User Guide for Ogmg: A Multigrid Solver for Overlapping Grids
Version 1.00

William D. Henshaw

Centre for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, 94551

August 18, 2012

UCRL-MA-134446

Abstract: This is the user guide to Ogmg, the Overlapping-Grid-MultiGrid-solver. This guide describes how to use Ogmg to solve elliptic boundary value problems. The various parameters associated with the solver are explained. The Overlapping-Grid-MultiGrid-solver, Ogmg, that can be used to obtain solutions to elliptic boundary value problems. Ogmg solves problems in two and three space dimensions on composite overlapping grids. Second and fourth-order accurate approximations are supported. Given an overlapping grid generated from the Ogen grid generator, Ogmg will generate the coarse grid multigrid levels using an automatic coarsening algorithm. The equations on the coarse grids can be determined automatically using a Galerkin averaging procedure. The multigrid solution algorithm has been optimised for some commonly occuring problems such as equations defined with the Laplace operator. Smoothers include Red-Black, Jacobi, Gauss-Seidel, line-zebra and line-Jacobi. Ogmg is particularly efficient when a majority of the grid points belong to cartesian component grids; this is often the case when grids become sufficiently fine. The fourth-order accurate approximations are solved directly with multigrid (as opposed to using a defect correction scheme). Convergence rates for the fourth-order approximations are often nearly as good as the convergence rates for second-order discretizations.

Contents

1 Introduction

Ogmg is a multigrid solver for use with Overture [2],[3]. Ogmg can solve scalar elliptic problems on overlapping grids. It has a variety of smoothers including Red-Black, Jacobi, Gauss-Seidel and line smoothers. Second-order accurate and fourth-order accurate approximations are supported. A sparse direct or sparse iterative solver such as GMRES can be used to solve the coarse grid equations – the Overture solver Oges is used for this purpose, thus allowing access to a variety of sparse matrix solvers such as those available with PETSc[1].

In the case of a general elliptic boundary value problem, the system of equations that Ogmg solves are specified as a "coefficient-array" grid function. The coefficient array can be created using the Overture operator classes.

Ogmg has been specifically optimised for a class of commonly occuring problems. These *predefined* equations are

$$\begin{aligned}
\Delta u &= f && \text{laplaceEquation} \\
\nabla\cdot(s(\mathbf{x})\nabla)u &= f && \text{divScalarGradOperator} \\
(I + c_0\Delta)u &= f && \text{heatEquationOperator} \\
(I + s(x)\Delta)u &= f && \text{variableHeatEquationOperator} \\
(I + \nabla\cdot(s(\mathbf{x})\nabla))u &= f && \text{divScalarGradHeatEquationOperator}
\end{aligned}$$

These equations are augmented with Dirichlet, Neumann or mixed boundary conditions. The above equations can be solved more quickly and with less storage than if the same equation had been defined through a general coefficient matrix.

Ogmg starts with an overlapping grid constructured with the overlapping grid generator Ogen. The coarse grids needed by the multigrid algorithm are automatically generated with a new coarsening algorithm. Very coarse grids can be formed by relaxing the accuracy requirements for interpolation on the coarser grids and allowing the overlap between grids to grow as the grids are coarsened.

The discrete approximations to the equations on the coarse grids are determined automatically using a Galerkin averaging procedure from finer grids.

An adaptive smoothing algorithm is used to improve the convergence rate. The number of sub-smooths performed on each component grid is adjusted to keep the residuals nearly the same.

Good multigrid convergence rates are usually obtained. Ogmg is particularly efficient when a majority of the grid points belong to cartesian component grids; this is often the case when grids become sufficiently fine. The overall convergence rates has been significanty improved from the original fortran version of the code [5]. Some of the factors that led to this improved performance were

- an adaptive smoothing algorithm that performs additional sub-smooths on component grids that are converging slowly.
- generating the coarse grid equations through operator averaging improves the convergence rate.
- careful attention to boundary conditions (especially for Neumann boundary conditions and fourth-order accurate discretzations) can improve the convergence rate.
- using over-relaxation for Red-Black smoothers is very helpful, especially in 3D.

Ogmg does not yet efficiently handle the case of singular problems such as a Poisson equation with all Neumann boundary conditions; in this case one must first determine the left null vector to the discrete operator.

2 The multigrid algorithm for overlapping grids

Ogmg uses the standard defect correction algorithm. The implementation on overlapping grids is relatively straightforward. See the paper [5] for further discussion.

- Typically Jacobi, red-black or line (zebra) smoothers are used.
- The fine to coarse *Restriction* operator is the *full weighting* operator except at boundaries.
- The coarse to fine Prolongation operator is second or fourth order interpolation; second-order by default.
- The **cycle** chosen is either adaptive or can be fixed to a desired one.

```
while not converged do
    smooth $\nu_1$ times or until the smoothing rate $> \eta$
        $v_1 \leftarrow S^{\nu_1} v_1$
    form the defect and transfer to the coarser grid
        $f_2 \leftarrow R^{1\rightarrow 2}(f - A v_1)$
    "solve" the defect equation (at least to an "accuracy" of $\delta$)
        $A_2 v_2 \approx f_2$
    correct the fine grid solution from the coarse grid solution
        $v_1 \leftarrow v_1 + P^{2\rightarrow 1} v_2$
    smooth $\nu_2$ times or until the smoothing rate $> \eta$
        $v_1 \leftarrow S^{\nu_2} v_1$
end while
```

The **smoothing step** represented by the operator S is a *composite-smooth* where each grid in turn is smoothed:

```
for each grid g in a CompositeGrid do
    smooth grid g $\nu_g$ times
    interpolate
end for
```

The smoother and the number of smooths may vary from component grid to component grid. We try to choose n_g, the number of smooths on each component grid, so that the residual stays about the same size on each component grid. The approximate rule we use is that

$$n_g \approx \frac{\|\text{ residual on grid } g\|}{\min_g \|\text{residual on grid } g\|}$$

The grid with the smallest residual will have $n_g = 1$.

3 Using Ogmg

3.1 Basic Steps

To use Ogmg to solve a problem the user should take the following steps:

1. Generate an overlapping grid with the grid generator `ogen`. The number of grid lines on each grid must be chosen appropriately to allow for some number of multigrid levels to be generated. Suppose that one would like $L > 1$ multigrid levels. If there are N grid lines in a given direction then $N-1$ must be divisible by 2^{L-1}.
2. If the equations one wishes to solve are not one of the predefined equations (**??**) then use the operator classes to build a coefficient matrix defining the discretization of an elliptic boundary value problem. There are a number of sample programs that show how to define such a problem.
3. Create an Ogmg object and assign parameters such as the type of smoother.
4. Define the right hand side function for the PDE, including boundary conditions. This need only be done at the finest level.
5. solve the problem.

3.2 Convergence criteria

Ogmg uses two possible convergence criteria. One can either measure the convergence through the maximum norm of the residual, or by an estimate of the error. **Both** criteria must be met for convergence.

The residual convergence criteria is

$$\|\text{residual}\|_\infty < \texttt{residualTolerance} \times \text{number of grid points}$$

where the '`residualTolerance`' is the user specified tolerance. We scale by the number of grid points as an attempt to make this tolerance somewhat independent from the number of grid points. Note that for a typical elliptic problem the very best value that can be expected for the residual is proportional to the round-off error, ϵ/h^2, where ϵ is the machine epsilon and h is the grid spacing (so that $1/h^2$ is approximately the number of grid points).

The second converge criteria is based on an error estimate,

$$E^n < \texttt{errorTolerance}$$

where '`errorTolerance`' is a user specified value. The estimate E^n for the error at iteration n is computed as

$$\delta^n = \frac{\|u^{n+1} - u^n\|}{\|u^n - u^{n-1}\|}$$
$$E^n = \frac{\delta}{1-\delta}\|u^n - u^{n-1}\|$$

where we use the maximum norm. This estimate follows assuming the solution is converging at a rate δ,

$$u^n - u^\infty \approx \delta^n v, \qquad (E^n \approx \|u^n - u^\infty\|_\infty)$$

3.3 Test routine ogmgt

The test routine `Overture/Ogmg/ogmgt.C` shows how to call Ogmg to solve Poisson's equation with either Dirichlet or Neumann boundary conditions:

$$\Delta u = f \qquad \text{for } \mathbf{x} \in \Omega$$
$$u = g, \text{ or } u_n = g \qquad \text{for } \mathbf{x} \in \partial\Omega.$$

The forcing functions f and g are chosen so that the true solution is known. We use the Twilight-Zone functions defined in the `OGPolyFunction` and `OGTrigFunction` classes to define the true solution and its derivatives. See the **OtherStuff** documentation [4] for further details on these classes.

