. First, stationary problems are considered, where the time step, l, is fixed and all of the presented methods will be compared. Then, real-life simulations will be carried out, where the performance of the best methods from the stationary problems will be further investigated. The computations are performed on an Intel Core 2 Duo (2.66 GHz) computer with a memory capacity of 8GB. The code is compiled with the Intel FORTRAN compiler, ifort, on LINUX. We note that several of the comparisons made here are extensions of those made in [54], where the optimal choice of parameters for DIC-CG was considered. Here, our focus is on the comparison of the solver developed in [54] with the multigrid approaches discussed in Section 3.3.

5.1. *Stationary Problems*

Stationary 3-D bubbly flows are considered, where a 1 cm^3 cube is fully filled with one fluid and m spherical bubbles of another fluid can appear. The bubbles have the same radius, s, and are located in a 'structured' way in the fluid. The geometry of some test cases can be found in Figure 1. The aim is to solve the resulting linear system (6), where n grid points are chosen in each spatial direction, so that the total number of degrees of freedom is $N = n^3$. We will examine all of the methods described in Section 3 and, for DIC-CG, will typically take $k = \frac{n}{8}$; thus $K = 4^3, 8^3, 16^3$ for $N = 32^3, 64^3, 128^3$.

The experiments are divided into several parts, examining the scalability of these solvers relative to various parameters of the problems: N (total number of degrees of freedom), m (number of bubbles), s (radius of each bubble) and ϵ (contrast between the phases). The results of the experiments will be given in terms of the required computing time for the pressure-correction system (CPU), including both the set-up and solution time, number of iterations or cycles (IT), and the obtained accuracy (RES), measured as the relative norm of the residual, $||b - Ax^{(L)}||_2/||b - Ax^{(0)}||_2$, where $x^{(L)}$ denotes the L-th iterate. While the solution of the pressure-correction system is not the only contribution to the total solution time, it is typically the dominant cost and the main contributor to a lack of scalability in the overall solution time. For this reason, we focus here only on the time required to solve this part of the problem.

For all methods, we choose the starting vector to be the zero vector (i.e., $x^{(0)} = 0$) and the stopping criterion to be

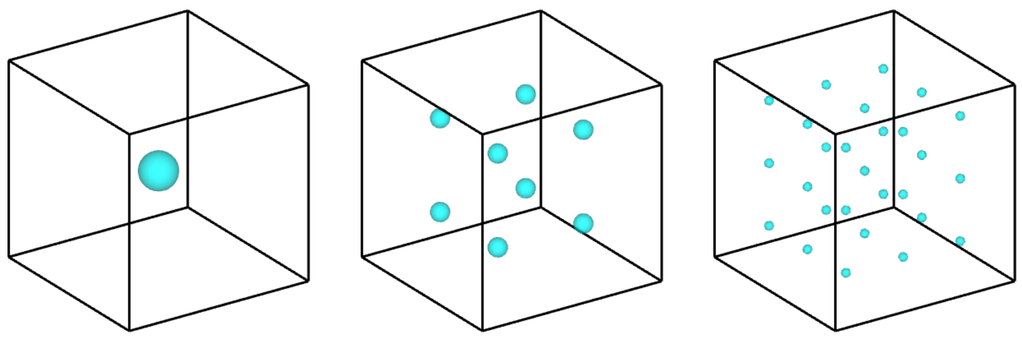

(a) $m = 1$ and $s = 0.1$ cm. (b) $m = 8$ and $s = 0.05$ cm. (c) $m = 27$ and $s = 0.025$ cm.

Fig. 1. Geometry of some stationary bubbly flows.

$$\|r^{(L)}\|_2 / \|r^{(0)}\|_2 < \delta = 10^{-8},$$

where $r^{(L)}$ is the residual calculated after L cycles. For the CG-accelerated methods, $r^{(L)}$ is taken to be the CG-generated residual, and not the so-called "true" residual, calculated directly as $b - Ax^{(L)}$. It is important to realize that, due to the accumulation of roundoff errors, the CG-generated residual may "drift" away from the true residual as the iteration proceeds [23, Section 7.3]. For this reason, we report the true residual reduction (RES) measured by calculating the true residual associated with the solution computed by CG. Note that this possible inaccuracy does not affect the stationary MG methods, where the residual is calculated directly at each stage of the cycle.

5.1.1. *Varying Number of Grid Points*

Table 1 presents results with varying grid points. A larger number of grid points leads to coefficient matrices that are more ill-conditioned, as mentioned in Section 3. It can be observed in Table 1 that only BoxMG, BoxMG-CG and DIC-CG show perfectly scalable iteration counts with respect to the number of grid points, although the computing times grow relatively quickly. Recall that, for DIC-CG, more deflation vectors are taken for larger N, which results in a decreasing number of iterations for DIC-CG. Moreover, observe that BoxMG and BoxMG-CG outperform the other methods both in terms of the number of iterations required and the computing time. For larger N, the number of iterations for CCMG-CG and AMG-CG grows very slowly; however, the large cost per iteration for these methods still makes them uncompetitive. As mentioned earlier, a lower bound for the cost of CCMG-CG can be given; in the case of $N = 128^3$, CCMG-CG would require at least 11.1 sec (instead of 38.0 sec for our current implementation) and, therefore, may be competitive with DIC-CG.

In this experiment, the scalability of the methods can be easily observed. Considering the computing time, it can be seen that the required CPU time for both DIC-CG and BoxMG increase by a factor of approximately 8 when doubling the number of grid points, n, in each direction. Slightly larger increases are seen for the other multigrid methods, with increases by factors of 9 or 10 seen for GMG-CG, CCMG-CG, and AMG-CG, as well as for CCMG and AMG without acceleration by CG. Iteration counts for all methods, except IC-CG, are nearly constant as problem size increases; IC-CG requires significantly more iterations than the other techniques and clearly does not scale effectively as the problem size increases. While AMG and AMG-CG are competitive in terms of the number of iterations required for convergence, they are clearly much more expensive; this is due to the extra costs associated with the unstructured-grid data structures used within AMG, and the extra setup required based on this assumption, as discussed in Section 4.2.

We also observe in Table 1 that the accuracy of DIC-CG is consistently the worst when compared with the other methods, although the differences are generally quite small. As mentioned in Section 4.1, this is caused by the fact that the deflated residuals are used in DIC-CG, leading to extra roundoff errors. The true final residuals for the other methods are quite comparable, although the relatively fast convergence of BoxMG-CG and AMG-CG shows in the consistently better residual reduction achieved in only a few iterations.

