Contents lists available at SciVerse ScienceDirect

Journal of Computational Physics

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

Short Note

On the performance of a simple parallel implementation of the ILU-PCG for the Poisson equation on irregular domains

Frédéric Gibou [a], Chohong Min [b,*]

[a] Computer Science and Mechanical Engineering Departments, University of California, Santa Barbara, CA 93106, United States
[b] Mathematics Department, Ewha Womans University, Seoul 120-750, Republic of Korea

ARTICLE INFO

Article history:
Received 13 November 2011
Received in revised form 17 February 2012
Accepted 17 February 2012
Available online 4 April 2012

Keywords:
Incomple LU factorization
Parallel algorithm
Cuthill–Mckee ordering
Lexicographical ordering
Level set method
Preconditioned conjugate gradient
Poisson equation

ABSTRACT

We report on the performance of a parallel algorithm for solving the Poisson equation on irregular domains. We use the spatial discretization of Gibou et al. (2002) [6] for the Poisson equation with Dirichlet boundary conditions, while we use a finite volume discretization for imposing Neumann boundary conditions (Ng et al., 2009; Purvis and Burkhalter, 1979) [8,10]. The parallelization algorithm is based on the Cuthill–McKee ordering. Its implementation is straightforward, especially in the case of shared memory machines, and produces significant speedup; about three times on a standard quad core desktop computer and about seven times on a octa core shared memory cluster. The implementation code is posted on the authors' web pages for reference.

1. Introduction

The Poisson equation is used to describe a wide range of physical and biological phenomena and is also a building block for solving the Navier–Stokes equations, the Poisson–Boltzmann equation, the diffusion or the heat equation and many more. In modern computer simulations, the Poisson equation is often used on irregular domains where boundary conditions are imposed on the irregular domain's boundary. For large problems or simply for speeding the solution process, it is necessary/advantageous to consider parallel implementations for solving the linear systems resulting from spatial discretizations. Those systems are pentadiagonal in two spatial dimensions and septadiagonal in three spatial dimensions. Depending on the spatial discretizations used and the treatment of the boundary conditions at the irregular interface, the corresponding linear system is either symmetric positive definite or leads to a nonsymmetric M-matrix. In those cases, iterative methods such as the Conjugate Gradient (symmetric case) or the BiCGSTAB (nonsymmetric case) are two algorithms that are widely used. In order to speed up the convergence, an incomplete Cholesky (symmetric case) or an incomplete LU preconditioning of the linear system is used.

Different approaches can be used to parallelize a Cholesky or LU factorization. Duff and Meurant [5] have investigated the effect of the ordering of the unknowns on the convergence of the preconditioned conjugate gradient method by considering a wide range of ordering methods including red–black, nested dissection, minimum degree, and preconditionings without fill-in. They have presented the differences in condition numbers and discussed the degree of inherent parallelism: The parallelization of red–black ordering is based on multi-coloring and has diagonal blocks that can be efficiently parallelized;

* Corresponding author.
E-mail addresses: chohong@ewha.ac.kr, chohongmin@gmail.com (C. Min).

0021-9991/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcp.2012.02.023

however the preconditioning is rather mediocre. The lexicographical ordering does not directly offer a block structure where one can parallelize diagonal blocks, but has a good preconditioning property. The Van der Vorst ordering has good preconditioning properties, but it is not straightforward to parallelize as it requires one to perform a domain decomposition. The Cuthill–McKee ordering offers a good compromise since the preconditioner has a relatively low condition number (on a par with the lexicographical ordering), which translates into a small number of iterations for the solution to be reach; and the forward and backward substitutions can be parallelized in a straightforward fashion. A good review on preconditioning techniques is given by Benzi [2] and details on Cuthill–McKee ordering can be found in Sadd [11]. Overall, this gives a procedure that is straightforward to implement and produces a significant speedup.