4 Smoothers

A variety of smoothers are available for use with Ogmg. These include

Red-Black: ω-RB-GS - Red-Black Gauss-Seidel with a relaxation coefficient ω, is generally the best smoother for the Laplace operator on grids that are not too highly stretched.

Jacobi: ω-Jac - under-relaxed Jacobi is a reasonable smoother. It gives a much smoother looking and symmetric defect than ω-RB-GS and so can be useful for debugging.

Gauss-Seidel: ω-GS.

Line-Jacobi: ω-xLJac, ω-yLJac, ω-zLJac, ω-aLJac

Line-Zebra: ω-xZGS, ω-yZGS, ω-zZGS,ω-aZGS, - x, y, z and alternating line-zebra are often the best smoothers for highly stretched grids. In three dimensions it pays to include a relaxation coefficient, ω.

5 Options and Parameters

The majority of options for Ogmg are contained in the **OgmgParameters** class.
Options can be set interactively or through function calls.

Smoothing parameters smoother type

```
"!Ogmg parameters",
">smoother",
  "jacobi",
  "gauss-seidel",
  "red black",
  "line jacobi direction 1",
  "line jacobi direction 2",
  "line jacobi direction 3",
  "line zebra direction 1",
  "line zebra direction 2",
  "line zebra direction 3",
  "alternating",
  "alternating jacobi",
  "alternating zebra",
  "smoother(grid)=[ja][rb][ld1][ld2][ld3][alj][alz]",
  "smoother(grid,level)=[ja][rb][ld1][ld2][ld3][alj][alz]]",
"<number of smooths",
"maximum number of iterations",
"number of cycles",
"number of smooths",
"number of smooths per level",
"number of sub-smooths",
"residual tolerance",
"error tolerance",
"fine to coarse transfer width",
"coarse to fine transfer width",
">boundary conditions",
  "use symmetry for Neumann BC on lower levels",
  "do not use symmetry for Neumann BC on lower levels",
  "use symmetry corner boundary condition",
  "do not use symmetry corner boundary condition",
  "boundary averaging option",
  "ghost line averaging option",
  "extrapolate fourth order boundary conditions",
  "use equation for fourth order boundary conditions",
  "useEquationForDirichletOnLowerLevels",
  "solve equation with boundary conditions",
  "do not equation with boundary conditions",
"<>miscellaneous",
  "maximum number of levels",
  "maximum number of extra levels",
  "smoothing rate cutoff",
  "number of iterations on coarse grid",
  "minimum number of initial smooths",
  "interpolate the defect",
  "do not interpolate the defect",
  "use automatic sub-smooth determination",
  "do not use automatic sub-smooth determination",
  "show smoothing rates",
  "output a matlab file",
  "do not show smoothing rates",
  "use optimized version",
  "do not use optimized version",
  "do not couple coarse grid equations",
  "average coarse grid equations",
  "do not average coarse grid equations",
  "do not average coarse curvilinear grid equations",
  "omega Jacobi",
  "omega Gauss-Seidel",
  "omega red-black",
  "omega line-Jacobi",
  "omega line-zebra",
  "variable omega scale factor",
  "use locally optimal omega",
  "do not use locally optimal omega",
  "use split step line solver",
  "do not use split step line solver",
  "maximum number of sub-smooths",
  "maximum number of line sub-smooths",
  "use full multigrid",
  "do not use full multigrid",
  "interpolate after smoothing",
  "do not interpolate after smoothing",
  "order of extrapolation for Dirichlet on lower levels",
  "forward ordering of grids in smooth",
  "reverse ordering of grids in smooth",
  "alternate ordering of grids in smooth",
  "alternate smoothing directions",
  "fully alternate smoothing directions",
  "do not alternate smoothing directions",
  "number of boundary layers to smooth",
  "number of boundary smooth iterations",
  "number of levels for boundary smooths",
  "number of interpolation layers to smooth",
  "number of interpolation smooth iterations",
  "number of levels for interpolation smooths",
  "use an F cycle",
"<debug",
">coarse grid options",
  "Oges parameters",
  "iterate on coarse grid",
  "number of coarse grid iterations",
```

6 Examples of overlapping grids and multigrid levels

Ogmg will automatically generate the sequence of coarse grids required by the multigrid algorithm. The difficult step in creating a coarse level is determining how to interpolate between different component grids.

Figures (1-2) show some examples of overlapping grids and their multigrid levels. Notice that the overlap between grids tends to increase as the grids are coarsened. The accuracy of interpolation may also be relaxed on the coarse grids.

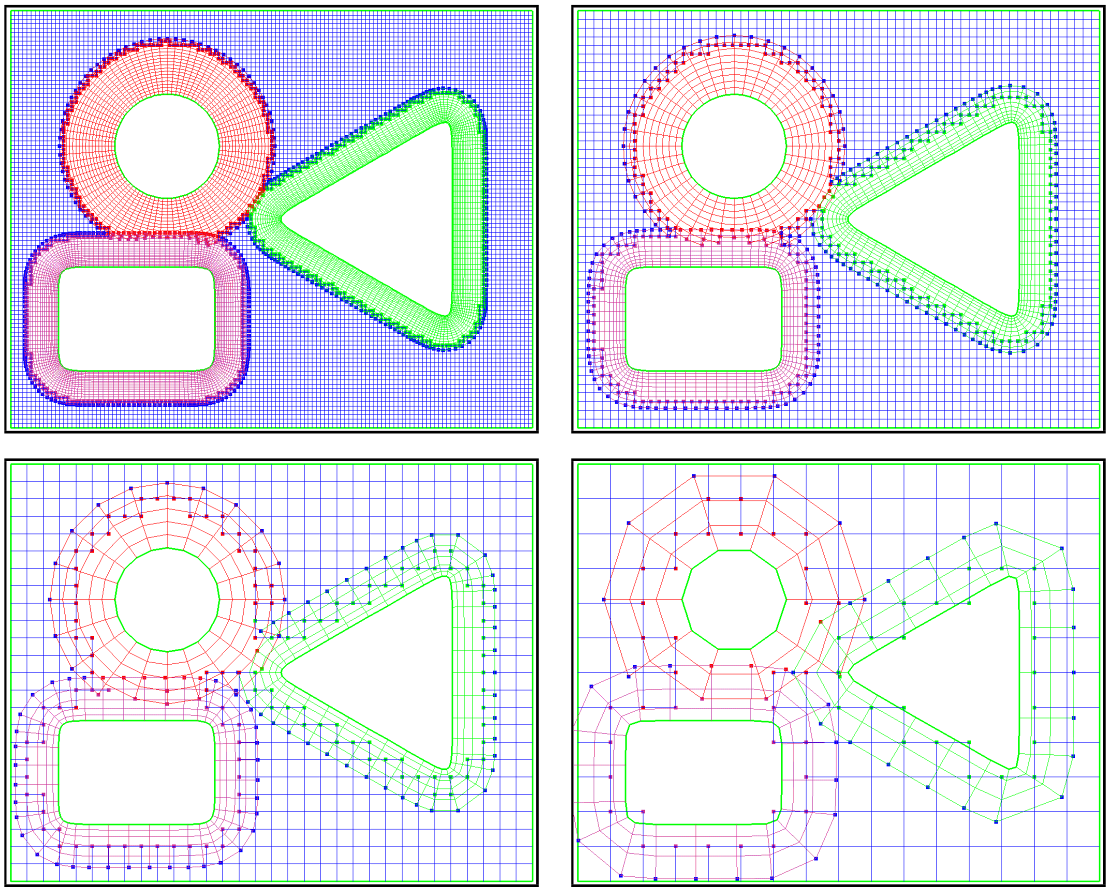


Figure 1: An overlapping grid for some shapes, 4 multigrid levels. Ogmg will automatically generate the coarser levels.

