# Schwarz Splitting Using ELLPACK

*Bennett W. Cleveland and Calvin J. Ribbens*

TR 88-2

# Schwarz Splitting Using ELLPACK

Bennett W. Cleveland[*] Calvin J. Ribbens[†]

February 3, 1988

**Abstract**

This report describes a high-level implementation of *Schwarz splitting*, an approach to solving partial differential equations which seems particularly attractive from the point of view of parallelism. We discuss the basic scheme of Schwarz splitting and an implementation using ELLPACK. We consider two numerical examples in detail. The appendices contain a listing of an ELLPACK program which implements Schwarz splitting, and a complete set of data from the two examples. The examples were done on a sequential machine, but they provide some clues as to the potential for parallelism exhibited by the Schwarz technique.

## 1 Introduction

The original motivation for the Schwarz alternating procedure, or Schwarz *splitting*, was to prove the existence of a solution to Laplace's equation with Dirichlet boundary conditions on the union of two overlapping regions (see Kantorovich and Krylov [4]). Miller [7] developed a numerical analog to this procedure for solving elliptic partial differential equations (PDEs). More recently, Schwarz splitting has attracted attention as a way to exploit parallelism in the numerical solution of PDEs. Rodrigue and Simon [12], [13] and Rodrigue [11] consider the convergence behavior of the Schwarz iteration. Meier [6] looks at improving the iteration by a SOR-like strategy. Potentials for parallelism are the main focus of Rice and Marinescu [10] and Tang [14]. Schwarz splitting must be distinguished from *domain decomposition*, another active area of interest. Domain decomposition methods also divide the domain into subdomains to enhance parallelism. There is no overlapping of subdomains in domain decomposition, however. Instead a capacitance matrix approach is used to solve for the unknowns on the interfaces between subdomains. Preconditioned conjugate gradient schemes have drawn much interest in connection with domain decomposition. References include Bjorstad and Widlund [1], Chan and Resasco [2], and Keyes and Gropp [5].

The basic idea of the Schwarz splitting iteration is simple. The domain is split into several overlapping subdomains. The PDE is repeatedly solved on each of the subdomains independently (thus the potential for parallelism). Boundary conditions for subdomain boundaries which are inside the original domain are supplied by using the approximate solution from the previous iteration. The iteration continues until some stopping criterion is satisfied (typically the relative change in the solution is less than some tolerance). Rodrigue [11] characterizes this process as an "inner" and an "outer" iteration. The outer iteration is the Schwarz iteration over the subdomains. The inner iteration refers to the method used to solve the PDEs on the individual subdomains. It is

[*]Department of Mathematics, Purdue University, West Lafayette, IN, 47907.

[†]Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, 24061.

```
1.  define subdomains and grids.
2.  initialize unknown $U^{(0)} = 0$.
3.  for $i = 1, 2, \ldots,$ until satisfied do
4.      for $k = 1, 2, \ldots,$ *ndomains* do
5.          solve for new unknown $U^{(i)}$ on subdomain $k$.
```

Figure 1: Basic Schwarz splitting algorithm.

not required that an iterative method be used to solve the subproblems, of course. In fact, one of our interests in this work is to investigate various choices for solving the subproblems, and also to compare methods for speeding up the outer iteration. The algorithm in Figure 1 is a simplification of the basic steps. It is easy to see that the interesting questions include: 1) how to speed up the outer iteration; 2) how to recognize convergence of the outer iteration; 3) how to solve for the unknowns at the innermost step.

Our purpose in this work is to explore the properties and potentials for parallelism of Schwarz splitting. As in any preliminary investigation, we find that more questions are raised than are answered at times. In the next section we describe a high level implementation of Schwarz splitting and report on two experiments using this implementation. We also discuss several miscellaneous issues that pertain to the research. In the final section we discuss future directions for research. The appendices contain a program listing and a complete set of data from our experiments.

## 2 Present Work

In this section we report on experiments carried out during the summer of 1987. We first describe an implementation of Schwarz splitting based on the ELLPACK system (see Rice and Boisvert [8]). We then report extensively on two examples which illustrate some of the properties and potential of Schwarz splitting algorithms. Finally, we discuss a few other issues which pertain to this research.

### 2.1 Initial ELLPACK implementation of Schwarz splitting

We first investigate the feasibility of implementing Schwarz splitting using ELLPACK. ELLPACK is a powerful system for numerically solving elliptic PDEs. It includes a very high-level problem-statement language and a library of over fifty precompiled problem solving modules. Several reasons for considering this approach are evident:

1. We take advantage of good implementations of well-known numerical methods.
2. We implement the Schwarz scheme in a high-level way. This should be easier and quicker to do than a complete low-level FORTRAN implementation of Schwarz splitting. It is also easier to understand and maintain, and it may make identifying parallelism easier as well.
3. We get rough estimates of potential parallel speed-ups, without implementing on a particular parallel architecture.
4. We get a convenient framework for investigating properties of Schwarz splitting; it is easy to vary parameters and splitting strategies, as well as discretization, indexing, and solution methods.

5. We would like to estimate the added cost of the high-level ELLPACK approach. This would require implementing a low-level FORTRAN version of Schwarz splitting for purposes of comparison.

Since the ELLPACK language is FORTRAN based, it is possible to do rather complicated things in an ELLPACK program. One simply includes the necessary FORTRAN code to control the other ELLPACK problem-solving steps, or in fact, to do anything you would like. The simplest ELLPACK program solves a single two-dimensional, linear, elliptic problem. However, by including extra FORTRAN code in the ELLPACK program, and providing additional FORTRAN subprograms, ELLPACK can be used to solve time-dependent problems, non-linear problems, and systems of elliptic equations (see Rice and Boisvert [8] for examples). The common theme in these more complicated ELLPACK applications is that the computation is arranged as an iterative process, in which the kernel is the solution of a single linear, elliptic problem. Obviously, Schwarz splitting is exactly this kind of scheme. In each Schwarz iteration we solve a single elliptic problem on each of the subdomains. Of course, in an ideal parallel implementation each of these solves would occur simultaneously, and the only iteration would be the outer one. In a sequential environment we are forced to solve on each subdomain in turn. Our initial ELLPACK implementation is on a sequential computer, the Ridge 32.

Our ELLPACK implementation is presented in considerable detail in the program in Appendix A. The in-line comments help to explain some of the details. Notice that the user is allowed to choose the number of subdomains in each direction and the relative overlap between subdomains. A few aspects of the implementation bear emphasis:

1. In order to switch subdomains, it is necessary to reset all of the variables in the ELLPACK *discrete domain interface*. These are the variables which define the domain and grid. If they are set to the correct values, any ELLPACK module invoked will treat the current subdomain as if it were the entire domain of interest. Our strategy is to define each of the subdomains and grids initially, save the information in private common blocks, and copy it into the official ELLPACK interface as needed. The extra cost of this copying is small compared to the cost of an entire computation, since it varies linearly with the size of the grid. In Example 1 below, for Method 1, a $3 \times 3$ subdomain partition, and a $9 \times 9$ grid on each subdomain, the time taken to do the copying is less than 1% of the total time. In a multiprocessor implementation, if each processor has local memory, the copying may not even be necessary.

2. The boundary conditions defined in the BOUNDARY segment are given in terms of functions GEAST, GWEST, GNORTH, and GSOUTH. If the specified point is on the boundary of the original domain, these functions return the value of the original boundary condition there. If the specified point is inside the original domain, these functions return the value of the current approximate solution $U$.

3. We store a table of unknowns for the solution on each subdomain (these are function values or coefficients of basis functions, in general). If a high order discretization method is used, spline interpolation is necessary, so we save the spline coefficients in this case. In order to evaluate the solution at a point where two or more subdomains overlap, we take the average of the unknowns defined on each of the subdomains involved.

4. Another major overhead of our implementation is unnecessary repetition of the discretization step. As the outer iteration proceeds, the discretization matrix for each subdomain does not

change. The changing values of the boundary conditions only affect the right side of the linear system. It is possible to build and save the discretization matrix exactly once for each subdomain. This would require very large storage arrays indeed. Hence, at present we simply recompute the discretization matrix at each iteration. This extra cost can be significant, especially for PDEs with variable (and expensive to evaluate) coefficients. For fine grids however, the cost of discretization tends to be small compared with the cost of solving the linear systems. For Example 1 below, with Method 2, $3 \times 3$ subdomains, and a $9 \times 9$ grid on each subdomain, 68% of the overall time is spent building the linear systems; but with a $65 \times 65$ grid, only 34% of the time is used to build the linear systems. It would clearly be worth the effort to avoid these redundant discretizations. Another possible improvement would be to factor the linear system once and for all with a direct method, and then only do forward and back substitutions for each new right hand side. However, the discussion in Section 2.4.1 below suggests that this may not be the best approach either, especially for fine grids. Finally, we note that in a parallel implementation it may be possible for each processor to keep track of its own discretization matrix.

