Computer Physics Communications 244 (2019) 35–39

Contents lists available at ScienceDirect

Computer Physics Communications

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

Composite matrix construction for structured grid adaptive mesh refinement

Mark F. Adams [a,*], Stephen L. Cornford [b], Daniel F. Martin [c], Peter McCorquodale [c]

[a] *Scalable Solvers Group, Lawrence Berkeley National Laboratory, Berkeley, CA, United States of America*
[b] *Department of Geography, College of Science, Swansea University, Swansea, UK*
[c] *Applied Numerical Algorithms Group, Lawrence Berkeley National Laboratory, Berkeley, CA, United States of America*

ARTICLE INFO

Article history:
Received 25 October 2018
Received in revised form 21 June 2019
Accepted 12 July 2019
Available online 23 July 2019

Keywords:
Algebraic multigrid
Preconditioning
PETSc
Adaptive mesh refinement

ABSTRACT

Structured-grid adaptive mesh refinement (SAMR) is an approach to mesh generation that supports structured access to data and adaptive mesh refinement for discretized partial differential equations (PDEs). Solution algorithms often require that an inverse of an operator be applied, a system of algebraic equations must be solved, and this process is often the primary computational cost in an application. SAMR is well suited to geometric multigrid solvers, which can be effective, but often do not adapt well to complex geometry including material coefficients. Algebraic multigrid (AMG) is more robust in the face of complex geometry, in both boundary conditions and internal material interfaces. AMG requires a stored matrix linearization of the operator. We discuss an approach, and an implementation in the Chombo block-structured AMR framework, for constructing composite grid matrices from a SAMR hierarchy of grids for use in linear solvers in the PETSc numerical library. We consider a case study with the Chombo-based BISICLES ice sheet modeling application.

1. Introduction

Geometric multigrid (GMG) is popular in structured-grid adaptive mesh refinement (SAMR) applications because coarse grid generation is often natural and GMG can be very efficient. Some problems, however, such as porous media flow with complex embedded boundary boundary conditions [1], and ice sheet modeling with a dynamic and strongly-varying material coefficient structure [2], can be challenging to solve with GMG. The more flexible and robust algebraic multigrid (AMG) can be useful in those cases [3]. Additionally, after SAMR blocks have been coarsened, GMG is often no longer natural to apply and, again, AMG can be useful. AMG, for all intents and purposes, requires an explicit stored matrix and a *composite grid* where cells without any degrees of freedom (i.e., ghost cells and cells that are fully covered by refinement) are eliminated and their stencil values are resolved appropriately. The resulting matrix, with only active degrees of freedom, is a *composite grid matrix*. This paper describes a matrix construction methodology and implementation for composite grid construction from a SAMR hierarchy of grids. We use the PETSc library for the linear algebra and solvers [4], and implement this method in the Chombo SAMR framework [5]. We demonstrate an implementation of this method in the ice sheet modeling application BISICLES [2].

At a high level, constructing composite matrices from SAMR grid hierarchies is the inverse of the operations required to construct discretizations for the ghost cells employed by the application of the operator in a SAMR method. Methods that use face quadrature instead of ghost cells have similar demands. We focus on ghost-cell methods herein. Fig. 1 (top left) shows a 2D, 5-point stencil example of a SAMR mesh with the cell types used in our algorithms.

Applying an operator in Chombo requires preparing several types of ghost cells: *process* ghost cells with an "exchange" process in distributed memory, *boundary* ghost cells with interpolation from interior cells in the domain, *coarse-fine* ghost cells with interpolation from coarse grid cells in the ghosted region of fine grid domain, and *fine-coarse* interpolation for cells that have been refined. These processes interpolate given data between *genuine* cells, or degrees of freedom, and dependent *ghost* cell values. Constructing a matrix requires a reverse process of interpolating non-genuine (ghost) cell *stencil* values to genuine cell stencil values. Our approach is to decompose this process into its components with a series of transformations of a minimal amount of data required from the application operator — stencils of the operator on a uniform, infinite grid. We have instantiated some standard forms of these transformations in our matrix construction object in Chombo, but the object is designed with the understanding that future users would extend the object with new methods, such as higher-order interpolation for boundary conditions and new types of boundary conditions. To date,

* Corresponding author.
E-mail address: mfadams@lbl.gov (M.F. Adams).

https://doi.org/10.1016/j.cpc.2019.07.006
0010-4655/© 2019 Elsevier B.V. All rights reserved.