Test Problem	$N = 32^3$			$N = 64^3$			$N = 128^3$		
Method	CPU	IT	RES	CPU	IT	RES	CPU	IT	RES
IC-CG	0.3	112	0.9E-8	5.8	244	0.1E-8	92.3	444	0.9E-8
DIC-CG	0.2	64	0.8E-7	1.7	54	0.4E-7	11.7	39	0.3E-7
GMG-CG	0.2	81	0.9E-8	3.9	132	0.9E-8	36.0	134	0.8E-8
CCMG-CG	0.4	14	0.3E-8	4.3	18	0.1E-7	38.0	19	0.9E-8
BoxMG-CG	0.1	12	0.2E-8	0.8	12	0.2E-8	7.0	12	0.3E-8
AMG-CG	0.8	10	0.4E-8	8.9	14	0.1E-8	89.0	15	0.2E-8
CCMG	0.8	44	0.1E-7	11.2	79	0.1E-7	99.1	85	0.8E-8
BoxMG	0.1	16	0.8E-8	0.8	17	0.3E-8	7.4	17	0.4E-8
AMG	0.9	20	0.1E-7	13.0	40	0.1E-7	129.2	45	0.8E-8

Table 1
Convergence results for the experiment with $m = 2^3$, $s = 0.05$ cm, $\epsilon = 10^{-3}$ and varying total number of degrees of freedom, N.

5.1.2. *Varying Contrasts*

The results for the test problems with varying contrast, ϵ, are given in Table 2.

A smaller ϵ typically corresponds to a linear system whose coefficient matrix is more ill-conditioned. Therefore, we expect most methods to need more iterations and computing time as ϵ decreases, as shown in Table 2. The performance of BoxMG and BoxMG-CG in this test, however, is independent of the contrast, giving perfect scaling in terms of both iteration counts and CPU time. Both AMG-CG and CCMG-CG show moderate increases in the iteration counts as ϵ decreases, while these methods alone (without CG) show much more significant growth (by factors of roughly 2 and 4, respectively). This shows that the CG acceleration adds significant robustness to the performance of these methods as the contrast between the phases grows. DIC-CG also performs well, with only a slight increase in the number of iterations required as ϵ decreases; scaling studies [55] show that, for large enough values of K, we can expect perfect scaling of DIC-CG-K as ϵ decreases. The iteration counts for GMG-CG increase, as expected, as ϵ decreases. In fact, for $\epsilon = 10^{-1}$, GMG-CG is competitive with BoxMG, BoxMG-CG, and DIC-CG, while for $\epsilon = 10^{-5}$, the iteration count for GMG-CG is comparable with that of IC-CG.

As mentioned in Section 4.2, CCMG-CG is not implemented as efficiently as possible. Nevertheless, we can get an idea of the cost of its optimal implementation by comparing with the performance of BoxMG. CCMG-CG is at least as expensive per iteration as BoxMG, so, in the case of $\epsilon = 10^{-3}$, CCMG-CG would require at least 1.2 sec instead of 4.3 sec and may, therefore, be slightly faster than DIC-CG. However, for $\epsilon = 10^{-5}$, CCMG-CG would need at least 2.5 sec, so DIC-CG appears to be faster than CCMG-CG for larger contrasts.

5.1.3. *Varying Radii*

The results for an experiment with varying the radii of the bubbles, s, are given in Table 3. The smallest radius is chosen to be $s = 0.01875$ cm, because the bubbles are no longer resolved for $s < 1/64 = 0.015635$ cm. In general, a smaller radius does not significantly affect the conditioning of the coefficient matrix, but it does change the form of the errors that are difficult to resolve, possibly making them more difficult to approximate.

In Table 3, it can be seen that there are changes in convergence behavior of the various methods for different radii. In general, a smaller s leads to a more favorable performance for several of the iterative methods, including IC-CG, GMG-CG and CCMG-CG. The other methods do not have a clear relation with respect to s. This even seems to hold for the stationary methods, CCMG, BoxMG and AMG, which we might expect to be sensitive to the size of the bubbles due to the challenges discussed in Section 3.3.5. BoxMG and BoxMG-CG seem to be fully insensitive to s. They are also the fastest methods in this experiment, followed by again DIC-CG. It is interesting to note that while GMG-CG is very ineffective in the case of

Test Problem	$\epsilon = 10^{-1}$			$\epsilon = 10^{-3}$			$\epsilon = 10^{-5}$		
Method	CPU	IT	RES	CPU	IT	RES	CPU	IT	RES
IC-CG	3.1	131	0.1E-7	5.8	244	0.1E-8	6.9	289	0.5E-8
DIC-CG	1.1	35	0.3E-7	1.7	54	0.4E-7	1.9	59	0.4E-7
GMG-CG	1.0	33	0.1E-7	3.9	132	0.9E-8	8.0	267	0.8E-8
CCMG-CG	3.3	10	0.1E-8	4.3	18	0.1E-7	6.7	37	0.4E-8
BoxMG-CG	0.8	12	0.1E-8	0.8	12	0.2E-8	0.8	12	0.2E-8
AMG-CG	8.0	9	0.3E-8	8.9	14	0.1E-8	9.2	16	0.2E-8
CCMG	4.0	17	0.6E-8	11.2	79	0.1E-7	40.9	338	0.1E-7
BoxMG	0.8	17	0.3E-8	0.8	17	0.3E-8	0.8	17	0.3E-8
AMG	8.9	15	0.8E-8	13.0	40	0.1E-7	21.5	92	0.9E-8

Table 2
Convergence results for the experiment with $N = 64^3$, $m = 2^3$, $s = 0.05$ cm and varying the contrast, ϵ.

large bubbles, its performance improves as the bubbles shrink and, for the case of $s = 0.01875$ cm, it becomes competitive with BoxMG and DIC-CG.

Test Problem	$s = 0.1$ cm			$s = 0.05$ cm			$s = 0.025$ cm			$s = 0.01875$ cm		
Method	CPU	IT	RES	CPU	IT	RES	CPU	IT	RES	CPU	IT	RES
IC-CG	6.0	250	0.1E-7	5.8	244	0.1E-8	3.8	159	0.8E-8	4.1	170	0.9E-8
DIC-CG	1.3	39	0.3E-7	1.7	54	0.4E-7	1.5	46	0.9E-7	1.5	45	0.8E-7
GMG-CG	4.3	143	0.9E-8	3.9	132	0.9E-8	2.6	85	0.8E-8	1.3	41	0.4E-8
CCMG-CG	5.1	24	0.4E-8	4.3	18	0.1E-7	3.6	12	0.5E-8	3.8	14	0.4E-8
BoxMG-CG	0.8	12	0.3E-8	0.8	12	0.2E-8	0.8	12	0.2E-8	0.8	12	0.1E-8
AMG-CG	8.3	11	0.6E-8	8.9	14	0.1E-8	8.2	12	0.7E-8	8.8	14	0.3E-8
CCMG	13.0	95	0.8E-8	11.2	79	0.1E-7	7.0	43	0.7E-8	10.1	69	0.9E-8
BoxMG	0.8	17	0.4E-8	0.8	17	0.3E-8	0.8	17	0.3E-8	0.8	17	0.3E-8
AMG	10.6	25	0.8E-8	13.0	40	0.1E-7	10.9	29	0.6E-8	12.8	39	0.7E-8

Table 3
Convergence results for the experiment with $N = 64^3$, $m = 2^3$, $\epsilon = 10^{-3}$ and varying the radius of the bubbles, s.