Our ELLPACK framework for Schwarz splitting is useful but far from perfect. It does allow quick implementation and easy experimentation. A more efficient scheme within ELLPACK would probably be to define a Schwarz TRIPLE module. This would make it easier to avoid some of the unnecessary overhead mentioned above. In terms of a parallel implementation of Schwarz splitting in ELLPACK, the task of designing a parallel version of ELLPACK itself may have to be considered first, and that will be a formidable task. It is likely that extensions to the language will be needed. If a parallel ELLPACK is built however, a parallelization of Schwarz splitting would be reasonable.

## 2.2 Example 1

We first consider a test problem with variable coefficients, Problem 1 from Rice, et al. [9]:

$$(e^{xy}u_x)_x + (e^{-xy}u_y)_y - \frac{1}{1+x+y}u = f(x,y),$$

where $f$ is chosen so that the true solution is

$$u(x,y) = 0.75e^{xy}\sin(\pi x)\sin(\pi y).$$

We impose Dirichlet boundary conditions on the unit square. We use two different variations of Schwarz splitting to solve this problem. In both cases the subdomain problems are solved using ELLPACK modules 5 POINT STAR (discretization), RED BLACK (indexing), and REDUCED SYSTEM CG (solution). We also use ELLPACK module SET UNKOWNS FOR 5 POINT STAR to initialize the vector of unknowns for the reduced system iteration. We use the solution from the previous Schwarz iteration to initialize the unknowns for each reduced system iteration. The only difference between the two methods for Example 1 is in the choice of Schwarz "outer" iteration scheme. Method 1 computes the new solution $U^{(i)}$ at Schwarz iteration $i$ as

$$U^{(i)} = \mathcal{F}(L, f, U^{(i-1)}),$$

where $\mathcal{F}$ represents the subdomain solution procedure just described, and $L$ is the linear elliptic operator. We term this a "jacobi" outer iteration. Method 2 on the other hand, computes $U^{(i)}$ by

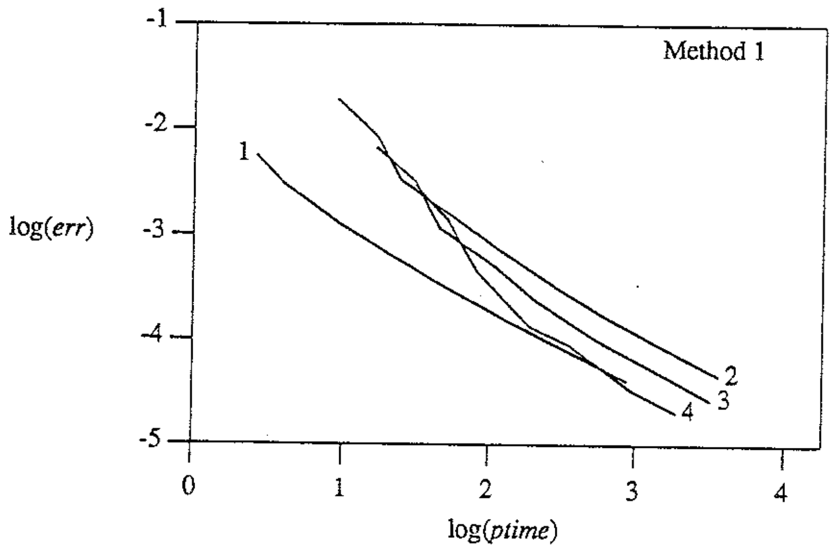
$$U^{(i)} = (1-\omega)U^{(i-1)} + \omega\mathcal{F}(L, f, U^{(i-1)}).$$

For Method 2 we color the subdomains in a typical red-black fashion (see Hageman and Young [3] for example). Subdomain solutions are computed on all the red subdomains first, using the solution from the previous Schwarz iteration. Then the black subdomain problems are solved, this time using the new red solutions just calculated. A scheme such as this is necessary if one wants to do an sor-like outer iteration, while still preserving as much potential parallelism as possible. Even so, the degree of parallelism is cut in half, since only half the subdomain solves can be done in parallel. For obvious reasons, the approach of Method 2 may be termed "red-black sor" outer iteration.

The data for Example 1 (reported in detail in Appendix B) is based on runs with 2, 3 and 4 subdomains in each direction (i.e. 4, 9, and 16 subdomains in all). The overlap parameter is taken as 0.25 in all cases. This means that 25% of a subdomain is overlapped by a neighboring subdomain on each side (in one dimension). For example, with $2 \times 2$ subdomains in the unit square, the internal subdomain boundaries are horizontal and vertical lines at 3/7 and 4/7. We varied the number of grid lines on each subdomain, and recorded error estimates, the number of Schwarz iterations, and time for each case.

Consider first the effect of increasing the number of subdomains. The plots in Figure 2 show the effect on each method as we increase the number of subdomains in each direction from 1 (no splitting) to 2, 3, and 4. Each plot shows log(*error*) vs. log(*ptime*). We estimate *error* by taking the maximum relative error over a $40 \times 40$ grid on each subdomain; *ptime* (parallel time) is the total time divided by the number of subdomain solves that could be done in parallel. The parallel time is thus a very rough estimate of the time the algorithm might take on a parallel machine. For the jacobi outer iteration (Method 1) we divide sequential time by the number of subdomains; for red-black sor outer iteration (Method 2) we divide sequential time by half the number of subdomains. Figure 2 suggests that as the number of subdomains increases, the Schwarz approach will win over the no-splitting case for Method 1, although the evidence is certainly not overwhelming. For Method 2 however, the results seem even less promising for Schwarz splitting. Problems with Method 2 (discussed below) make the performance of the splitting runs less than expected. It does appear that increasing the number of subdomains does help, however.

Next, we fix the number of subdomains, and compare the performance of the two methods. The series of plots in Figure 3 seems to indicate that as the number of pieces increases, Method 1 (jacobi outer iteration) is better than Method 2 (red-black sor outer iteration). This is not at all what we had expected, especialy since the number of Schwarz iterations for a given grid size is significantly reduced by using Method 2—seemingly enough to compensate for the loss in potential parallelism due to red-black coloring. However, we notice that for a given grid size, Method 1 is more accurate than Method 2. This suggests that something besides discretization error is affecting the error estimates, especially for Method 2. We find, in fact, that this is the case. It appears that we are actually doing too many Schwarz iterations for both Methods 1 and 2, in the sense that the best solution is achieved several iterations before the convergence criterion is satisfied. The solution actually gets worse during these extra iterations. Table 1 illustrates this situation for the $3 \times 3$ subdomain case. Notice that both methods would have achieved better accuracy if the Schwarz iteration had been stopped earlier. We conjecture that the red-black sor outer iteration (Method 2) suffers more from this problem simply because of its speed of convergence. Not only does it converge quicker, but if too many iterations are done, the answer begins too diverge more quickly also. We are not sure how to remedy this problem at present. We had hoped that our choice of stopping criteria *eps* would not unfairly penalize any one method. It appears that this is very difficult to do. One obvious idea, for experimental purposes, would be to set $eps = ||e||$, rather than $0.1||e||$ (see the discussion in Section 2.4.2). We are not at all confident that this is a



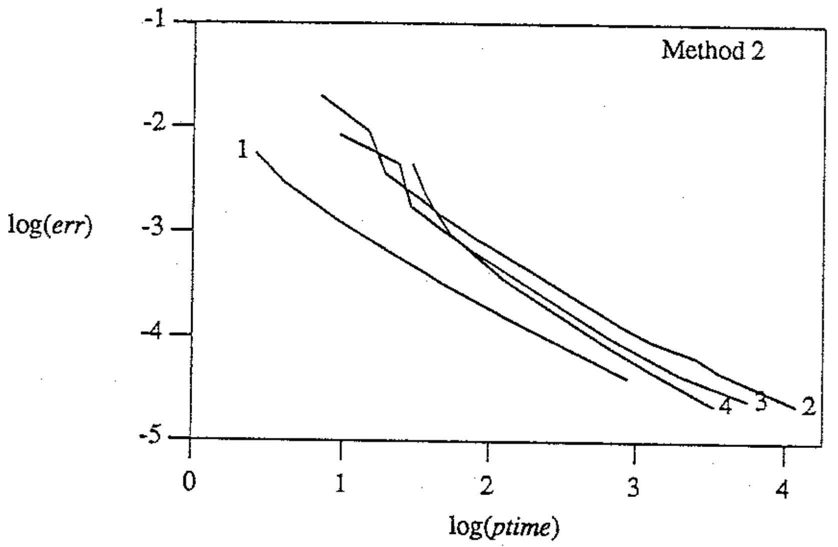


Figure 2: Log-log plot of error vs. time for the two methods on Example 1. The curves are labeled by the number of subdomains in each direction.