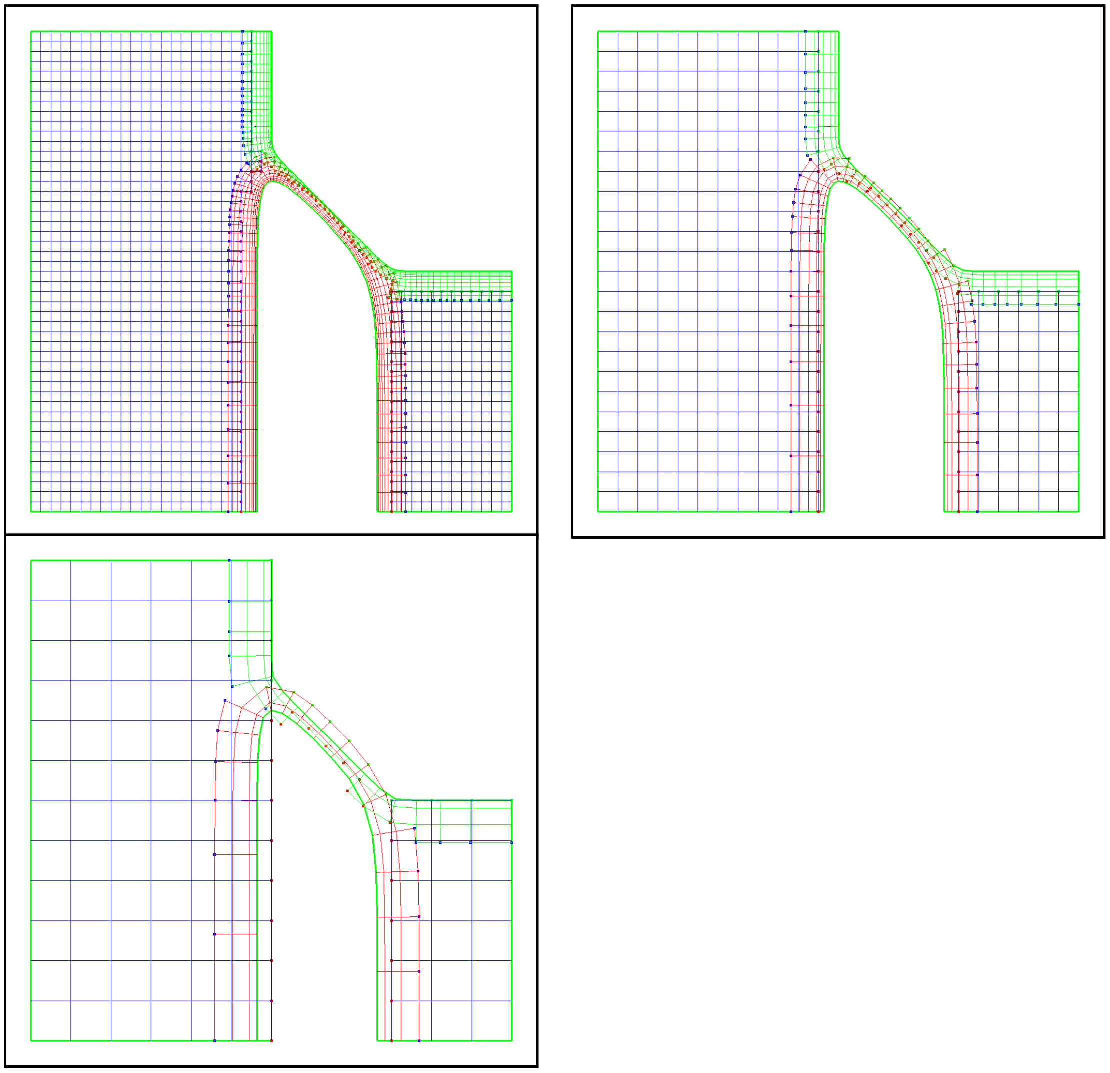


Figure 2: An overlapping grid for a valve, 3 multigrid levels.

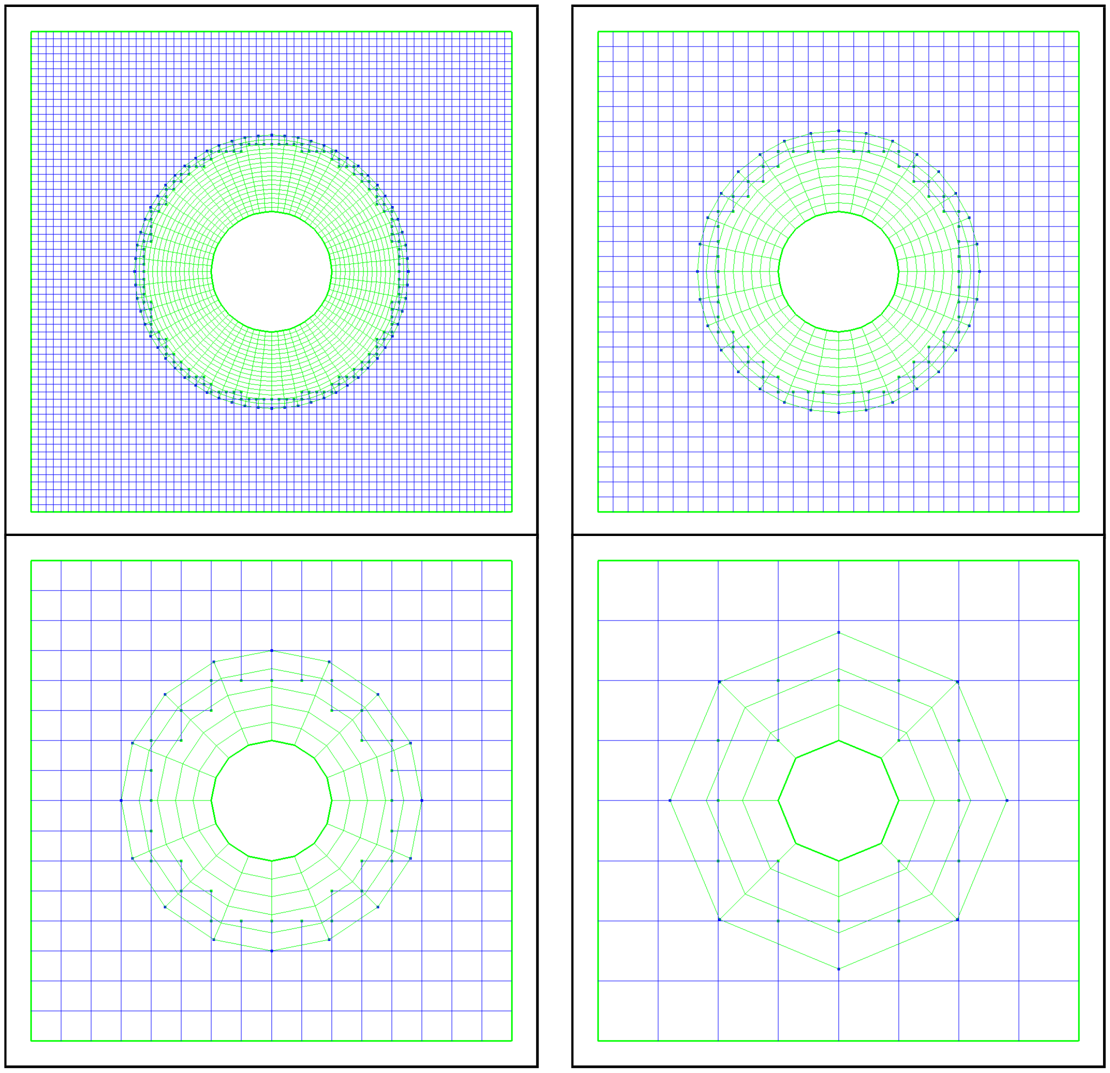


Figure 3: An overlapping grid for a circle in a channel, 4 multigrid levels.

