JOURNAL OF COMPUTATIONAL PHYSICS **132,** 226–232 (1997)
ARTICLE NO. CP965466

Comparison of Second- and Fourth-Order Discretizations for Multigrid Poisson Solvers

Murli M. Gupta, Jules Kouatchou, and Jun Zhang

Department of Mathematics, The George Washington University, Washington, DC 20052

Received July 24, 1995; revised on January 30, 1996

We combine a compact high-order difference approximation with multigrid V-cycle algorithm to solve the two-dimensional Poisson equation with Dirichlet boundary conditions. This scheme, along with several different orderings of grid space and projection operators, is compared with the five-point formula to show the dramatic improvement in computed accuracy, on serial and vector machines.

1. INTRODUCTION

We consider the two-dimensional Poisson equation with Dirichlet boundary conditions,

$$\begin{aligned} -\Delta u(x,y) &= f(x,y), \quad (x,y) \in \Omega, \\ u(x,y) &= g(x,y), \quad (x,y) \in \partial\Omega. \end{aligned} \tag{1}$$

Here $\Delta = \partial^2/\partial x^2 + \partial^2/\partial x^2$ is the two-dimensional Laplace operator and Ω is a bounded convex domain. When (1) is solved by finite differences, the most commonly used approximation is the five-point formula ($\mathcal{FPF}$):

$$4u_{i,j} - [u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j}] = h^2 f_{i,j}, \tag{2}$$

where h is the uniform meshsize. The approximation (2) has a truncation error of order h^2. An approximation of order h^4 can also be used to solve (1):

$$\begin{aligned} 20u_{i,j} - 4[u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j}] \\ - [u_{i+1,j+1} + u_{i+1,j-1} + u_{i-1,j+1} + u_{i-1,j-1}] \\ = \frac{h^2}{2}[f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} + 8f_{i,j}]. \end{aligned} \tag{3}$$

This compact nine-point formula ($\mathcal{NPF}$) is generally called "Mehrstellen" and has been known for many years [8, 16]. In recent years, several authors have derived high-order finite difference approximations to solve various partial differential equations (e.g., [11, 12, 17]), all of which reduce to (3) for the case of the Poisson equation.

The multigrid method is among the most efficient iterative methods to solve linear systems arising from discretizing elliptic differential equations. It solves the error correction (coarse-grid-correction) sub-problem on the coarse grids and interpolates the error correction solution back to the fine grids. Considerable computational time is saved by doing major computational work on the coarse grids.

One iteration of a simple multigrid V-cycle consists of smoothing the error using a relaxation technique (e.g., Gauss–Seidel, Jacobi), solving an approximation to the smooth error equation on a coarse grid, interpolating the error correction to the fine grid, and finally adding the error correction into the approximation. An important aspect of the multigrid method is that the coarse grid solution can be approximated by recursively using the multigrid idea. That is, on the coarse grid, relaxation is performed to reduce high frequency errors followed by the projection of a correction equation on yet another coarser grid, and so on. Thus, the multigrid method requires a series of different problems to be solved on a hierarchy of grids with different mesh sizes. Each coarse grid provides a coarse-grid-correction to the solution on the next fine grid. A multigrid V-cycle is the process that goes from the finest grid down to the coarsest grid and back from the coarsest up to the finest. We summarize one iteration of this procedure in Fig. 1. A common variation of the V-cycle is to do two correction cycles at each level before returning to the next finer level; this is the W-cycle. However, in this paper, we restrict our attention to the multigrid V-cycle. A V(1,1)-cycle is a multigrid V-cycle algorithm which performs one relaxation at each level before projecting the residual to the coarse grid space (presmoothing) and performs one relaxation after interpolating the solution back to the fine grid space (postsmoothing). For more details on multigrid, readers are referred to [3, 7, 20] and the reference therein.

Figure 1 indicates that the solution on the coarsest grid is obtained by a direct method. Because there are few unknowns on the coarsest grid, the cost of the direct method is minimal. In practice, we carry out a few relaxation sweeps on the coarsest grid to avoid coding a direct solver.