5.1.4. *Varying Number of Bubbles*

In Table 4, we present results demonstrating the performance of the various solvers with variation in the number of bubbles, m. Note that the test case with $m = 0$ corresponds to the Poisson equation with a constant coefficient, i.e., a domain with only one phase. It is known that increasing the number of bubbles leads to the appearance of more large eigenvalues in the original coefficient matrix, A, and small eigenvalues in the Incomplete Cholesky preconditioned coefficient matrix, $M^{-1}A$, (see, e.g., [55,65]), resulting in a more difficult linear system to solve, although both the original and preconditioned coefficient matrices are not necessarily worse conditioned.

It can be seen in Table 4 that, for most methods, the convergence worsens with increasing m, as expected. Moreover, GMG-CG is the best method in the case of $m = 0$, but quickly loses efficiency for $m > 0$. The number of iterations required by CCMG also grows rapidly with m, whereas it grows gradually for CCMG-CG, AMG-CG and AMG. A single iteration of these methods, however, is more expensive than one of BoxMG or DIC-CG, as mentioned in Section 4. For $m > 0$, BoxMG and BoxMG-CG are always the fastest methods, followed by DIC-CG. The performance of BoxMG-CG degrades only a little with increasing m, while the iteration counts for BoxMG increase more substantially. For a sufficiently large number of deflation vectors, K, DIC-CG would be less sensitive to changes in m.

Test Problem	$m = 0$			$m = 1$			$m = 2^3$			$m = 3^3$		
Method	CPU	IT	RES	CPU	IT	RES	CPU	IT	RES	CPU	IT	RES
IC-CG	3.0	125	0.8E-8	3.9	163	0.7E-8	5.8	244	0.1E-8	8.2	342	0.8E-8
DIC-CG	1.0	31	0.2E-7	1.5	47	0.2E-7	1.7	54	0.4E-7	2.0	60	0.7E-7
GMG-CG	0.3	8	0.1E-8	3.7	124	0.8E-8	3.9	132	0.9E-8	9.8	329	0.8E-8
CCMG-CG	3.2	9	0.7E-8	3.7	13	0.7E-8	4.3	18	0.1E-7	6.5	35	0.4E-8
BoxMG-CG	0.8	12	0.1E-8	0.8	12	0.1E-8	0.8	12	0.2E-8	1.0	15	0.2E-8
AMG-CG	7.7	7	0.9E-8	8.0	9	0.2E-8	8.9	14	0.1E-8	8.9	14	0.8E-8
CCMG	4.0	17	0.4E-8	10.0	68	0.9E-8	11.2	79	0.1E-7	27.9	223	0.1E-7
BoxMG	0.8	17	0.3E-8	0.8	17	0.3E-8	0.8	17	0.3E-8	1.3	29	0.7E-8
AMG	8.2	11	0.6E-8	11.8	33	0.6E-8	13.0	40	0.1E-7	14.1	45	0.7E-8

Table 4
Convergence results for the experiment with $N = 64^3$, $s = 0.05$ cm, $\epsilon = 10^{-3}$ and varying the number of bubbles, m.

5.1.5. *Difficult Test Problem*

We end the stationary experiments with a test case where the most difficult parameters (taken based on the previous experiments) are chosen. The results associated with this experiment can be found in Table 5, using stopping tolerance $\delta = 10^{-8}$. We note that a higher accuracy than $\delta = 10^{-8}$ can not be reached, due to the accumulation of roundoff errors and the effects of finite precision arithmetic.

As in the other experiments, BoxMG-CG and BoxMG perform the best, in terms of both the number of iterations and the computing time. They are again followed by DIC-CG, which also performs rather well. GMG-CG, CCMG, and AMG all typically fail to converge within the allowed number of iterations for $\delta = 10^{-8}$ (and also for the weaker tolerance, $\delta = 10^{-6}$. CCMG-CG and AMG-CG do converge but, as always, are not competitive in terms of true CPU time.

Method	CPU	IT	RES
IC-CG	195.4	942	0.1E-7
DIC-CG	19.6	65	0.5E-7
GMG-CG	–	> 1000	–
CCMG-CG	114.0	92	0.1E-7
BoxMG-CG	7.4	13	0.4E-8
AMG-CG	107.6	29	0.8E-8
CCMG	–	> 1000	–
BoxMG	7.8	18	0.7E-8
AMG	–	> 1000	–

Table 5
Convergence results for the difficult test problem. The following parameters are kept constant: $N = 128^3$, $m = 3^3$ $s = 0.025$ cm, $\epsilon = 10^{-5}$, and $\delta = 10^{-8}$.

While neither AMG nor CCMG converge within the allowed number of iterations as standalone solvers, both perform reasonably well as preconditioners. In fact, both unaccelerated solvers do converge, but very slowly, with AMG converging marginally faster than CCMG. While it may be possible to improve this performance somewhat by using different relaxation schemes, or by changing some of the parameters in the AMG setup stage, this is beyond the scope of the current study. Clearly BoxMG (and BoxMG-CG) and DIC-CG are both efficient techniques for solving these problems, and so we restrict the remainder of our study to comparing these two methods with one another, and with a single-level preconditioner, IC-CG.

5.2. *Time-Dependent Problems*

In this section, we consider simulations of three “real-life” bubbly flows. In order to obtain sophisticated geometries, these flows are considered without surface tension. The pressure-correction method is adopted to solve the Navier-Stokes equations, as described in Section 2. The interface advection is carried out using the mass-conserving level set method [61, 62]. Our main interest in each time step is the pressure solve, which takes the bulk of the computing time in each simulation, especially when the total degrees of freedom, N, is relatively large. The actual time step, Δt, is restricted by

$$\Delta t \leq \beta := \frac{h\alpha}{2\left(|u|_{\max} + |v|_{\max} + |w|_{\max}\right)},$$

where h is the distance between grid points in one direction and $\alpha = 0.35$ is the CFL number, see [62] for more details. That means that we use an adaptive time stepping procedure by considering the time-step restrictions due to convection of the bubbly flow. At each time step, l, the actual time, t, is increased with an increment, Δt, that obeys

$$\Delta t = \min\left(\beta, \Delta t_{\max}\right),$$

where we choose $\Delta t_{\max} = 0.0005$ sec.