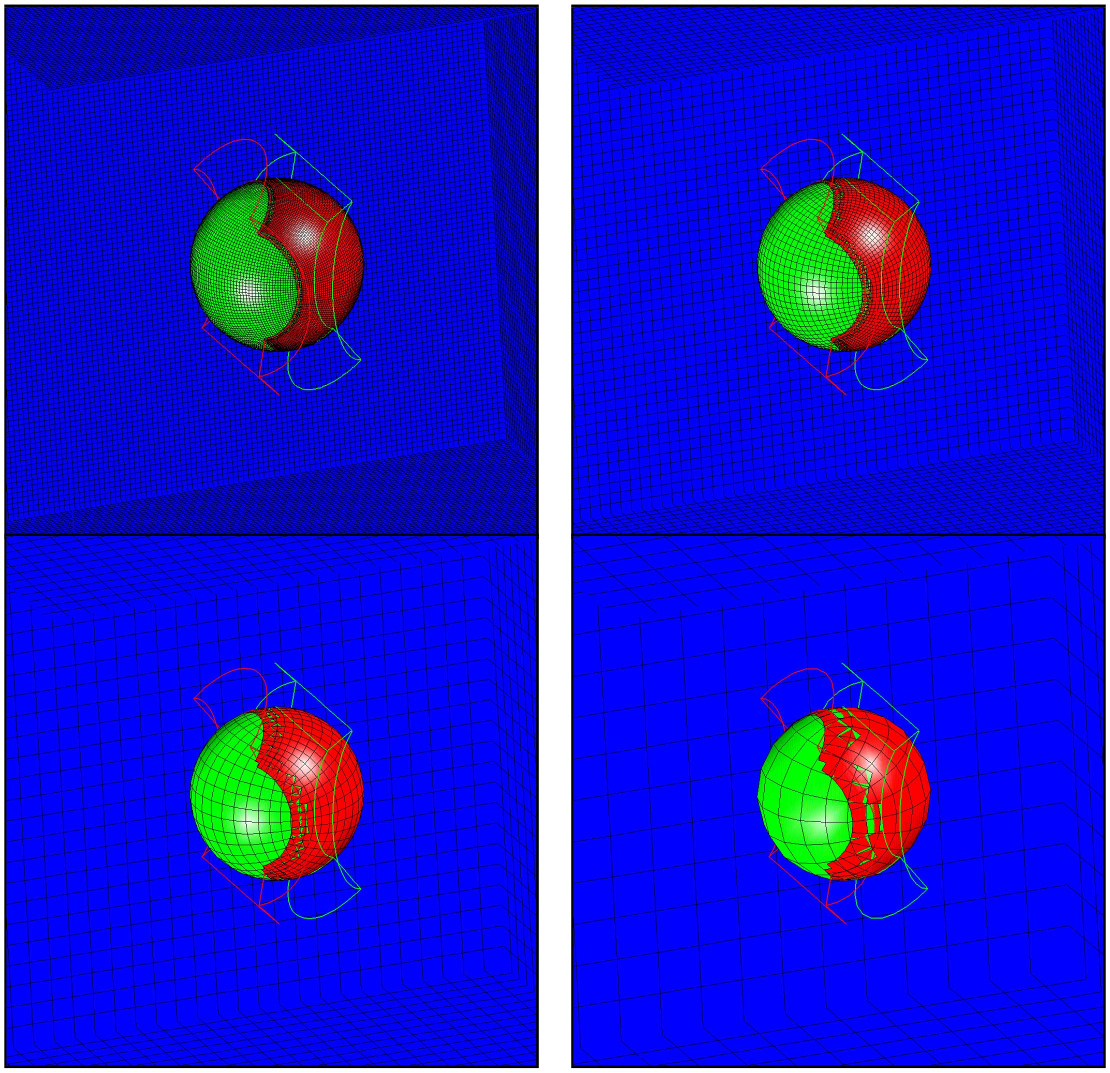


Figure 4: An overlapping grid for a sphere-in-a-box, 4 multigrid levels.

7 Numerical Results

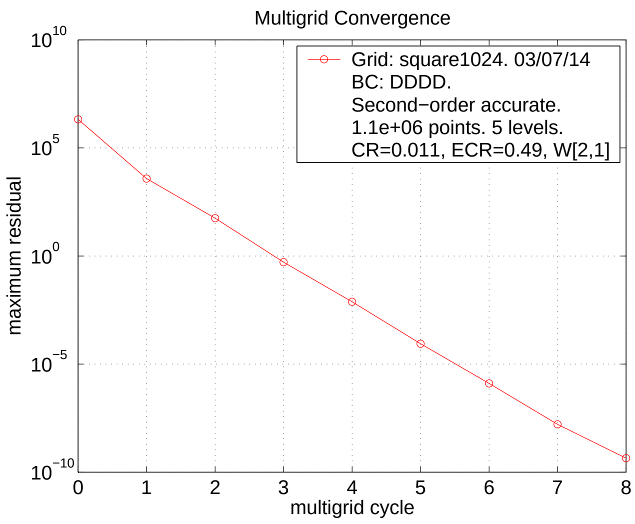
In this section we present some sample numerical results for some two and three dimensional problems, second- and fourth-order accuracy, Dirichlet, Neumann and mixed boundary conditions.
Notation:

$$\begin{aligned}
\text{WU(i)} &= \text{number of work units for iteration i} \\
\text{res(i)} &= \text{residual for iteration i} \\
\text{rate(i)} &= \text{convergence rate, res(i)/res(i-1)} \\
\text{ECR(i)} &= \text{effective convergence rate} \\
&= \left(\frac{\text{res}(i+1)}{\text{res}(i)}\right)^{1/WU(i)} \\
\text{err(i)} &= \text{maximum error in the solution for iteration i} \\
n_s &= \text{number of smooths per level}
\end{aligned}$$

A work unit is defined to be the amount of work (number of multiplications) required for a single Jacobi iteration. The work units reported here are only reasonable approximations.

The effective convergence rate (ECR) is a normalized convergence rate that takes into account the amount of work required for each multigrid iteration. The ECR is the convergence rate that a Jacobi iteration would have to achieve per iteration in order to be as good as the multigrid algorithm. Recall that the Jacobi iteration has a convergence rate for standard elliptic problems that quickly approaches 1, $ECR = 1 - O(h^2)$ as the mesh size h goes to zero. Optimal SOR is better with $ECR = 1 - O(h)$. Multigrid effective convergence rates are generally in the range .5 to .8, independent of the mesh size. (The convergence rate for a full cycle is usually about .1). Since the convergence rate does not depend on h the method has optimal complexity – the work required to compute the solution to a given accuracy requires a fixed number of iterations, independent of h, and is thus proportional to the number of unknowns.

7.1 Square



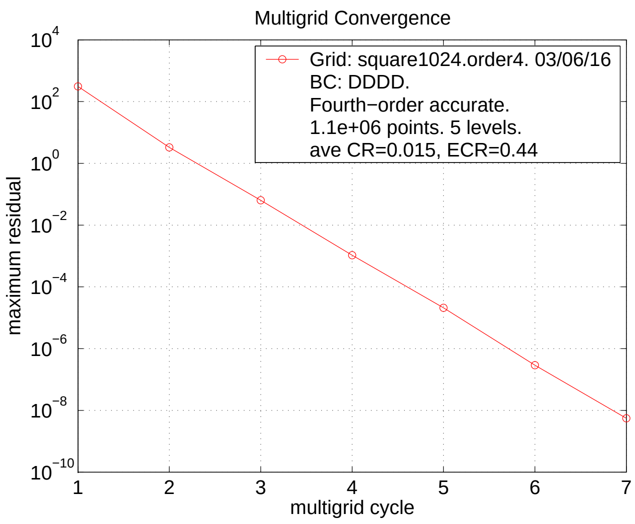


Figure 5: Convergence history for square1024, second- and fourth-order accuracy, V(2,1) cycle. The convergence rate for the fourth-order accurate discretization is almost as good as the second-order accurate case.

7.2 Circle in a channel

The circle in a channel, figure 3, is an example of a simple overlapping grid. As the grid becomes finer, the radius of the annulus grid is reduced, and thus the relative number of points on the cartesian grid increases. This explains why the convergence rate improves as the grid is refined.