In this paper, we report the speedup on typical Poisson problems on irregular domains with Dirichlet or Neumann boundary conditions. In addition, given the trend of shared memory, multi-threaded modern machines that are readily available and often used in research groups, we focus on shared memory machines — although we note that this approach can be applied to distributed machines as well, albeit using message passing programming. The algorithm is straightforward to implement and gives a significant speedup on standard modern machines. It is therefore worth considering for any group working on solving PDE-based physical and biological systems.

2. Spatial discretization

Consider a scalar function $u = u(\boldsymbol{x})$ satisfying the Poisson equation

$$\nabla\cdot(\beta\nabla u) = f \quad \text{in } \Omega^-, \tag{1}$$

where $\beta = \beta(\boldsymbol{x})$ is a function bounded below by a positive constant and $f = f(\boldsymbol{x})$ is a given function of the spatial variable $\boldsymbol{x}$. The computational domain $\Omega = \Omega^- \cup \Omega^+$ consists of an interior irregular domain Ω^- and its complement Ω^+. The boundary of Ω^- is defined as Γ. It is convenient to represent implicitly an irregular domain since in the case where the boundary is moving, the changes in topology are handled automatically. We therefore use the level-set formalism [9] and define the irregular domain by the set $\{\boldsymbol{x}|\phi(\boldsymbol{x}) \leqslant 0\}$, where ϕ is a Lipschitz continuous function. Fig. 1(a) illustrates an implicit representation of an irregular domain.

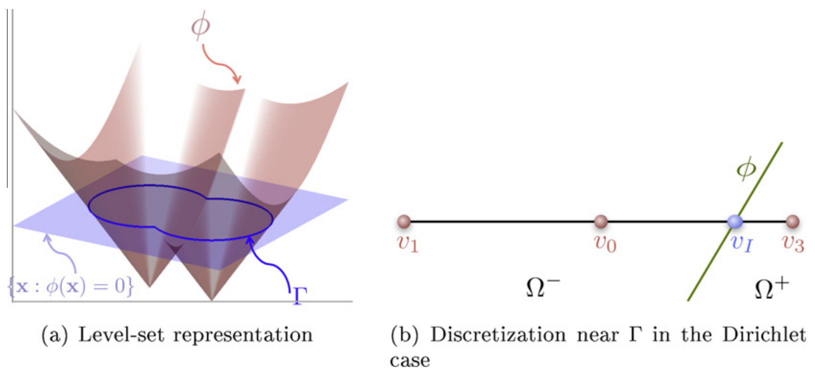
In the case where a Dirichlet boundary condition is to be imposed on Γ, we follow the approach of Gibou et al. [6]. In particular, [6] approximates (1) using standard finite difference approximations for grid points that are not adjacent to Γ and for grid points that are adjacent to the interface, the approximation is modified as to incorporate the Dirichlet boundary condition. For example, consider the case depicted in Fig. 1(b), where the node v_3 is outside the irregular domain Ω^-, the discretization is modified to directly include the Dirichlet condition at the interface point v_I:

$$u_{xx}(v_0) = \left(\frac{u_I - u_0}{\theta\Delta x} - \frac{u_0 - u_1}{\Delta x}\right)\frac{1}{\Delta x} = f_0,$$

where $\theta\Delta x$ is the distance between v_0 and v_I. The second-order derivatives in the other spatial dimensions are approximated similarly. It is shown in [6] that the corresponding linear system is symmetric positive definite and we refer the interested reader to [6] for more details on this approach. We also note that in the case where the Dirichlet boundary condition is imposed more accurately, the linear system is nonsymmetric. However, the parallel implementation we discuss here will still apply.