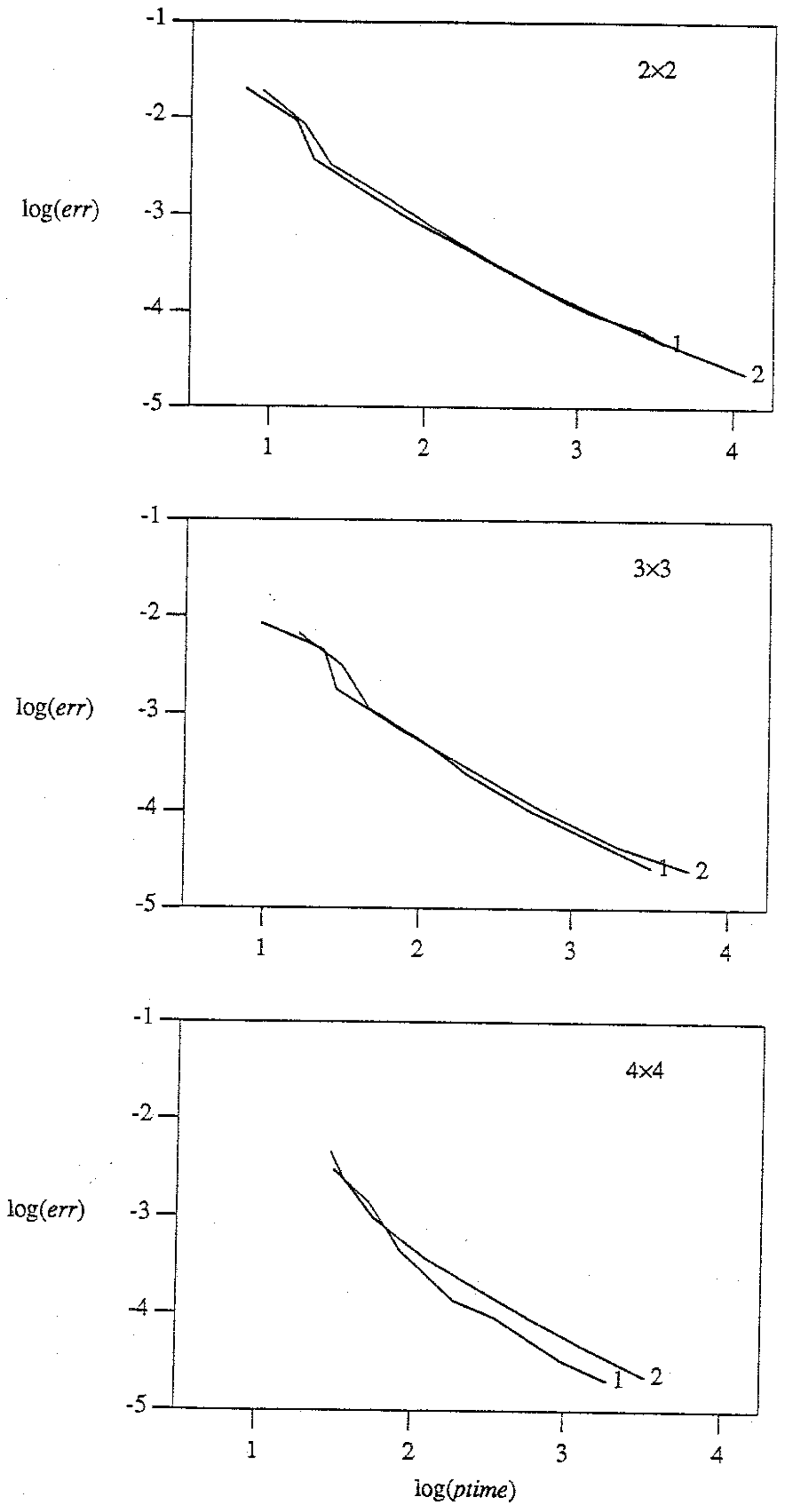


Figure 3: Log-log plot of error vs. time for 2, 3, and 4 subdomains in each direction. The curves are labeled by method number.

Table 1: Number of Schwarz iterations and error for both methods on Example 1, with $3 \times 3$ subdomains. The best iterate and the final iterate are listed for each grid size (the number of grid lines in each direction on one subdomain).

| | Method 1 | | | | Method 2 | | | |
|---|---|---|---|---|---|---|---|---|
| | Best | | Final | | Best | | Final | |
| Grid | Iteration | Error | Iteration | Error | Iteration | Error | Iteration | Error |
| 5 | 32 | 6.4e-3 | 38 | 6.7e-3 | 10 | 8.5e-3 | 11 | 8.6e-3 |
| 7 | 39 | 2.0e-3 | 46 | 3.2e-3 | 11 | 2.3e-3 | 18 | 4.5e-3 |
| 9 | 43 | 8.2e-4 | 47 | 1.2e-3 | 9 | 1.3e-3 | 15 | 1.8e-3 |
| 13 | 50 | 3.6e-4 | 54 | 5.1e-4 | 11 | 4.2e-4 | 17 | 7.7e-4 |
| 17 | 55 | 1.6e-4 | 58 | 2.5e-4 | 13 | 2.4e-4 | 19 | 4.3e-4 |
| 25 | 61 | 7.2e-5 | 64 | 1.0e-4 | 15 | 9.8e-5 | 21 | 1.8e-4 |

safe thing to do in all cases, however.

Finally, we consider the effect of refining the grid on the number of Schwarz iterations. Figure 4 shows that for the most part, the number of Schwarz iterations rises quite slowly with respect to the number of grid lines. The independent variable in these plots is $1/h$, where $h$ is the grid spacing. We do not yet understand the nonsmooth behavior for $2 \times 2$ and $3 \times 3$ subdomains using Method 2.

## 2.3 Example 2

As a second example we consider the Helmholtz equation

$$u_{xx} + u_{yy} - u = f(x, y).$$

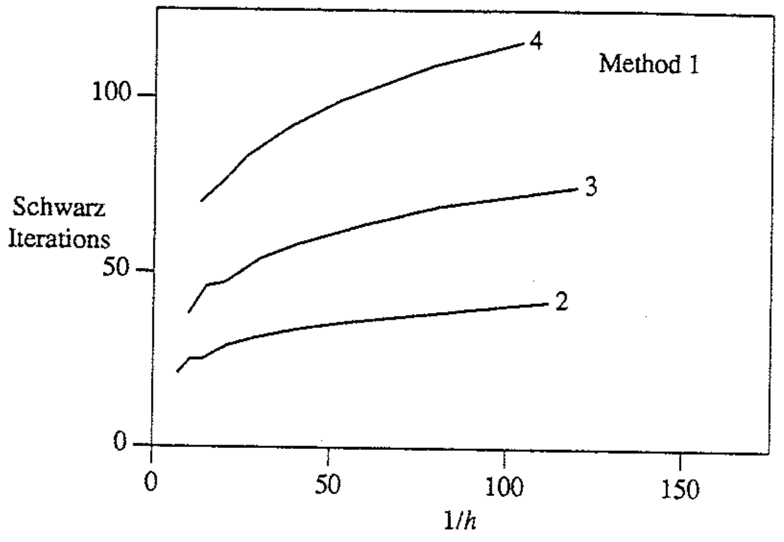
Again we impose Dirichlet boundary conditions on the unit square, and select $f$ so that the true solution is the same as for Example 1. Very fast elliptic solvers may be applied to this problem. We use ELLPACK module HODIE FFT with a 4th order accurate discretization. We compare three variations of Schwarz splitting for this problem:

**Method 1.** Subdomain solves are done with ELLPACK modules 5 POINT STAR, RED BLACK, and REDUCED SYSTEM CG (as in Example 1); the outer iteration is red-black sor.

**Method 2.** Subdomain solves are done with ELLPACK module HODIE FFT; the outer iteration is jacobi.

**Method 3.** Subdomain solves are done with ELLPACK module HODIE FFT; the outer iteration is red-black sor.

The first set of plots (see Figure 5) show log(*error*) vs. log(*ptime*) for each method. We plot one curve for each number of subdomains. Also plotted are the curves for the no-splitting cases: one using 5 POINT STAR (F) and one using HODIE FFT (H). We see that Method 1 improves very slightly as the number of pieces increases, to the point where it may be competitive with the no-splitting 5 POINT STAR curve; but it is no where near as efficient as HODIE FFT with no splitting. Method 2 shows no improvement at all as the number of subdomains increases from two to four. In fact, the performance appears to degrade a little. Method 3 shows no significant improvement



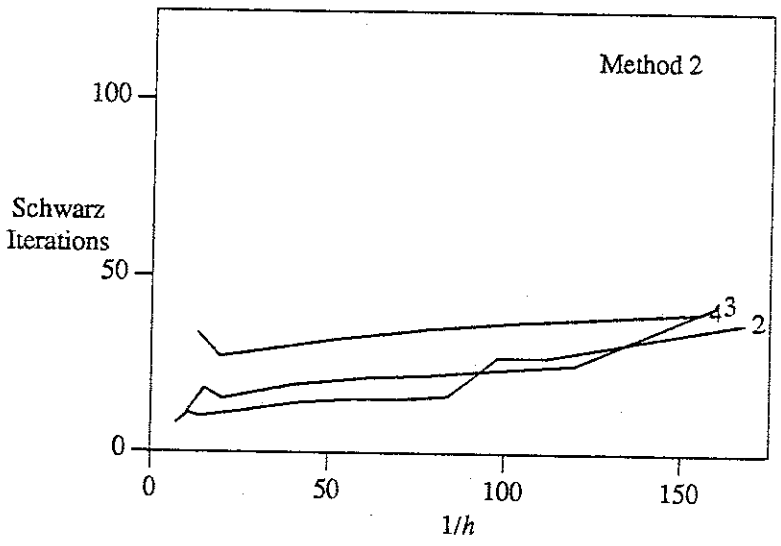


Figure 4: Plot of Schwarz iterations vs. $1/h$ for each method on Example 1. The curves are labeled by the number of subdomains in each direction.