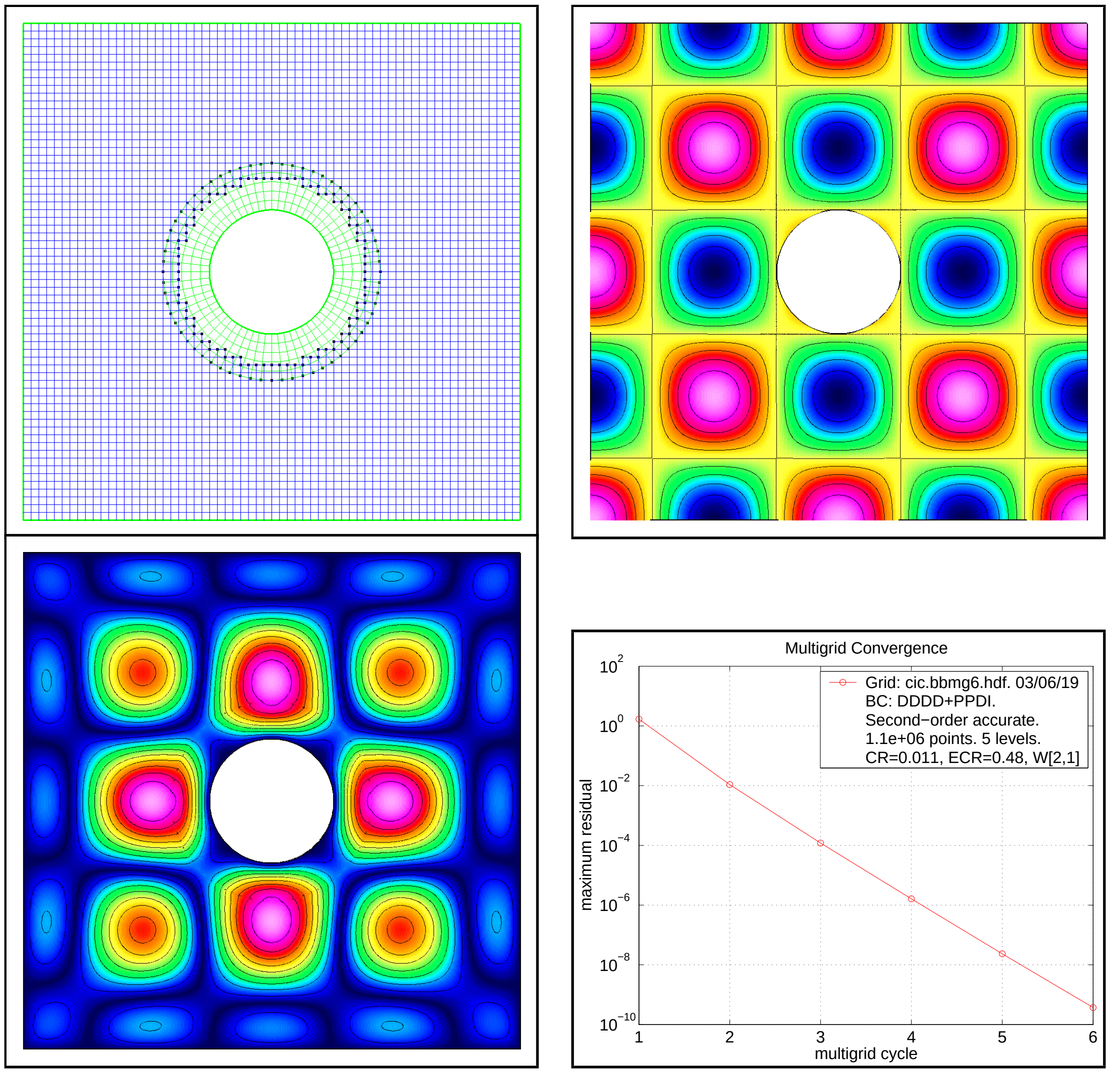


Figure 6: Top left: An overlapping grid for a circle in a channel, (fifth multigrid level of cic.bbmg6). Top right: computed solution. Bottom left: Error. Bottom right: convergence history.

7.2.1 Fourth-order accurate circle-in-a-channel

7.3 Shapes

The “shapes” geometry is shown in figures 1 and 8. The majority of the grid points on the the fine versions of the composite grid are on the cartesian grid.

7.4 Airfoils in a channel

7.5 Box

7.6 Sphere in a box

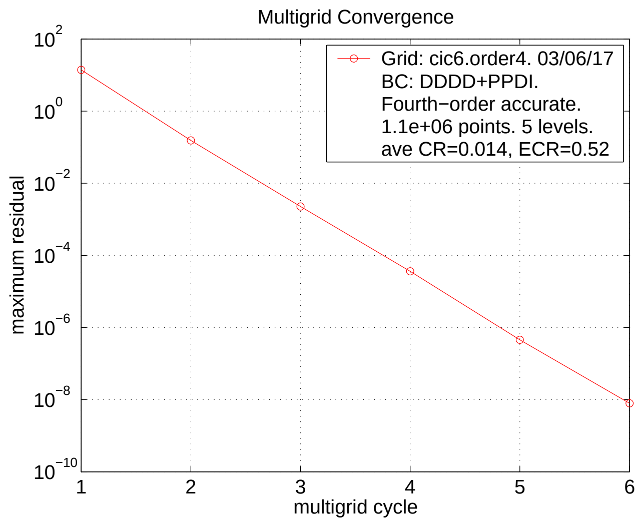
7.6.1 Fourth-order accuracy

7.7 Ellipsoid-in-a-Box

The ellipsoid in a box grid is shown figure 11. Also shown are the computed solution and error.

7.8 Multiple spheres in a box

Figure **??** shows results for a geometry consisting of a collection of spheres in a box.



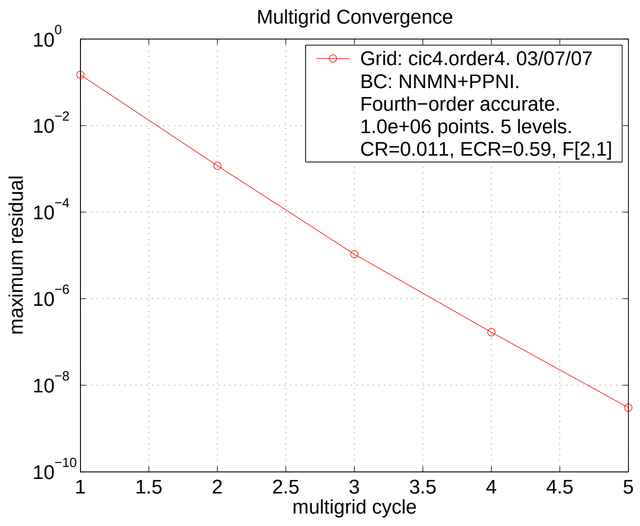


Figure 7: Convergence history for a circle-in-a-channel, fourth-order accurate, with IBS smoothing. Left: W[2,1], cic6.order4. Right: F[2,1], cic4.order4.

7.9 Sub-in-a-Box

The sub in a box grid is shown figure 13. Also shown are the computed solution and error.

7.10 Performance and Comparison to other methods

In table (3) we show a comparison of the multigrid solver to an iterative solver (stabilized bi-conjugate-gradient from PETSc) and a direct sparse matrix solver (YALE). In table (4) we show some timings and memory usage for a selection of problems.

In table 1 we compare the results for a variety of Krlov solvers and preconditioners. We consider two- and three-dimensional problems with second-order discretizations. In two dimensions bi-CG-stab with ILU(5) seems to give about the best results in terms of CPU time.

Solver	grid	pts	its	$\|\|res\|\|_\infty$	CPU	setup	solve	reals/pt
gmres(20), ILU(0)	cic	1.1e6	2657	8.9e-9	1135		1102	49.0
gmres(20), ILU(5)	cic	1.1e6	435	1.0e-8	271		236	65.0
biCG-stab, ILU(0)	cic	1.1e6	554	8.6e-9	342	32	310	33.3
biCG-stab, ILU(3)	cic	1.1e6	225	9.1e-9	190		157	37.4
biCG-stab, ILU(5)	cic	1.1e6	144	8.9e-9	152		117	53.5
biCG-stab, ILU(10)	cic	1.1e6	100	6.8e-9	155		117	57.6
gmres(20), ILU(0)	ellipsoid2a	2.0e6	218	5.2e-10	306.	70.	236	56.5
biCG-stab, ILU(0)	ellipsoid2a	2.0e6	113	3.7e-10	264.		187	41.6
biCG-stab, ILU(1)	ellipsoid2a	2.0e6	62	3.9e-10	259.		180	45.1
biCG-stab, ILU(2)	ellipsoid2a	2.0e6	46	4.1e-10	222.		116	70.3
biCG-stab, ILU(3)	ellipsoid2a	2.0e6	36	2.2e-09	299.		126	77.3

Table 1: A comparison of various Krylov solvers and preconditioners. biCG-stab is a stabilized bi-Conjugate Gradient solver. gmres(20) is a generalized miniumum residual solver with a restart length of 20. ILU(k) is an incomplete LU preconditoner with k extra levels of fillin. The grid cic is a second-order accurate circle-in-a-channel grid.

In table 3 we compare results from Ogmg to those of some good Krylov solvers.

For the two-dimensional grid cic with about 1.1 million grid points the solve time for Ogmg is about 30 times faster and uses about 10 times less storage than biCG-stab, ILU(5). Compared to biCG-stab, ILU(0) Ogmg is about 80 times faster and uses about 6 times less memory (for the solve step).