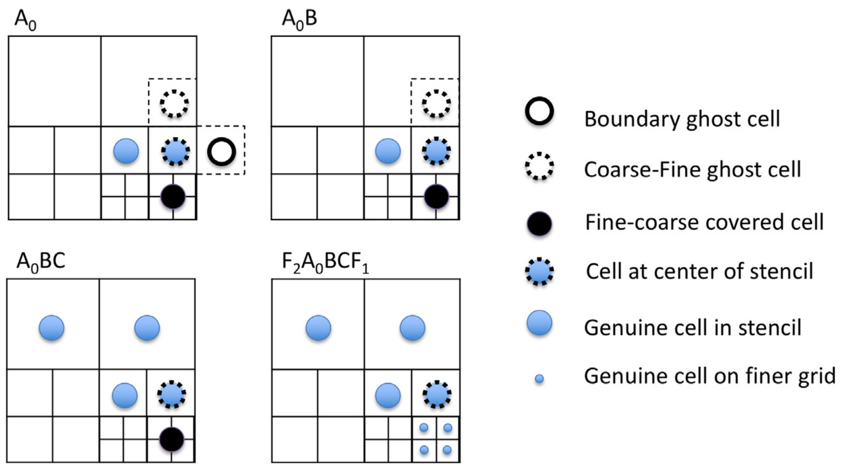

Fig. 1. Diagram of stencil transformations: generic 5-point stencil (top left); boundary ghost cell reference removed (top right); coarse-fine ghost cell reference removed (bottom left); fine-coarse covered cell reference removed (bottom right).

two applications use this technology, Chombo-Crunch [1] and BISICLES [2].

2. Methodology

Our approach requires that the developer of an operator provide a method that generates a stencil, assuming a uniform, infinite grid, at an arbitrary point in the domain. This is the minimal information needed to generate a matrix and would be the whole matrix for the operator on a uniform grid without boundary conditions. This method is called for each genuine cell on each level. This stencil is a row of a sparse matrix. In general these stencils contain non-genuine cells (columns that are not in the matrix) that must be removed to generate a matrix with only true degrees of freedom, which is a convenient form for generic algebraic solvers, but not strictly necessary. Our method iterates over all the levels and all of the genuine degrees of freedom on each level, and calls this user-provided stencil method. Chombo provides some instantiations of some simple operators, but in general this is application-specific and must be provided by the user. These stencils can be aggregated into a rectangular matrix, at least conceptually, with more columns than rows. The extra columns are a result of fact that genuine-cell (row) stencils reference non-genuine cells (columns) as well as genuine cells.

The philosophy of our approach is to require the user provide the minimal information necessary to generate the matrix, short of a higher-level PDE language to generate stencils (although one could use such an approach in the user-provided function), and then decompose the operations required to make a square matrix linearization of the operator for use in generic algebraic solvers. These operations or transformations remove stencil entries to ghost cells and interpolate their values into new stencil entries to genuine cells or, in some rare cases, other ghost cells that will be removed by later transformations.

3. Example and details of methodology

For example, a standard five-point stencil of the 2D Laplacian for cell i, with mesh spacing h, can be expressed as a list of tuples:

$$\Big\{\langle\langle i,i\rangle,\frac{4}{h^2}\rangle,\langle\langle i,i+1\rangle,-\frac{1}{h^2}\rangle,\langle\langle i,i-1\rangle,-\frac{1}{h^2}\rangle,\\ \langle\langle i+1,i\rangle,-\frac{1}{h^2}\rangle,\langle\langle i-1,i\rangle,-\frac{1}{h^2}\rangle\Big\}.$$

The transformations of our method distribute the stencil values (e.g., the $-\frac{1}{h^2}$ on any cells that are not genuine cells) to genuine cells. One can express these transformations as matrix operations, and implement them with sparse matrices. We do not use the matrix approach in the implementation, however, it is useful for defining the transformations.

There are three types of transformation: (1) Boundary ghost cells (B), (2) coarse-fine ghost cells (C), and (3) fine-coarse covered cells (F_1). In matrix notation, where these cell-level transformations are aggregated into matrices, given a rectangular input matrix of the stencils A_0, the output matrix can be computed with $A = F_2A_0BCF_1$. The F_2 matrix just removes the covered coarse cells and is included for completeness of the matrix form. We iterate over cells and compute all the transformations on each cell. This in effect fuses the loops of these matrix–matrix products. Fig. 1 shows a diagram of these transformations on a simplified three-level SAMR mesh for a five-point stencil.