0021-9991/97 $25.00

PROGRAM MG(f^h, u^h, h) to solve the system of linear equations $A^h u^h = f^h$.

if (h = coarsest) **then** $u \leftarrow A^{h-1} f^h$ (Use direct solver on the coarsest grid.)
else
  $u \leftarrow \text{relax}(f, u, h)$
  $r \leftarrow f - Au$
  $\bar{r} \leftarrow Pr$ (P is a projection operator from fine to coarse grids.)
  $v \leftarrow 0$
  $v \leftarrow \text{MG}(\bar{r}, v, 2h)$
  $u \leftarrow u + Iv$ (I is an interpolation operator from coarse to fine grids.)
  $u \leftarrow \text{relax}(f, u, h)$
end if

FIG. 1. One iteration of a multigrid V-cycle.

Iyengar and Goyal [15] used the fourth-order difference schemes with multigrid algorithm to solve three-dimensional Poisson equation in cylindrical coordinates. They tested V-cycle and a so-called ''sawtooth''(S-)cycle algorithm and showed that V-cycle and S-cycle algorithms achieve the same convergence rate, but S-cycle was preferred since smaller number of smoothings needs to be carried out. However, their experiments only employed two or three levels of grids and the fourth-order formula was only used in calculating the residual on the finest grid. A second-order formula was used to do the relaxations on all levels and calculate the residual on the coarse levels. In some sense, this work falls into the category described by Brandt as ''double discretization'' (see [5, 6]).

Nine-point discretization was used by de Zeeuw and van Asselt [21, 22] to develop a so-called ''Black-Box'' multigrid solver for general linear second-order elliptic partial differential equations in two dimensions. The solver employs ''sawtooth'' multigrid cycling, matrix-dependent grid transfers and incomplete line LU relaxation techniques.

On vector machines, Barkai and Brandt [2] reported some experiments on vectorized multigrid Poisson solver for the CDC CYBER 205 machine. They employed the full multigrid (FMG) method, which used the standard five-point discretization (2), and the red–black Gauss–Seidel relaxation method. The grid space was colored in a red-black (checkerboard) fashion. To exploit the vectorization, the red points are stored together, so are the black points. Two vectors were used. The relaxation, projection, and interpolation processes were modified properly to maximize the benefit of the vector machines (i.e., to form long vectors).

More information on the development of general multigrid solvers can be found in [1, 14, 18].

Nine-point discretization of the Poisson equation was analyzed extensively by Stüben and Trottenberg [19] in the context of the smoothing factor. It was shown that, for $\mathcal{NPF}$, four-color ordering of grid space and Gauss–Seidel relaxation with full-weighting (FW) projection operator has the smallest smoothing factor. However, there seems to be no reported result on the practical implementation and tests of $\mathcal{NPF}$ for the Poisson equation.

For two-dimensional Poisson equation with $\mathcal{FPF}$, the most cost-effective smoother in multigrid V-cycle is probably the Gauss–Seidel relaxation with red–black ordering of grid space (RBGS) [19, p. 85]. In practice, half-injection projection (HI) and bi-linear interpolation are commonly employed for $\mathcal{FPF}$. Few multigrid practitioners have reported results with $\mathcal{NPF}$. (Some people even mistakenly think that $\mathcal{NPF}$ converges only when a damped Jacobi relaxation, full-weighting projection and four-color ordering of grid space are used.) In this paper, Gauss–Seidel relaxation and bi-linear interpolation operator are used in all cases, but different orderings of grid space and projection operators are tested and compared. Standard coarsening (the coarse grid meshsize is double of the fine grid meshsize) is used in all cases. Our numerical experiments show that $\mathcal{NPF}$ is superior to $\mathcal{FPF}$ in both accuracy and computational efficiency. Our attention is restricted to compare $\mathcal{NPF}$ and $\mathcal{FPF}$ as relaxation methods in standard multigrid V-cycle algorithm. No effort is made to compare them with other non-standard algorithms.