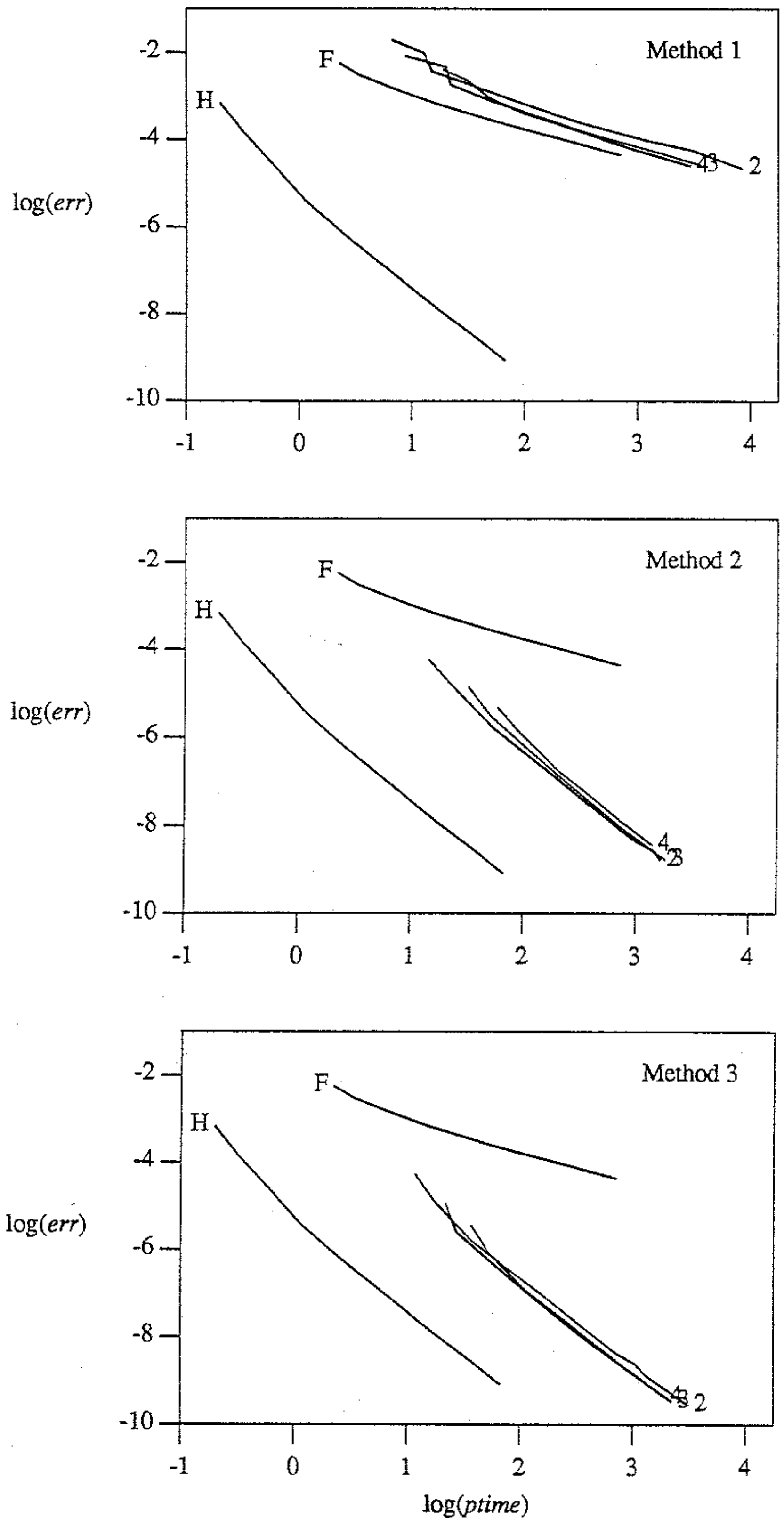


Figure 5: Log-log plot of error vs. time for the three methods on Example 2. The curves are labeled by the number of subdomains in each direction.

either, and needs a great deal of improvement before it could be considered competitive with HODIE FFT with no splitting.

Figure 6 contains a set of plots which compare the three methods, for a given number of subdomains. As expected, in all three cases Methods 2 and 3 are preferable to Method 1; and it appears Method 3 is slightly better than Method 2. With the Hodie methods, we did not encounter the problem described in the previous section where too many Schwarz iterations caused the accuracy to get worse. Instead, the solution seemed to converge and then remain essentially stationary as more Schwarz iterations were done. Method 1 does exhibit this problematic behavior, however.

Finally, the plots in Figure 7 show again that the number of Schwarz iterations rises fairly slowly with respect to increasing numbers of grid lines. Notice the dramatic decrease in the number of Schwarz iterations as we switch from Method 2 to Method 3.

## 2.4 Other issues

In this section we discuss several other issues which pertain to our investigation.

### 2.4.1 Complexity analysis

It is often instructive to use computational complexity analysis to estimate the relative performance of numerical methods. Consider first the choice of linear system solution method for the subproblems if the standard 5-point finite difference discretization is used. Suppose we have an $n \times n$ grid on each subdomain. Gauss elimination may be used to factor the banded system (bandwidth $= 2n+1$) at a cost of $O(n^4)$ operations. With appropriate changes to our implementation, this factorization could be done only once. At each outer iteration of the Schwarz scheme only the right side of the linear equation is changed. The cost of a forward and back solve at each iteration is $O(n^3)$. Hence the total cost for $k$ Schwarz iterations is $O(n^4 + kn^3)$ per subdomain.

By comparison, if an iterative method such as SOR is used to solve the linear systems at the inner iteration, we have $O(5n^2)$ operations per SOR iteration (we use Reduced System CG in the examples above; its overall performance is usually better than SOR). Suppose we do $p$ SOR iterations per subdomain, and $k$ Schwarz iterations in all. The total cost per subdomain is then $O(5kpn^2)$. An estimate of the required number of SOR iterations $p$ is $1.1n$ (see Rice and Boisvert [8]). Hence, an iterative method is likely to be better if

$$5.5kn^3 < n^4 + kn^3$$

or

$$5.5k < n + k.$$

If $k$ were a constant with respect to $n$, then this analysis would suggest that as $n$ grows, the iterative method would be faster. We cannot treat $k$ as a constant with respect to $n$ however, since the number of Schwarz iterations may grow as $n$ does. We conjecture however, that $k$ grows much more slowly than $n$; and in fact our experiments so far have born this conjecture out. If $k$ does grow quite slowly with respect to $n$, then the analysis still favors the iterative methods, especially for very large $n$.