The base class of this matrix construction object has virtual function implementations of each of these transformations, operating on one cell, which can be overridden if the user wished to add custom implementations. This base class has only one pure virtual function for the stencil construction (A_0). Note, one could imagine decomposing this stencil method further where divergence and gradient operators are provided and, for instance, the Laplacian could be implemented with a gradient operator on the cell identity, generating a face-centered gradient field, a material field could be applied on these faces to get fluxes, and then a divergence operator could generate the Laplacian stencil. The current implementation does not provide this level of refinement in the base class, but could implement a derived class with this approach.

3.1. Example of cell transformation for coarse-fine interface cell

Consider the stencil above, enriched with a level index l, and dropping h for clarity (it is implied with the level on a regular grid), and in 1D,

$$\{\langle\langle\langle i\rangle,l\rangle,2\rangle,\langle\langle\langle i-1\rangle,l\rangle,-1\rangle,\langle\langle\langle i+1\rangle,l\rangle,-1\rangle\}.$$

Assume the point $\langle\langle i+1\rangle,l\rangle$ is a ghost within the domain, a coarse-fine interface cell, and it is interpolated to two cells j and k, with interpolation weights α_1 and α_2 ($\alpha_1+\alpha_2=1.0$), on the next coarse level, then the transformed stencil would be of the form,

$$\{\langle\langle\langle i\rangle,l\rangle,2\rangle,\langle\langle\langle i-1\rangle,l\rangle,-1\rangle,\langle\langle\langle j\rangle,l-1\rangle,-\alpha_1\rangle,\\ \langle\langle\langle k\rangle,l-1\rangle,-\alpha_2\rangle\}.$$

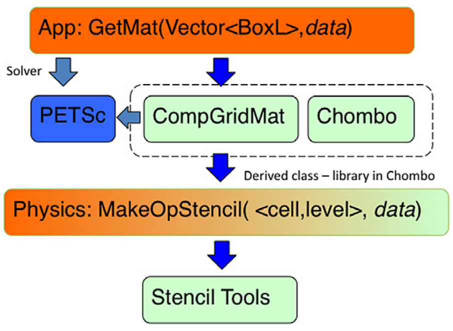
App: GetMat(Vector<BoxL>,data)
Solver
PETSc
CompGridMat
Chombo
Derived class – library in Chombo
Physics: MakeOpStencil(<cell,level>, data)
Stencil Tools

Fig. 2. Class architecture of composite grid matrix class, showing application code (orange), Chombo code (green), and PETSc library code (blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.2. Architecture

Fig. 2 shows a diagram of the architecture of the matrix construction object in Chombo. An application creates a class of the operator, and provides it with any specialized data like material parameters. The application calls a method to construct and return a PETSc matrix. The application creates a PETSc solver with the provided matrix, and solves the system. Utility methods which take data in a vector of Chombo level data objects and put them on a PETSc vector, and methods for the reverse of this operation, are provided in the base class, along with stencil tools useful in creating the stencil as a part of the stencil construction method (MakeOpStencil). The base class and some derived classes for specific operators are provided in the Chombo release.

3.3. Refluxing, preconditioning, and matrix-free solvers

The matrix that is constructed with the methods and code described here is meant to be used as the preconditioner matrix for the true operator. The true operator is usually implemented with, or wrapped in, a PETSc shell matrix. PETSc solvers are constructed with two matrices, the operator and preconditioner matrix, for this reason. It is difficult to linearize the operator exactly and so this shell matrix is usually required. Even the simple operators developed to date are not consistent for two reasons. First, we do not implement refluxing [6]. We simply average the fine-grid covered cells to the coarse grid for the stencil transformations on the coarser grid. While this method is still second-order accurate, it leads to errors in the discretization because flux is not conserved across this fine-coarse boundary. Chombo's finite-volume methods add a refluxing process to their operators to balance this flux [5], but we have not implemented this in our matrix construction object.

The second potential inconsistency is that we use the relatively new high-order coarse-fine interpolation in Chombo [7], while some Chombo operator implementations do not. This high-order interpolation was critical to achieving second-order accuracy in uniform refinement of a base SAMR grid. The nature of this high-order interpolation, which is described in the following section, provides the precise data required for the transformation, unlike simpler methods that are often composed of several steps, which must each be addressed individually.