This paper is organized as follows: Section 2 gives some analyses on the cost of implementing $\mathcal{NPF}$ with different options of projection operators; the cost is compared with that of $\mathcal{FPF}$. In Section 3, three test problems are given and numerical tests are done on both vector and serial machines. Some concluding remarks and suggestions on future research are given in Section 4.

2. PROGRAMMING AND COST ANALYSIS

2.1. *Relaxation and Storage Cost*

$\mathcal{FPF}$ has five floating-point operations and $\mathcal{NPF}$ has 15 if counted directly from (2) and (3). As the right-hand side of (3) is not updated at any step, we may define $F_{i,j}$ by

$$F_{i,j} = \tfrac{1}{2}[f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} + 8f_{i,j}]. \tag{4}$$

Now (3) becomes

$$\begin{aligned} 20u_{i,j} - 4[u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j}] \\ - [u_{i+1,j+1} + u_{i+1,j-1} + u_{i-1,j+1} + u_{i-1,j-1}] = h^2 F_{i,j}. \end{aligned} \tag{5}$$

The computation of $F_{i,j}$ for grid points close to the boundary requires the knowledge of $f(x, y)$ on the boundary. We assume that $f(x, y)$ is extended naturally to $\partial\Omega$.

To utilize the computational space more efficiently, practical multigrid solvers usually use a long vector to store $u_{i,j}$ and $f_{i,j}$ ($F_{i,j}$) for all the grids (on the coarse grids, $u_{i,j}$ and $f_{i,j}$ are coarse grid correction and residual, respectively).

Furthermore, $F_{i,j}$, as defined by (4), are only evaluated once on the finest grids and may be stored at the place where $\mathcal{FPF}$ stores its $f_{i,j}$. This can be done when the initialization of data is performed. Hence $\mathcal{NPF}$ does not require any more storage space than $\mathcal{FPF}$ and a relaxation sweep using $\mathcal{NPF}$ (5) has 10 floating-point operations. The codes for the two solvers are almost identical.

Our conclusion is that $\mathcal{NPF}$ relaxation could cost twice as much as $\mathcal{FPF}$ on the same grid space, but requires no more storage space. This is to say a $\mathcal{NPF}$ relaxation sweep could take roughly double CPU time.

2.2. *Grid Transfer Cost*

The cheapest projection operators are probably by injections of some kind. The residuals corresponding to the coarse grid points are injected (projected) to the coarse grid space. Only those residuals needed to be injected are calculated on the fine grid and the cost is roughly equivalent to one relaxation on the coarse grid. If the grid space is in natural (lexicographic) order, a full-injection (FI) projection operator may be used. For RBGS, a half-injection (HI) projection is commonly used. The residuals are directly transferred to the corresponding coarse grid points weighted by $\frac{1}{2}$. The factor of $\frac{1}{2}$ is motivated by the fact that the fine grid residual is zero at black fine grid points; hence the other residuals should be multiplied by $\frac{1}{2}$ to represent the correct average [2, p. 219]. This works for $\mathcal{FPF}$ only. For $\mathcal{NPF}$ with RBGS, the grid space is not completely de-coupled and the half-injection may not be accurate.

More accurate projection operators are full-weighting (FW) and half-weighting (HW). The residuals are computed on all of the fine grid points (the cost is roughly equivalent to one relaxation sweep on the fine grid) and weighted to the coarse grid points by the formulas

$$\begin{aligned}\bar{r}_{i/2,j/2} = [4r_{i,j} + 2(r_{i+1,j} + r_{i-1,j} + r_{i,j+1} + r_{i,j-1}) \\ + (r_{i+1,j+1} + r_{i+1,j-1} + r_{i-1,j+1} + r_{i-1,j-1})]/16\end{aligned} \tag{6}$$

for FW and

$$\bar{r}_{i/2,j/2} = [4r_{i,j} + (r_{i+1,j} + r_{i-1,j} + r_{i,j+1} + r_{i,j-1})]/8 \tag{7}$$

for HW, respectively. Here $r_{i,j}$ is the residual at the fine grid point (i, j), $\bar{r}_{i/2,j/2}$ is the quantity to be transferred to the corresponding coarse grid point $(i/2, j/2)$. i and j are assumed to be even numbers.