At each time step, the resulting linear system (6), with $N = n^3$, is solved using both BoxMG-CG and DIC-CG, as these are shown to be the most stable and efficient methods for the stationary problems considered in Section 5.1. DIC-CG with K deflation vectors will be denoted as DIC-CG$-K$, where K is chosen to minimize the required CPU time. IC-CG is used as a benchmark in the experiments. The initial guess for each solve is chosen to be the previous solution, except for the first 10 time steps, where the zero starting vector is used (to avoid problems with achieving a too-strict relative residual reduction when the flow is initialized). The stopping criterion of all methods is chosen as in the stationary experiments. For more details on the physical problems simulated here, see [61, 62].

5.2.1. *Rising Air Bubble in Water*

We first consider a test problem where a cube of 1 cm^3 is filled with water to a height of 0.6 cm. For these experiments, we take the density of water to be 820 times that of air (i.e., $\epsilon = 1.22 \times 10^{-3}$). At the initial time step, $l = 0$, a spherical air bubble with radius of 0.125 cm is located in the middle of the domain at a height of 0.35 cm. The exact material constants and other relevant conditions for this simulation can be found in [62, Sect. 7.2]. The evolution of the geometry during 500 time steps is given in Figure 2. Here, we take a grid with $N = 100^3$; in this case, the optimal value for K in DIC-CG$-K$ appears to be $K = 20^3$. Results of the experiment can be found in Figure 3, showing both the number of iterations and computing time required for each method for the pressure solves at each time step, l.

It can be readily observed from Figure 3 that, for each time step, both DIC-CG and BoxMG-CG converge in fewer iterations and require less computing time than to IC-CG. Due to the zero starting vector in the first 10 iterations, one can observe a peak in the IC-CG cost around these first iterations, whereas this phenomenon cannot be clearly seen for DIC-CG and BoxMG-CG. Moreover, BoxMG-CG shows better performance than DIC-CG. We remark that both methods behave smoothly in time and seem to be more-or-less independent of the (sometimes complicated) geometry of the density field. This is in contrast to IC-CG, whose convergence is rather erratic. Only some small outliers can be observed in the convergence of DIC-CG and BoxMG. For example, a small peak can be seen around $l = 325$ in DIC-CG and around $l = 390$ and $l = 410$ in BoxMG-CG. This is likely related to particular changes in the density field at these time steps, but it is difficult to pinpoint the cause, due to the complicated surface dynamics at these time steps. Moreover, it can be seen that, for $l \in [150, 350]$, more iterations are required especially for DIC-CG, because the geometry is most complicated in this period, due to the interaction of the bubble with the interface and the appearance of many droplets, as can be observed in Figure 2.

5.2.2. *Falling Water Droplet in Air*

In the next simulation, we consider a 1 cm^3 cube filled with water to a height of 0.45 cm. At the initial time step, $l = 0$, a spherical water droplet with radius 0.125 cm is located in the middle of the domain at a height of 0.6 cm. The same material constants and conditions are used as in the previous simulation. The evolution of the geometry during the 500 time steps is depicted in Figure 4. Again, the grid resolution is $N = 100^3$ and the optimal number of deflation vectors is $K = 20^3$.

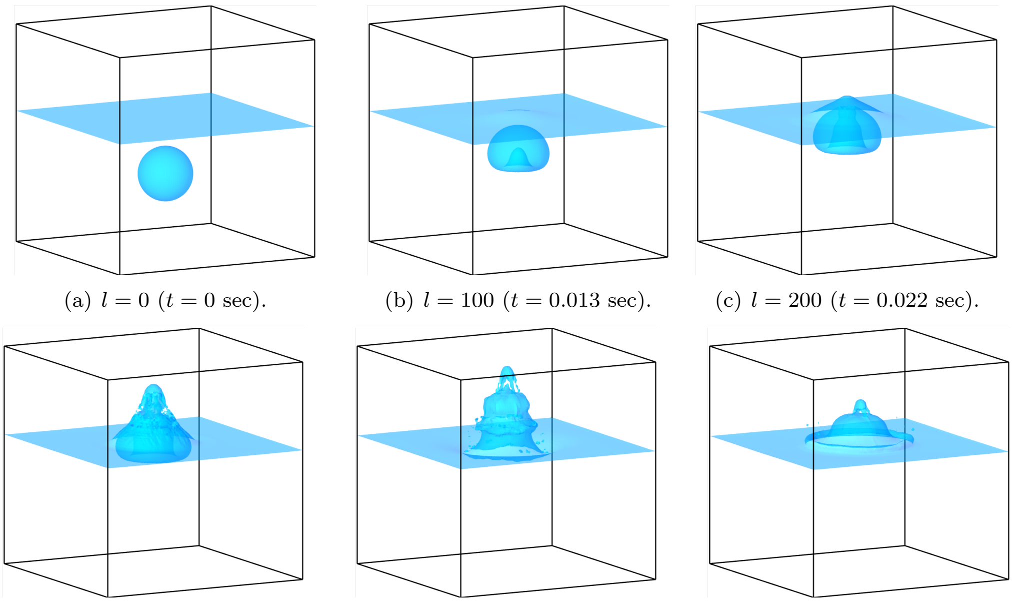
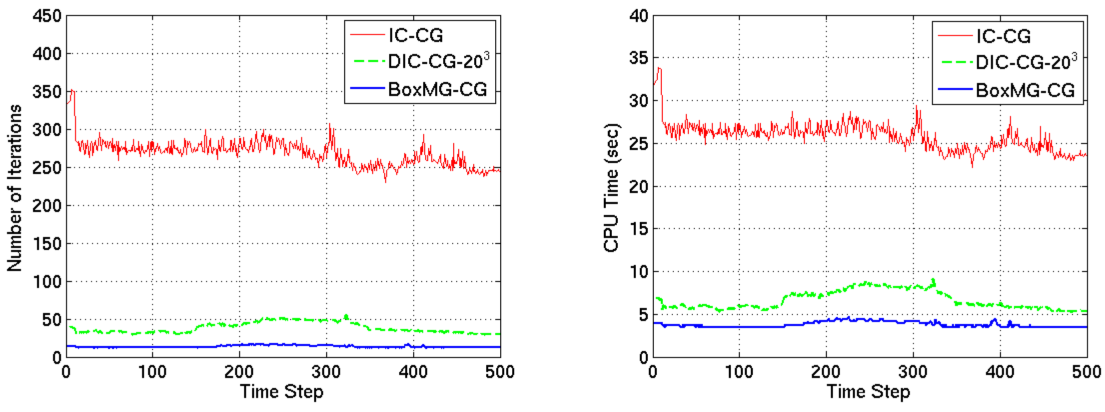

(a) $l = 0$ ($t = 0$ sec). (b) $l = 100$ ($t = 0.013$ sec). (c) $l = 200$ ($t = 0.022$ sec).
(d) $l = 300$ ($t = 0.032$ sec). (e) $l = 400$ ($t = 0.050$ sec). (f) $l = 500$ ($t = 0.064$ sec).