In the case where a Neumann boundary condition of $\partial u/\partial \boldsymbol{n} = 0$ is to be imposed on Γ, we follow the approach of Ng et al. [8] and Purvis et al. [10]: The standard five point scheme in two spatial dimensions is replaced by:

$$L_{i+\frac{1}{2},j}\frac{u_{i+1,j}-u_{i,j}}{\Delta x} - L_{i-\frac{1}{2},j}\frac{u_{i,j}-u_{i-1,j}}{\Delta x} + L_{i,j+\frac{1}{2}}\frac{u_{i,j+1}-u_{i,j}}{\Delta y} - L_{i,j-\frac{1}{2}}\frac{u_{i,j}-u_{i,j-1}}{\Delta y} = \int_{C_{ij}\cap\Omega^-} f d\Omega,$$



ϕ
$\{\mathbf{x} : \phi(\mathbf{x}) = 0\}$
Γ
(a) Level-set representation
ϕ
v_1
v_0
v_I
v_3
Ω^-
Ω^+
(b) Discretization near Γ in the Dirichlet case

Fig. 1. Left: level-set representation of an irregular domain with boundary Γ. Right: schematic representing the case where the level-set crosses between the nodes v_0 and v_3 at location v_I.

where C_{ij} is a computational cell centered on (i,j) and on a cell's face $(i-\frac{1}{2}) \times [j-\frac{1}{2}, j+\frac{1}{2}]$, $L_{i-\frac{1}{2},j}$ represents the length fraction of the face covered by the irregular domain. The integral is approximated by the geometric integration of Min and Gibou [7]. The corresponding linear system is symmetric positive definite. Similar discretizations are obtained in three spatial dimensions and we refer the interested reader to [8] for the details.

3. Preconditioning

The discretizations of Section 2 produce symmetric positive definite linear systems of the form $A\mathbf{x} = \mathbf{b}$, with A a matrix and $\mathbf{x}$ and $\mathbf{b}$ vectors. Preconditioning refers to transforming this linear system into another linear system with improved spectral properties — smaller spectral condition number, and/or eigenvalues clustered around 1. The modified system is written as $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$, where M is a nonsingular matrix that approximates A. We take the Cuthill–McKee ordering (see Duff and Meurant [5]) of the variables, as depicted in Fig. 2(a). The ILU preconditioner for the corresponding matrix takes the simple form $M = (L+D)D^{-1}(D+U)$, where L and U are the strictly lower and upper triangular matrices of A and D is a diagonal matrix that is determined from the diagonal D_0 of A:

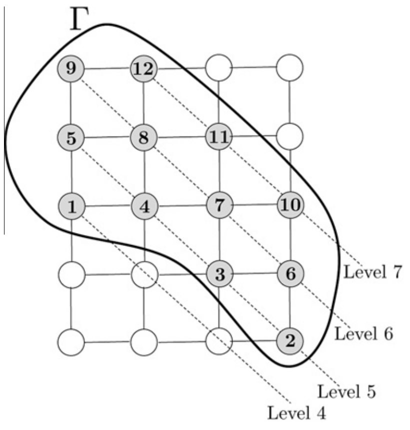
```
$D := D_0,$
for $(i,j) = (1,1),(2,1),(1,2),(3,1),\cdots,(imax,jmax)$, i.e. following the Cuthill—McKee ordering :
  if $(x_i,y_j) \in \Omega^-$,
    $D_{(i+1,j),(i+1,j)} := D_{(i+1,j),(i+1,j)} - (A_{(i,j),(i+1,j)})^2/D_{(i,j),(i,j)},$
    $D_{(i,j+1),(i,j+1)} := D_{(i,j+1),(i,j+1)} - (A_{(i,j),(i,j+1)})^2/D_{(i,j),(i,j)}.$
```

This algorithms apply to the three-dimensional case as well.