A second useful complexity analysis deals with using a fast direct method such as HODIE FFT for the inner solution. This is not always possible, of course, but in problems where fast methods do apply, the question is: "can Schwarz splitting ever be effective compared to simply solving the

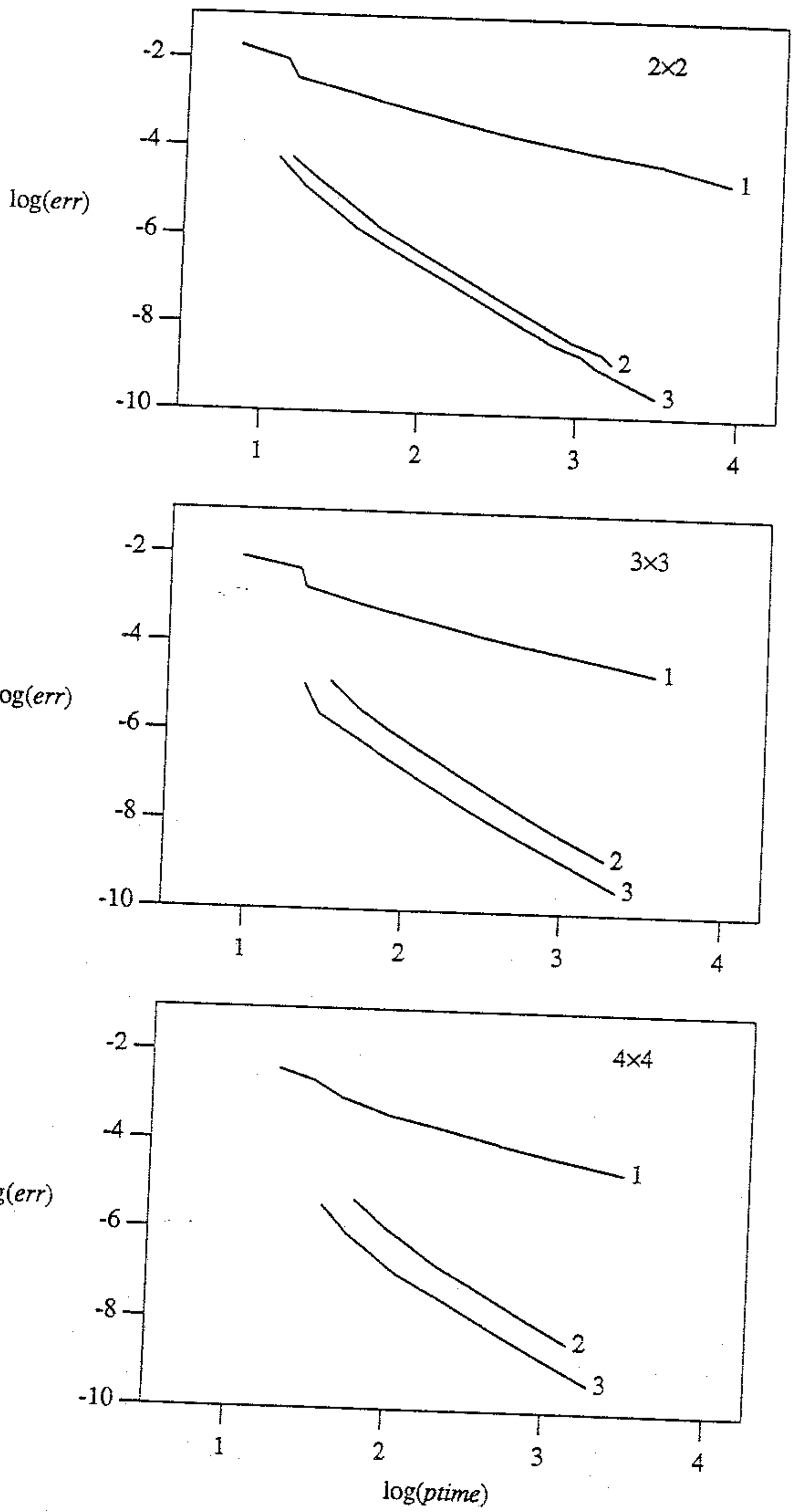


Figure 6: Log-log plot of error vs. time for 2, 3, and 4 subdomains in each direction. The curves are labeled by method number.

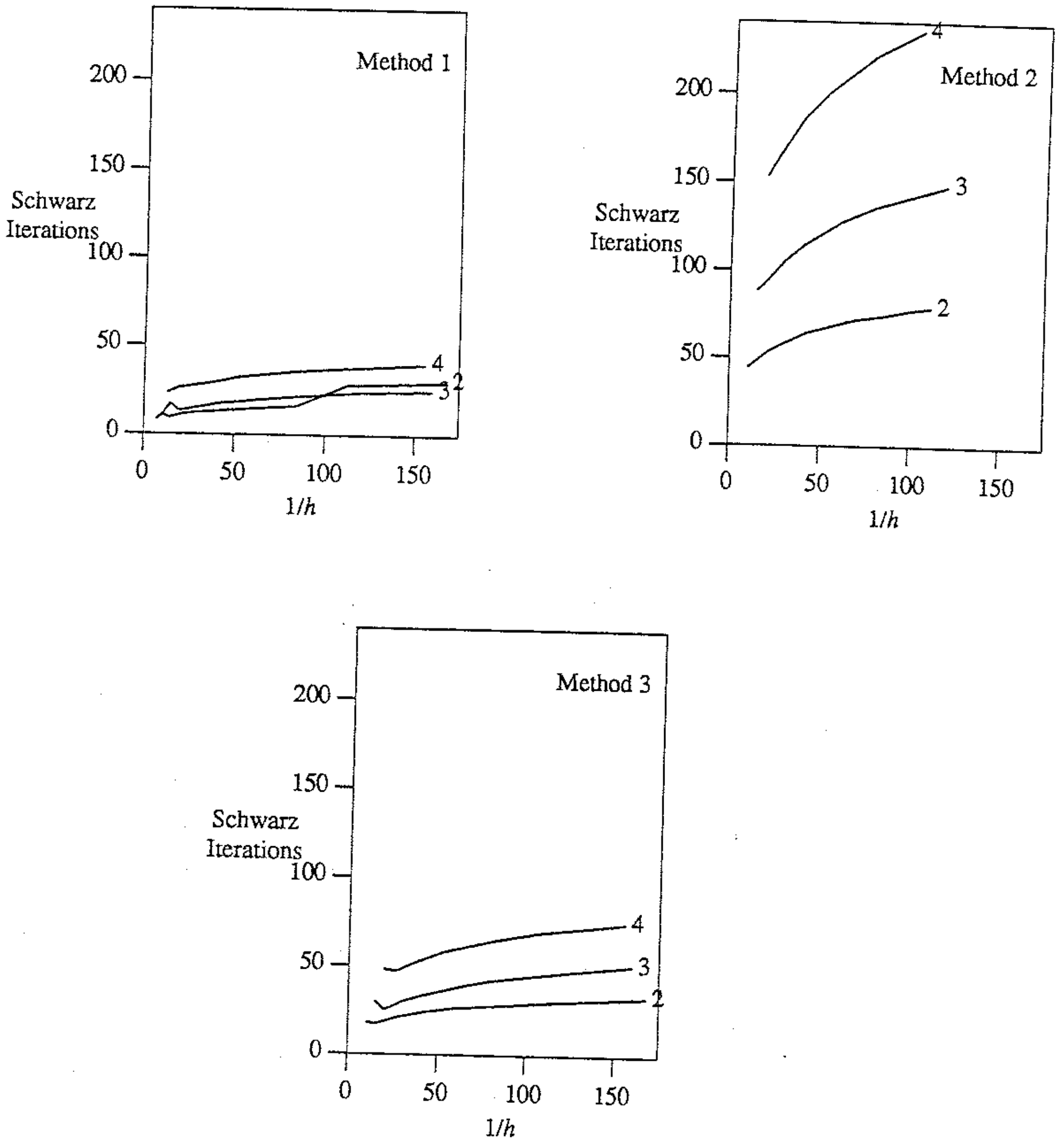


Figure 7: Plot of Schwarz iterations vs. $1/h$ for each method on Example 2. The curves are labeled by the number of subdomains in each direction.

problem on the original domain (with no splitting) with a sufficiently fine grid?". Suppose we solve the PDE without splitting using an $n \times n$ grid. HODIE FFT is expected to take time proportional to $n^2 \log n$. If we split the domain into $m \times m$ subdomains, and define an $n/m \times n/m$ grid on each subdomain (so that the grid spacing is approximately the same as in the no-splitting case), then the time for HODIE FFT on each subdomain is $O(k(n/m)^2 \log(n/m))$, where $k$ is the number of Schwarz iterations. Assuming perfect parallelism (all subdomain solves done in parallel) the splitting approach is best if

$$k(n/m)^2(\log n/m) < n^2 \log n$$

or

$$k < m^2 \frac{\log n}{\log n - \log m}.$$

For typical choices of $n$ and $m$ ($n \gg m$) the expression on the right in this inequality is only slightly larger than $m^2$. So to be preferable, the Schwarz iteration must converge in about as many iterations as there are subdomains. We have not been able to achieve this kind of rapid convergence so far.

### 2.4.2 Stopping criterion for outer iteration

A difficult question for any iterative numerical method is a stopping criterion. Our initial approach was to stop when the maximum relative change in the solution was less than some fixed threshold *eps*. Meier [6] uses this approach. Other researches have used reduction in the residual as a criterion, but the residual is notorious for giving misleading information and our attempts at using it were very unsatisfactory. The problem with picking a single threshold for the relative change in $U$ is that it may not be fair to compare two methods using one fixed *eps*. For example, if for method A the stopping criterion is such that the iteration stops just after discretization error is reached, while for method B the iteration continues until the change in $U$ is two orders of magnitude smaller than the discretization error, then we unfairly penalize method B by our choice of *eps*.