4. Coarse-fine interfaces

A critical component in the construction of these matrices is the treatment of the coarse-fine ghost cells, or interpolants for (ghost) cells in the problem domain that are not refined on the level of the stencil center cell. High (greater than 2nd) order is required to maintain the second-order convergence of the method. This section describes, for completeness, the approach used in Chombo and utilized in the default coarse-fine ghost cell transformation in the base class, following the presentation in [7]. Ghost-cell values are interpolated from cells at the next coarser level. The interpolation stencil for a particular ghost cell consists of the coarser-level cell that contains it, together with other, neighboring coarser-level cells, as shown in Fig. 3.

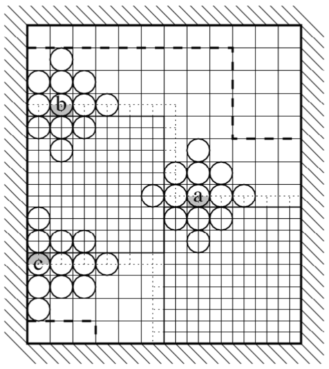

Fig. 3. Stencils for ghost-cell interpolation. In this 2D example, there are two levels, with a refinement ratio of 2, and the coarser level covers the whole rectangular domain, with boundary indicated by hatching. The finer level has one layer of ghost cells, which are shown with dotted outlines. The six shaded ghost cells, two inside each of the coarse cells marked (a), (b), and (c), are interpolated from stencils consisting of those and neighboring coarse cells marked with circles. For each of the coarse cells (a), (b), and (c), there is a coarse-fine interpolation transformation to the ghost cells within the coarse cell from the coarse cells in the stencil. Note that every stencil includes coarse cells that are covered by the finer level.

The stencil for the ghost cells within a particular coarse cell is determined by whether that coarse cell is on the boundary or is separated from the boundary by one cell. Modulo translation, reflection, and permutation of axes, the three stencils shown in Fig. 3 are all of the possibilities that can arise in 2D. Note that every stencil includes coarse cells that are covered by the finer level. The dashed lines in Fig. 3 mark the limit of coarse cells that are used in stencils to interpolate to any of the fine ghost cells, illustrating the required nesting radius of 3. In 2D, each such stencil has 13 cells, or 12 if the coarse cell is near the boundary, and transforms from values on the stencil cells to the 10 coefficients of a bivariate polynomial of degree 3, using a least-squares approximation with a conservation constraint. This polynomial is then evaluated to find values for the ghost cells contained within the coarse cell. The composition of the two operations of finding the polynomial coefficients and then evaluating the polynomial is a linear transformation from coarse-cell values to ghost-cell values, and this transformation depends only on the particular grids. The interpolation weights computed with this least squares solve are used for the weights α in the example in Section 3.1.

The size of the interpolation stencils determines a proper-nesting condition on the grids: because stencils extend two coarse cells in each dimension beyond each coarse cell containing ghost cells, we require that for every pair of successive levels, there be at least three cells at the coarser level beyond the finer-level patches, except where the finer-level patches abut the boundary.

5. Example application: BISICLES

The BISICLES ice sheet model [2] solves a 2D nonlinear coupled viscous tensor equation on an AMR hierarchy for the ice velocity

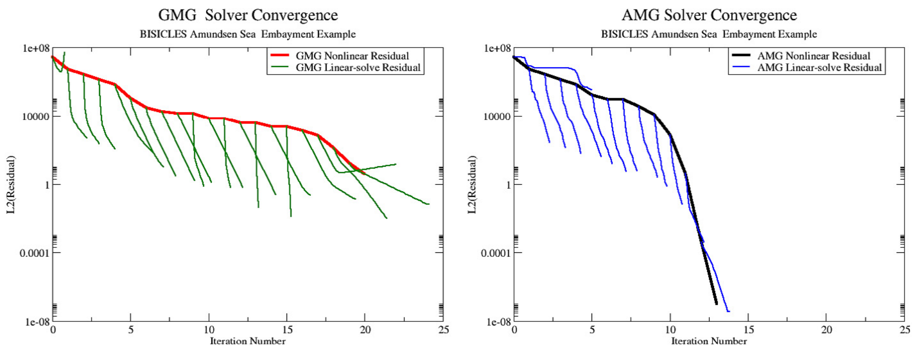

Fig. 4. Convergence history of a typical real-world BISICLES application, to the Amundsen Sea Embayment glaciers, using GMG (left) and AMG (right) approaches. Thick red and black lines show nonlinear residuals for GMG and AMG, respectively, thinner green and blue lines show linear-solver convergence.