Fig. 2. Evolution of a rising bubble in water.



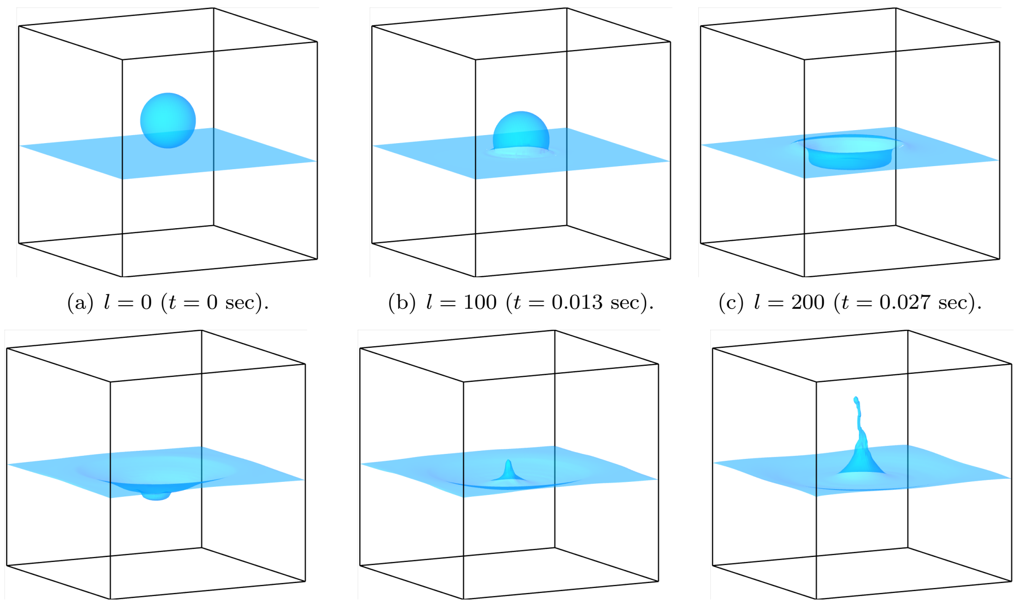
(a) Number of iterations versus time step. (b) CPU time versus time step.

Fig. 3. Results for IC-CG, DIC-CG-20^3 and BoxMG-CG for the pressure solves during the real-life simulation with a rising air bubble in water.

The results of the experiment can be found in Figure 5. It can again be noticed that IC-CG performs worse than both DIC-CG and BoxMG-CG. BoxMG-CG is always faster than DIC-CG, although the differences are small in this experiment; the number of iterations and computing time per time step are approximately the same for both methods. We observe a very smooth behavior of the corresponding performance curves, because very few additional bubbles or droplets appear during the simulation. In this experiment, BoxMG and DIC-CG appear to be more-or-less insensitive to the geometry of the density field, while it can be readily observed that the performance of IC-CG does depend on it.

5.2.3. *Two Rising and Merging Air Bubbles in Water*

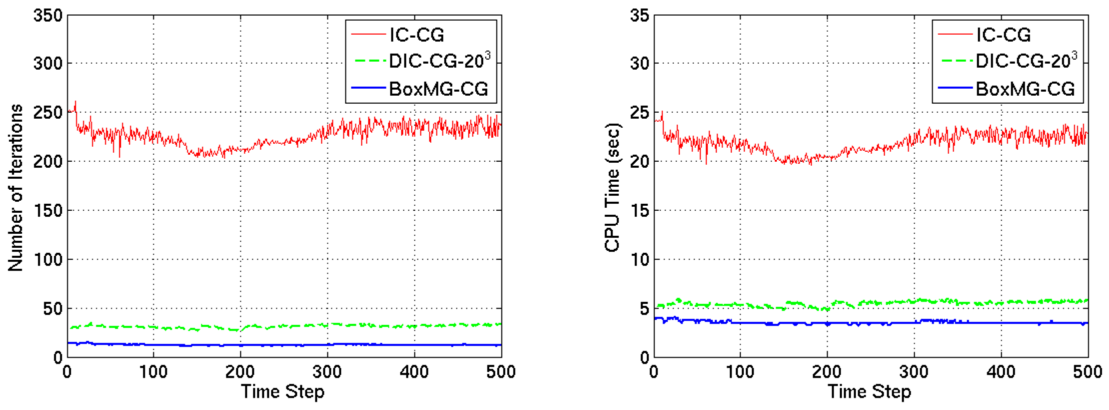
In the final simulation, we consider a test problem where a 1 cm^3 cube is filled with water to a height of 0.65 cm. At the initial time step, $l = 0$, two air bubbles of radius 0.1 cm are located with centers at coordinates $(0.5, 0.5, 0.37)$ and $(0.5, 0.3, 0.15)$. The evolution of the geometry during 2500 time steps can be



(a) $l = 0$ ($t = 0$ sec). (b) $l = 100$ ($t = 0.013$ sec). (c) $l = 200$ ($t = 0.027$ sec).

(d) $l = 300$ ($t = 0.044$ sec). (e) $l = 400$ ($t = 0.050$ sec). (f) $l = 500$ ($t = 0.059$ sec).

Fig. 4. Evolution of a falling droplet in air.



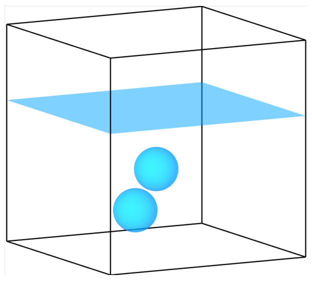
(a) Number of iterations versus time step. (b) CPU time versus time step.