Computational costs of FW and HW are similar as in both cases, all of the residuals on the fine grid points need to be computed.

For $\mathcal{FPF}$, the general understanding is that HI is the most cost-effective for RBGS. Since the residuals at the black points are zero. We note that, from (7), HW and HI are the same in this special case. For $\mathcal{NPF}$, FW converges fastest for RBGS.

3. COMPUTATIONAL ACCURACY AND EFFICIENCY

Our $\mathcal{FPF}$ and $\mathcal{NPF}$ multigrid V-cycle programs have essentially the same structure, except for the obvious differences in relaxation computation and in choosing suitable projection operators. The programs were coded and debugged on a SUN SPARCstation 1+ using FORTRAN 77 programming language in double precision. The same code is also used on a vector machine Cray C90 (in single precision, which is equivalent to double precision on SUN) at the Pittsburgh Supercomputing Center. No extra work is done especially to exploit benefits of vectorization, all such work being left to be done by the Cray FORTRAN compiler.

We used the following three test problems on a unit square to test the performance of the multigrid solvers on both serial and vector machines and applied the multigrid V(1,1)-cycle algorithm with either $\mathcal{FPF}$ or $\mathcal{NPF}$ for different values of $N(=1/h)$ and the stopping criteria ε. The programs terminated when the absolute residual in L_2-norm is less than ε. The maximum errors reported are the overall absolute discrete errors in L_∞-norm. The solution on the coarsest grid for both $\mathcal{NPF}$ and $\mathcal{FPF}$ are obtained by performing two relaxation sweeps on the coarsest grid.

TEST PROBLEM 1.

$$f(x, y) = -x^2(1 - x^2)(2 - 12y^2) - y^2(1 - y^2)(2 - 12x^2);$$
$$g(x, y) = x^2y^2(1 - x^2)(1 - y^2).$$

TEST PROBLEM 2.

$$f(x, y) = -(x^2 + y^2)e^{xy};$$
$$g(x, y) = e^{xy}.$$

TEST PROBLEM 3.

$$f(x, y) = 52\cos(4x + 6y);$$
$$g(x, y) = \cos(4x + 6y).$$

3.1. *Performance Comparison on Cray* C90

We first compare $\mathcal{NPF}$ with (FW) and $\mathcal{FPF}$ with (HI). Test Problem 1 is solved for $\varepsilon = 10^{-10}$. Table I contains the results.

We note that for the same values of N and ε, $\mathcal{NPF}$ achieves significantly better accuracy than $\mathcal{FPF}$. The CPU costs are comparable, which is remarkably better than the *a priori* estimate of Section 2. If we seek a required accuracy, $\mathcal{NPF}$ terminates with fewer iterations and far less CPU cost by doing computations on coarser grids. For example, from Table I, $\mathcal{NPF}$ achieves overall accuracy of 10^{-6} with $N_9 = 16$ in 3.50×10^{-3} s with 10 V-cycles. But $\mathcal{FPF}$, with $N_5 = 256$ and 12 V-cycles, takes almost 290 times more CPU time. We note that $N_9 = \sqrt{N_5}$.

TABLE I

Performance Comparison on Vector Machine Cray C90

	$\mathcal{NPF}$ with RBGS and full weighting			$\mathcal{FPF}$ with RBGS and half injection		
N	Iteration	CPU(s)	Max error	Iteration	CPU(s)	Max error
8	09	1.58(−3)	1.22(−05)	09	1.34(−3)	7.64(−04)
16	10	3.50(−3)	7.96(−07)	10	2.79(−3)	1.97(−04)
32	10	7.29(−3)	4.98(−08)	11	6.00(−3)	4.91(−05)
64	10	1.68(−2)	3.11(−09)	12	1.40(−2)	1.23(−05)
128	10	4.26(−2)	1.93(−10)	12	3.42(−2)	3.07(−06)
256	10	1.32(−1)	6.45(−12)	12	1.02(−1)	7.68(−07)

Note. $\mathcal{NPF}$ with RBGS relaxation and full-weighting, compared with $\mathcal{FPF}$ with RBGS and half-injection for Test Problem 1 with $\varepsilon = 10^{-10}$.