Our approach is to use an "optimal" *eps* for each method/grid combination. With this choice we lose something in terms of what is realistically possible in a real-world problem, but we remove the misleading influence of a fixed stopping threshold. We are primarily interested in comparing Schwarz splitting methods, not choices for *eps*. By "optimal" we mean that *eps* is chosen as $0.1||e||$, where $||e||$ is an estimate of the size of the discretization error. We estimate $||e||$ by assuming $||e|| = Ch^p$, where $C$ is a constant, $h$ is the mesh size, and $p$ is a known constant which depends on the discretization method. We estimate $C$ by solving the PDE initially on a few coarse grids without splitting, and using the known true solution to our test problems. If the true solution is not available, one could solve the problem on a coarse grid and on one twice as fine, and use the change in the solution to estimate the discretization error.

### 2.4.3 Stopping criterion for REDUCED SYSTEM CG

The choice of stopping tolerance *zeta* for ITPACK module REDUCED SYSTEM CG has a considerable effect on the number of iterations needed to converge. We choose the reduced system stopping tolerance as

$$\mathit{zeta} = \min(0.001, 0.01 \max(\mathit{eps}, \mathit{dfmin})),$$

where

*eps* = convergence threshold for outer iteration

*dfmin* = minimum relative change in the unknowns in the last two Schwarz iterations

The reasoning behind this choice of *zeta* is as follows. The requirement $zeta \leq 0.001$ ensures that the tolerance will never get too big. (Obviously, these numbers need to be scaled by the magnitude of the solution; for our examples the true solution is of size 1). Further, $zeta \geq 0.01eps$ because there is no point in solving a subdomain problem to several orders of magnitude more accuracy than you expect based on the discretization error. Finally, in the range $0.01eps \leq zeta \leq 0.001$, *zeta* gets smaller as *dfmin* does. The idea is that *zeta* can be fairly large during the early Schwarz iterations because the solution is so far from the final answer, and is changing significantly from iteration to iteration. As we near convergence of the outer iteration, *zeta* must get smaller, because we want to solve the subproblems more and more accurately. At each step we must solve the subdomain problems with enough accuracy to make sure the Schwarz iteration continues toward convergence.

### 2.4.4 Relaxation factor for red-black sor outer iteration

An optimal relaxation factor $\omega$ is known for linear systems resulting from discretizing many simple linear elliptic operators (see Hageman and Young [3]). However, for the red-black sor outer iteration, the choice of $\omega$ is not so well-founded. Meier [6] uses the optimal $\omega$ associated with the PDE and gets good results. We experimented with using the $\omega$ chosen by the ELLPACK SOR module for solving the linear system at the inner iteration, but this did not yield the experimentally observed best $\omega$. In the examples above we use $\omega = 1.25$ for $2 \times 2$ subdomains, and $\omega = 1.4$ for $3 \times 3$ and $4 \times 4$ subdomains. These are values that we observe to be nearly optimal. We have no theoretical basis for our choices at this time. We plan to investigate this question more in the future.

## 3 Future work

There are many aspects of our investigation of Schwarz splitting which need further work. The most important next step is probably to implement our scheme on a parallel computer. We plan to develop a parallelized version of our Schwarz splitting ELLPACK program on a Sequent Balance 21000. This machine is a shared memory design. The configuration we have available has 12 processors. This should give us just enough realistic parallel computing power to investigate the speed-ups achievable with Schwarz splitting. We plan to implement a prototype parallel version of ELLPACK as part of this investigation. This version of ELLPACK should make the handling of the multiple processors and processes as transparent as possible to the user. Obviously, this work has implications beyond Schwarz splitting alone.

There are several improvements that could be made to our present implementation. For one, the unnecessary discretization steps mentioned above should be avoided. This will take a bit more programming and ELLPACK expertise than we were willing to expend on our initial implementation, but it should be looked at soon. Furthermore, a better understanding of such problem parameters as overlap and the number of subdomains is needed. We also plan to investigate the effect of the elliptic operator on the performance of Schwarz splitting. For example, do non-smooth coefficients degrade the convergence behavior of the outer iteration?

Finally, and most importantly, we plan to pursue other possibilities for speeding up the algorithm as a whole. We have looked already at using fast direct methods for the inner iteration. This needs more attention. Other discretization methods (eg. collocation) should also be considered.

The outer iteration too, needs to be improved, possibly by applying a conjugate-gradient-like acceleration scheme, or different subdomain coloring strategies. We plan to investigate these possibilities in the near future. We believe the Schwarz splitting approach can be improved enough to be very competitive as a parallel algorithm. Many researchers have concentrated on capacitance matrix methods (often called "domain decomposition" methods) for splitting domains. It is not clear at this point whether the Schwarz splitting approach is better or worse (in terms of ultimate parallel performance) than domain decomposition. Both theoretical and experimental analysis is needed.

## References

[1] P. E. Bjorstad and O. B. Widlund, "Solving elliptic problems on regions partitioned into substructures", in *Elliptic Problem Solvers II*, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp 245–255.

[2] T. F. Chan and D. C. Resasco, "A domain-decomposed fast poisson solver on a rectangle", *SIAM J. Sci. Stat. Comput.*, 8(1987), pp s14–s26.

[3] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

[4] L. V. Kantorovich and V. I. Krylov, *Approximate Methods of Higher Analysis*, Interscience, New York, 1964.

[5] D. E. Keyes and W. D. Gropp, "A comparison of domain decomposed techniques for elliptic pdes and their parallel implementation", *SIAM J. Sci. Stat. Comput.*, 8(1987), pp s166–s202.

[6] U. Meier, "Two parallel SOR variants of the Schwarz alternating procedure", *Parallel Comput.*, 3(1986), pp 205–215.

[7] K. Miller, "Numerical analogs to the Schwarz alternating procedure", *Numer. Math.*, 7(1965), pp 91–103.

[8] J. R. Rice and R. F. Boisvert, *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, New York, 1985.

[9] J. R. Rice and E. N. Houstis and W. R. Dyksen, "A population of linear, second order elliptic partial differential equations on rectangular domains", *Math. Comp.*, 36(1981), pp 479–484.

[10] J. R. Rice and D. C. Marinescu, "Analysis and modeling of Schwartz splitting algorithms for elliptic pde's" in *Advances in Computer Methods for Partial Differential Equations VI*, R. Vichnevetsky and R.S. Stepleman, eds., IMACS, New Brunswick, N.J., 1987, pp 383-386.

[11] G. Rodrigue, "Inner/outer iterative methods and numerical Schwarz algorithms", *Parallel Comput.*, 2(1985), pp 205–218.

[12] G. Rodrigue and J. Simon, "Jacobi splittings and the method of overlapping domains for solving elliptic pdes", in *Advances in Computer Methods for Partial Differential Equations V*, R. Vichnevetsky and R.S. Stepleman, eds., IMACS, Brussels, 1984, pp 383-386.

[13] G. Rodrigue and J. Simon, "A generalization of the numerical Schwarz algorithm", in *Computing Methods in Applied Sciences and Engineering VI*, R. Glowinski and J.L. Lions, eds., North-Holland, Amsterdam, 1984, pp 273-283.

[14] W. P. Tang, "Schwarz splitting: a model for parallel computations", Ph.D. thesis, Stanford University, 1987.