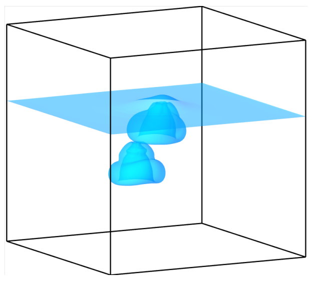
Fig. 5. Results for IC-CG, DIC-CG-20^3 and BoxMG-CG for the pressure solves during the real-life simulation of a falling water droplet in air.

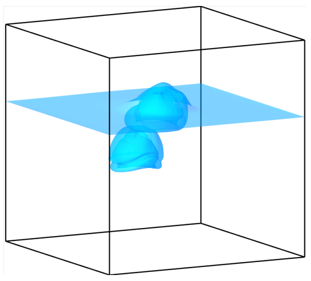
found in Figure 6. This test problem is, obviously, harder to solve than the previous two test problems, since there is interaction between the two bubbles at the same time as they interact with the water interface. In addition, we now consider a refined grid with $N = 200^3$, resulting in a strongly ill-conditioned coefficient matrix and making the problem very complicated to solve. DIC-CG$-K$ with $K = 25^3$ appears to be optimal in terms of the required CPU time.

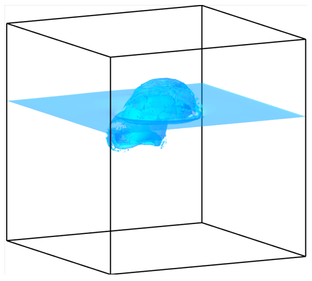
Results are presented in Figure 7. IC-CG is omitted in these results, since it is extremely slow in this difficult test case, requiring, on average, over 700 iterations and 500 seconds of CPU time per time step during the first 100 time steps, which have relatively simple dynamics. It can be observed that the number of iterations, and, therefore, also the computing time, increases gradually during the simulation for both methods, but especially for DIC-CG-25^3. This is due to the fact that the geometry of the problem becomes progressively more sophisticated as the simulation proceeds. Apparently, the influence of the deflation vectors depends heavily on the time step. This even holds if we increase K. Obviously, BoxMG is always faster than

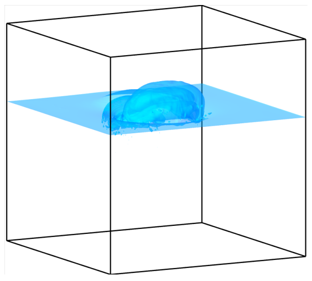
DIC-CG (especially for $l \in [1000, 2500]$), both with respect to the number of iterations and the computing time.


(a) $l = 0$ ($t = 0$ sec).


(b) $l = 500$ ($t = 0.025$ sec).


(c) $l = 1000$ ($t = 0.035$ sec).


(d) $l = 1500$ ($t = 0.045$ sec).


(e) $l = 2000$ ($t = 0.056$ sec).

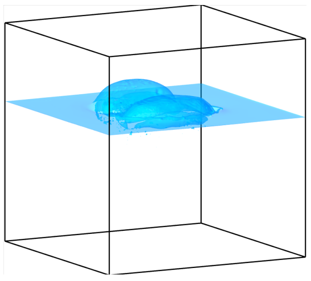
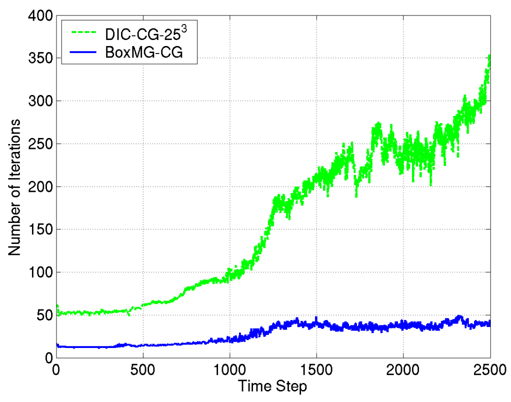
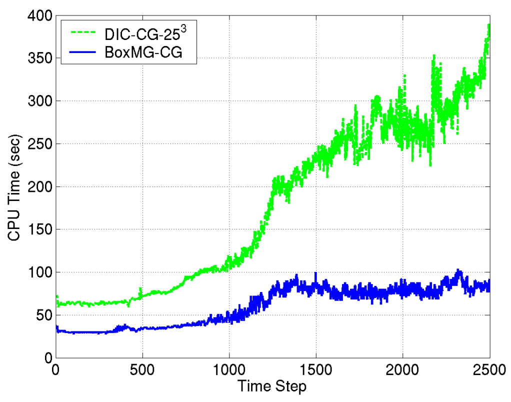

(f) $l = 2500$ ($t = 0.066$ sec).

Fig. 6. Evolution of two rising air bubbles in water.



(a) Number of iterations versus time step.



(b) CPU time versus time step.

Fig. 7. Results for DIC-CG-25^3 and BoxMG-CG for the pressure solve during the real-life simulation with two rising air bubbles in water. IC-CG is omitted in these results.

6. Conclusions

We present a comparison of several multilevel techniques that may be considered as efficient solvers for the pressure-correction equations in two-phase bubbly flows. In particular, two families of algorithms are considered; the DIC-CG algorithm, based on the principles of deflation for classical preconditioned CG techniques, and multigrid algorithms. For the multigrid algorithms, we consider a range of approaches, including standard geometric, robust geometric and algebraic multigrid variants. The solvers are compared on a series of stationary problems in three dimensions, where it is shown that BoxMG-CG and DIC-CG are the most stable and efficient techniques. Time-dependent simulations are also given, showing efficient and scalable solution of the pressure-correction equations using these techniques for three-dimensional problems with up to eight million degrees of freedom.

From the time-dependent simulations, we conclude that BoxMG-CG performs better than DIC-CG, especially for relatively large grid sizes. BoxMG-CG appears to be more scalable and requires fewer iterations and less computing time in all experiments, for all time steps. In addition, it seems to have a low sensitivity to the density fields, gives accurate solutions and is very robust in all cases. Improvement of DIC-CG to give performance comparable to BoxMG-CG is a subject for future research. The relatively large coarse grids required by DIC-CG to achieve good convergence properties suggest that there is a need for a better solver for the coarse linear systems in DIC-CG in order to make the method more efficient and scalable. Overall, we have demonstrated that solution of the pressure-correction equations within bubbly flow applications can be significantly accelerated using the methods studied here.

Acknowledgments

The authors would like to thank Bruce Boghosian, Kees Oosterlee, and Pieter Wesseling for many helpful discussions during the preparation of this work. They would also like to thank Markus Berndt, Joel Dendy, and David Moulton for making the BoxMG code available (`https://software.lanl.gov/boxmg/`).