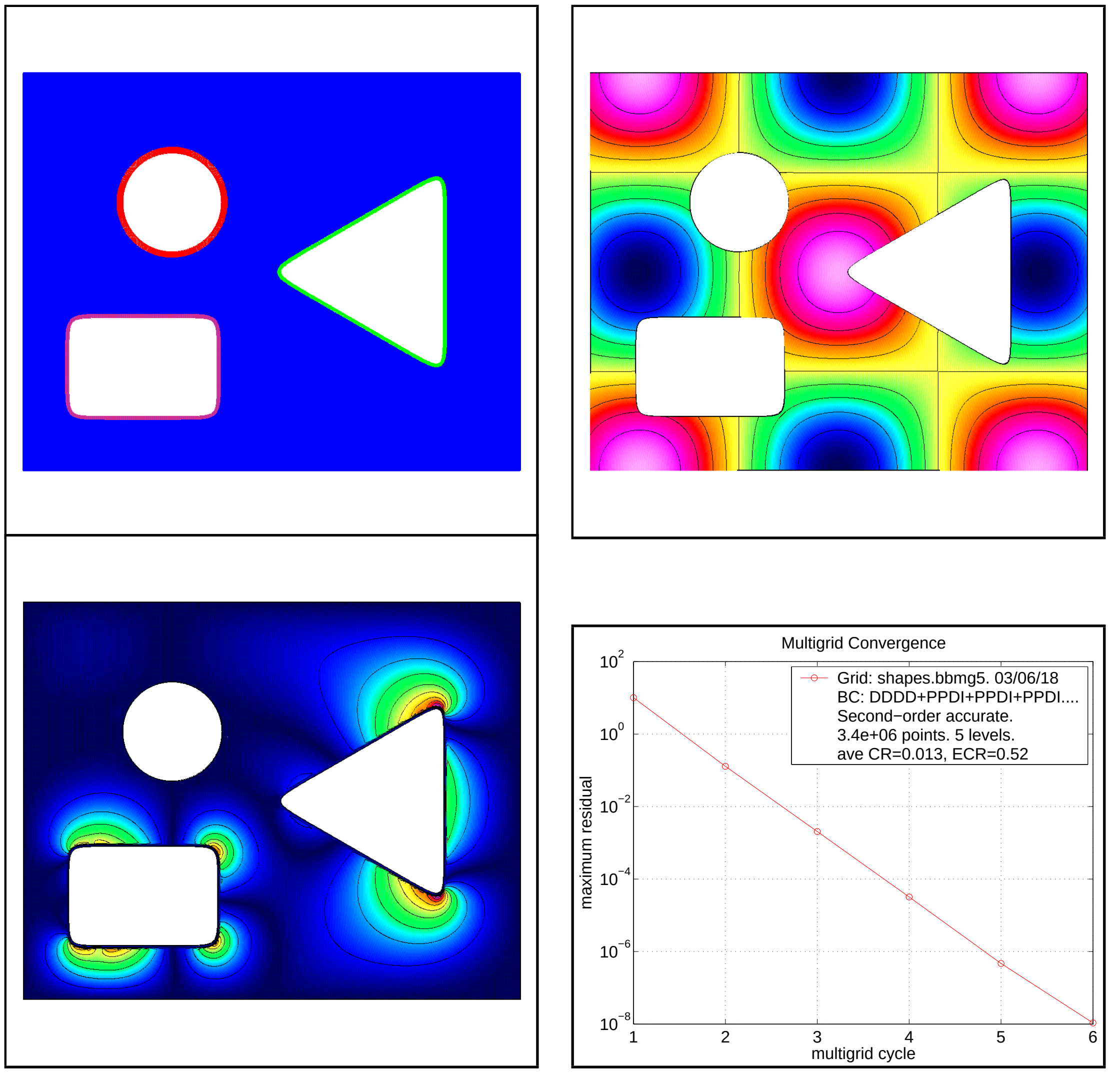


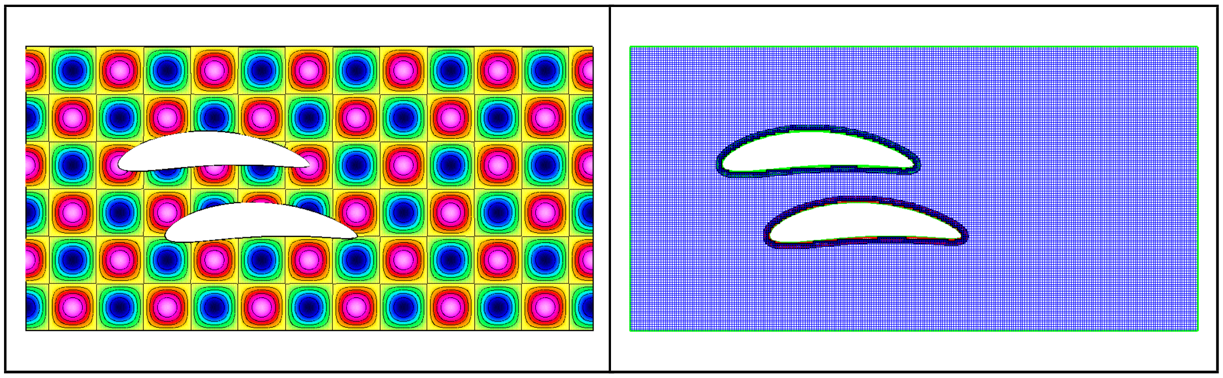
Figure 8: Top left: An overlapping grid for some shapes (shapes.bbmg4). Most of the grid points are on the cartesian grid. Top right: computed solution. Bottom left: Error. Bottom right: convergence history.

The setup time for Ogmg includes the time needed for generation of the multigrid levels and generation of the coarser grid operators by averaging. The setup time for the Krylov solvers includes the time required to copy the matrix coefficients from Overture to PETSc and the time needed to build the preconditoner.

Solver	grid	pts	its	$\|\|res\|\|_\infty$	CPU	setup	solve	reals/pt
Ogmg V[1,1] FMG	cic	1.1e6	7	5.7e-10	3.21	.49	2.7	4.6
biCG-stab, ILU(5)	cic	1.1e6	144	8.9e-9	152	35.	117	53.5
biCG-stab, ILU(0)	cic	1.1e6	554	8.6e-9	342	32	310	33.3
Ogmg V[1,1] FMG	ellipsoid2a	2.0e6	10	3.3e-10	21.5	4.52	17.0	9.9
biCG-stab, ILU(2)	ellipsoid2a	2.0e6	46	4.1e-10	222.	106	116	70.3
biCG-stab, ILU(0)	ellipsoid2a	2.0e6	113	3.7e-10	264.	77.	187	41.6

Table 2: A comparison of the multigrid solver Ogmg to Krylov based solvers. The biCG-stab ILU(5) is faster than biCG-stab ILU(0), but requires significantly more memory. The grid cic is a second-order accurate circle-in-a-channel grid.

In table 4 we show some timings and memory usage for a selection of problems.



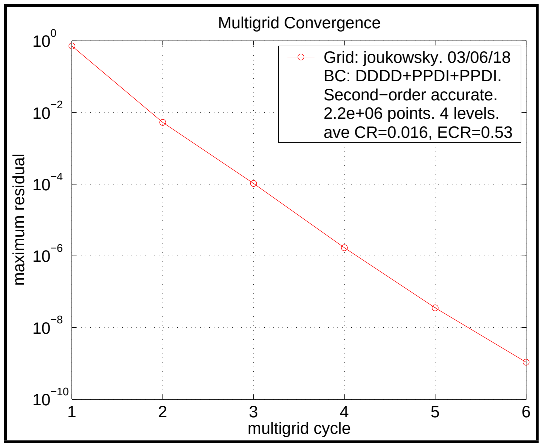


Figure 9: Top: computed solution for two airfoils in a channel. Middle: The overlapping grid on level 3 (joukowsky). Bottom : convergence history.

Solver	grid	pts	its	$\|\|res\|\|_\infty$	CPU	setup	solve	reals/pt
Ogmg	cic	1.1e6	9	2.e-8	9.7	.53	9.2	5.2
PETSc	cic	1.1e6	1268	2.e-8	934	43.	891.	26.5
Ogmg	cic4.order4	1.0e6	8	1.3e-6	11.2	1.26	9.9	9.4
PETSc	cic4.order4	1.0e6	458	1.6e-6	360.	8.	352.	55.4
Ogmg	cic4.order4	1.0e6	10	1.3e-8	14.6	1.42	13.2	9.4
PETSc	cic4.order4	1.0e6	666	1.6e-8	405.	8.	397.	55.4
Ogmg	ellipsoid	7.4e5	10	3.e-7	19.6	2.9	16.7	19.4
PETSc	ellipsoid	7.4e5	50	3.e-7	44.4	23.7	20.7	55.6
Ogmg	ellipsoid	2.0e6	10	2.e-10	38.6	4.5	34.1	10.7
PETSc	ellipsoid	2.0e6	154	1.e-9	350.	36.	314.	55.2

Table 3: A comparison of the setup and solution times for various solution algorithms. The iterative solve is bi-CG-stab.

References

[1] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *The portable extensible toolkit for scientific computation*, Tech. Rep. http://www.mcs.anl.gov/petsc/petsc.html, Argonne National Laboratory, 1999.