Also from Table I, we note that the errors of $\mathcal{NPF}$ decrease by a factor of 16 and the errors of $\mathcal{FPF}$ decay by a factor of 4 when N is doubled. Thus $\mathcal{NPF}$ solutions demonstrate fourth-order convergence and $\mathcal{FPF}$ solutions demonstrate second-order convergence.

Next, instead of taking the same stopping criterion for $\mathcal{NPF}$ and $\mathcal{FPF}$, we choose $\varepsilon_9 = \sqrt{\varepsilon_5}$. The results for Test Problem 2 are given in Table II. Note that, in all cases, $\mathcal{NPF}$ costs less CPU time than $\mathcal{FPF}$, and achieves higher accuracy.

Further $\mathcal{NPF}$ relaxation, combined with different ordering patterns of the grid space and projection operators, is tested on Test Problem 3 to compare its performance and efficiency. Table III gives the V-cycle numbers and CPU times. The second and third columns contain data with RBGS and full-weighting projection. The fourth and fifth columns contain results with two-color red–black ordering of the grid space and half-injection projection. The sixth and seventh columns contain the information with four-color ordering of the grid space and full-weighting projection. In this pattern, the groups of grid points are completely de-coupled. One Gauss–Seidel relaxation sweep on the grid space is equivalent to four Jacobi sweeps, each carried out on roughly a quarter of the grid points. The eighth and ninth columns contain data with natural (lexicographic) ordering of the grid space and full-injection projection. All the variations give almost the same accuracy.

The results in Table III show that the convergence of RBGS with a HI operator deteriorates as the discretization gets finer. This is not a true multigrid performance. Contrary to expectation, the complete de-coupling of the relaxation sweep with four-color ordering of the grid space does not help. This is contrary to the theoretic analysis given by Stüben and Trottenberg [18] which shows that the four-color ordering of the grid space gives the smallest smoothing factor. It seems that the more colors are employed, the closer the Gauss–Seidel relaxation tends to the Jacobi relaxation. The natural (lexicographic) ordering results in the slowest convergence. This agrees with Barkai and Brandt's test of $\mathcal{FPF}$ [2]. In conclusion, all these modifications are inefficient, compared with RBGS with a FW projection operator.

3.2. *Performance Comparison on SUN SPARCstation* 1+

The same code and test problems in double precision have been tested on a SUN SPARCstation 1+. Table IV

TABLE II

Performance Comparison for Test Problem 2 on Vector Machine Cray C90

	$\mathcal{NPF}$ RBGS with FW and $\varepsilon = 10^{-4}$			$\mathcal{FPF}$ RBGS with HI and $\varepsilon = 10^{-8}$		
N	Iteration	CPU(s)	Max error	Iteration	CPU(s)	Max error
8	06	1.19(−3)	1.80(−06)	08	1.22(−3)	4.60(−05)
16	06	2.28(−3)	6.60(−08)	10	2.78(−3)	1.20(−05)
32	06	4.59(−3)	1.40(−07)	12	6.49(−3)	3.07(−06)
64	07	1.21(−2)	1.39(−08)	13	1.49(−2)	7.69(−07)
128	07	3.18(−2)	1.87(−08)	14	3.96(−2)	1.92(−07)
256	08	1.12(−1)	1.59(−09)	14	1.13(−1)	4.80(−08)

Note. Red–black Gauss–Seidel relaxation and full-weighting are used for $\mathcal{NPF}$ with $\varepsilon = 10^{-4}$, compared with red–black Gauss–Seidel relaxation and half-injection for $\mathcal{FPF}$ with $\varepsilon = 10^{-8}$.