References

[1] J. C. Adams, MUDPACK: Multigrid portable FORTRAN software for the efficient solution of linear elliptic partial differential equations, Applied Mathematics and Computation 34 (2) (1989) 113–146.

[2] D. Alber, L. Olson, Parallel coarse-grid selection, Numer. Linear Algebra Appl. 14 (8) (2007) 611–643.

[3] R. E. Alcouffe, A. Brandt, J. E. Dendy, J. W. Painter, The multigrid method for the diffusion equation with strongly discontinuous coefficients, SIAM J. Sci. Stat. Comput. 2 (1981) 430–454.

[4] T. Austin, M. Berndt, B. K. Bergen, J. E. Dendy, J. D. Moulton, Parallel, scalable, and robust multigrid on structured grids, T-7 Research Highlight LA-UR-03-9167, Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM, USA (2003).

[5] A. Behie, P. A. Forsyth, Multi–grid solution of three–dimensional problems with discontinuous coefficients, Appl. Math. Comput. 13 (1983) 229–240.

[6] J. B. Bell, P. Colella, H. M. Glaz, A second-order projection method for the incompressible Navier-Stokes equations, J. Comput. Phys. 85 (2) (1989) 257–283.

[7] D. Braess, On the combination of the multigrid method and conjugate gradients, in: Multigrid methods, II (Cologne, 1985), vol. 1228 of Lecture Notes in Math., Springer, Berlin, 1986, pp. 52–64.

[8] A. Brandt, S. F. McCormick, J. W. Ruge, Algebraic multigrid (AMG) for sparse matrix equations, in: D. J. Evans (ed.), Sparsity and Its Applications, Cambridge University Press, Cambridge, 1984, pp. 257–284.

[9] B. Bunner, G. Tryggvason, Dynamics of homogeneous bubbly flows. part 1. rise velocity and microstructure of the bubbles, J. Fluid Mech. 466 (2002) 17–52.

[10] A. J. Chorin, Numerical solution of the Navier-Stokes equations, Math. Comp. 22 (1968) 745–762.

[11] A. J. Chorin, On the convergence of discrete approximations to the Navier-Stokes equations, Math. Comp. 23 (1969) 341–353.

[12] F. S. de Sousa, N. Mangiavacchi, L. G. Nonato, A. Castelo, M. F. Tomé;, V. G. Ferreira, J. A. Cuminato, S. McKee, A front-tracking/front-capturing method for the simulation of 3d multi-fluid flows with free surfaces, J. Comput. Phys. 198 (2) (2004) 469–499.

[13] J. E. Dendy, Black box multigrid, J. Comput. Phys. 48 (3) (1982) 366–386.

[14] J. E. Dendy, Two multigrid methods for three-dimensional equations with highly discontinuous coefficients, SIAM J. Sci. Stat. Comput. 8 (1987) 673–685.

[15] J. E. Dendy, Black box multigrid for periodic and singular problems, Appl. Math. Comput. 25 (1, part I) (1988) 1–10.

[16] Y. Erlangga, R. Nabben, Multilevel projection-based nested Krylov iteration for boundary value problems, SIAM J. Sci. Comput.To appear.

[17] A. Esmaeeli, G. Tryggvason, Direct numerical simulations on bubbly flows. Part 1. Low Reynolds number arrays, J. Fluid Mech. 377 (1998) 313–345.

[18] A. Esmaeeli, G. Tryggvason, Direct numerical simulations on bubbly flows. Part 2. Moderate Reynolds number arrays, J. Fluid Mech. 385 (1999) 325–358.

[19] R. D. Falgout, J. E. Jones, Multigrid on massively parallel architectures, in: Multigrid methods, VI (Gent, 1999), vol. 14 of Lect. Notes Comput. Sci. Eng., Springer, Berlin, 2000, pp. 101–107.

[20] M. Fortin, Old and new finite elements for incompressible flows, Internat. J. Numer. Methods Fluids 1 (4) (1981) 347–364.

[21] J. Frank, C. Vuik, On the construction of deflation-based preconditioners, SIAM J. Sci. Comput. 23 (2001) 442–462.

[22] V. Girault, P.-A. Raviart, Finite element approximation of the Navier-Stokes equations, vol. 749 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1979.

[23] A. Greenbaum, Iterative methods for solving linear systems, vol. 17 of Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 1997.

[24] V. E. Henson, U. M. Yang, BoomerAMG: a parallel algebraic multigrid solver and preconditioner, Appl. Numer. Math. 41 (1) (2002) 155–177.

[25] J. Hua, J. Lou, Numerical simulation of bubble rising in viscous liquid, J. Comput. Phys. 222 (2) (2007) 769–795.

[26] T. Inamuro, T. Ogata, S. Tajima, N. Konishi, A lattice boltzmann method for incompressible two-phase flows with large density differences, J. Comp. Phys. 198 (2004) 628–644.

[27] J. E. Jones, S. F. McCormick, Parallel multigrid methods, in: Parallel numerical algorithms (Hampton, VA, 1994), vol. 4 of ICASE/LaRC Interdiscip. Ser. Sci. Eng., Kluwer Acad. Publ., Dordrecht, 1997, pp. 203–224.

[28] E. F. Kaasschieter, Preconditioned conjugate gradients for solving singular systems, J. Comput. Appl. Math. 24 (1-2) (1988) 265–275.

[29] R. Kettler, J. Meijerink, A multigrid method and a combined multigrid-conjugate gradient method for elliptic problems with strongly discontinuous coefficients in general domains, Tech. Rep. 604, Shell Oil Company (1981).

[30] M. Khalil, P. Wesseling, Vertex-centered and cell-centered multigrid for interface problems, J. Comput. Phys. 98 (1992) 1–20.

[31] M. Kilmer, E. de Sturler, Recycling subspace information for diffuse optical tomography, SIAM J. Sci. Comput. 27 (6) (2006) 2140–2166.

[32] D. Kwak, C. Kiris, J. Dacles-Mariani, An assessment of artificial compressibility and pressure projection methods for incompressible flow simulations, in: Sixteenth International Conference on Numerical Methods in Fluid Dynamics, vol. 515 of Lecture Notes in Physics, Springer, 1998, pp. 177–182.

[33] L. Mansfield, Damped Jacobi preconditioning and coarse-grid deflation for conjugate gradient iteration on parallel computers, SIAM J. Sci. Stat. Comput. 12 (6) (1991) 1314–1323.