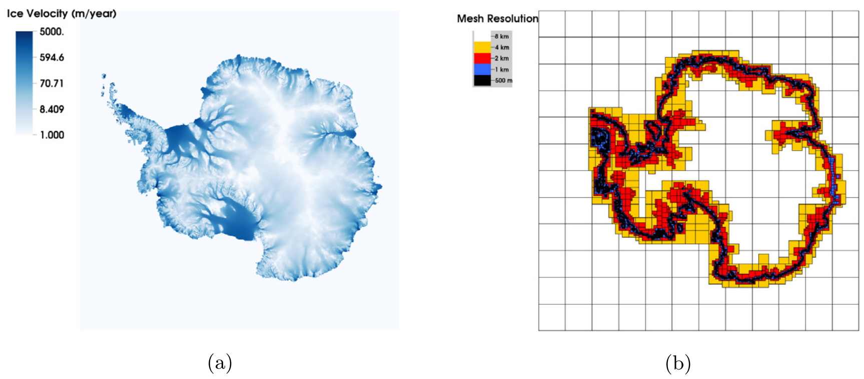

Fig. 5. Problem setup for BISICLES Antarctic example: (a) initial computed Antarctic ice-velocity field (b) Mesh resolution distribution. The coarsest mesh is 8 km and spans the entire domain. There are 4 levels of refinement, each a factor 2 finer, resolving down to 500 m resolution on the finest mesh.

field at the base of the ice $\vec{u_b}$:

$$\beta^2(u_b)\vec{u_b}+\vec{\nabla}\cdot[h\mu(\dot{\epsilon}^2,T)(\vec{\nabla}+\vec{\nabla}^T)\vec{u_b}-2\mu(\dot{\epsilon}^2,T)(\vec{\nabla}\cdot\vec{u_b})] = -\rho gh\vec{\nabla}s, \tag{1}$$

where β^2 is the basal friction coefficient, h is ice thickness, g is gravity, ρ is the ice density, and s is the vertical elevation of the ice surface. The viscosity μ varies as an inverse power law relationship with the strain rate,

$$\mu(\dot{\epsilon}^2,T) = A(T)(\dot{\epsilon}^2)^{\frac{(1-n)}{2}} \tag{2}$$

where $\dot{\epsilon}^2$ is the strain rate invariant and $A(T)$ is the temperature-dependence of ice viscosity. As a result, μ can vary over orders of magnitude, while the nonlinearity tends to concentrate velocity gradients (and thus changes in μ) into relatively narrow shear bands. Standard geometric multigrid can struggle to represent the resulting sharp coefficient gradients on coarse levels, slowing or even preventing convergence. The nonlinear solver in BISICLES is a hybrid of Picard and Newton. The solver starts with Picard and when convergence of the nonlinear residual slows (or after a specified number of iterations) switches to Newton. We use a Jacobian-Free Newton–Krylov (JFNK) nonlinear solver [8] with the *boomeramg* solver in the hypre library [9], which is supported by PETSc.

As a demonstration of our approach, we first present a case where the standard geometric multigrid approach fails. We then follow with a demonstration of the AMG solver on a full-scale problem.

The Amundsen Sea Embayment glaciers of the Antarctic Ice Sheet present a typical BISICLES problem, with fast sliding ice streams flowing into ice shelves. The basal drag coefficient β is low or zero across much of the domain. Although the standard geometric multigrid often works adequately in this region, real world applications involve an optimization problem, where (1) must be solved across a range of $\beta(x,y)$ and $\mu(x,y)$. It is common to find cases where the standard approach fails, or at least performs poorly, but the AMG approach is more robust. Fig. 4 shows the progress of the nonlinear solver for one such example: the notable feature is that while the GMG linear solver works well enough to reduce the residual of (1) by some orders of magnitude, the AMG solver allows the problem to be solved to machine-precision and with the growing-rate convergence expected of Newton's method.

To demonstrate the effectiveness of this approach on a full-scale problem, we solve a benchmark problem similar to that presented in [10], advancing the Antarctic Ice Sheet for 6 timesteps, entailing 6 nonlinear solves. A representative velocity field and AMR mesh configuration are shown in Fig. 5(a). The resulting solver convergence history is shown in Fig. 6, and demonstrates the effectiveness of the composite-grid AMG approach, even on a fairly complex AMR hierarchy with strongly varying material coefficients.