TABLE III

Performance Comparison on Vector Machine Cray C90

	RBGS and FW		RBGS and HI		4-color and FW		Natural and FI	
N	Iter.	CPU(s)	Iter.	CPU(s)	Iter.	CPU(s)	Iter.	CPU(s)
8	09	1.53(−3)	09	1.50(−3)	09	2.25(−3)	10	2.11(−3)
16	09	3.20(−3)	12	3.86(−3)	11	5.85(−3)	13	8.19(−3)
32	10	7.31(−3)	15	9.88(−3)	11	1.29(−2)	14	3.12(−2)
64	10	1.66(−2)	18	2.70(−2)	12	3.13(−2)	15	1.27(−1)
128	10	4.27(−2)	22	8.28(−2)	12	7.49(−2)	16	5.28(−1)
256	11	1.40(−1)	26	2.92(−1)	13	2.25(−1)	17	2.22(+0)

Note. Different orderings of grid space and different projection operators using with $\mathcal{NPF}$ for Test Problem 3 with $\varepsilon = 10^{-8}$. RBGS stands for red–black Gauss–Seidel relaxation, FW for full-weighting, HI for half-injection, FI for full-injection; 4-color for four-color ordering of grid space, Natural for natural (lexicographic) ordering of grid space.

contains the results for Test Problem 1. The numbers of V-cycles and the maximum errors do not vary very much. Again, $\mathcal{NPF}$ relaxation with full-weighting achieves great accuracy improvement over $\mathcal{RPF}$ with half-injection, although the CPU time is increased by about 50%. To achieve the overall accuracy of 10^{-6}, we need $N = 256$ and 202 s with $\mathcal{RPF}$. On the other hand, we need only $N = 16$ and 0.81 s of CPU time with $\mathcal{NPF}$. In this case, $\mathcal{NPF}$ is almost 250 times faster than $\mathcal{RPF}$ to obtain the same accuracy.

Different orderings of grid space and projection operators, combined with $\mathcal{NPF}$ relaxation, are also tested for Test Problem 2. Table V contains the results for RBGS with full-weighting (columns 2 and 3), RBGS with half-injection (column 4 and 5), and natural ordering GS with full-injection (columns 6 and 7). Again, RBGS with full-weighting is the most efficient; other modifications deteriorate the convergence rate.

Finally, we choose $\varepsilon_9 = \sqrt{\varepsilon_5}$ for $\mathcal{NPF}$ and $\mathcal{RPF}$, respectively. The results for Test Problem 3 are given in Table VI. Note that $\mathcal{NPF}$ costs almost the same as $\mathcal{RPF}$, but it achieves much higher accuracy.

All of our tests show that $\mathcal{NPF}$ is much more accurate and efficient than $\mathcal{RPF}$ on serial machines if a FW projection operator is employed.

We conclude this section with some remarks.

Remark 1. If both (2) and (5) are evaluated on the same grid N with the same stopping criterion ε, we expect $\mathcal{NPF}$ to give much higher accuracy than $\mathcal{RPF}$ with comparable CPU cost.

Remark 2. If we want to solve (2) and (5) on the same grid N and expect comparable CPU cost, we may be able to set a lower stopping criterion, say ε_9 for (5), and a higher one, say ε_5 for (2). We may use $\varepsilon_9 = \sqrt{\varepsilon_5}$. In this case, $\mathcal{NPF}$ will give better accuracy.

4. CONCLUSIONS AND FUTURE RESEARCH

We have shown that the nine-point discretization formula, combined with full-weighting projection operator, is much more accurate than the five-point discretization formula, on both vector and serial machines. The interesting conclusions from our tests are that $\mathcal{NPF}$ is even more attractive on vector machines as the higher accuracy is achieved with almost the same computational cost as $\mathcal{RPF}$.