[34] E. Marchandise, P. Geuzaine, N. Chevaugeon, J. Remacle, A stabilized finite element method using a discontinuous level set approach for the computation of bubble dynamics, J. Comput. Phys. 225 (1) (2007) 949–974.

[35] J. A. Meijerink, H. A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Math. Comp. 31 (137) (1977) 148–162.

[36] M. Mohr, R. Wienands, Cell-centred multigrid revisited, Comput. Vis. Sci. 7 (3-4) (2004) 129–140.

[37] J. D. Moulton, J. E. Dendy, J. M. Hyman, The black box multigrid numerical homogenization algorithm, J. Comput. Phys. 141 (1998) 1–29.

[38] R. Nabben, C. Vuik, A comparison of Deflation and Coarse Grid Correction applied to porous media flow, SIAM J. Numer. Anal. 42 (2004) 1631–1647.

[39] R. A. Nicolaides, On the l^2 convergence of an algorithm for solving finite element equations, Math. Comp. 31 (1977) 892–906.

[40] R. A. Nicolaides, Deflation of conjugate gradients with applications to boundary value problems, SIAM J. Numer. Anal. 24 (2) (1987) 355–365.

[41] A. Padiy, O. Axelsson, B. Polman, Generalized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems, SIAM J. Matrix Anal. Appl. 22 (3) (2000) 793–818.

[42] M. Parks, E. de Sturler, G. Mackey, D. Johnson, S. Maiti, Recycling Krylov subspaces for sequences of linear systems, SIAM J. Sci. Comput. 28 (5) (2006) 1651–1674.

[43] E. Puckett, A. Almgren, J. Bell, D. Marcus, W. Rider, A high-order projection method for tracking fluid interfaces in variable density incompressible flows, J. Comp. Phys. 130 (1997) 269–282.

[44] M. Raw, Robustness of coupled algebraic multigrid for the Navier-Stokes equations, Technical Paper 96-0297, AIAA Press, Washington, D.C. (1996).

[45] J. W. Ruge, K. Stüben, Algebraic multigrid (AMG), in: S. F. McCormick (ed.), Multigrid Methods, vol. 3 of Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 1987, pp. 73–130.

[46] Y. Saad, M. Yeung, J. Erhel, F. Guyomarc'h, A deflated version of the conjugate gradient algorithm, SIAM J. Sci. Comput. 21 (5) (2000) 1909–1926.

[47] R. Singh, W. Shyy, Three-dimensional adaptive cartesian grid method with conservative interface restructuring and reconstruction, J. Comput. Phys. 224 (1) (2007) 150–167.

[48] K. Stüben, An introduction to algebraic multigrid, in: U. Trottenberg, C. Oosterlee, A. Schüller (eds.), Multigrid, Academic Press, San Diego, CA, 2001, pp. 413–528.

[49] M. Sussman, E. Puckett, A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows, J. Comput. Phys. 162 (2) (2000) 301–337.

[50] M. Sussman, K. M. Smith, M. Y. Hussaini, M. Ohta, R. Zhi-Wei, A sharp interface method for incompressible two-phase flows, J. Comput. Phys. 221 (2) (2007) 469–505.

[51] R. A. Sweet, Vectorization and parallelization of FISHPAK, in: Parallel processing for scientific computing (Houston, TX, 1991), SIAM, Philadelphia, PA, 1992, pp. 637–642.

[52] J. M. Tang, S. P. MacLachlan, R. Nabben, C. Vuik, Theoretical comparison of two-level preconditioners based on multigrid and deflation, DIAM Report 08-05, Delft University of Technology, Delft (2008).
[53] J. M. Tang, R. Nabben, C. Vuik, Y. A. Erlangga, Theoretical and numerical comparison of various projection methods derived from deflation, domain decomposition and multigrid methods, DIAM Report 07-04, Delft University of Technology, Delft (2007).
[54] J. M. Tang, C. Vuik, Efficient deflation methods applied to 3-D bubbly flow problems, Elec. Trans. on Num. Anal. 26 (2007) 330–349.
[55] J. M. Tang, C. Vuik, New variants of deflation techniques for bubbly flow problems, J. Numer. Anal. Indust. Appl. Math. 2 (3–4) (2007) 227–249.
[56] J. M. Tang, C. Vuik, On deflation and symmetric positive semi-definite matrices, J. Comput. Appl. Math. 206 (2) (2007) 603–614.
[57] O. Tatebe, The multigrid preconditioned conjugate gradient method, in: N. D. Melson, T. A. Manteuffel, S. F. McCormick (eds.), Sixth Copper Mountain Conference on Multigrid Methods, vol. CP 3224, NASA, Hampton, VA, 1993.
[58] U. Trottenberg, C. W. Oosterlee, A. Schüller, Multigrid, Academic Press, London, 2001.
[59] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, Y.-J. Jan, A front-tracking method for the computations of multiphase flow, Journal of Computational Physics 169 (2001) 708–759.
[60] G. Tryggvason, A. Esmaeeli, J. Lu, S. Biswas, Direct numerical simulations of gas/liquid multiphase flows, J. Comput. Phys. 38 (9) (2006) 660–681.
[61] S. P. van der Pijl, A. Segal, C. Vuik, P. Wesseling, A mass-conserving Level-Set method for modelling of multi-phase flows, Int. J. Numer. Methods Fluids 47 (2005) 339–361.
[62] S. P. van der Pijl, A. Segal, C. Vuik, P. Wesseling, Computing three-dimensional two-phase flows with a mass-conserving level set method, Comput. Vis. Sci. (2008) to appear.
[63] J. van Kan, A second-order accurate pressure-correction scheme for viscous incompressible flow, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 870–891.
[64] J. van Kan, C. Vuik, P. Wesseling, Fast pressure calculation for 2D and 3D time dependent incompressible flow, Num. Linear Algebra Appl. 7 (2000) 429–447.
[65] C. Vuik, A. Segal, J. Meijerink, G. Wijma, The construction of projection vectors for a Deflated ICCG method applied to problems with extreme contrasts in the coefficients, J. Comp. Phys. 172 (2001) 426–450.
[66] C. Vuik, A. Segal, J. A. Meijerink, An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients, J. Comp. Phys. 152 (1999) 385–403.
[67] X.-H. Wen, J. J. Gómez-Hernández, Upscaling hydraulic conductivities in heterogeneous media: An overview, Journal of Hydrology 183 (1996) ix–xxxii.
[68] P. Wesseling, Cell-centered multigrid for interface problems, J. Comput. Phys. 79 (1988) 85–91.
[69] P. Wesseling, An introduction to multigrid methods, Pure and Applied Mathematics (New York), John Wiley & Sons Ltd., Chichester, 1992.