4. Parallel implementation

An iteration of the PCG algorithm involves three vector updates, two inner products, one matrix–vector multiplication, and the operations associated with the preconditioning. It is well known [11] that the bottleneck of the algorithm is associated with the preconditioning operations, and that all the other operations are straightforward to parallelize. The preconditioning operations involve the setup of the preconditioner (only done once) and the iterations to solve the linear system associated with the factored matrix $(L+D)D^{-1}(U+D)$, i.e. the forward substitution using the lower triangular matrix $I + LD^{-1}$ and the backward substitution using the upper triangular matrix $D + U$. In the case of the lexicographic ordering (or raster-scan ordering) the spatial discretizations of Section 2 result in diagonally banded matrices for which the preconditioning is very effective; however the operations associated with this preconditioning are hard to parallelize. In contrast, the red–black ordering (or checker-board ordering) allows for a very efficient parallel implementation at the expense of the preconditioning efficiency; in some examples, the iteration number using a red–black ordering can be tenfold that using a lexicographic ordering, as demonstrated in [5].

We consider the Cuthill–McKee ordering, because it strikes a good balance between parallelism and efficient preconditioning: Indeed, it allows for a natural parallel implementation of the corresponding matrix diagonal blocks as it is the case of the red–black ordering (albeit with less inherent parallelism), and its preconditioning is as efficient as that associated with the lexicographic ordering. The level l of a grid node (x_i,y_j) is defined as its index sum, i.e. $l = i + j$. The Cuthill–McKee ordering



(a) Cuthill-McKee ordering

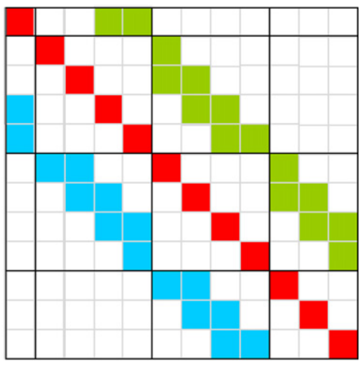

(b) Sparsity pattern

Fig. 2. Cuthill–McKee ordering (left) and sparsity pattern of the corresponding matrix (right). Note the diagonal block structures of the sparsity pattern, which implies that the nodes associated with the diagonal blocks can be updated in parallel during the forward and backward substitutions.

increases from smaller levels to larger ones as illustrated in Fig. 2(a); the grid nodes with the same level are ordered lexicographically. In the original Cuthill–Mckee ordering [4], the nodes within the same level are ordered from lowest to highest degree, where the degree of a grid node refers to the number of its neighboring nodes. Since we assume uniform grids and the Cuthill–Mckee ordering in [5] was thoroughly tested to ensure the most efficient preconditioner among 17 different ones, we take the simple lexicographical ordering within each level-set. In the forward substitution, the updates for the nodes with level l are independent to each other and depend only on the nodes with level $l-1$, as depicted in Fig. 2. Therefore the nodes with level l can be distributed evenly among the available threads and updated in parallel. The backward substitution is similar since the nodes with level l only depend on the nodes with level $l-1$. In three spatial dimensions, the Cuthill–McKee ordering also provides a natural parallelism, namely the nodes with $i+j+k=l$ can be updated in parallel.

Algorithm 1 compares two implementations of the preconditioning operation: the serial implementation in lexicographical ordering and the parallel implementation in Cuthill–Mckee ordering.

Algorithm 1. Preconditioning: $(U+D)^{-1}(LD^{-1}+I)^{-1}X$

```
Serial in lexicographical ordering
for i = 1:imax
  for j = 1:jmax
    $X_{(i,j)} := X_{(i,j)} - A_{(i,j),(i+1,j)} \frac{X_{(i+1,j)}}{D_{(i+1,j),(i+1,j)}}$
                    $- A_{(i,j),(i,j+1)} \frac{X_{(i,j+1)}}{D_{(i,j+1),(i,j+1)}}$
for i = imax:1
  for j = jmax:1
    $X_{(i,j)} := X_{(i,j)} - A_{(i,j),(i-1,j)} X_{(i-1,j)}$
                    $- A_{(i,j),(i,j-1)} X_{(i,j-1)}$
    $X_{(i,j)} := \frac{X_{(i,j)}}{D_{(i,j),(i,j)}}$
```

```
Paralllel in Cuthill–Mckee ordering
for sum = 2:imax + jmax
  parallel for (i,j) such that i + j = sum
    $X_{(i,j)} := X_{(i,j)} - \frac{A_{(i,j),(i+1,j)}}{D_{(i+1,j),(i+1,j)}} X_{(i+1,j)}$
                    $- \frac{A_{(i,j),(i,j+1)}}{D_{(i,j+1),(i,j+1)}} X_{(i,j+1)}$
for sum = imax + jmax:2
  parallel for (i,j) such that i + j = sum
    $X_{(i,j)} := X_{(i,j)} - A_{(i,j),(i-1,j)} X_{(i-1,j)}$
                    $- A_{(i,j),(i,j-1)} X_{(i,j-1)}$
    $X_{(i,j)} := \frac{X_{(i,j)}}{D_{(i,j),(i,j)}}$
```