TABLE IV

Performance Comparison between $\mathcal{RPF}$ and $\mathcal{NPF}$ for Test Problem 1 on SUN SPARCStation

	$\mathcal{NPF}$ with RBGS and full weighting			$\mathcal{RPF}$ with RBGS and half injection		
N	Iteration	CPU(s)	Max error	Iteration	CPU(s)	Max error
8	09	1.60(−1)	1.22(−05)	09	9.00(−2)	7.64(−04)
16	10	8.10(−1)	7.96(−07)	10	4.30(−1)	1.97(−04)
32	10	3.78(+0)	4.98(−08)	11	2.19(+0)	4.91(−05)
64	10	1.67(+1)	3.11(−09)	12	1.07(+1)	1.23(−05)
128	10	7.20(+1)	1.94(−10)	12	4.53(+1)	3.07(−06)
256	10	2.92(+2)	1.22(−11)	13	2.02(+2)	7.68(−07)

Note. RBGS relaxation with full-weighting projection is used for $\mathcal{NPF}$, RBGS relaxation with half-injection is used for $\mathcal{RPF}$; $\varepsilon = 10^{-10}$.

TABLE V

Performance Comparison on SUN SPARCStation among Different Orderings of Grids Using $\mathcal{NPF}$ for Test Problem 2 with $\varepsilon = 10^{-8}$

	2-color and full weighting		2-color and half injection		Natural and full injection	
N	Iteration	CPU(s)	Iteration	CPU(s)	Iteration	CPU(s)
8	09	1.70(−1)	08	1.10(−1)	10	1.40(−1)
16	09	7.20(−1)	12	7.60(−1)	13	8.10(−1)
32	10	3.77(+0)	15	4.40(+0)	15	4.38(+0)
64	10	1.68(+1)	19	2.47(+1)	16	2.08(+1)
128	10	7.09(+1)	23	1.27(+2)	17	9.37(+1)
256	11	3.21(+2)	27	6.15(+2)	18	4.10(+2)

Note. Gauss–Seidel relaxation is used in all cases.

It will be interesting to test fully vectorized or parallelized high-order discretization schemes on vector and parallel machines, as Barkai and Brandt did for the five-point scheme [4, 2].

Another promising direction is to combine the fourth-order and second-order schemes in a multigrid algorithm to exploit the full efficiency of these two schemes and to design more cost-effective schemes. This has been discussed in the context of double discretization, in which the high-order formula is used to compute the residual. We think using high-order formula for coarse grid relaxation may be more rewarding.

Although our results may not readily convince multigrid practitioners to move to the high-order formula immediately, it lays down the foundation for further research in this direction. We believe that high-order compact formulas are promising in multigrid. Especially in practical applications such as solving convection-diffusion equations, a multigrid algorithm with $\mathcal{FPF}$ is often problematic because $\mathcal{FPF}$ becomes divergent as the mesh size gets coarse. Golub and Tuminaro [9] used a cyclic reduction to precondition $\mathcal{FPF}$; the resulting reduced problem is a (non-compact) nine-point formula which is solved by multigrid. However, there exist some compact nine-point formulas similar to (3) for the convection–diffusion equation [10, 11]. Our initial computations show that these compact nine-point formulas with multigrid are convergent for any convection–diffusion equations. No preconditioner is needed. Encouraging results in this direction have been obtained and will be reported separately [13].

REFERENCES

1. J. C. Adams, *Appl. Math. Comput.* **34**, 113 (1989).
2. D. Barkai and A. Brandt, *Appl. Math. Comput.* **13**, 215 (1983).
3. A. Brandt, *Math. Comput.* **31**, 333 (1977).
4. A. Brandt, "Multigrid Solvers on Parallel Computers," in *Elliptic Problem Solvers*, edited by M. H. Schultz (Academic Press, New York, (1981), p. 39.
5. A. Brandt, *Multigrid Techniques: 1984 Guide with Applications to*