# Appendices

## A ELLPACK program

In this appendix we include a listing of the ELLPACK program that implements Schwarz splitting. This particular program is for Example 1, Method 2. We do not show several FORTRAN functions and subroutines that must also be supplied. The comments in this program indicate what these subprograms do.

```
*********************************************************
*
* Schwarz splitting program.
*
*  inner iteration: 5ps + reduced system cg
*  outer iteration: red-black sor (user picks omega)
*
*********************************************************

*********************************************************
*
*    options
*
*     output level 0
*     max x and y grid lines
*
*********************************************************

options.
      level = 0
      max x points = 65
      max y points = 65

*********************************************************
*
*    declarations
*
*     maxpcx,maxpcy     max pieces in each direction
*     maxpcs            max total number of pieces
*     maxits            max number of iterations
*     npcs              current number of pieces
*     npcsx, npcsy      number of pieces in each direction
*     kpc               current piece
*     nxmax, nymax      max number of x and y grid lines
*     kordmx            max order of spline interpolant
*     intype            interpolation type (0=quad, 1=splines)
*     unkcur            saved unknowns for each piece (current)
*     unkold            saved unknowns for each piece (old)
*     xmin, xmax         x grid ranges
*     ymin, ymax         y grid ranges
*     ngrdx, ngrdy       number of grid points for each piece
*     hgrdx, hgrdy       grid spacing for each piece
*     ugrdx, ugrdy       uniform grid switch for each piece
*     gridx, gridy       x and y grid lines for each piece
*
*    other variables
*     eps                epsilon for stopping criterion
*     errcon             estimate of constant c such that
*                           err = c*h**2
*     z                  estimate for itpack parameter zeta
*
*****************************************************

declarations.
      parameter (maxpcx=4,
     +           maxpcy=4,
     +           maxpcs=16,
     +           maxits=100)
      common / csplit / npcs, npcsx, npcsy, kpc, nxmax, nymax,
     a                  kordmx, intype
      common / cunkcu / unkcur($i1ngrx,$i1ngry,maxpcs)
      common / cunkol / unkold($i1ngrx,$i1ngry,maxpcs)
      common / csplax / xmin(maxpcx)
      common / csplbx / xmax(maxpcx)
      common / csplay / ymin(maxpcy)
      common / csplby / ymax(maxpcy)
      common / csplnx / ngrdx(maxpcs)
      common / csplny / ngrdy(maxpcs)
      common / csplhx / hgrdx(maxpcs)
      common / csplhy / hgrdy(maxpcs)
      common / csplux / ugrdx(maxpcs)
      common / cspluy / ugrdy(maxpcs)
      logical           ugrdx, ugrdy
      common / cgridx / gridx($i1ngrx,maxpcs)
      common / cgridy / gridy($i1ngry,maxpcs)
      integer           iters(maxpcs)

equation.  (exp(x*y)ux)x + (exp(-x*y)uy)y - 1./(1.+x+y)u = f(x,y)

boundary.   u = gwest(x,y)  on x = 0.
            u = geast(x,y)  on x = 1.
            u = gsouth(x,y) on y = 0.
            u = gnorth(x,y) on y = 1.
```

```
fort.
      nxmax  = $i1ngrx
      nymax  = $i1ngry
      intype = 0
      errcon = 0.67

c
c     Input parameters
c
      print *, 'Input number of pieces in x and y'
      read *, npcsx, npcsy
      npcs = npcsx * npcsy
      print *, 'Input overlap factor p'
      read *, p
      print *, 'Input initial grid size (nx,ny)'
      read *, nx, ny
      print *, 'Input number of grids'
      read *, ngrids
      print *, 'Input omega'
      read *, xomega

      print 900, npcsx, npcsy, p, xomega
900   format(//'Number of pieces in x =', i3
     a        /'Number of pieces in y =', i3
     b        /'Overlap factor        =', f8.3
     c        /'Relaxation parameter  =', f8.3)

c
c     Set piece boundaries.
c
      call initpc (npcsx, p, xmin, xmax)
      call initpc (npcsy, p, ymin, ymax)

c
c     Big loop over grids
c
      do 300 ig = 1, ngrids
        soltim = 0.0
        call q1time(t0)
        nxny = nx * ny
        print 1000, nx, ny
1000    format(///' New grid size =', 2i3
     a          //' iter   maxdiffu       zeta',
     b           '    reduced system iterations')
```

```
c
c       Define and save grids. Initialize solution to 0.
c       Pieces are numbered left to right, bottom to top.
c
        do 100 j = 1, npcsy
        do 100 i = 1, npcsx
          kpc = i + (j-1)*npcsx
          r1axgr = xmin(i)
          r1bxgr = xmax(i)
          r1aygr = ymin(j)
          r1bygr = ymax(j)
c
c         define grid and initialize solution to zero
c
grid.     nx x points $ ny y points
triple.   set(u=zero)
fort.

c
c         save unknowns and grid
c
          call unksav (r1tabl, nxny, unkold(1,1,kpc))
          call grdsav (kpc, gridx(1,kpc), gridy(1,kpc))
  100   continue

c
c       set stopping criterion eps as function of h**2
c       since 5ps is 2nd order method.
c
        h = amax1(hgrdx(1),hgrdy(1))
        eps = 0.1 * errcon * h*h
        print 1100, eps
 1100   format(' eps =', e11.3)

c
c       schwarz iteration until change in u is less than eps
c
        iter = 0
        dfmin = 1.0
   50   continue
          iter = iter + 1
          z = amin1(0.01*amax1(eps,dfmin),1.0e-3)
          dfmax = 0.0
```

```
c
c         red black loop
c
          do 210 irdbl = 0, 1
c
c           loop over pieces
c
            do 200 kpc = 1, npcs
              j = (kpc-1)/npcsx + 1
              i = kpc - (j-1)*npcsx
              if (mod(i+j,2) .ne. irdbl) goto 200
              call grdres (kpc, gridx(1,kpc), gridy(1,kpc))
              call q1time(tt0)

disc.         5 point star
proc.         set unknowns for 5 point star (uest=uold)
ind.          red black
sol.          reduced system cg (zeta=z)

fort.
              call q1time(tt1)
              soltim = soltim + tt1 - tt0
              iters(kpc) = iparm(1)
              call q35pvl
              l1newd = .false.

c
c             do sor step (puts new answer in unkcur)
c
              call sor (xomega, unkcur(1,1,kpc), r1tabl,
     a                 unkold(1,1,kpc), nxny)

c
c             compute max change in u on this piece
c             and update max and min over all pieces
c
              dfu = diffu (unkcur(1,1,kpc), unkold(1,1,kpc), nxny)
              dfmax = amax1(dfmax,dfu)
              dfmin = amin1(dfmax,dfu)
  200     continue
```

```
c
c           copy red (black) unknowns from unkcur to unkold
c
            do 205 kpc = 1, npcs
              j = (kpc-1)/npcsx + 1
              i = kpc - (j-1)*npcsx
              if (mod(i+j,2) .ne. irdbl) goto 205
              call unksav (unkcur(1,1,kpc), nxny, unkold(1,1,kpc))
  205       continue

  210     continue

          print 1030, iter, dfmax, z, (iters(k), k = 1, npcs)
 1030     format(2x,i3,2e12.4,x,16i4)

c
c         check for convergence
c
          if ((dfmax .gt. eps) .and. (iter .lt. maxits)) goto 50

        call q1time(t1)

        print 1020, iter, t1-t0, soltim
 1020   format(/'  iterations    ',i8
     a         /'  total time    ',f8.2
     b         /'  solution time',f8.2)
c
c       put grid on entire domain and evaluate error
c
        r1axgr = 0.0
        r1bxgr = 1.0
        r1aygr = 0.0
        r1bygr = 1.0

grid.   20 x points $ 20 y points
out.    max(abserr,40,40)
fort.
        nx = 2*nx-1
        ny = 2*ny-1
  300 continue
```

```
subprograms.
      function true(x,y)
      common  / c1rvgl /  r1epsg, r1epsm, pi
      true = .75*exp(x*y)*sin(pi*x)*sin(pi*y)
      return
      end

      function f(x,y)
      common  / c1rvgl /  r1epsg, r1epsm, pi
      px = pi*x
      py = pi*y
      spx = sin(px)
      spy = sin(py)
      exy = exp(x*y)
      f = .75*(exy*exy*spy*((2.*y*y-pi*pi)*spx+3.*pi*y*cos(px))+
     $    pi*spx*(x*cos(py)-pi*spy)-exy*spx*spy/(1.+x+y))
      return
      end
end.
```

## B Experimental data

Table B.1: Data for Example 1, Methods 1 and 2. Column N is the number of grid lines in each direction per subdomain, SI is the number of Schwarz iterations, ST is sequential time, PT is parallel time, Error is the maximum relative error over a 40 × 40 grid on each subdomain.

2 × 2 Subdomains

| | Method 1 | | | | Method 2 | | | |
|---|---|---|---|---|---|---|---|---|
| N | SI | ST | PT | Error | SI | ST | PT | Error |
| 5 | 21 | 36 | 9 | 1.9e-02 | 8 | 14 | 7 | 2.0e-02 |
| 7 | 25 | 66 | 17 | 8.6e-03 | 11 | 29 | 15 | 9.3e-03 |
| 9 | 25 | 97 | 24 | 3.3e-03 | 10 | 38 | 19 | 3.7e-03 |
| 13 | 29 | 230 | 58 | 1.5e-03 | 11 | 86 | 43 | 1.6e-03 |
| 17 | 31 | 423 | 106 | 7.9e-04 | 12 | 162 | 81 | 8.8e-04 |
| 25 | 34 | 1108 | 277 | 3.3e-04 | 14 | 442 | 221 | 3.9e-04 |
| 33 | 36 | 2239 | 560 | 1.8e-04 | 15 | 894 | 447 | 2.1e-04 |
| 49 | 39 | 6691 | 1673 | 8.0e-05 | 16 | 2583 | 1292 | 9.2e-05 |
| 65 | 42 | 14686 | 3672 | 4.6e-05 | 27 | 7164 | 3582 | 4.8e-05 |
| 97 | - | - | - | - | 37 | 23953 | 11977 | 2.3e-05 |