For the septadiagonal matrix on an N^3 grid, the three vector updates require $8N^3$ number of operations, the two inner products $4N^3$, one matrix–vector multiplication $14N^3$, and one preconditioning $18N^3$. The PCG routine of Cuthill–Mckee ordering is implemented fully in parallel, so that its speedup can be increased as desired. However, in the case of lexicographical ordering, all the other operations can be implemented in parallel, but the preconditioning is in serial. The parallel fraction of the code in the case is $f_{par} = \frac{26}{42} \simeq 0.619$. By the Amdahl's law [1], its speedup is limited by $\frac{1}{1-f_{par}} \simeq 2.62$.

5. Numerical results

We provide an example in three spatial dimensions for each of the Dirichlet case and the Neumann case. In each of those cases, we report the speedup on a standard four-core 3.4 GHz Intel i7 PC and on an Intel X5570 eight-core processors shared memory cluster. The PCG routine is iterated until the convergence criteria $r^n \cdot M^{-1} r^n < 10^{-14} r^0 \cdot M^{-1} r^0$ is satisfied. Here r^p refers to the pth iteration of the residual. The run time in each example is measured in seconds.

All of our codes are implemented in the C++ language. In a standard implementation of three dimensional arrays, data access is performed using three indices, e.g. $X[i][j][k]$, and the data is stored in the lexicographical ordering. Therefore in memory, the data $X[i][j][k+1]$ is adjacent to $X[i][j][k]$ while the data $X[i][j+1][k]$ is N memory units away and $X[i+1][j][k]$ is N^2 memory units away. In the case of the lexicographically ordered PCG, the inner iteration of the preconditioning, which access the data the most, takes the same i and j and thus is at most N memory units apart to each other. However, in the case of Cuthill–Mckee ordering, the inner iteration access data ranging from $X[1][1][sum-2]$ to $X[sum-2][1][1]$ so that the data range accessed by the inner iteration can be N^3 memory units apart. In standard computer architecture, the recently used data temporarily resides in cache memory, making its subsequent access fast. For this reason, three dimensional arrays in the case of the Cuthill–Mckee ordering is stored in sequential arrays whose ordering is the same as the memory ordering. With this implementation, the inner iteration accesses data that is at most N units apart to each other. When a single thread is used, this difference in implementation makes the Cuthill–Mckee ordering slightly faster than the lexicographical ordering (even though they should be the same from an algorithm standpoint). We stress that the run time may become faster or slower than ours based on the amount of optimization technique used; however, the focus here is on the parallization speedup obtained with more threads.

5.1. Irregular domain with Dirichlet boundary condition

Consider a torus-shaped domain defined by $\phi(x,y,z) = (\sqrt{x^2+y^2}-2)^2 + z^2 - 1$, a variable coefficient $\beta(x,y,z) = 1 + \frac{1}{2}\sin(x+y+z)$, and an exact solution of $u(x,y,z) = \sin(x)\cos(y)e^{-x^2-z^2}$. We solve Eq. (1) with Dirichlet boundary condition on the boundary of the irregular domain. Here, the Dirichlet boundary condition is given by the exact solution. Table 1 gives the order of accuracy of the method and compares the required iterations of the Cuthill–Mckee ordering and

Table 1
Order of accuracy for example 5.1.