[2] W. D. Henshaw, *Overture: An object-oriented framework for solving PDEs in moving geometries on overlapping grids using C++*, in Proceedings of the Third Symposium on Overset Composite Grid and Solution Technology,

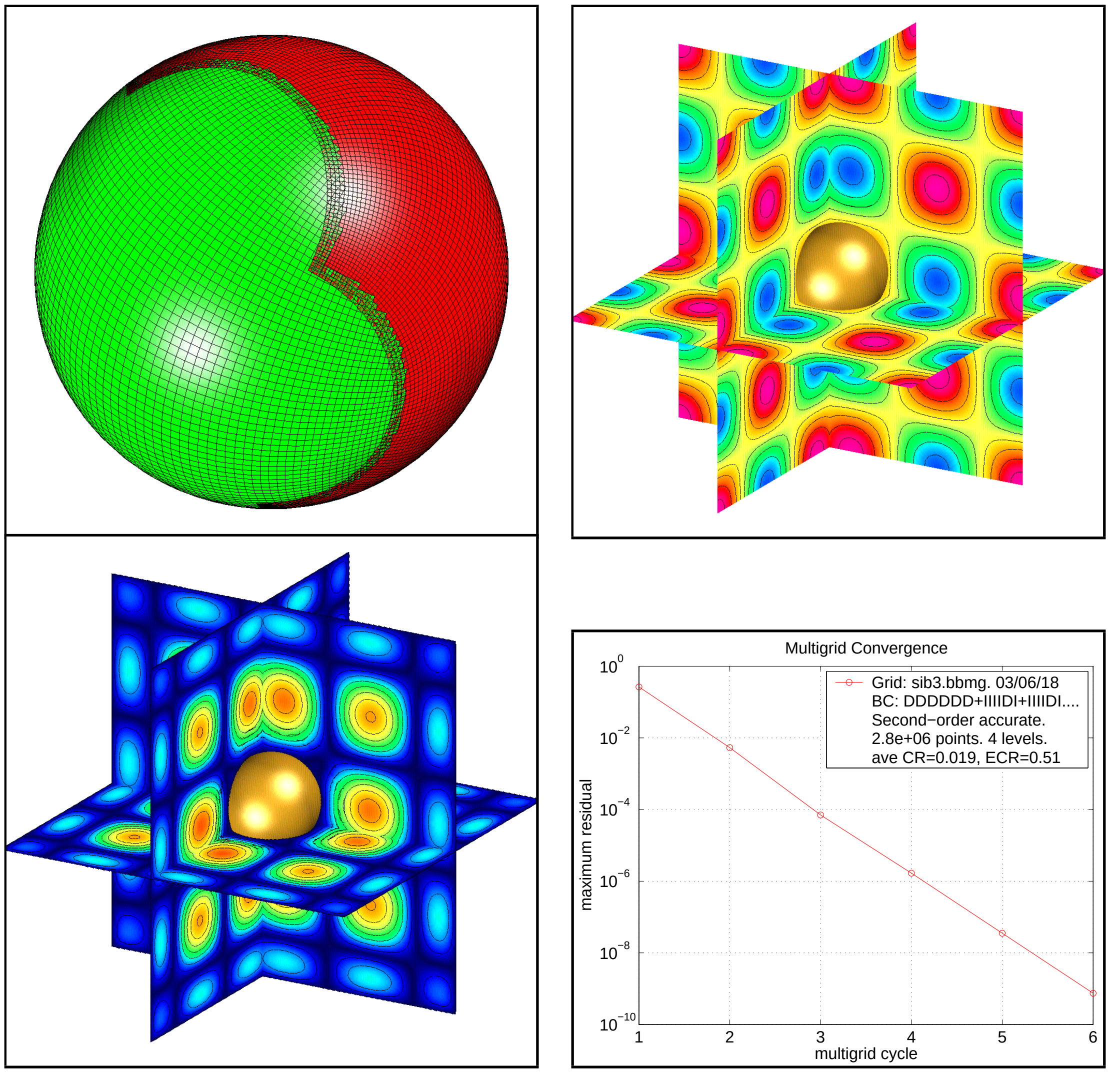


Figure 10: Top left: An overlapping grid for a sphere-in-a-box, 2.8 million grid points, (sib3.bbmg). Top right: computed solution. Bottom left: Error. Bottom right: convergence history.

problem	grid points	CR	cpu/cycle	Memory	reals/grid-point
circle in a square (2D)	1.1 million	.056	.93	48 M	5.2
two circles (2D)	6.3 million	.043	4.7	287 M	5.7
ellipsoid in a box (3D)	4.7 million	.12	8.5	442 M	11.7
two spheres (3D)	10.2 million	.085	29.2	1420 M	17.4

Table 4: Performance and memory usuage results for the predefined Laplace operator which has been optimised for Cartesian grids; 2.2 GigaHertz Pentium workstation.

1996.

[3] ——, *Oges user guide, a solver for steady state boundary value problems on overlapping grids*, Research Report UCRL-MA-132234, Lawrence Livermore National Laboratory, 1998.

[4] ——, *Other stuff for Overture, user guide, version* 1.0, Research Report UCRL-MA-134292, Lawrence Livermore National Laboratory, 1999.

[5] W. D. Henshaw and G. S. Chesshire, *Multigrid on composite meshes*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 914–923.

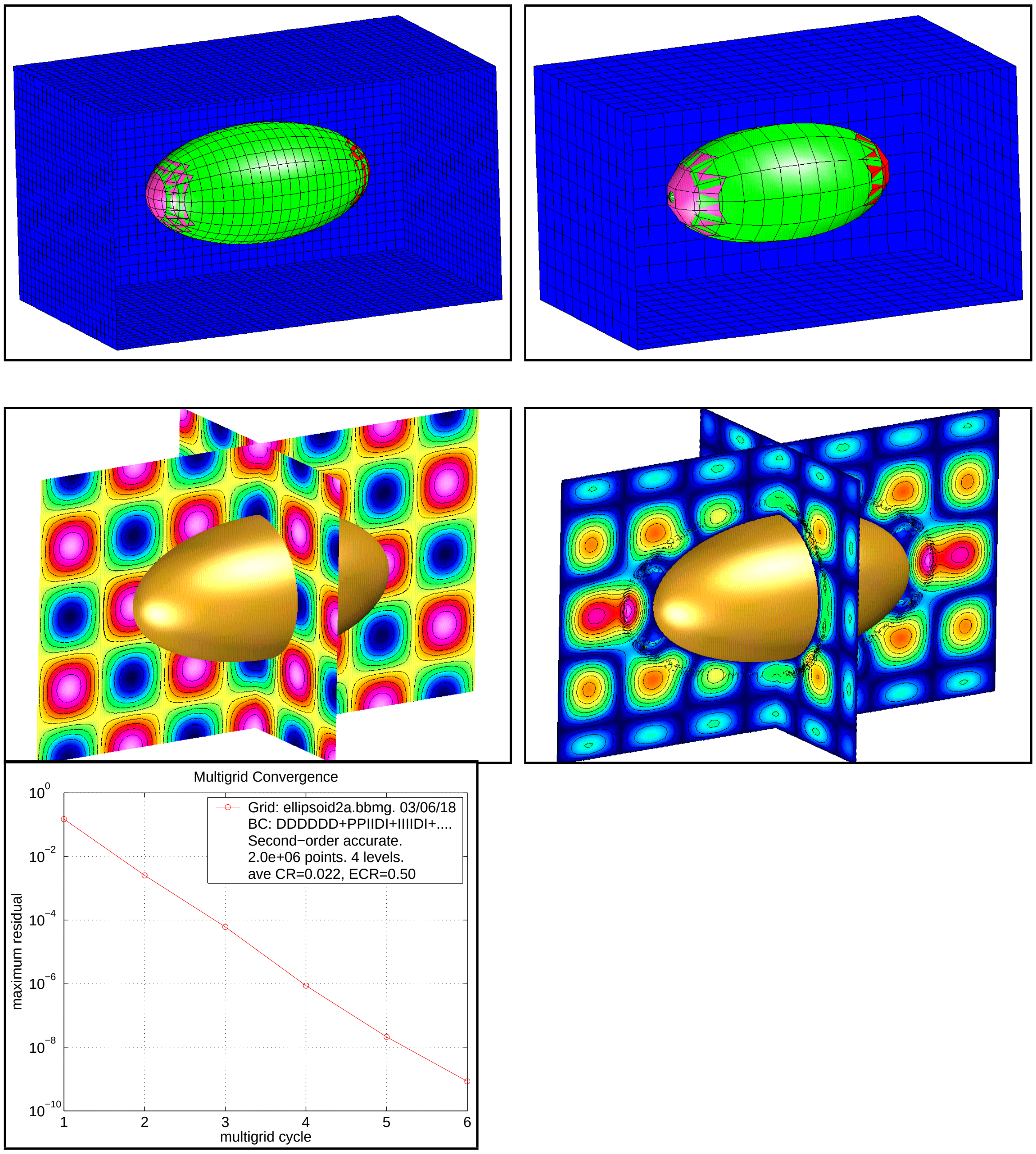


Figure 11: Top: Levels $l = 2$ and $l = 3$ for an ellipsoid in a box (ellipsoid2a.bbmg). Middle left: computed solution. Middle right: error. Bottom: convergence history.