6. Conclusion

We have outlined a process of building a composite matrix linearization of a semi-structured adaptive mesh grid hierarchy and described an instantiation of this approach in the Chombo framework. The advanced high-order accurate fine-coarse cell interpolation methods in Chombo proved to simplify this work considerably. We have demonstrated the use of this approach for use in nonlinear solvers with the PETSc numerical library in the BISICLES ice sheet model.

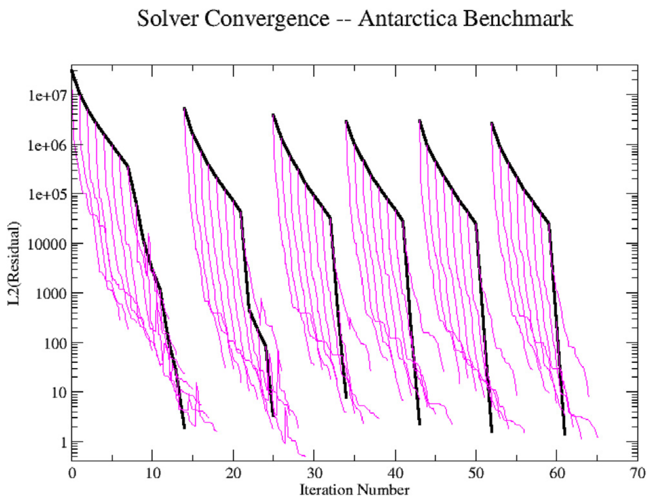

Fig. 6. Convergence history of a benchmark BISICLES AMR Antarctic run using the hypre solver (accessed through PETSc) over 8 timesteps, including 6 nonlinear multilevel viscous-tensor solves. Black lines show nonlinear residuals, magenta lines show linear-solver convergence. Iteration number on *x*-axis is the nonlinear iteration number. Each iteration for the linear solver is 0.1 on this scale.

Acknowledgments

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of Nuclear Physics, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute and the NUCLEI project under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory and the Director, Office of Science, Offices of Advanced Scientific Computing Research (ASCR) and Biological and Environmental Research (BER), of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, as a part of the ProSPect SciDAC Partnership. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] D. Trebotich, M.F. Adams, C.I. Steefel, S. Molins, C. Shen, Comput. Sci. Eng. (2014) http://dx.doi.org/10.1109/MCSE.2014.77.
[2] S.L. Cornford, D.F. Martin, D.T. Graves, D.F. Ranken, A.M. Le Brocq, R.M. Gladstone, A.J. Payne, E.G. Ng, W.H. Lipscomb, J. Comput. Phys. 232 (1) (2013) 529–549.
[3] Ulrich Trottenberg, Cornelis Oosterlee, Anton Schuller, Multigrid, Elsevier Academic Press, London, 2001.
[4] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang, PETSc Users Manual, Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015, http://www.mcs.anl.gov/petsc.
[5] M.F. Adams, P. Colella, D.T. Graves, J.N. Johnson, N.D. Keen, T.J. Ligocki, D.F. Martin, P.W. McCorquodale, D. Modiano, P.O. Schwartz, T.D. Sternberg, B. Van Straalen, Chombo Software Package for AMR Applications - Design Document, Technical Report LBNL-6616E, LBNL, Berkeley, 2014, Available at http://chombo.lbl.gov.
[6] D. Martin, P. Colella, J. Comput. Phys. 163 (2000) 271–312.
[7] P. McCorquodale, P. Colella, Commun. Appl. Math. Comput. Sci. 6 (1) (2011) 1–25.
[8] D.A. Knoll, D.E. Keyes, J. Comput. Phys. (ISSN: 0021-9991) 193 (2) (2004) 357–397, http://dx.doi.org/10.1016/j.jcp.2003.08.010.
[9] Robert D. Falgout, Ulrike Meier Yang, in: Peter M.A. Sloot, Alfons G. Hoekstra, C.J. Kenneth Tan, Jack J. Dongarra (Eds.), Computational Science — ICCS 2002, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN: 978-3-540-47789-1, 2002, pp. 632–641.
[10] S.L. Cornford, D.F. Martin, V. Lee, A.J. Payne, E.G. Ng, Ann. Glaciol. 57 (73) (2016) http://dx.doi.org/10.1017/aog.2016.13.