Grid	L^1 error	Order	L^∞ error	Order	# Iterations of PCG	
					Cuthill–Mckee	Lexicographical
25^3	3.23×10^{-3}		4.76×10^{-4}		12	12
50^3	9.24×10^{-4}	1.81	1.20×10^{-4}	1.98	24	24
100^3	2.50×10^{-4}	1.89	2.98×10^{-5}	2.01	44	44
200^3	6.82×10^{-5}	1.87	7.38×10^{-6}	2.01	80	80

Table 2
Speedup for example 5.1.

Ordering machine # thrs	Cuthill–Mckee								Lexicographical							
	PC				Cluster				PC				Cluster			
	1	2	3	4	1	2	4	8	1	2	3	4	1	2	4	8
Run time	45.6	24.1	17.5	15.5	82.8	42.6	21.9	12.2	66.7	48.2	42.2	38.8	109	76.5	60.4	52.5
Speedup	1	1.89	2.61	2.94	1	1.94	3.78	6.78	1	1.38	1.58	1.71	1	1.42	1.80	2.08

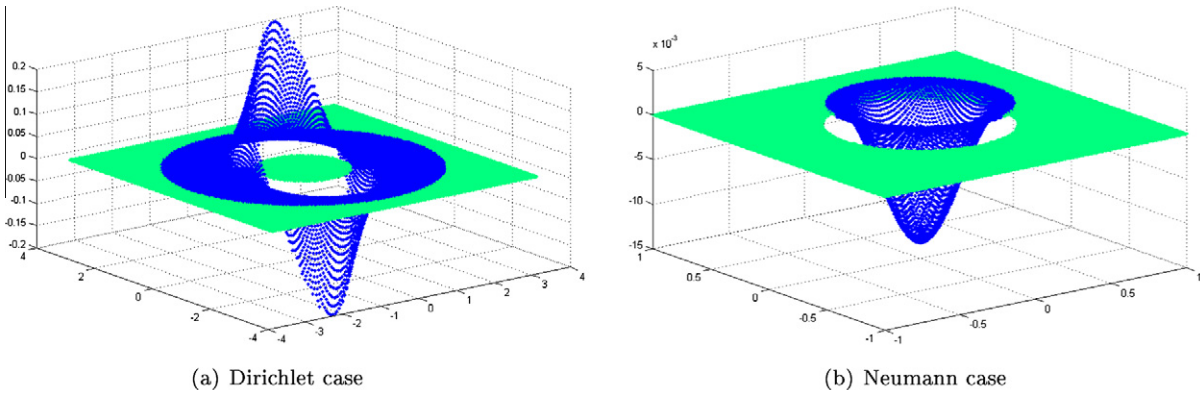


Fig. 3. Cross-section along the 0-z axis of the three dimensional solution of examples 5.1 and 5.2. The solution in Ω^- is plotted in blue, while the solution outside is set arbitrarily to 0. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 3
Order of accuracy for example 5.2.

Grid	L^1 error	Order	L^∞ error	Order	# Iterations of PCG	
					Cuthill–Mckee	Lexicographical
25^3	2.14×10^{-3}		6.85×10^{-4}		25	25
50^3	4.33×10^{-4}	2.30	1.82×10^{-4}	1.91	57	62
100^3	8.98×10^{-5}	2.27	3.59×10^{-5}	2.34	72	63
200^3	2.40×10^{-5}	1.90	9.27×10^{-6}	1.95	99	98

Table 4
Speedup for example 5.2.