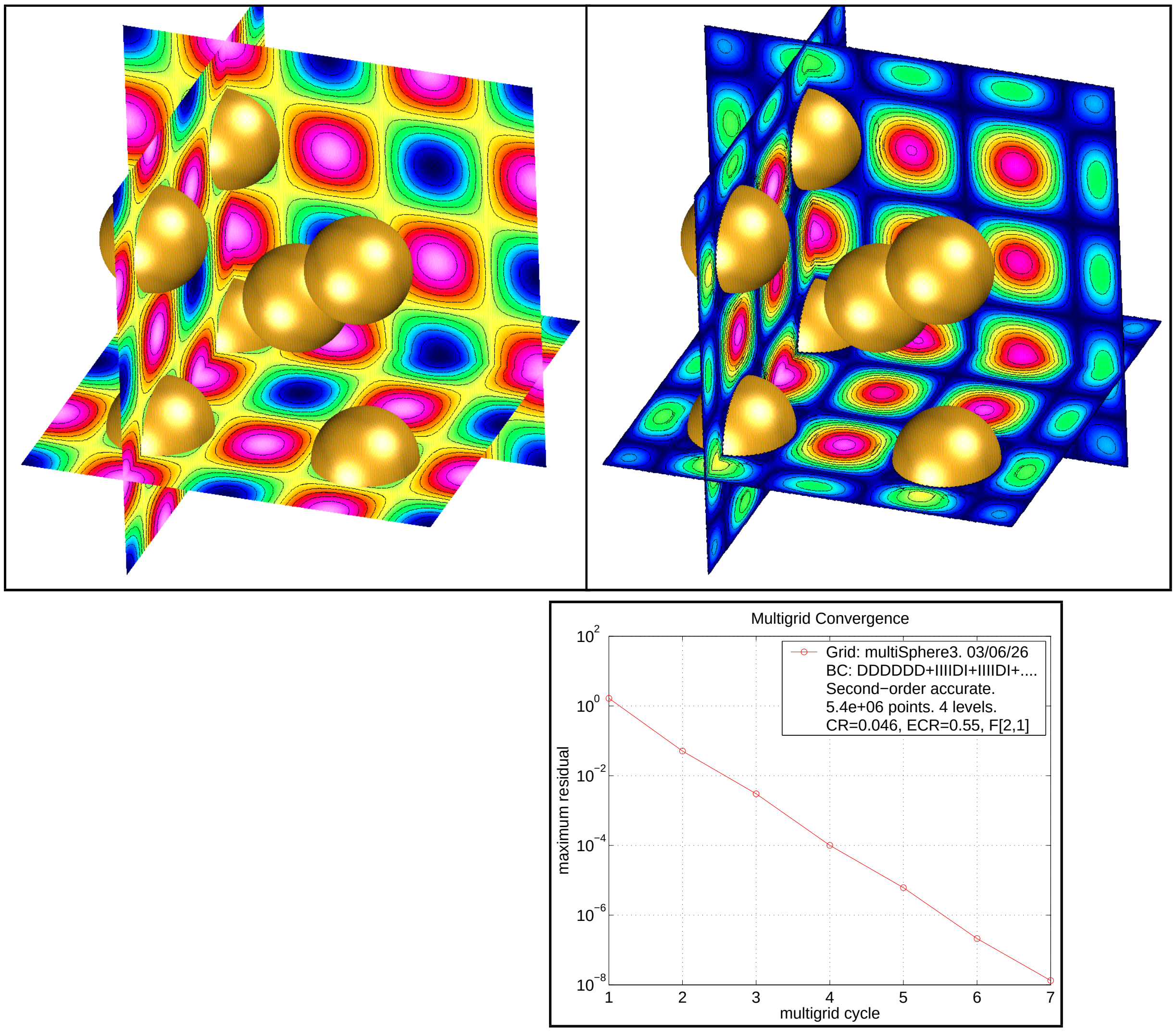


Figure 12: Top: computed solution for spheres in a box (multiSphere3). Middle right: error. Bottom: convergence history.

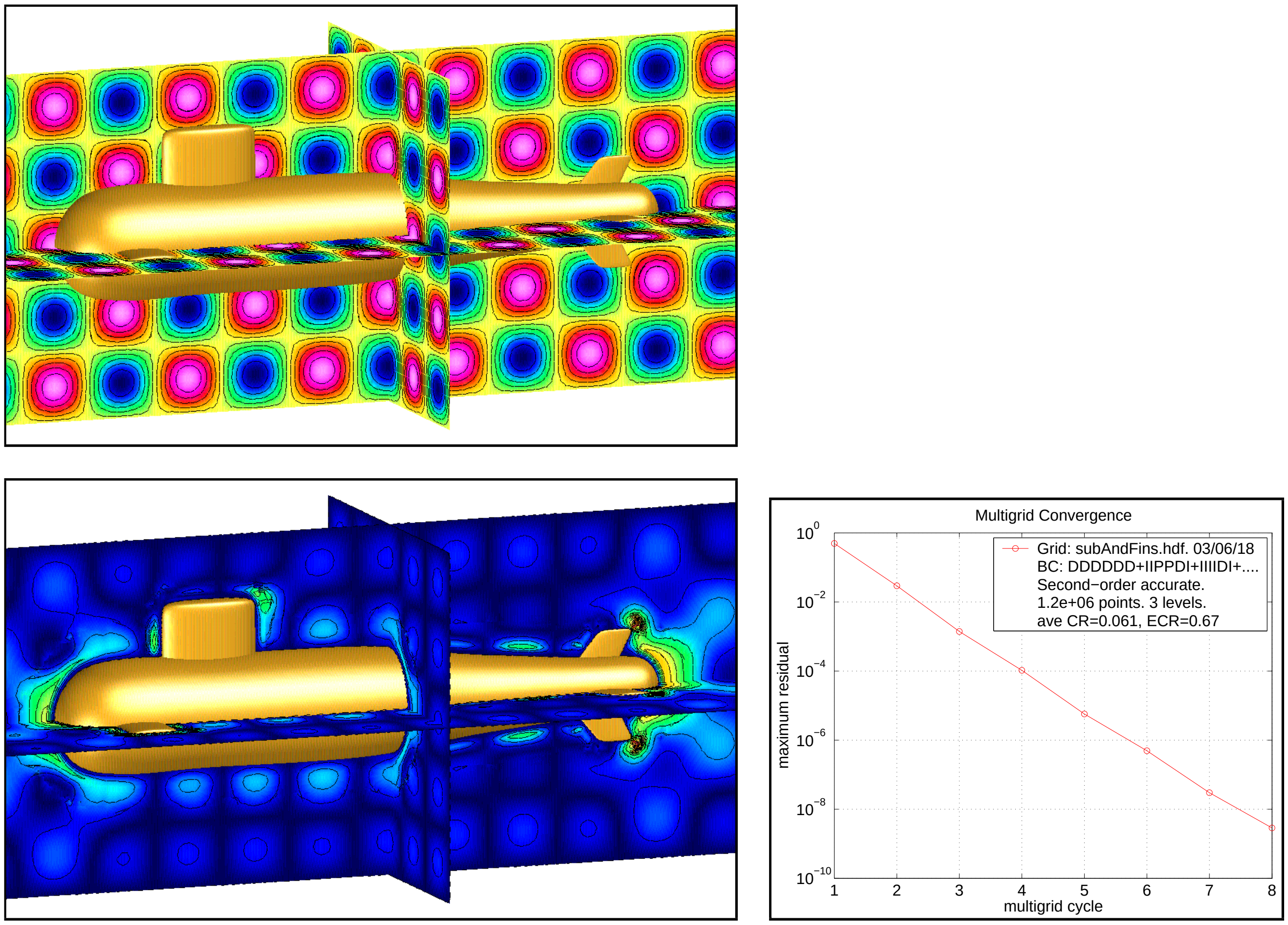


Figure 13: Top: First two levels for an sub in a box (subAndFins). Top left: computed solution. Bottom left: error. Right: convergence history.