TABLE VI

Performance Comparison on SUN SPARCStation for Test Problem 3

	$\mathcal{NPF}$ RBGS with FW and $\varepsilon = 10^{-4}$			$\mathcal{FPF}$ with HI and $\varepsilon = 10^{-8}$		
N	Iteration	CPU(s)	Max error	Iteration	CPU(s)	Max error
8	05	9.00(−2)	4.35(−05)	08	8.00(−2)	4.59(−01)
16	06	5.00(−1)	1.56(−06)	10	4.30(−1)	1.14(−02)
32	06	2.31(+0)	1.33(−07)	11	2.20(+0)	2.83(−03)
64	07	1.18(+1)	9.29(−09)	12	1.07(+1)	7.08(−04)
128	07	4.99(+1)	7.23(−09)	13	4.90(+1)	1.77(−04)
256	07	2.05(+2)	9.30(−09)	14	2.17(+2)	4.42(−05)

Note. RBGS relaxation and full-weighting are used for $\mathcal{NPF}$ with $\varepsilon = 10^{-4}$, compared with RBGS relaxation and half-injection for $\mathcal{FPF}$ with $\varepsilon = 10^{-8}$.

Fluid Dynamics, monograph, GMD-Studien, Nr. 85, Postfach 1240, Schloss Birlinghoven, D-5205 St. Augustin 1, 1984.

6. A. Brandt, "Multilevel Computations: Review and Recent Development," in *Multigrid Methods: Theory, Applications and Supercomputing,* edited by S. F. McCormick, Lect. Notes in Pure and Appl. Math., Vol. **110** (Dekker, New York, 1988), p. 35.
7. W. L. Briggs, *A Multigrid Tutorial* (SIAM, Philadelphia, 1987).
8. L. Collatz, *The Numerical Treatment of Differential Equations* (Springer-Verlag, Berlin, 1960).
9. G. Golub and R. Tuminaro, Manuscript NA-92-14, Numerical Analysis Project, Computer Science Dept., Stanford University, 1992 (unpublished).
10. M. M. Gupta, R. Manohar, and J. W. Stephenson, "A Fourth Order, Cost Effective and Stable Finite Difference Scheme for the Convection–Diffusion Equation," in *Numerical Properties & Methodologies in Heat Transfer*, *Proceedings*, *Second National Symposium* (Hemisphere, Washington, DC, 1983), p. 201.
11. M. M. Gupta, R. P. Manohar, and J. W. Stephenson, *Int. J. Numer. Methods Fluids* **4**, 641 (1984).
12. M. M. Gupta, *J. Comput. Phys.* **55**, 166 (1984).
13. M. M. Gupta, J. Kouatchou, and J. Zhang, *J. Comput. Phys.* **132,** 123 (1977).
14. P. W. Hemker, R. Kettler, P. Wesseling, and P. M. de Zeeuw, *Appl. Math. Comput.* **13**, 311 (1983).
15. R. K. Iyengar and A. Goyal, *J. Comput. Appl. Math.* **33**, 163 (1990).
16. L. V. Kantorovich and V. I. Krylov, *Approximate Methods of Higher Analysis* (Wiley, New York, 1964).
17. R. E. Lynch and J. R. Rice, "The Hodie Method and Its Performance for Solving Elliptic Partial Differential Equations," in *Recent Advances in Numerical Analysis*, edited by C. De Boor and G. H. Golub (Academic Press, New York, 1978), p. 143.
18. K. Stüben, U. Trottenberg, and K. Witsch, "Software Development Based on Multigrid Techniques, PDE Software: Modules, Interfaces and Systems, in *Proceedings, IFIPWG2.5 Working Conference*, edited by B. Engquist and T. Smedsaas (North-Holland, Amsterdam, 1984).
19. K. Stüben and U. Trottenberg, *Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications,* monograph, GMD-Studien Nr. 96, Postfach 1240, Schloss Birlinghoven, D-5205 St. Augustin 1, 1984.
20. P. Wesseling, *An Introduction to Multigrid Methods*, Pure and Appl. Math. (Wiley, Chichester, 1992).
21. P. M. de Zeeuw and E. J. van Asselt, *SIAM J. Sci. Stat. Comput.* **6**, 492 (1985).
22. P. M. de Zeeuw, *J. Comput. Appl. Math.* **33**, 1 (1990).