3 × 3 Subdomains

| | Method 1 | | | | Method 2 | | | |
|---|---|---|---|---|---|---|---|---|
| N | SI | ST | PT | Error | SI | ST | PT | Error |
| 5 | 38 | 147 | 16 | 6.7e-03 | 11 | 42 | 9 | 8.6e-03 |
| 7 | 46 | 277 | 31 | 3.2e-03 | 18 | 106 | 24 | 4.5e-03 |
| 9 | 47 | 415 | 46 | 1.2e-03 | 15 | 129 | 29 | 1.8e-03 |
| 13 | 54 | 979 | 109 | 5.1e-04 | 17 | 300 | 67 | 7.7e-04 |
| 17 | 58 | 1821 | 202 | 2.5e-04 | 19 | 577 | 128 | 4.3e-04 |
| 25 | 64 | 4782 | 531 | 1.0e-04 | 21 | 1507 | 335 | 1.8e-04 |
| 33 | 69 | 9837 | 1093 | 6.0e-05 | 22 | 2993 | 665 | 1.0e-04 |
| 49 | 75 | 29409 | 3268 | 2.7e-05 | 25 | 8918 | 1982 | 4.4e-05 |
| 65 | - | - | - | - | 42 | 25828 | 5740 | 2.5e-05 |

4 × 4 Subdomains

| | Method 1 | | | | Method 2 | | | |
|---|---|---|---|---|---|---|---|---|
| N | SI | ST | PT | Error | SI | ST | PT | Error |
| 5 | 70 | 488 | 31 | 3.0e-03 | 34 | 233 | 29 | 4.5e-03 |
| 7 | 76 | 821 | 51 | 1.4e-03 | 27 | 286 | 36 | 2.3e-03 |
| 9 | 83 | 1331 | 83 | 4.5e-04 | 28 | 441 | 55 | 9.5e-04 |
| 13 | 92 | 3013 | 188 | 1.4e-04 | 30 | 964 | 121 | 3.8e-04 |
| 17 | 99 | 5656 | 354 | 9.1e-05 | 32 | 1787 | 223 | 2.1e-04 |
| 25 | 109 | 14863 | 929 | 3.4e-05 | 35 | 4619 | 577 | 9.1e-05 |
| 33 | 116 | 30267 | 1892 | 2.0e-05 | 37 | 9275 | 1159 | 5.1e-05 |
| 49 | - | - | - | - | 40 | 26859 | 3357 | 2.2e-05 |

Table B.2: Data for Example 2, Methods 1, 2 and 3.

2 × 2 Subdomains

| | Method 1 | | | | Method 2 | | | | Method 3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | SI | ST | PT | Error | SI | ST | PT | Error | SI | ST | PT | Error |
| 7 | 11 | 26 | 13 | 9.6e-03 | 44 | 58 | 15 | 5.8e-05 | 18 | 24 | 12 | 5.5e-05 |
| 9 | 9 | 30 | 15 | 3.6e-03 | 47 | 95 | 24 | 1.4e-05 | 17 | 35 | 18 | 1.3e-05 |
| 13 | 11 | 73 | 37 | 1.6e-03 | 53 | 209 | 52 | 1.8e-06 | 19 | 76 | 38 | 1.6e-06 |
| 17 | 12 | 134 | 67 | 9.1e-04 | 57 | 364 | 91 | 5.7e-07 | 21 | 136 | 68 | 4.9e-07 |
| 25 | 13 | 343 | 172 | 4.0e-04 | 64 | 859 | 215 | 1.0e-07 | 24 | 326 | 163 | 8.9e-08 |
| 33 | 14 | 707 | 354 | 2.2e-04 | 68 | 1539 | 385 | 3.1e-08 | 26 | 596 | 298 | 2.6e-08 |
| 49 | 16 | 2197 | 1099 | 9.9e-05 | 74 | 3622 | 906 | 5.4e-09 | 28 | 1387 | 694 | 4.5e-09 |
| 65 | 28 | 6385 | 3193 | 5.5e-05 | 79 | 6655 | 1664 | 1.7e-09 | 30 | 2559 | 1280 | 1.4e-09 |
| 97 | 30 | 17102 | 8551 | 2.2e-05 | - | - | - | - | 33 | 6295 | 3148 | 2.8e-10 |

3 × 3 Subdomains

| | Method 1 | | | | Method 2 | | | | Method 3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | SI | ST | PT | Error | SI | ST | PT | Error | SI | ST | PT | Error |
| 7 | 17 | 92 | 20 | 4.6e-03 | 88 | 293 | 33 | 1.4e-05 | 30 | 101 | 22 | 1.1e-05 |
| 9 | 13 | 99 | 22 | 1.8e-03 | 93 | 460 | 51 | 3.2e-06 | 25 | 125 | 28 | 2.5e-06 |
| 13 | 15 | 225 | 50 | 7.7e-04 | 105 | 997 | 111 | 5.4e-07 | 30 | 288 | 64 | 3.8e-07 |
| 17 | 17 | 438 | 97 | 4.4e-04 | 114 | 1741 | 193 | 1.6e-07 | 33 | 509 | 113 | 1.1e-07 |
| 25 | 19 | 1156 | 257 | 1.9e-04 | 127 | 4011 | 446 | 2.7e-08 | 38 | 1214 | 270 | 1.8e-08 |
| 33 | 21 | 2446 | 544 | 1.0e-04 | 136 | 7177 | 797 | 8.1e-09 | 42 | 2242 | 498 | 5.4e-09 |
| 49 | 24 | 7638 | 1697 | 4.7e-05 | 148 | 16736 | 1860 | 1.7e-09 | 47 | 5378 | 1195 | 1.1e-09 |
| 65 | 25 | 16201 | 3600 | 2.6e-05 | - | - | - | - | 51 | 9988 | 2220 | 3.4e-10 |

4 × 4 Subdomains

| | Method 1 | | | | Method 2 | | | | Method 3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | SI | ST | PT | Error | SI | ST | PT | Error | SI | ST | PT | Error |
| 7 | 26 | 253 | 32 | 2.2e-03 | 153 | 954 | 60 | 4.8e-06 | 48 | 301 | 38 | 3.6e-06 |
| 9 | 27 | 376 | 47 | 9.6e-04 | 165 | 1531 | 96 | 1.2e-06 | 47 | 440 | 55 | 7.9e-07 |
| 13 | 29 | 794 | 99 | 3.9e-04 | 186 | 3269 | 204 | 1.8e-07 | 53 | 937 | 117 | 1.1e-07 |
| 17 | 32 | 1503 | 188 | 2.3e-04 | 200 | 5614 | 351 | 6.2e-08 | 58 | 1645 | 206 | 3.7e-08 |
| 25 | 35 | 3870 | 484 | 1.0e-04 | 221 | 12692 | 793 | 1.1e-08 | 64 | 3701 | 463 | 6.9e-09 |
| 33 | 37 | 7892 | 987 | 5.6e-05 | 235 | 22485 | 1405 | 3.7e-09 | 69 | 6660 | 833 | 2.2e-09 |
| 49 | 40 | 23617 | 2952 | 2.5e-05 | - | - | - | - | 75 | 15386 | 1923 | 4.4e-10 |

Table B.3: Data for Example 1 with no splitting. ELLPACK modules used are 5 POINT STAR, RED BLACK, and REDUCED SYSTEM CG. N is the number of grid lines in each direction. Time is given in seconds. Error is estimated by the maximum relative error over a 40 × 40 grid.

| N | Time | Error |
|---|---|---|
| 5 | 1.3 | 8.67e-2 |
| 7 | 1.5 | 2.91e-2 |
| 9 | 1.7 | 1.31e-2 |
| 13 | 2.6 | 5.63e-3 |
| 17 | 4.0 | 2.99e-3 |
| 25 | 9.7 | 1.23e-3 |
| 33 | 19.2 | 6.85e-4 |
| 49 | 55.1 | 2.96e-4 |
| 65 | 121.2 | 1.64e-4 |
| 97 | 385.7 | 7.30e-5 |
| 129 | 885.5 | 4.08e-5 |

Table B.4: Data for Example 2 with no splitting, using two methods: 5 POINT STAR + RED-BLACK + REDUCED SYSTEM CG, and HODIE FFT.

| | 5 POINT STAR | | HODIE FFT | |
|---|---|---|---|---|
| N | Time | Error | Time | Error |
| 7 | 1.4 | 2.94e-2 | 0.2 | 6.77e-04 |
| 9 | 1.6 | 1.31e-2 | 0.3 | 1.49e-04 |
| 13 | 2.2 | 5.64e-3 | 0.7 | 1.84e-05 |
| 17 | 3.4 | 3.02e-3 | 1.1 | 4.11e-06 |
| 25 | 7.7 | 1.30e-3 | 2.5 | 6.77e-07 |
| 33 | 15.3 | 6.98e-4 | 4.3 | 2.18e-07 |
| 49 | 44.0 | 3.07e-4 | 9.7 | 4.18e-08 |
| 65 | 96.7 | 1.74e-4 | 16.8 | 1.33e-08 |
| 97 | 310.8 | 7.72e-5 | 39.2 | 2.62e-09 |
| 129 | 714.7 | 4.28e-5 | 67.9 | 8.28e-10 |