Ordering machine # thrs	Cuthill–Mckee								Lexicographical							
	PC				Cluster				PC				Cluster			
	1	2	3	4	1	2	4	8	1	2	3	4	1	2	4	8
Run time	64.5	34.2	24.7	21.6	116	59.9	30.7	16.8	89.4	62.6	53.8	49.8	147	100	77.9	65.4
Speedup	1	1.88	2.61	2.98	1	1.93	3.77	6.90	1	1.42	1.66	1.79	1	1.47	1.88	2.24

lexicographical ordering. Table 2 gives the speedup obtained with the parallel implementation given in this paper. Note that the parallel speedup in Cuthill–Mckee ordering is close to the optimal value, the number of threads employed, but the speedup in lexicographical ordering is bounded above by 2.62 accordingly to the Amdahl's law. Fig. 3(a) depicts a slice of the solution.

5.2. Irregular domain with Neumann boundary conditions

The Poisson equation with Neumann boundary conditions is a very relevant problem since it is one of the main building blocks in solving the Navier–Stokes equations using a projection method [3]. Consider a spherical domain with radius $\frac{1}{2}$ centered at the origin. The exact solution is taken as $u = \left(r^2 - \frac{1}{4}\right)^3$, the weight as $\beta = 1$, and the right-hand-side of (1) is computed accordingly. We solve the linear system on the vector space of sum zero to guarantee a unique solution. Table 3 gives the order of accuracy of the method and Table 4 the speedup obtained with the parallel implementation given in this paper. As in the previous example, the parallel speedup in Cuthill–Mckee ordering is close to the optimal value, but the speedup in lexicographical ordering is bounded above by 2.62. Fig. 3(b) depicts a slice of the solution.

6. Conclusion

We have applied the Cuthill–McKee ordering to the ILU preconditioning and reported on the speedup for solving the Poisson equation on irregular domain with Dirichlet or Neumann boundary conditions. This parallel implementation is straightforward and we found that one can obtain good speedup with several processors. Given that research groups have often access to multicore/multithreaded machines, we believe that this framework will offer a favorable speedup with very minimal effort. The source code is freely available on the authors' web pages.

Acknowledgement

The authors thank Miles Detrixhe for helping in running those tests. The work of C. Min was supported by the Ewha Womans University Research Grant of 2010, the Priority Research Centers Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010–0028298) and by the Korea Research Foundation Grant funded by the Korean Government (KRF-2011–0013649). The research of F. Gibou was supported in part by ONR N00014–11-1–0027, NSF CHE 1027817, DOE DE-FG 02–08ER15991, ICB W911NF-09-D-0001 and by the W.M. Keck Foundation. The authors acknowledge computational resources from the 'Center for Scientific Computing at UCSB' and NSF Grant CNS-0960316.

References

[1] G. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities, in: AFIPS Conf. Proc., 1967, pp. 483–485.
[2] M. Benzi, Preconditioning techniques for large linear systems: a survey, J. Comput. Phys. 182 (2002) 418–477.
[3] A. Chorin, A numerical method for solving incompressible viscous flow problems, J. Comput. Phys. 2 (1967) 12–26.
[4] E. Cuthill and J. Mckee, Reducing the bandwidth of sparse symmetric matrices, in: Proc. 24th Nat. Conf. ACM, 1969, pp. 157–172.
[5] S. Duff, G. Meurant, The effect of ordering on preconditioned conjugate gradients, Bit 29 (1989) 635–657.
[6] F. Gibou, R. Fedkiw, L.-T. Cheng, M. Kang, A second–order–accurate symmetric discretization of the Poisson equation on irregular domains, J. Comput. Phys. 176 (2002) 205–227.
[7] C. Min, F. Gibou, Geometric integration over irregular domains with application to level set methods, J. Comput. Phys. 226 (2007) 1432–1443.
[8] Y.-T. Ng, C. Min, F. Gibou, An efficient fluid–solid coupling algorithm for single-phase flows, J. Comput. Phys. 228 (2009) 8807–8829.
[9] S. Osher, J.A. Sethian, Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations, J. Comput. Phys. 79 (1988) 12–49.
[10] J.W. Purvis, J.E. Burkhalter, Prediction of critical mach number for store configurations, AIAA J. 17 (1979) 1170–1177.
[11] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing, 1996.